A Modular Architecture For Systematic Text Categorisation

Andrew Barnes

A thesis submitted to the University of Huddersfield in partial fulfilment
for the degree of Doctor of Philosophy

May 2013

Copyright Statement

Abstract

This work examines and attempts to overcome issues caused by the lack of formal standardisation when defining text categorisation techniques and detailing how they might be appropriately integrated with each other. Despite text categorisation's long history the concept of automation is relatively new, coinciding with the evolution of computing technology and subsequent increase in quantity and availability of electronic textual data. Nevertheless insufficient descriptions of the diverse algorithms discovered have lead to an acknowledged ambiguity when trying to accurately replicate methods, which has made reliable comparative evaluations impossible.

Existing interpretations of general data mining and text categorisation methodologies are analysed in the first half of the thesis and common elements are extracted to create a distinct set of significant stages. Their possible interactions are logically determined and a unique universal architecture is generated that encapsulates all complexities and highlights the critical components. A variety of text related algorithms are also comprehensively surveyed and grouped according to which stage they belong in order to demonstrate how they can be mapped.

The second part reviews several open-source data mining applications, placing an emphasis on their ability to handle the proposed architecture, potential for expansion and text processing capabilities. Finding these inflexible and too elaborate to be readily adapted, designs for a novel framework are introduced that focus on rapid prototyping through lightweight customisations and reusable atomic components.

Being a consequence of inadequacies with existing options, a rudimentary implementation is realised along with a selection of text categorisation modules. Finally a series of experiments are conducted that validate the feasibility of the outlined methodology and importance of its composition, whilst also establishing the practicality of the framework for research purposes. The simplicity of experiments and results gathered clearly indicate the potential benefits that can be gained when a formalised approach is utilised.

Dedication

This work is dedicated to my partner and parents; my only links to the outside world during my dark days of eternal typing.

... And also that voice in my head for providing entertainment and keeping me going when logical reasoning failed.

List of Contents

Word Count (excluding appendices): 92,866

List of Tables

List of Figures

1 Introduction

1.1 Problem Concepts and Characteristics

The problem of text categorisation is well known with a long history dating back to the creation of organised text documents, while the first automated systems were produced as far back as 1960 [1]. However it was not until the late 1980's that research began to favour machine learning techniques over knowledge engineering approaches, as they permitted a greater degree of automation. Prior to this change documents were processed in a predominantly manual way, requiring experts with sufficient domain knowledge to either categorise the individual texts or devise specialised rules that others could follow.

These traditional methods were time consuming and resource intensive, frequently involving the creation of applications that were difficult to modify and near impossible to develop beyond their original intended area. This made them highly impractical and limited, especially as advances in technology and electronic communication increased the capacity for information generation and sharing on a global scale [2]. As a result the advent of techniques that allowed superior automation and improved flexibility became a necessity.

Many different versions of the text categorisation problem definition are available [1][2][3][4], though each expresses the same common meaning; that it is the assignment of previously unseen text documents to one or more predefined categories based on their content. Although this generalised definition may appear simplistic it is not trivial and forms the basis of multiple real world tasks, a number of which influence events over a diverse collection of important subjects. There are also a variety of complex concepts and characteristics associated with the majority of textual data, which pose a challenge when trying to attain adequate performance from automated machine learning approaches.

1.1.1 Types of Text Classification

Though all forms of text categorisation essentially involve grouping documents according to the assignment of categories based on their content, there are a variety of criteria that can be used to determine how this is achieved. Different combinations of these criteria are often referred to as sub types of the general categorisation task, each having intricacies that can make them better suited to particular classification techniques. While some problems have traits that overlap several sub types and make them difficult to explicitly describe, for example language identification [5], sentiment analysis [6] or cause of death determination [7], many belong to one of the three predominant types regularly encountered.

Topic classification or topic spotting is the most common type and involves determination of the subject or category of text, normally being based on the information related to context or content meaning. It is the primary focus for a substantial amount of research into the investigation of novel algorithms and their comparative evaluation against existing approaches.

Genre classification is the correlation of documents according to theme or type rather than subject matter and grew in demand as datasets became available that were not homogenous in type, such as those derived from various Internet repositories [8]. It is often employed to compliment topical classification [9] but is also important in its own right, distinguishing features about the creation of a text, its vocabulary and the intended audience [8].

Authorship classification collates manuscripts that are produced by the same author, or are otherwise from an equivalent originating source, and is exploited in a range of noteworthy applications [10]. It tends to concentrate more on literary style, linguistic analysis and semantic relationship occurrence statistics rather than depending on actual content or context and precise semantic meaning [11].

1.1.2 Classification and Categorisation

Frequently and indiscriminately intermixed throughout the field of generalised and text categorisation [12][13], there is in fact a subtle distinction between the two terms ‘classification’ and ‘categorisation’ [14][15]. Classification is the concept of a machine processing the data, with elements grouped purely by logical assumptions founded on related derived characteristics as opposed to an exact knowledge of the content’s contextual meaning. In contrast categorisation is the notion of correlations made by a human, where associations are established because elements share similar semantic traits and connotations. This is the form of information that is generally most useful, but is difficult for machines to accomplish as the subjective nature requires a certain degree of domain knowledge and even humans may disagree on the results.

1.1.3 Category and Document Pivoted

There two possible perspectives when performing text categorisation and both are beneficial in different circumstances with each having its own advantages and drawbacks. It is also a practical concern as to which should be employed, depending primarily on how data becomes available rather that the techniques that a solution is comprised of, as the majority can be applied regardless with only a few specific exceptions [4].

Document-pivoted categorisation, referred to as 'DPC' and also known as 'category-ranking' or 'on-line' classification [1][1], assigns categories to documents and is the most common as it occurs more often in realistic situations. It involves ranking all categories by relevance against each document and is essential for any system where the data is not entirely available at the moment of classification, for example in continually ongoing processes. Obviously those methods that require calculations reliant on statistics extracted from a global overview cannot be used in this type of categorisation if the information is not wholly present.

Category pivoted categorisation, denoted as 'CPC' or 'document-ranking' classification [1][1], assigns documents to categories and entails ordering every document against each separate category. For this to be effective all test data must be available simultaneously, though it is possible for any method to be accommodated and it offers the benefit of being able to cope with the addition or removal of categories without the need to repeat the full categorisation procedure.

1.1.4 Decision Matrix

If D_j represents a particular document from the overall set of documents D, which is of size $|D|$, and if C_i represents a particular category from the total set of categories C, of size $|C|$, then the confidence value for a specific document and category pair can be described by v_{ij}. Where $1 \leq i \leq |C|$ and $1 \leq j \leq |D|$ and either $v_{ij} = \{0, 1\}$ or $0 \leq v_{ij} \leq 1$, when v is normalised.

	D_1	D_2	…	D_j	…	$D_{\|D\|}$
C_1	v_{11}	v_{12}	…	v_{1j}	…	$v_{1\|D\|}$
C_2	v_{21}	v_{22}	…	v_{2j}	…	$v_{2\|D\|}$
…	…	…	…	…	…	…
C_i	v_{i1}	v_{i2}	…	v_{ij}	…	$v_{i\|D\|}$
…	…	…	…	…	…	…
$C_{\|C\|}$	$v_{\|C\|1}$	$v_{\|C\|2}$	…	$v_{\|C\|j}$	…	$v_{\|C\|\|D\|}$

Table 1.1 : Text Categorisation Decision Matrix Example

A decision matrix is used to analyse the relationship between a set of criteria and values, which in text categorisation is a mapping of the document and category pairings with the associated relevance of each [1]. The information relating to every pair takes the form of either a binary or scored value that denotes the estimated confidence that making that particular assignment is correct. With higher

[1] Note there is a slight error in the referenced publication where the terms have been confused but is corrected here.

values signifying a connection is more likely and lower values indicating it is less appropriate, while zero shows a complete lack of association.

In the case of ‘binary’ or ‘hard’ categorisation the correlation of a particular document and category pairing can either be valid or invalid, which creates a matrix containing only true or false to represent whether specific assignments should be made or not. Whereas in ‘soft’ categorisation a variable measure of suitability is generated instead, allowing assignments to be ranked in order of priority so another mechanism can be applied to produce the final binary decisions [12]. This is favourable when an application requires a list of potential candidates, for example in a partially automated system where the options are first narrowed before a human expert manually selects the final choices [4].

1.1.5 Dimensionality

Typically problem dimensions refer to the features extracted from the textual data, which are the basis for determining the similarity between documents that contain them and their relevance to categories associated with them. Text categorisation inherently has a very high dimensional feature space due to tokenisation, which splits each document into fundamental semantic or syntactic components that can be consumed more easily by machines. Though necessary for automated systems this process frequently gives rise to a considerable number of features, often in the order of thousands or tens of thousands even for a relatively small dataset [16][17][18], which can prove a hindrance to many approaches.

Reduction of the dimensional space is highly desirable where possible, as it not only makes tasks more practical but has been shown to sometimes improve performance in terms of accuracy [16][18]. This is because noise is removed from the dataset, which takes the form of irrelevant, redundant or misleading features that have a tendency to cause a bias toward associations that should not be so prominent. While a few datasets do retain a significant proportion of redundant features [16] there are also those where the majority of extracted aspects hold useful information [17], so it is essential that appropriate reduction techniques are employed.

It should further be noted that while dimensions normally denote the text features this may not always be the case, for instance a subtype of text categorisation involves category structures with each tier being identified as a dimension [19]. This is known as ‘hierarchical’ text categorisation and unlike the more common ‘flat’ version does not treat each category independently, instead taking category tree or directed acyclic graph formations into account [20]. Research has shown that this approach can lead to superior performance when a problem has a natural inclination to organise data into groups with distinct layers [21][22].

1.1.6 Multi-Lingual

Research in text categorisation and information retrieval generally only consider datasets that each contains a single language, though this language may vary between applications. However there are many real world situations where documents within the same collection are based on a range of separate dialects or have individual texts which include sections composed in multiple different languages [23]. Despite this and a growing number of linguistic utilities that handle a diverse mix of vocabularies [24][25][26], these issues pose an increased level of relative hardness and are rarely investigated. Most current tools also tend to focus on just one language as several have overlapping terms that give rise to a substantial degree of ambiguity, which often makes language detection the principal component of multi-lingual solutions.

1.1.7 Single and Multi-Class

Categories are regularly referred to as topics, labels or classes, which are all acknowledged expressions but may also be used to describe certain dataset characteristics. For example the term ‘single-class’ signifies what is sometimes known as a ‘one-class’ or ‘binary’ problem [2], which contains only a solitary predefined category that every document either does or does not belong to. Whereas the phrase ‘multi-class’ denotes there are multiple possible categories that each document might be related to, though not necessarily that they should be assigned to more than one [2].

Single-class problems are conventionally believed to be easier to handle than multi-class and the bulk of traditional machine learning algorithms are actually unable to deal with a choice of several categories at once. Instead they break the selection down into a distinct task for each category using an approach called ‘one-vs-all’, which determines a relevancy score before compiling all of the results to obtain a ranked list of candidate assignments [13]. This can be extremely resource intensive and is incapable of exploiting correlations between the categories [27], such as those typically present in hierarchical categorisation.

1.1.8 Single and Multi-Label

Often the class and label expressions are arbitrarily interchanged [3], though there is a recognised consensus that they stipulate markedly different characteristics [2][16][28]. Whilst the use of ‘class’ concerns the total distinct categories within a problem, the ‘label’ terminology relates to the number of potential assignments that can be applied to each document. Therefore ‘single-label’ means that only a solitary category can be explicitly assigned per document and likewise ‘multi-label’ indicates that at least one of them has zero or more associations.

Obviously if a problem is multi-label it is inferred as being multi-class with category associations that are not considered mutually exclusive, however the opposite assumption is not necessarily true. These varied overlapping collections are substantially harder to correctly resolve and require advanced evaluation metrics that account for both the accuracy and completeness of results, regardless of whether hard or soft categorisation is performed. Traditional methods still necessarily employ the same divide and conquer strategy of splitting the task into several discrete single-class, and consequently single-label, 'binary classifiers' [13][28].

1.1.9 Structured and Unstructured

Individual document content can be structured, unstructured or semi-structured, which is a mixture of both organised and disordered information [15]. These different compositions can have a significant effect on the performance of a system and processes should be chosen that maximise the potential benefits gained from any standardised features. This is especially important in structured data, such as official reports or articles, as the controlled formatting, styles or available metadata commonly provide an intuitive direction for methods that generally makes them easier to categorise. Conversely unstructured data is nearly always more challenging, frequently being random or chaotic in nature, as is often encountered when dealing with manuscripts that depict personal thoughts or emotions.

1.1.10 Supervised and Unsupervised

Text categorisation is inherently a supervised problem, where a set of previously labelled historic or purposely generated data is available for the creation of a solution, and this is typically manually classified to the highest possible level of correctness. If it is unsupervised, with no pre-labelled data, the task becomes that of grouping similar content together without any explicit points of reference, which is analogous to a clustering issue [12]. However cross over between these areas of knowledge discovery does occur, as dedicated clustering techniques are regularly used to aid in the filtering and removal of irrelevant noise or for combining multiple similar terms. This collaboration also arises in semi-supervised situations, where a small set of labelled documents guide the clustering and categorisation of a much larger unlabelled set [29].

1.1.11 Training and Testing

There are two core functions involved in text categorisation; the first being the training of a predictive model by means of a supervised learning procedure, while the second is the use of such a model for the assignment of categories to unseen or validation testing data. Each uses a different collection of documents and in the case of research a single labelled dataset is usually divided into separate

training and testing subsets, occasionally with the training set further segmented to provide a means of solution optimisation.

Both tasks normally require a number of distinct stages, which in turn may entail several steps that have a similar purpose, but while some are dedicated to either training or testing operations there are many that need to be implemented by both. Typically any specific text processing prior to actual construction or application of the classification model must be repeated, ensuring that identical features are extracted and filtered.

1.1.12 Word Sense Disambiguation

Perhaps the most difficult characteristic to deal with is that of word sense disambiguation where a particular feature can relate to multiple dissimilar meanings, giving it the potential to significantly alter the underlying semantics of the text that contains it [30]. It is generally a very difficult trait for machines to deal with as contextual information must be taken into account, sometimes necessitating specialist domain knowledge that might even make it challenging for humans.

This poses a couple of problems in text categorisation largely because of the widespread use of individual words as features and the common assumption that all extracted features are independent entities [31]. For instance elements subject to polysemy, where the same word has several connotations, will generate identical features and always be treated the same regardless of specific context, lowering their discriminative capability. Likewise those elements displaying synonymy also suffer, as though they embody an equivalent inherent meaning each produces a discrete feature, which are consequently treated as being unrelated. Synonymy is easier to resolve, as relatively simple thesaurus and dictionaries can be utilised, however polysemy is notably harder and demands complex methods such as named entity extraction [32] and part of speech tagging [33].

1.2 Research Motives

Text categorisation dates back to the first text manuscripts, but the concept of automation only rose in popularity during the late 1980's inline with the evolution of computing technology and subsequent increase and improved availability of electronic text mediums and databases [12]. This discovered enthusiasm and vigour for the research area lead to the discovery and refinement of various solution approaches, until the subject seems to turn stale again in early 2000 with an apparent reduction in the frequency of novel publications [34]. However, due to the continued progression of technology and the ever present desire to create and consume information, the volume of textual data is expanding exponentially and the current pace of research is unlikely to meet demands.

Persistent growth combined with the increasing diversity of text categorisation applications, such as hypertext tagging, email filtering, manuscript authorship, survey encoding, speech recognition and many more, means that existing 'state-of-the-art' solutions are not as useful as they once were. While this has resulted in an assortment of new and refined categorisation solutions over the past decade, no standardised structure has been defined for their accurate reproduction or evaluation. Due to this some systems deliberately or unintentionally employ biased datasets specifically designed for that particular approach [10][35] or are compared exclusively against algorithms not considered state-of-the-art for the problem under investigation [36]. Furthermore, in the majority of cases there is simply not enough information supplied to exactly duplicate the described system or confirm the results claimed [37][38]. In combination these issues severely hinder the advancement of research in text categorisation, with the many debatable contributions leading to an uncertain and chaotic direction.

Adding to the concern is a generally low level of innovation in some publications, with many surveys [4][13][39][35] reiterating the same basic information found in previous reports originally composed several years earlier [40][1][41]. There are also more specialised articles reporting results that are impossible to verify or that differ substantially from other findings [42]. Even literature specifically aimed at presenting effective comparisons between approaches does not contain particularly decisive information, often being composed from an assortment of second hand data that was evaluated under varying criteria [35]. It has now reached a point where a few authors have acknowledged how severe the situation has become, with some commenting on the incompatibility of datasets or assessment criteria and the lack of critical information presented for the systems being reported [35][43][1][42][44].

Further compounding this issue is a growing requirement to investigate problems that relate more closely to real world scenarios, such as those consisting of complex datasets exhibiting multi-class or multi-label characteristics. These tend to give rise to solutions that require elevated levels of intricacy and normally mean an increase in the number of variables and steps involved in the categorisation process. As a result there are more individual components that, if not properly explained, may be misinterpreted or neglected under a misconception that they are inconsequential. Examples of this are 'Associative Classification' (AC) algorithms [45], a relatively new addition to the domain, which are

composed of several phases that each require a set of finely tuned parameters in addition to those usually needed for processing textual data. The reported findings for AC based systems initially seem promising [3][36], but a lack of result analysis and detail regarding basic construction and data processing makes assessing their true capability unfeasible without performing further investigation.

Performance evaluation criteria pose yet another issue with many different techniques available, each with its own particular strengths and weaknesses, causing researchers to differ in opinion as to which is superior. While accepted that at present no single measure adequately illustrates all of the potential benefits, drawbacks or intricacies of a complex solution, the inability to exactly reproduce systems and thus re-evaluate them using different performance criteria greatly limits comparisons that can be made. Fortunately, one appraisal method known as the 'Break-Even Point' (BEP), which is based on precision and recall values and accounts for multi-label problems, has taken precedence over others. However, this has several major limitations, especially considering its feasibility outside of a research environment due to assumptions made in respect to parameter tuning, but as enhanced performance measures are discovered these comparison issues will continue to become more evident.

Many applications that utilise text categorisation use it in a time critical manner, either to prevent backlog of high volume data or provide instantaneous real-time results, yet the notion of observing time statistics for input processing or classification is almost unprecedented at current. The main reasons for this are a general assumption that hardware advances fast enough to meet requirements and a common acceptance that effectiveness is more important than efficiency. Another reason may be due to the rapid prototyping approach typically used in research, as creation of inflexible and disposable solutions with as little resource and effort consumption as possible will often lead to inefficient systems that make the value of time tracking questionable.

Indeed, with the rate technology evolves and the ability to scale applications through parallel, cloud or grid computing, the idea of time scrutiny might seem irrelevant until it is balanced against the associated economic and environmental impact. Some applications might also just favour a trade of speed over accuracy, particularly those dealing with multi-class or multi-label problems, as very few algorithms can process these without a divide-and-conquer strategy that requires multiple iterations over the classification process.

1.3 Thesis Overview

1.3.1 Statement

Analysis of available text classification solutions and their performance comparisons reveal major issues that significantly hinder advancement of research in the text categorisation domain: widespread use of inconsistent definitions and terminology, conflicting or incomplete evaluations and inadequate explanation and standardisation of the exact methodologies being employed.

1.3.2 Specific Terminology

Aside from the terminology previously outlined and any others that are commonly associated with general text categorisation or data mining, there are also a number of additional expressions present that have a specific meaning within the context of this thesis. Some are explained on a case-by-case basis as they are introduced, such as those that name explicit parts of the framework described in chapter five, while others are encountered more commonly throughout the entire document.

The term ‘solution’ is used to refer to one or more algorithms that have been grouped together for the purpose of achieving a single overall task. As such a ‘text categorisation solution’ relates to a set of components that have been intentionally combined in order to undertake the process of text categorisation. Note that publications tend to apply the phrase ‘algorithm’ for this concept, but this has purposely not been adopted to emphasise that multiple distinct algorithms are required to solve each individual aspect of a text categorisation problem, as demonstrated in chapters two and three.

In the context of this thesis ‘architecture’ corresponds to the generalised structure or blueprint of a solution, therefore representing the variety of possible or necessary stages and interactions that are involved in its construction. For the problem of text categorisation this notion is examined in further detail during chapter two, where existing processes and methodologies are considered and utilised to produce a comprehensive universal architecture.

Where the word ‘framework’ is employed it denotes a reusable software platform or selection of code libraries, which enable the production of software applications with a standardised structure or defined arrangement of several discrete components. A number of current data mining frameworks are investigated in chapter four, whilst a high-level design for a new framework is introduced in chapter five, followed by the description of a practical working implementation.

1.3.3 Contributions

Several key issues within the domain of text categorisation have their primary causes identified and investigated, while innovative ways to remedy them and advance the general research area are proposed. As part of the process intuitive naming conventions are recommended for a number of aspects commonly referred to by disparate aliases, encouraging the standardisation of many labels and expressions. Guideline definitions are also specified for distinct concepts to reduce the confusion from inconsistent and ambiguous terminology, which is easily misinterpreted by researchers of every experience level. Details of principle text resources are given as well, highlighting critical problems with the way they are currently used and outlining basic information that assists in the creation of practical experiments.

The full complexity of the text categorisation process is revealed in a manner not previously achieved with the independence and intricacy of its distinct stages being established, while the necessity and benefit of segmenting solutions into atomic components is demonstrated. A suitable methodology for the structuring of complete and partial categorisation solutions is also proposed, comprising of the individual stages they might contain and details of how these could potentially be configured and interact.

A survey of popular algorithms that regularly form the foundation of most complete solutions has been produced, intentionally organised according to the proposed stages to emphasise their purpose and show how they correspond to existing components. While the validity and clarity of all descriptions has been cross examined over multiple sources, ensuring clear and accurate explanations are provided for every concept with any disparities or areas of possible confusion being noted where appropriate. Equations have also been defined throughout using consistent mathematical notation and are supplied in their most commonly acknowledged format along with information regarding prominent equivalent or specialised alternatives.

Desired attributes have been determined for applications that are built to investigate novel text categorisation approaches, with a focus on the requirements needed to cope with the complexity and modularisation of the individual stages proposed. Based on these attributes a selection of prevalent data mining frameworks and utilities have been reviewed, with particular regard to their aptitude for handling the categorisation process and capacity for modification or integration of new components. The notable benefits and concerns encountered during this analysis have been assessed and employed to generate an idealistic combination of features that offer rapid prototyping within a stable framework, permitting unprecedented control over solution composition and duplication.

Unable to find an adequate means to produce solutions comprised of the previously outlined stages, a new framework has been designed that is highly adaptable, having the flexibility to deal with fully modularised systems and the capacity for infinite expansion to accommodate further content. A preliminary implementation of the design is also introduced, in addition to basic reusable components

for the generation of trials to reinforce assertions made about the categorisation process, providing a proof of concept to confirm its feasibility and reveal any unexpected limitations.

Using this new framework a series of experiments have been devised to test the potential impact of the individual stages, with each directly based on a central benchmark that allows an impartial and fair comparison of both effectiveness and efficiency. Results obtained from these trials demonstrate the importance of every stage and highlight the benefits to be gained by modular solution architectures. In order to facilitate swift generation of solution components a new independent library of text processing utilities has also been created, retaining complete separation of the individual algorithms to ensure their accurate portrayal.

1.3.4 Structure

This initial chapter provides a synopsis of the thesis and a brief overview of the text categorisation problem, summarising its fundamental aspects and dominant concepts, whilst establishing a standard set of terminology and explicit descriptions based on those found in existing literature. It also outlines motives for undertaking the research, stating the poor availability of comparative results and a general deficiency in the descriptions and standardisation of published solutions.

Chapter two analyses the constitution of a text categorisation solution, dividing it into atomic elements and examining the different types of essential components and how they might interact. It considers the various combination possibilities and how they could be applied in a practical and meaningful way to maximise research options and productivity.

An assortment of algorithms and techniques are surveyed in chapter three, with each being organised according to the particular stage they are part of. They have been chosen due to being regularly cited or employed in related publications and are carefully described to resolve any confusing issues and show how they might fit into the proposed solution architecture. Popular text document resources are also detailed, along with considerations that should be made when selecting a dataset to investigate.

The fourth chapter appraises current applications that have a minimum level of text processing capability against a predefined list of desired traits, attempting to match them to the various stages and determining their value in a research based environment. Several leading open source data mining frameworks are also reviewed, focusing on their current text related functionality and capacity for expansion, with the overall positive and negative findings being compiled and analysed.

The previous discoveries are utilised to guide design of a novel framework in chapter five, which is specifically aimed at the rapid generation of modular prototype solutions that can be easily adapted and exactly replicated. It then describes an implementation of these designs, including details of the

key elements forming its core and a brief summary of the programming concepts they involve. A list of perceived limitations is also given, along with recommendations on how they might be resolved if the implementation were to be taken beyond the trial stage.

Chapter six introduces several text categorisation related components that are compatible with the trial framework and employs them to construct a simple benchmark solution and a number of variants based on it. Each variant follows the proposed architecture and contains an alteration to just one of the benchmark stages, making it possible to measure the impact it has on performance. The potential importance of individual stages is examined and the whole process is fully explained, with obtained results and experimental components described in enough detail for them to be duplicated.

A brief summary of the problems encountered within text categorisation, the research motives, overall objectives and thesis contributions is provided in chapter seven. It also reiterates the notion of a standardised and segmented solution structure and offers a critical examination of its worth, taking into consideration the variety of approaches available, the potential for future research and results obtained from the previous experiments. Secondary to the main objective the new framework design and implementation is also evaluated and compared to the existing options, with notes on unexpected limitations that need to be resolved and enhancements to make it more practical. Finally a summary of likely future directions for both the exploration of solution architectures and further development of the framework is supplied.

2 The Text Categorisation Process

2.1 Chapter Contributions

This chapter critically examines the configuration of present solutions created to deal specifically with the text categorisation problem and that of data mining in general. It considers the ability of each to portray the inherent complexities of the issue and their capacity to encapsulate the required details necessary for any solution to be defined in a standardised global perspective. Finally it outlines a new architecture that attempts to resolve the potential shortfalls discovered and then leads onto the next chapter, which shows how common algorithms correspond to the proposed structure.

Utilising existing descriptions of the text categorisation process a set of stages is defined that can be combined in a variety of ways, allowing reproduction of all presently known solutions in a flexible and modular manner. It divides current interpretations into their smallest independent components and groups them together into distinct collections by distinguishing those that share similar features or perform equivalent operations. These collections are then assessed for intended functionality and employed data formats in order to derive generalised stages, which are mapped to presently available solutions to permit their potential interactions to be logically determined.

An overview of every stage is provided, detailing the main purpose of each and demonstrating how it fits into the categorisation process. It shows the diversity and complexity of aspects involved and emphasises that all have specific roles in an overall solution, highlighting it is not just the actual classification algorithms that are important. This further stresses that merging separate stages into a single entity should be avoided and offers a mechanism for creating multifaceted solutions, where elements are easily substituted to gain the maximum benefit from different algorithm combinations.

2.2 Existing Solution Architectures

Despite the long history, level of interest and importance of research in the field of text categorisation, so far only a relatively small number of attempts have been made to define a formalised architecture that can be used as a standardised template for the construction of solutions. Of those that do exist few are aimed specifically at text categorisation, while the most well-known are related to generalised data mining and have no particular association with any distinct subject.

Furthermore, even the methodologies that are specialised do not cover the entire process in sufficient detail to encapsulate all of the involved elements and often portray an idealistic linear progression through the limited aspects they entail. However, nearly all of them do tend to contain a degree of overlap within their descriptions, which can be used to determine several common subsets that give a logical indication of what stages should be part of the complete solution architecture.

2.2.1 Generic Data Mining

There are a limited number of formal methodologies available that relate to data mining or knowledge discovery in databases (KDD), but they are typically generic and have a bias toward larger enterprise level applications for use in industry or commerce. None of them directly encompass the nuances and intricacies of text categorisation and are inclined to give a higher degree of abstraction that disregards particular solution techniques or problems. As a result they are rarely appropriate for situations where greater detail is required to account for unique traits or subtleties, such as those encountered during research that involves algorithm prototyping.

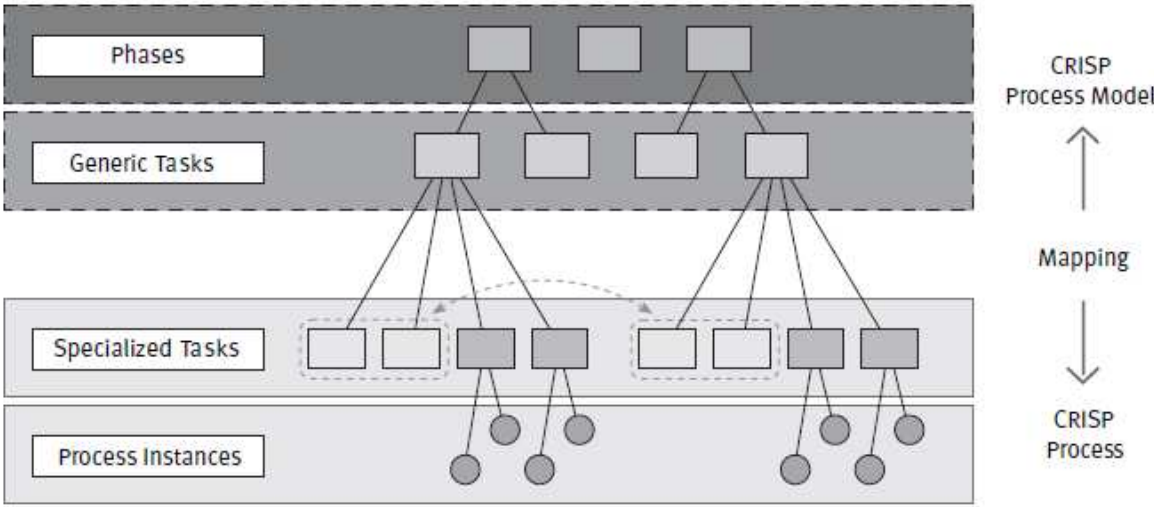


Figure 2.1 : Hierarchical abstraction of CRISP-DM

Perhaps the largest of these generic architectures is CRISP-DM (Cross Industry Standard Process for Data Mining) [46], which was originally conceived in 1996 by a consortium of companies with an interest in data mining. In 1997 the project gained funding from the European Commission and grew in popularity until the first official version was presented in 1999, with discussions for an updated specification taking place in 2006. The methodology has also been embraced by many companies, including IBM who have integrated it into their SPSS predictive analytics product, but despite this the current project owner is uncertain and the main website noted in some publications [47] has not been accessible for several years.

It is designed to give a structured approach when planning a data mining project and is broken down into four hierarchical layers of abstraction, which begin as a broad overview and become increasingly more specific as they progress downward. The topmost layer consists of six core phases, starting with problem understanding and traversing through the entire creation process until a final deployment stage is reached. In general the architecture is notably similar to the standard practices employed in modern software development and, whether deliberately or otherwise, most projects will tend to utilise subsets of the defined stages, whilst also promoting hierarchical inheritance and module recycling.

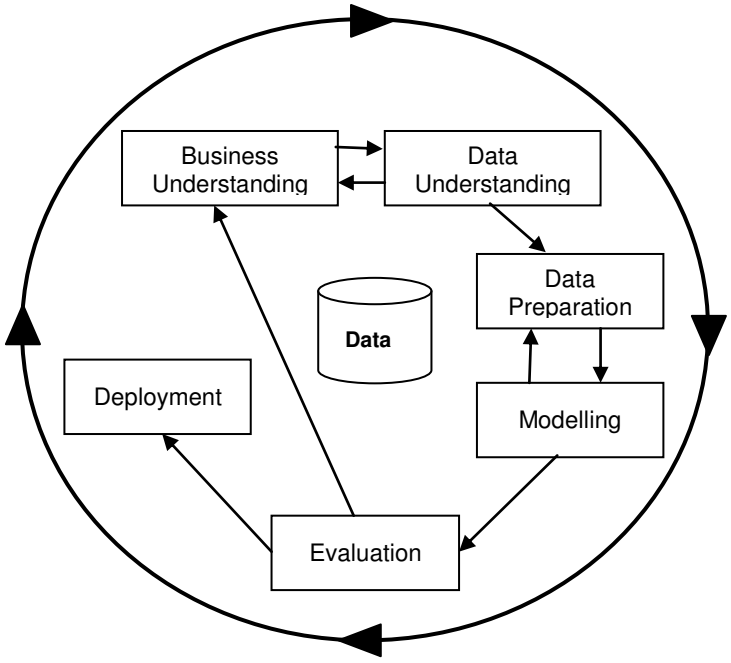


Figure 2.2 : Phases of CRISP-DM

There are a small number of similar alternatives to CRISP-DM and the majority have comparable high-level phases, for example SEMMA (Sample, Explore, Modify, Model and Assess) contains five analogous stages [48]. Originally created for the SAS Enterprise Miner software, SEMMA is frequently encountered as a standalone methodology and is typically considered to be less business orientated,

though more ambiguous when applied outside the application it was designed for [49]. Likewise, other architectures with equivalent aspects include those derived from the KDD process [47] and even more generic ones, such as the Scientific Method [50], which requires only minor modification if employed for data mining tasks.

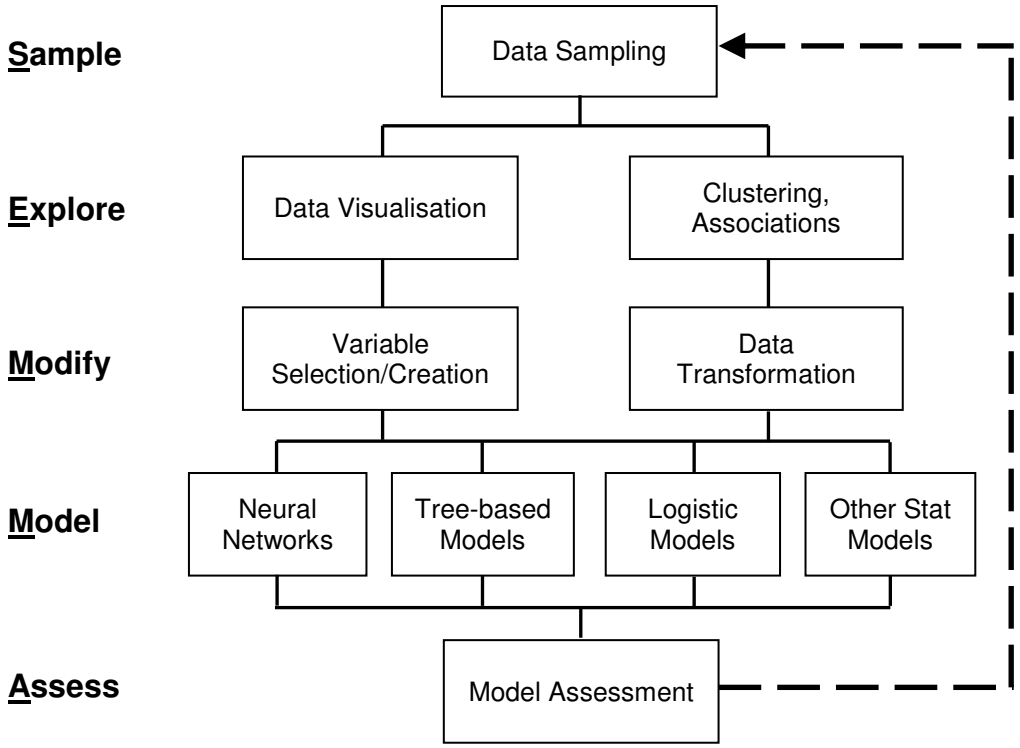


Figure 2.3 : Phases of SEMMA

Despite their original size and scope, or maybe because of them, few of these broad methodologies appear to be in widespread use throughout the text categorisation or general text related research communities. Instead they are predominantly embedded within commercial applications, as is to be expected when considering the phases they entail and the emphasis placed on business processes, reporting and deployment concerns. While these are important issues, they are not the primary focus when rapidly prototyping new algorithms or defining requirements for distinctive problems like text categorisation [47].

2.2.2 Text Categorisation Specific

Since the advent of automated text categorisation there has been some effort to outline the elements required when constructing a solution, though mostly this takes the form of tutorials that list prevalent algorithms in roughly the order they would be applied [51][2][1][4][13][52] . While many publications go further and provide brief explanations of the purpose behind each type of algorithm, even these do not

cover the complete scope of possible interactions and intricacies available. A few also deliberately focus only on particular techniques, principally those considered to be popular or highly sophisticated, rather than establishing a universal methodology that is relevant in all situations [53][54].

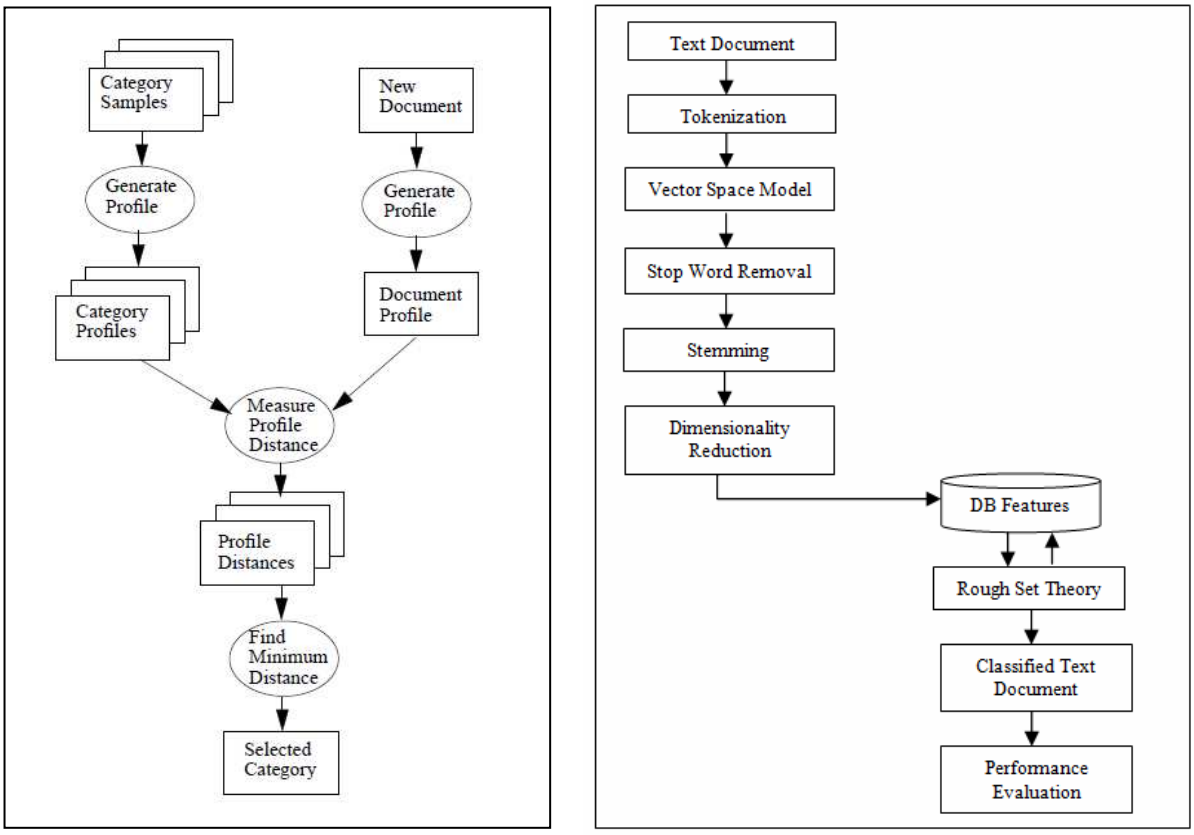


Figure 2.4 : N-Gram-Based Text Categorisation [54] (1994), left, and Hybrid Intelligent Technique for Text Categorisation [55] (2012), right

Aside from the tutorial format there have been limited attempts to visually illustrate the process flow of automated text categorisation as well [54][56][57][53][58][55], though they share similar inadequacies and have an apparent vagueness due to unwarranted oversimplification. Often stages are missed or components that perform fundamentally different tasks become merged, for example the grouping of tokenisation and index weighting into a single data representation task [53][56][58]. Frequently fixed aspects relating to specific algorithms rather than generalised phases are also exhibited, for instance the inclusion of explicit data pre-processing techniques [55][57], which introduce undesirable rigidity and eliminate potential solution design options.

The sequential nature consistently depicted is another issue, as it implies that any given stage will never be revisited and can only be employed once. In reality this basic assumption proves remarkably constrictive and may be the reason why certain interpretations include a number of set elements with closely associated functionality, such as the stemming and stopword removal data preparation steps.

Lack of differentiation between classifier construction and execution is a further shortcoming, as it can make it difficult to determine which processes are always required or should be employed just for training or testing.

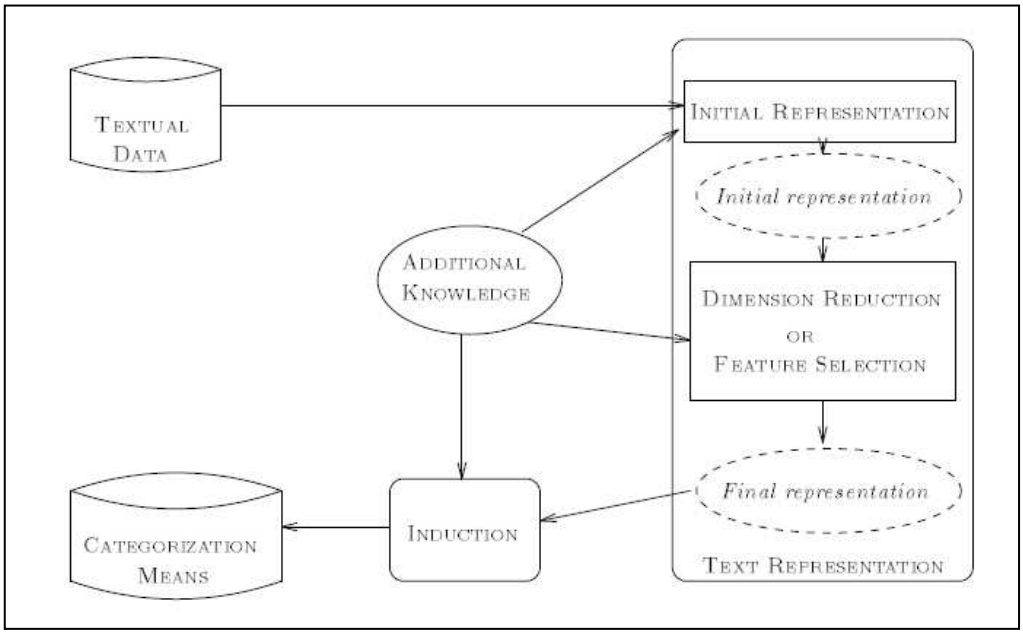


Figure 2.5 : A Framework for Comparing Text Categorization Approaches [56] (1996)

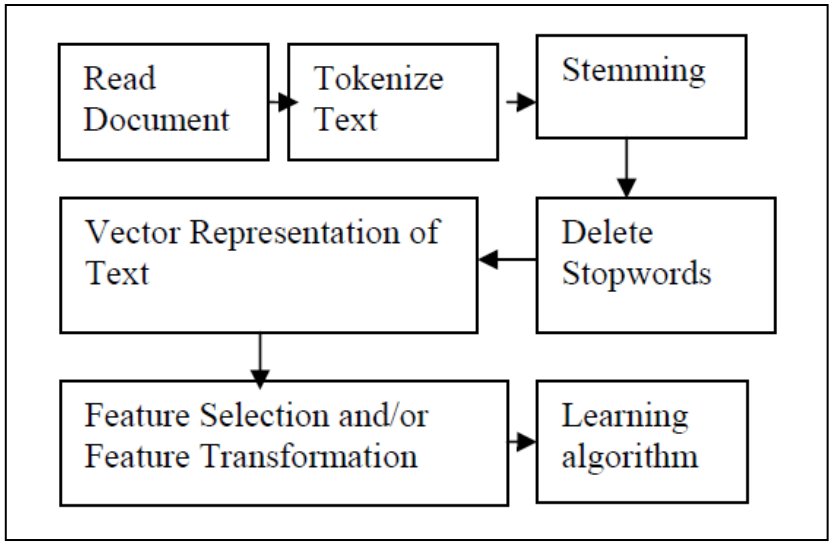


Figure 2.6 : Text Classification using Machine Learning Techniques [57] (2005)

Examining the change in explanations for the text categorisation process over the last two decades clearly shows that few significant improvements have been made. There is no decisive convergence

towards a standardised description and much of the ambiguity and rigidity still remains, with linear organisation, extensive oversimplification and depictions that contain fixed aspects relating to specific algorithms. Nevertheless, it is feasible to discern common elements shared between them and extract the overlapping features to derive a simple methodology, which can then be adapted according to the information gathered from tutorials and surveys.

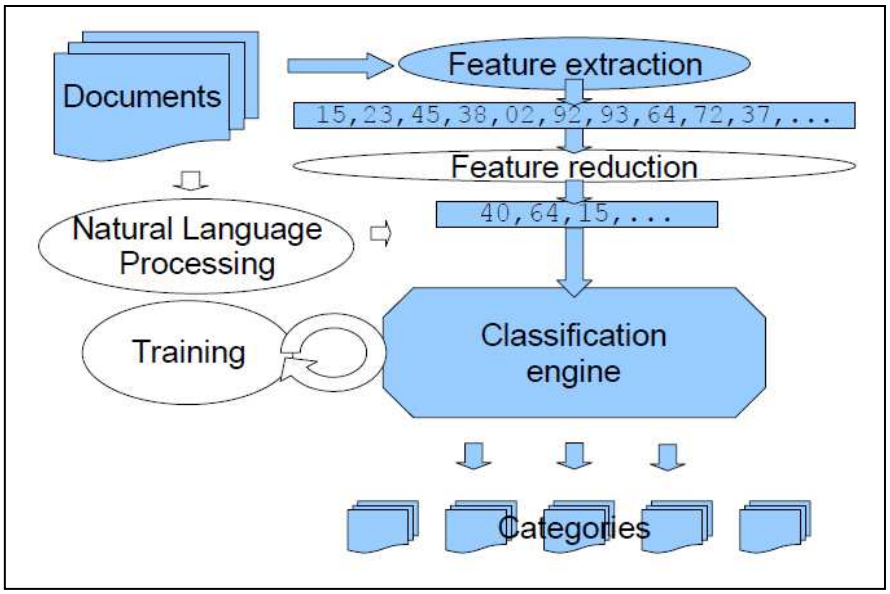


Figure 2.7 : Text classification method review [53] (2007)

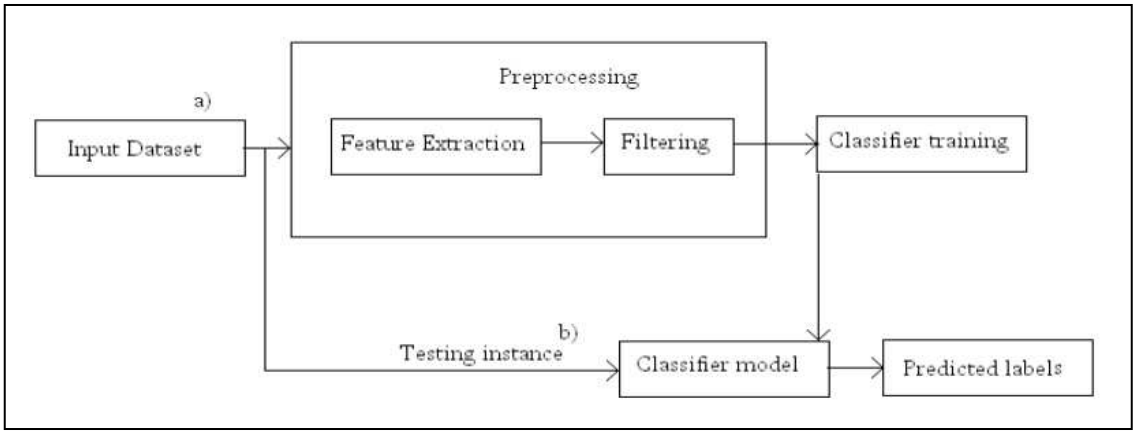


Figure 2.8 : A Novel Multi Label Text Classification Model using Semi supervised Learning [58] (2012)

2.3 Proposed Architecture Stages and Interactions

By breaking down existing text categorisation architectures into common components and logically restructuring them based on the algorithms currently in use, it has been possible to define a universal set of stages and their potential interactions. Each of these proposed stages should be considered independently of the others when generating a new solution, regardless of the overall approach and combination of techniques adopted. This is because any of them can be essential for the production or practical replication of a solution and various approaches already exist for all of them, though previous over simplification often means some get merged together or ignored [59][53][57][13][54].

A fuzzy boundary is present between a few stages that have similar functionality or utilise identical calculations, for instance there are pre-processing techniques seen as a form of dimensionality reduction and index weights based on the same metrics as dimensional reduction. However, every stage has a definite purpose and the overlap between them primarily occurs only as an inadvertent side effect when accomplishing that purpose. This notion becomes even more blurred as solutions grow in complexity and move away from the traditional linear flow, but is important to promote good practice when describing and reusing individual components.

The stages proposed generally occur in the following order for simple categorisation systems, but the possibility of reiterating a particular stage or set of stages may also be necessary for more involved solutions that accomplish specific tasks. For example several distinct data representations could be used during pre-processing or multiple indexing weights might be employed during the dimensional reduction steps, which can make the overall composition surprisingly complex.

- Data Selection
- Pre-processing
- Data Representation
- Data Indexing
- Dimensionality Reduction
- Classification
    - Classifier Construction
    - Classifier Execution
- Result Evaluation

A brief overview of each stage follows, while detailed explanations and a survey of the most popular algorithms and techniques associated with them are provided in the next chapter, deliberately sorted according to the respective area of the architecture that they belong to. Note that although these stages might be applicable to a variety of different formats and data mining tasks, the information and methods outlined are specifically aimed at those applied for text categorisation.

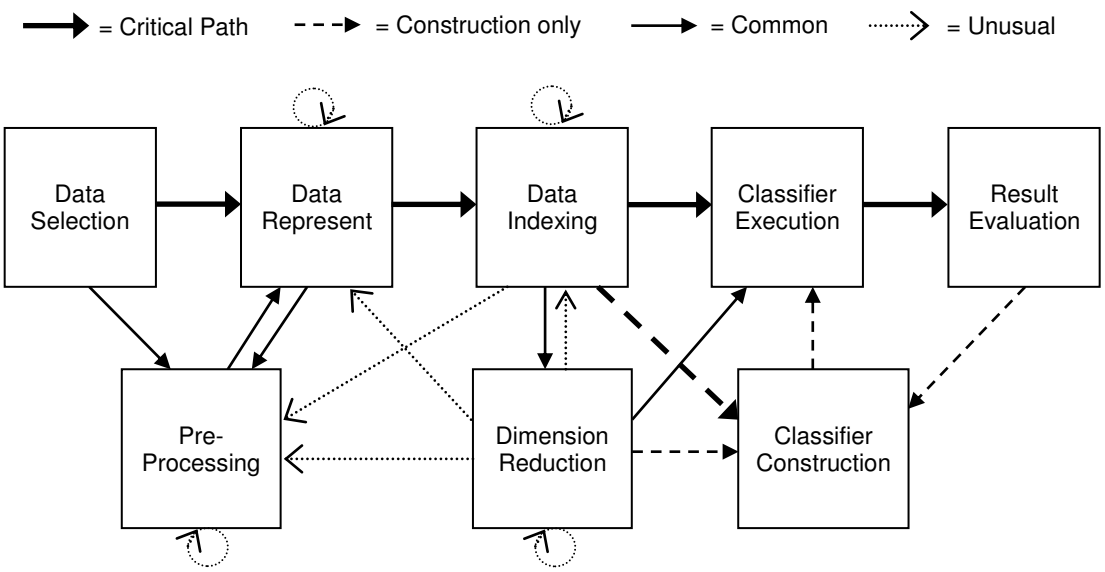


Figure 2.9 : Interaction of Text Categorisation Solution Stages

2.3.1 Data Selection

Data selection is the process of determining appropriate datasets, in this case in the form of text document corpora, for the particular domain being investigated and loading them into the categorisation solution. Depending on the data source it might take any number of formats and must be loaded into a standardised format that can be used by the categorisation system. The choice of corpora should typically be restricted to match a specific profile of desired data characteristics or annotations, especially when dealing with real-world applications. Often it is possible for a single large dataset to be split in to several smaller subsets, each of which may be associated with specialised annotations or better emulate certain characteristics. This subset division might be performed during the data loading step or as part of a separate pre-processing stage.

2.3.2 Pre-Processing

Once loaded and prior to being passed to other stages of a categorisation solution the data generally undergoes some form of cleansing or processing. The purpose of which is to insert additional informative metadata or remove known non-informative noise, relevant to the particular task being undertaken. Depending on the particular problem under investigation this could mean the removal, modification or generation of supplemental data by using certain characteristics defined from the entire dataset scope. It is also primarily during this stage that specialised processes relating directly to

textual data occur, such as language or writing system specific manipulations, and though only basic alterations are made they may have a significant intentional or unintentional impact to overall performance.

2.3.3 Data Representation

At least once during the categorisation process the textual data will need to be partitioned into a collection of symbols that will form the base representation used in later stages. This process is also known as tokenisation and it is not uncommon for the resultant lexicon to bear little resemblance to the original data or generate illegible output. The intention is normally to produce a concise version of the initial text that retains as many information rich features and attributes from the source as needed.

Depending on the solution employed, representation of the data may change numerous times and possibly involve the use of several different formats within a single stage. In some cases the final outcome may actually be more verbose than the original in an attempt to capture enhanced semantic relationships that might lead to superior categorisation performance.

2.3.4 Data Indexing

Data indexing is the mechanism used to produce a succinct machine readable version of each text document, or the previously processed data, into a format that is memory efficient and suitable for the next stage in the solution. Depending on the structure, complexity and weighting metric employed, the created index might represent only a basic synopsis of each document or sufficient detail to replicate the original in full. Most indexes relate the specific terms contained within a document to a value that represents how important or informative that term is when identifying the content and possibly the category labels assigned to it. As a result most index weighting metrics tend to be based on statistics discovered only within the scope of a single document and possibly the categories it is directly related to.

Frequently this stage is considered to be closely associated with data representation and both are often thought of as a single process, which is likely because every index uses the lexicon of terms derived from the output of the representation stage. Another cause is probably due to many existing solutions not exploiting different representations for separate purposes, tending instead towards only a single representation that is retained throughout the entire solution. In actuality the representation and indexing stages are distinct, as the symbols denoting terms in no way affect the inner workings of the index weighting algorithms employed, though they will of course influence the extracted values used within calculations.

2.3.5 Dimensionality Reduction

Text document data habitually has a large problem scope due to the volume of possible character and symbol combinations in any given language writing system. Depending on the corpus and choice of techniques employed, the scope of a problem can also become several orders of magnitude larger than that found in the associated writing system due to the addition of meta-data and use of verbose data representations. Dealing with such high dimensionality requires excessive resource overheads and computation so a variety of reduction techniques are usually applied to decrease it.

The complexity of techniques differs greatly, but there is strong evidence to suggest that, while making a solution more resource efficient, substantial pruning might also increase categorisation effectiveness. While some degree of arbitrary data reduction normally occurs at several points in a categorisation system, this stage specifically attempts to distinguish the least informative data and remove as much as is feasible without negatively affecting classification performance. Typically term based reduction using metrics relative to particular category scopes prove most effective, so this tends to be the main focus, though many techniques also include or rely upon details discovered at the global dataset or individual document level.

There are two main types of dimensional reduction; feature selection prunes low information terms from the data, while feature extraction attempts to combine multiple terms into a single new or existing one that contains all the information of the independent variables. Regardless of technique the overall aim is to remove as much superfluous data as possible without eliminating any potentially valuable or informative aspects. Each technique requires conversion of the data into a suitable representation and often involves the generation of an index to produce a machine readable format. This resultant representation and index combination may only be used for one particular method or be applicable to several, so the ratio of resource overhead to gained benefit needs to be considered relative to the specific problem requirements.

2.3.6 Classification

With the data converted into a resource efficient and machine readable format, the actual process of classification is able to be implemented. This involves the application of one or more algorithms to the processed data in order to predict possible category assignments to a collection of previously unseen text documents. A large variety of classification algorithms are available, with most encapsulating numerous discrete parts that perform certain functions, however in general the majority involve two distinct stages.

2.3.6.1 Classifier Construction

All classifiers require a set of previously categorised or 'training' documents as a basis for comparison and most utilise these to build static models that can be employed to predict the category labels of previously unseen documents. Due to the high dimensionality and potential issue of concept drift over time with real world textual data, especially for those with multi-label characteristics, these static models are frequently resource intensive to develop and maintain. In order to deal with this it is sometimes possible to introduce incremental steps to a system already constructed that counteract some of the negative traits, though commonly in exchange for a decrease in classifier effectiveness.

Even with the potential drawbacks pre-constructed classifier models are a widely accepted strategy as they often lead to streamlined execution, enabling high volumes of new documents to be quickly processed. Only a small number of algorithms forgo the construction stage entirely and instead make use of volatile short-term classification models or dynamically calculated comparison metrics. These methods completely negate the effects of concept drift and the initial overhead of generating a static classification model, but normally require increased resources during the execution stage.

2.3.6.2 Classifier Execution

The execution stage is the point at which the chosen approach and, if generated, static classification model is applied to previously unseen or 'test' data in order to predict which category labels should be assigned. As classification based on pre-labelled example data is a supervised process, execution output for test or validation data may be fed back into the construction stage multiple times to fine tune parameters until optimal results are produced.

Validation sets can either take the form of a single derived subset from the training data or utilise a more comprehensive mechanism such as cross-fold validation, which uses multiple training and validation partitions and averages parameter values over all of them. Reusing the execution output in such a manner during construction can blur the distinction between the two stages, but they should still be treated as separate because validation is not essential for construction and once all variables are optimal execution is a standalone procedure.

2.3.7 Result Evaluation

Once a set of previously unseen or test documents has been classified the performance of the applied categorisation solution is measured to enable comparison with other approaches. These performance measures are usually based exclusively in terms of label assignment effectiveness and provide a generalised assessment with respect to the entire dataset in the form of a single derived

value signifying the percentage of appropriately classified documents. In the case of single-label text categorisation problems the measure of correctly assigned labels alone, known as ‘precision’, is sufficient for determining one classifier as being superior to another. However in multi-label problems, where documents may be assigned a mix of correct and incorrect labels, precision is not adequate on its own and is typically combined with another measure such as ‘recall’.

2.4 Chapter Summary

This chapter described a universal set of stages generated from existing interpretations of the text categorisation process, which can be combined in a variety of ways to define any foreseeable solution composition. It outlined the purpose of each and highlighted the general roles they have in the overall process, noting critical components and showing the complex interactions that may potentially occur between them. Constructing solutions from these independent stages provides a level of flexibility not currently achievable, as their functional separation allows straightforward addition of components and easy substitution of those with similar operations. Subsequently the architecture permits the greatest combination of individual techniques and means systematic approaches can be employed to simplify the discovery of optimal configurations.

It is clear text categorisation involves more features and intricacies than typically acknowledged by present explanations of the process or systems that deal with it, though several common features are prevalent throughout all of them. Unfortunately these shared facets often receive the focus, while others are sometimes considered trivial and become neglected or deliberately disregarded as a consequence. However, designing distinct algorithms that adhere to the proposed structured stages forces methods to take the entire procedure into account, but means every component only needs to be developed according to the particular area it is meant to affect. Despite the slight boundary overlap this also prevents unintentional modification of aspects not under investigation, as these parts do not require alteration and can simply be recycled from previously established benchmarks.

Unlike the existing generic and specific methodologies that have been summarised, the architecture presented is highly flexible and able to be directly mapped to a wider variety of problem solutions. It greatly reduces ambiguity and makes the description of new techniques straightforward, as prior configurations can be given as a base reference and just the modified elements must be fully detailed. The next chapter reiterates this by examining each stage in turn and surveying collections of discrete algorithms that belong to them, demonstrating how they fit present techniques and can be treated as atomic entities.

3 Composition of Text Categorisation Stages

3.1 Chapter Contributions

This chapter supplies further information about each of the distinct stages outlined in the previous section. It describes their purpose in greater detail and surveys an assortment of existing algorithms and techniques associated with them, whilst highlighting any considerations that should be taken into account. By doing this it also serves to demonstrate that current solutions can be easily broken down into simple modular components and shows how they are mapped to the proposed architecture. At present it is not feasible to apply this procedure to most published text categorisation solutions in their entirety, as the majority have key elements omitted from their descriptions.

A thorough overview of existing methodologies relating to the individual stages of the categorisation process is provided, dealing with any confusing subject matter caused by contradictions or ambiguity found in literature. It expands on the sparse content of many other surveys by not focusing on only a single step or limited subset of algorithm types believed to be the most significant and instead reviews the entire range of elements involved in the process. The diversity and complexity of every discrete stage is emphasised, not just those responsible for actual classification, clearly portraying the critical role that each could have in the overall procedure.

By illustrating the potential for elaborate solutions it shows that underestimating the inherent intricacy of the categorisation process is bad practice and that no aspect should be neglected or intentionally disregarded as trivial, as each is a valid and important area of research in its own right. It also helps to stress that the combination of separate stages into a specific algorithm should be avoided, with it being more appropriate for them to be atomic components of an overall solution rather than part of a single entity that amalgamates several independent concepts.

3.2 Data Selection

Despite the wide diversity of applications involving some form of text categorisation, there are only a few notable document datasets that are readily available as comparative benchmarks [60][61]. Even those that are well established still have a number of issues that might limit their value for a wide range of practical investigations, as though many are initially derived from genuine problems they normally undergo heavy refinement [62][43][63]. These cleansing processes can give rise to specific subsets that detract from the authenticity and eliminate certain traits [43][64][65], leading to experiments less representative of actual situations. Cases also exist where alterations are ongoing for decades, with dated examples still utilised that portray just a small selection of the features that might be encountered in current scenarios.

While several plausible reasons exist for this crucial lack of resources, their absence is an important factor that significantly contributes toward the slow progression in the area. This is because any feasible solution must be tested under realistic conditions and while the present benchmarks allow for limited comparison and evaluation, they seldom provide adequate simulation diversity or scalability. Ideally all research should be employed on acknowledged datasets with clearly defined splits that include a wide range of controlled and reproducible problem characteristics, but this would require an unprecedented level of standardisation and collaboration. Though an appropriate variety of such freely accessible material would discourage the continued exploitation of custom datasets and atypical subdivisions, which in turn is likely to improve advancement within the field.

The principal reason for the low number of superior text resources, particularly those of a suitable size with appropriate labels, is that their production involves a considerable investment of time and effort. This is due to each dataset requiring the collection, formatting and correct category assignment of a substantial volume of items, which is usually a predominantly manual and extremely labour intensive task [60]. However it is essential that adequate quantities are gathered, as they need to be partitioned into several distinct groups that facilitate the training, optimisation and testing of feasible solutions. Complete and accurate labelling of the entire set is also vital, being the foundation of the experiments conducted, with strong evidence showing that low quality data containing missing or miscategorised information affects the validity of perceived performance [63].

Additional issues also contribute toward the difficulties encountered when creating viable datasets, which cause an increase to financial cost or in some cases even prevent the collection of information entirely. One such challenge is common in the investigation of bespoke applications, as the necessary generation of useful data frequently demands specialist knowledge that is prohibitively expensive or unattainable. Some problems are also impeded by associated confidentiality or privacy constraints, for instance the categorisation of realistic emails, formal examinations or restricted corporate archives. While others suffer from vague or deficient information that complicate precise

category assignment, such as partially complete or open format surveys, hypertext with broken links or unstructured content referring to contradictory subject matter.

In an attempt to counteract the manual collection and labelling issues there is potential for fabrication of artificial documents, though manufacturing feasible textual data in satisfactory amounts can still prove exceptionally hard. The most challenging aspect of producing synthetic content is obtaining an acceptable balance of realism and complexity, without yielding any undesirable bias toward particular characteristics. While this may be considered moderately easy for limited types of information, for instance those with a formal structure and strictly defined themes, there are significantly more where it is unrealistic without extensive manual participation.

Unfortunately it is the more demanding circumstances where the need for benchmark datasets is greatest, as the majority of existing resources emulate similar content and tend to lack the obscure traits that are present in many varieties of restricted information. As a result and despite the obstacles imitation corpora have been created that endeavour to mimic realistic text document collections [60][66], though they are rarely made openly available and only limited descriptions seem to be published.

3.2.1 Considerations

A small selection of benchmark and popular text document datasets are available for use in text categorisation experiments, though most have been split into a number of subsets containing varying characteristics and distributions. There are also datasets covering specific areas of the problem domain that are not readily available, further complicating the issue of deciding which corpus to use when investigating a new approach. Despite this diversity there are no specific rules regarding the choice of dataset, but several considerations should always be kept in mind while making a selection.

Obviously if the research project is in relation to a real-world or industrial application a dataset explicitly designed for or taken directly from that particular application should be used, but in the case of comparative or academic based experiments more thought is needed.

3.2.1.1 Type of Research/Specific Problem Area

All problems in the text categorisation domain have some similarities and share the basic concept of assigning unseen text documents to one or more predefined categories derived from previously labelled training data. However, individual problems also have their own features that are often unique and likely to affect the choice of methodology for gaining an optimal solution, therefore, it is important to select a corpus that accurately represents the research being conducted. Examples of problem sub domains that exhibit influential characteristics with similar roots, but subtle and significant differences,

are; e-mail spam detection [67], e-mail filtering [66][68][69], tracking web spam [70] and hypertext filtering [15][71].

3.2.1.2 Dataset Characteristics

Many factors exist within corpora that affect the effectiveness of different approaches by a varying amount and these are the reason that no single technique is able to dominate in every situation. The primary dataset attributes that influence results the most are given below:

- Inclusion of labelled/unlabelled documents
- Dataset noise, in the form of irrelevant or misleading information
- Volume of documents and category frequencies
- Distribution of categories in chosen selection of data
- Quality of data annotation and meta information
- Composition of text content (structured, semi-structured, unstructured)
- Presence of multi-class or multi-label traits

The decision to use a sample of a larger corpus also requires careful consideration, as there is sufficient evidence to show that subsets rarely provide a true reflection of the overall encompassing dataset. Quality of the labelled training documents is another aspect that plays a significant role, as datasets containing misclassified data have the ability to substantially restrict the upper performance bounds of any categorisation system applied to them [63].

3.2.1.3 Specialised Algorithms

If the research conducted introduces, or is to be compared against, solution approaches that consist of techniques combined with distinctive traits designed to solve particular problems, then experimental datasets exhibiting characteristics of that problem should be used. This ensures that gathered results portray an adequate appraisal of the algorithms, without a bias that might make some seem inferior if they are deliberately not complimented by features of the chosen data. Obviously the inverse is also true and wherever possible a selected dataset should not be so fine tuned toward an individual approach that results become unjustifiable or incomparable with other algorithms.

Every algorithm has data characteristics that suit them better than others, even the popular and consistently competent approach of Support Vector Machines [17] benefits from category distributions that fit the employed kernel function. Conversely statistical algorithms depend less on category distribution but tend to suffer from high levels of noise [72], which bias calculations and degrade performance. Similarly nearest neighbour techniques are also less sensitive to category distribution,

though experience decline in efficiency when unlabelled documents or extreme category frequencies are present, as rare categories may be ignored as insignificant.

3.2.1.4 Solution Scalability

Commonly only a subdivision of a dataset is utilised for testing categorisation solutions [43] and very little information is obtained on how they might cope when applied to larger datasets that reflect dimensions found in many real world circumstances. This is especially the case for research based on the “OHSUMED” medical journal dataset [73], which is a widely used benchmark with many derived subsets that each represents only a minor percentage of the entire corpus. As these subsets may produce inaccurate predictors of the parent dataset, very few solutions tested on them can be guaranteed to adequately scale in performance or efficiency in order to solve problems containing the increased set of problem dimensions. Neural Networks are one such method that rapidly becomes impractical and highly infeasible as problem dimensions grow [18].

There are only a few ways to handle the issue of scalability, allowing expansion of small scale algorithms to deal with substantially large scale problems, with the two main ones being more efficient data representations and the application of highly aggressive dimensional reduction. Both of these areas have been heavily researched [74] [16], particularly in the case of dimension reduction where limitations regarding the trade off between smaller feature space and performance degradation have been comprehensively studied [75] [76].

3.2.1.5 Comparability Issues

One of the principal issues with text classification experiments is that of inconsistent results and test conditions that are either difficult to reproduce or incompatible with each other [35][43]. There is little use in implementing new concepts if the outcome is not justifiable by means of legitimate comparative evaluation against solutions known to perform well on the same or notably similar problems. As such careful consideration of all the factors involved is necessary to ensure standardisation of results to a minimum level where they might be validated by further research.

Possibly the most important factor is the choice of corpus or data subdivision employed during experiments, as this determines which other existing research results can be directly compared to any conducted trials. The use of an abnormal dataset, even if originating from a popular or benchmark collection, will mean only indirect or less reliable comparisons can be achieved, such as the use of a common baseline solution. Assuming such a baseline could be found, assessments would only be relative to the similarities between the different datasets it is applied to and determination of efficiency would be difficult. For a fair appraisal every stage except those under scrutiny would also need to be

kept identical throughout all tests, which becomes harder when performing comparisons across multiple datasets.

3.2.1.6 Resource Availability

In the event that a custom dataset must be used for a particular investigation, where possible a copy of the data should be made publically available to allow others to perform their own evaluations. Unfortunately this is not always practical, for example when data is classified such as personal e-mails [66] or essay papers [38], and in these cases dataset characteristics should be supplied instead with as much detail as feasible so that similarities to other potential databases can be ascertained.

It is not only the data itself that must be taken into account when considering the availability of resources. As technology advances, general access to hardware is also rapidly increasing and while this allows faster construction and execution of solutions, the widespread lack of defined memory and computation information makes it impossible to accurately determine data processing requirements. This can become a highly restrictive factor when solutions are tested on platforms with differing resource availability, particularly as there is such a sizeable range of imposed physical hardware and software limitations between the various machines in common use. It is therefore important that the selected data be of a reasonable size, with characteristics and problem dimensions that allow the solutions under investigation to run to completion with acceptable levels of resource consumption.

3.2.2 Dataset Sectioning

During investigations text datasets are divided into two mutually exclusive groups, referred to as the training set, which is used to train a classification model, and the testing set, which has the trained model applied to it for the purpose of evaluation [4]. For any particular corpus the splits may not always be comprised of the same elements or utilise the entire document collection and several variations are usually defined, offering a selection of problem characteristics and varying degrees of relative hardness [43]. This means they can also yield significantly different results, even with an identical categorisation solution, so it is imperative that they are duplicated exactly when performing comparative analysis.

$$\text{Generality}_c = |\{dc : dc \in D\}| \,/\, |D|$$

The ‘generality’ for any derived subset of documents D is the total percentage of enclosed documents d_c that are associated to a particular category c and measures how well that category is represented by the group [4], where D can relate to a testing, training or validation set. Higher values show that the category has greater discriminative information available, while lower values portray the opposite with

zero denoting a complete lack of data. Comparing generality makes it possible to approximate how a model might cope when given a specific category, for instance a score of zero in the training set but not in the test set highlights a new or unknown category that the solution is unable to handle.

3.2.2.1 Testing Set

The testing set is normally a smaller group and is not used at all during classifier creation, instead being treated as unlabelled and passed to the trained system so it can attempt to estimate the appropriate category assignments. Once this process is completed for every item the predicted labels are compared against the actual labels, with statistics extracted and supplied to evaluation metrics that calculate the level of correctness.

3.2.2.2 Training Set

In comparison to the testing set, the training set is nearly always larger, as it should ideally contain sufficient examples to adequately cover all possible text features and category combinations that may occur within the test data. This is essential as the relationship between these aspects determines the predictive model composition and if more are available there is a better chance of high categorisation accuracy, while absences potentially reduce performance. It is also important that the testing set is not part of the training process, or bias will occur toward those specific documents and prevent a realistic assessment of the outcome when genuine unlabelled data is categorised.

3.2.2.3 Validation Set

Often the training collection is further separated to create another small distinct group, known as a 'validation' or 'hold-out' set, for the purpose of fine tuning any parameters. There are several ways to generate this split, with the most common simply using a fixed number or percentage of random documents, while a more complex method is to employ a technique called 'cross-validation' [77]. Regardless of the specific partitioning mechanism, the solution is optimised by repeatedly re-training the model with adjusted variables until the best results are achieved, then a final model is typically produced from the full training data and these parameters [4].

There are two primary types of cross validation used in text categorisation the most widespread being 'k-fold cross validation', where k discrete divisions or folds of approximately equal size are produced and k predicted models are trained, each keeping a different division aside as a validation set. Alternatively 'leave-k-out cross-validation' may be utilised, though this requires additional resources, as an explicit model is needed for every possible permutation that can form the k divisions.

No strict definitions exist for calculating a suitable value for k but those most regularly encountered create either ten or the maximum applicable number of folds, which is equivalent the total quantity of documents in the training set and means that each one is independently validated. Obviously as the size of k increases the computational requirement also grows and all the outputs must be combined in some manner and appropriately averaged to provide an optimal configuration.

3.2.3 Reuters-21578

The Reuters dataset is a collection of newswire stories gathered during 1987[1] and has been subject to extensive revision since its initial creation by Carnegie Group Inc. and Reuters Ltd. as part of development for the CONSTRUE text categorisation system [78]. Originally it was made available in 1990, but was heavily modified and reformatted until it was publically released in 1993 and designated as 'Reuters-22173, Distribution 1.0' due to the number of contained documents [79].

It was decided that an updated version with less ambiguous content should be produced in 1996 and this was supplied with documentation that defined a standardised way of utilising the dataset, along with various typographical, formatting and labelling corrections. A total of 595 exact duplicates were also removed from the collection, leaving 21,578 documents which lead to the designation 'Reuters-21578, Distribution 1.0'. This highly popular edition was actively maintained for several more years by a notable group of text mining researchers, until an entirely new dataset known as RCV1 was generated to supersede it [62].

While still a commonly mentioned corpus in publications [80][63][81] experiments tend to be conducted on one of the numerous subsets rather than the entire collection, some of which are widely recognised and clearly defined [62]. The reason for forming the majority of these splits is due to the presence of excessive noise, mainly in the form of unlabelled test or training items which negatively impact performance evaluation. However, there are also many unusual divisions [35] and even adaptations of the identified subsets [43], each with distinct characteristics that make comparison difficult between the solutions applied to them [4][1].

The latest version is available as a gzipped tarball archive that is 8.2 MB and 28.0 MB uncompressed, with the preferred download location currently being the KDD archive at the University of California, Irvine (UCI) [82]. It is separated over twenty two chronologically ordered files and formatted in SGML, with one thousand documents in each except the last which contains the remaining five hundred and seventy eight. Individual items are comprised from a variety of attributes and elements that aid in

[1] Note that many publications stating the articles were collected between 1987 and 1991 are incorrect and subsets cannot be produced by using the year.

creation of the recommended subsets, with a Document Type Definition (DTD) that describes item organisation and several other files that outline category information and basic details.

Five different category groupings are independently noted for each document, though experiments typically only employ the 135 labels from the 'TOPICS' group, as the others have features considered to be of little practical worth [62]. Statistics supplied about category distribution and density are strangely varied throughout literature, as is information regarding the quantity and composition of the textual content and unique terms, which is most likely due to frequent use of dissimilar subsets [2].

However, the data does exhibit multi-label traits with a highly skewed category density, with the densest being present in several thousand documents while the majority of labels apply to less than ten instances. In addition, for any specific category grouping the maximum number of assignments to a single document is fourteen and the minimum is none. Although a lack of association is either because an item is a true negative example, where it legitimately does not belong to any categories, or due to missing labels and misclassification of the original data [62].

3.2.3.1 ModApte Split

The Modified Apte or 'ModApte' split of Reuters-21578 is the most widely used subset for text categorisation and one of three recommended by the dataset creators, being based on the partitions employed by experiments conducted on the original Reuters-22173 version [62]. It contains a total of 9,603 training and 3,299 testing instances, leaving the remaining 8,676 documents unused, while allocating all those created on the 7th April 1987 and before to the training set and those from 8th April and onwards to the test set.

Only the 135 labels belonging to the 'TOPICS' category grouping are considered with a total of just 118 actually being utilised, while the mix of category assignments includes some that are exclusive to either the training or testing set. A selection of documents are left unlabelled to permit investigation of the potential impact caused by missing labels and there are also a number of empty samples present, where a document has a title but no entry in the main body of content.

These distinct features are important as a another subset named 'ApteMod' is also mentioned in literature [42], which is created by selecting categories with at least one sample present in both the training and testing partitions and then removing all unlabelled documents. A process that yields a division of 7769 training and 3019 testing instances that are associated with only 90 categories, making it produce results that are incompatible with those from the full ModApte split [35]. There are also many other cases that suffer from similar compatibility issues, such as where every unlabelled and empty document is ignored, which generates 7,063 training and 2,658 testing instances [2].

3.2.4 RCV1

Reuters Corpus Volume 1 (RCV1) [65] is a dataset consisting of 806,791 newswire stories published from the 20th August 1996 to the 19th Aug 1997 that was originally released in 2000, it is formatted in XML and distributed exclusively on compact disc having an uncompressed size of around 2.5 GB. Though used in the 2001 and 2002 Text Retrieval Conference (TREC) Filtering Track workshops, since 2004 it has only been available upon approval of organisation and personal level requests which must be submitted to the National Institute of Science and Technology (NIST) [83]. While intended to supersede the older and significantly smaller Reuters-21578 corpus its adoption has been limited, seemingly due to licensing issues that require disputes be settled in England, leading to rejection by many American universities [43].

A multilingual resource is also available, designated 'RCV2', which contains over 487,000 articles in thirteen different languages that employ a variety of alphabets and writing systems. They were collected over the same timeframe as RCV1, but some languages do not span the whole period and the stories were composed by separate local sources. In addition a substantially larger and more recent dataset, the 'Thomson Reuters Text Research Collection' (TRC2), is obtainable via the Internet and includes 1,800,370 news stories gathered between the 1st January 2008 and the 28th February 2009. Both of these resources are subject to the same agreements that cover RVC1 though and need to be requested and approved in a similar manner [83].

3.2.4.1 LYRL2004 Split

Reuters Corpus Volume 1 version 2 (RCV1-v2), also referred to as RCV1-v2/LYRL2004 in reference to the authors of the introductory publication [65], is a publicly accessible corrected version of RCV1 consisting of 804,414 documents that are deemed to have valid category codes. It is preformatted with several data preparation stages already applied, which serves to scramble the original content in a manner that avoids any possibility of breaching the licence agreements. While this enforces certain pre-processing and indexing criteria, it does make open distribution viable and subsequently the data can be easily acquired as a set of eighteen electronic journal appendices [84].

The data is only available as a term-by-document matrix or cosine normalised TF-IDF vector representations, with universal processing that includes the elimination of 571 stopwords and stemming, which consequently removes most of the punctuation. It has further been partitioned so that the first 23,149 chronological documents, created between 20th August 1996 and 31st August 1996 inclusive, are used as training instances and the remaining 781,265 are designated for testing. Three distinct category groups are also present, with a total of 103 assignable 'Topic' labels, 101 of which have at least one positive training example, 354 'Industry' and 366 'region' categories. Each of

these displays multi-label traits and is hierarchical in nature, with any missing ancestor labels being added as part of the update.

3.2.5 OHSUMED

The OSHUMED collection was originally defined at the Oregon Health Sciences University to assist with Information Retrieval research [73]. It contains a subset of 348,566 references from more than seven million that are part of the on-line medical information database 'MEDLINE' and comprises a mixture of titles and abstracts from 270 medical journals over a five year period from 1987 to 1991. After agreement by the National Library of Medicine the MEDLINE references became accessible, on provision that the data is only employed in an experimental manner and that users are explicitly made aware that the content is incomplete and obsolete.

Strict dataset divisions were used in individual tasks of the 2000 TREC-9 Filtering Track workshops [64] and though the track itself was last hosted in 2002, when it was based around the RCV1 corpus, updated information about OHSUMED was released in 2007. Currently it is openly available as a gzipped tarball archive, which is approximately 132 MB or 400 MB uncompressed, from the Text Retrieval Conference (TREC) website sponsored by the National Institute of Standards and Technology (NIST) [85].

Each text item has several possible fields, including the title, abstract, author, source, publication type and a list of Medical Subject Headings (MESH) indexing terms, which form a hierarchical category structure with only the most specific typically recorded [64]. Consisting mainly of unrefined bibliographical references the data is considered relatively complex [18] and requires significant computational resource to process, meaning it is rarely used in its entirety. As with other datasets it is commonly split into a number of smaller subsets intended to eliminate the majority of noise and reduce problems to a practical size, however these suffer from a variety of unusual partitions that make comparing solutions difficult [35][17].

3.3 Pre-Processing

Data loaded in a categorisation system frequently needs preparation, such as cleansing unwanted noise, removing obvious irrelevant information, conversion of relevant information to a beneficial form and deriving potentially useful metadata. The principal aim of these processes is to purge or modify extraneous aspects from the entire dataset that will clearly not be of use to the chosen classification solution or that may even impede performance.

Unfortunately this stage is often regarded as trivial in many publications, which have a tendency to completely neglect it [37][86] or state only that general pre-processing took place with omission of exact details [16][3][87]. However it has been proven that not only can the application of these techniques typically give positive effects on classifiers performance [88][89][90], but the manner in which they are executed might also make a significant difference to results [91][92].

3.3.1 Formatting/Annotation Removal

A common pre-processing step, formatting and annotation removal serves to normalise the layout of text documents and distinguish unwanted comments, symbols or meta-information embedded within a dataset. For example many corpora, including the popular Reuters-21578 variations [79][62][65], include markup annotation or tags to separate, label and expand the individual text documents they contain. While this information is of importance for many research tasks, including the organisation of data subsets and accurate comparative result evaluations, it is rarely desired in a raw format when being applied to the training and testing of categorisation solutions.

The actions carried out during this stage are problem dependent, but may include; conversion to lower case text, removal of font types, standardisation of whitespace, deletion of punctuation and obscure characters, culling the beginning or end of the data, standardisation or exclusion of numerical values, removal of markup annotation (for example HTML or XML tags) and elimination of any header, comment or other supplemental information not forming part of the main document content. Due to the nature of text classification tables, pictures or diagrams included as part of the content are also normally excluded, though any captions may be retained. In most cases the end result consists of only lowercase plain-text with single spaced words that contain the most basic symbols of the relevant writing-system.

Despite these seemingly major changes the intrinsic content of the data is rarely considered as being altered, with any relevant content that gets lost normally seen as acceptable compared to the gain in practicality. This has probably become the accepted trend as the bag-of-words representation, used almost exclusively throughout text categorisation research, is unable to suitably represent much of the removed information anyway, making it appear mostly irrelevant.

3.3.2 Stopword Elimination

The majority of text documents contain words that exist purely to enhance readability of the content, without influencing any of the intrinsic meaning. Generally these are referred to as ‘stopwords’, ‘stop words’, ‘noise words’ and the ‘negative dictionary’ and often embody a large portion of document content, sometimes more than forty percent, while accounting for only a tiny minority of the overall unique vocabulary [93]. As a result they are frequently removed as a pre-processing step in order to greatly reduce data size and resource requirements, whilst theoretically maintaining all of the useful information.

Words usually considered for elimination include pronouns, prepositions and conjunctions, though different systems often have varying notions on what should be a stopword and therefore tend to have their own specific lists. Obviously this can create discrepancies in generated results if exact details of each list are not provided or utilised. This is commonly the case in text categorisation research, where deviations seem to be disregarded due to not substantially affecting classification performance, though there is little proof of this and variations can sometimes be considerable. A fact easily seen by comparing publically available lists defined for general text processing tasks, for instance the default SMART information retrieval system list [94] and the one recommended as part of Snowball [95] differ by 397 words.

Application Source	Reference	Length
SAO/NASA Astrophysics Database System (astronomy)	[96]	813
SAO/NASA Astrophysics Database System (physics)	[96]	813
Onix Text Retrieval Toolkit (default SMART)	[94]	571
MySQL Full-Text Search List (version 5.1)	[97]	544
Onix Text Retrieval Toolkit (Common list)	[98]	429
Snowball recommended	[95]	174
Derived Internet Search Engine	[99]	36

Table 3.1 : Sample Stopword List Sources and Lengths

Aside from the type of problem being investigated another important factor is the exclusivity toward a particular language, as lists contain a static set of predefined words that are only useful with the language they were designed for. While possible to determine lists for any language it is not a trivial task, as grammatical and syntactic variation between writing systems makes direct translation infeasible, however independent lists are available for several languages [25][26]. Limited study has also been conducted in the area of multi-lingual stopwords, but due to the considerable word

collisions most solutions first attempt to automatically determine the language and then simply apply a relevant predefined list.

I	At	for	la	this	who
a	Be	from	of	to	will
about	By	how	on	was	with
an	Com	in	or	what	und
are	De	is	that	when	the
as	En	it	the	where	www

Figure 3.1 : Estimated Stopword List for Internet Search Engine

In an effort to avoid the lengthy process of stopword list construction and requirement of domain expertise when maintaining or producing them for specialised applications, attempts have been made at automatic generation using information statistically derived from the data. However, a closer inspection makes it apparent that all the methods are actually a form of dimensional reduction, for instance one employs dataset wide feature selection based on term frequency and a dynamically derived threshold [92]. The distinction is fuzzy though, as the removed words are determined via a sample subset and used to generate a true stopword list that is then applied to the remaining data. Use of this mixed pre-processing and dimensional reduction approach, called “Term-based sampling”, addresses the conventional issues associated with static stopword lists and computationally heavy term reduction techniques. Stated performance is be generally lower than the traditional approaches, though quantifiable values are not divulged and it relies on user defined parameters that can only be tuned by collecting substantial empirical data.

3.3.3 Word Stemming

Referred to as ‘conflation’ or less correctly ‘lemmatisation’, the process of stemming is a specialised type of term substitution that involves removing affixes to convert words into their base stem or root, which is a more generalised form containing the same inherent conceptual meaning. The first published stemming algorithm was developed by Julie Beth Lovins in 1968 and has become known as “Lovins stemmer” [100], influencing all subsequent stemmers. Production of stemming algorithms is a time consuming manual process and consists of generating a fixed set of language specific transformations that often need to be applied in a particular order. Due to predominantly manual construction combined with the extensive scope, variable nature and complexities inherent in most written languages, it is not uncommon for different algorithms to produce slightly different output formats [101].

This difference in output between algorithms occurs partly because most approaches, especially those that are rule based, seldom produce authentic forms of the root words, instead supplying only approximate representations. However, as the representations are always kept consistent for each particular algorithm results can still be used in a practical manner and only stemming imperfections or combining algorithms with incompatible representations will impact performance. For some tasks the lack of understandable root words may be considered unfavourable though, preventing manual adjustment and increasing analysis complexity. In these instances dictionary based techniques can convert representations to their true form, but require a considerable increase in resources to perform lookups and maintain the lexicon.

Sample Word	Actual Root	Porter Stem
consort	consort	consort
consorting	consort	consort
conspicuous	conspicuous	conspicu
conspicuously	conspicuous	conspicu
conspiracy	conspire	conspiraci
conspirator	conspire	conspir
conspire	conspire	conspir

Table 3.2 : Example of Stemmed Word Output

Aside from the potential of different output formats, it is also highly improbable that algorithms will always yield completely accurate results and two popular metrics have been developed to quantify the level of error [91]. Overstemming records false positives, where words are stemmed to the same root when they should actually be treated as distinct, for example “university” and “universe” are often stemmed to the same base despite having unique meanings. Understemming is the opposite and records false negatives, where two words should be stemmed to the same root but are not, such as the derivation “ran” which is rarely stemmed correctly to the root “run”.

The aim of all stemming algorithms is to minimise the occurrence of both Understemming and Overstemming, though the reduction of one can often cause an increase in the other. In this situation text classification generally takes preference toward Understemming as many classification solutions are less sensitive to this form of error and merely index each derived base word as a separate term. By comparison Overstemming entirely removes the distinction between individual terms and the more informative and relevant those terms are for category determination the more harmful this type of error becomes.

Stemmers have been applied to many text applications, where they have been shown to increase performance [88] [89], lower problem dimensions [102] and permit enhanced user interaction [101]. A good example of this is demonstrated by large scale information retrieval systems, such as Internet

search engines, which necessitate indexing extreme volumes of textual data and employ stemming to not only return exact query matches but also closely associated variations. Integration of stemmers is also common in document routing and filtering, including the use of novel applications such as applying customised stemmers to clarify content prior to performing spam detection [88].

Though stemmers are available for many languages, including Swedish [89], Spanish [103], Farsi [104] and many others [25][26], none are presently able to handle multi-lingual text and must first determine the language of each word and then apply a suitable algorithm [24]. Currently one of the largest collections of implemented algorithms is for English, due to its widespread use throughout the world and relatively simple morphology that has the ability to easily split words into individual morphemes or other linguistic units. In comparison Semitic languages, like Hebrew and Arabic, are considerably harder to stem as word formation frequently involves direct modification of the root word rather than just prefix or suffix addition. This behaviour is also experienced in English although to a much lesser extent, for example the plural “feet” is a modification of its root word “foot”, and is the cause of most inaccuracies.

3.3.3.1 Rule Based Stemmers

The majority of stemming algorithms consist of a fixed group of morphological rules that are applied in a specific order to transform a given word into a root form. Typically these are fast to derive the root forms, have low resource consumption and attempt to derive a base form for any word, which can be a positive or negative trait as they tend to focus more on inflection and do not always cope well with derivation.

Of the English stemmers the most extensively used is a rule based one called “Porter stemmer” [102], originally developed using BCPL in 1979 by Martin Porter as part of a large information retrieval project. Since inception many versions have been created for various development platforms, but due to ambiguity of the initial defined rules implementations often exhibit differing degrees of functionality [105]. There are also several other algorithms in assorted languages referred to as “Porter stemmer” simply because they are rule based, though most have little resemblance.

The original Porter’s stemming algorithm first converts a word to a standardised format and then removes a static set of prefixes before applying progressive stages to attempt inflexional and derivational suffix removal and finally ends with a mostly cosmetic stage.

1. Initialisation Phase – Converts to lower case and removes punctuation
2. Prefix Removal – Removes set list of prefixes
3. Suffix Trimming 1 – Prepares plurals and past participles
4. Suffix Trimming 2 – Fixed suffix modification
5. Suffix Trimming 3 – Fixed suffix modification
6. Suffix Trimming 4 – Fixed suffix modification
7. Suffix Trimming 5 – Final clean up stage

Figure 3.2 : Basic Stages of Porter's Stemming Algorithm

A new version, designated “Porter2” and also created by Martin porter, has been developed in an attempt to solve the issues with the original Porter stemmer [106]. This has improved accuracy and is open to revision, though the actual stages and most of the core rules remain the same and less than five percent of words are estimated to be affected by the updates made so far. For practical applications it is recommended to use the newer algorithm, but for research purposes the original version is preferred as it is strictly defined and not subject to change, making comparative results more reliable. Unfortunately this makes it likely that experimental results will be subject to those initial flaws for the foreseeable future.

3.3.3.2 Brute Force Stemming

The simplest way to perform stemming tasks is by maintaining a lookup list containing direct one-to-one mappings between each word root and all possible derivations, as done in the lexical database “WordNet” [107]. Not only is this highly accurate, but it provides human understandable output and requires little domain expertise to keep updated. On the downside, storing the list and performing lookups requires extensive resource, the initial designs can be complicated to keep efficient and the absence of a lookup word will result in complete failure [108].

3.3.3.3 Dictionary Stemmers

Dictionary based methods are similar to brute force approaches but carry out word modification, from basic inflectional suffix removal to complex lemmatisation, prior to performing a lookup for the root form. The advantage is the possibility for precise detection of obscure root associations, especially if additional contextual information can be extracted by lemmatisation, whilst also potentially maintaining a smaller lookup list than brute force methods alone. However it has complete reliance on

the core modifications and available scope of the dictionary, which can be complex to implement and extremely resource intensive to use.

1. Suffix removal
    a. Conversion of plural to single
    b. Conversion of past tense to present
    c. Removal of “ing” ending
2. Final dictionary look up for root discovery

Figure 3.3 : Stages of Krovetz Stemmer (K-Stem)

A prevalent dictionary stemmer is “Krovetz Stemmer” or “K-Stem” [109], which performs highly accurate suffix removal followed by a dictionary lookup focused on derivations, allowing the determination of root forms based on syntactic categories.

3.3.3.4 Statistical Stemmers

Based on the concept that affixes will have a different probability distribution than the root word they are attached to and that a root will always have a significantly similar form, statistical stemmers attempt to establish and remove affixes. This is done using a variety of string analysis techniques [110], including examination of word component frequency, probability of word structure and string similarity clustering.

Statistical stemmers suffer severely from high resource consumption and an inability to appropriately handle obscure derivational root forms, which may lead to lower performance than alternative types of stemmer. However they have the advantage of being fully automated, unsupervised and completely language independent, making them an important resource in areas that lack native stemming implementations.

3.3.3.5 Hybrid Stemmers

Hybrid solutions have the ability to capitalise on the potential benefits of multiple algorithms, which could mean mixing strong inflectional rules or combining custom derivational and inflection stemming capability. The former is reported to have been tested to an extent by combining the accurate suffix

removal of Krovetz stemmer with the compact format of Porter stemmer, though exact details are difficult to find [108].

3.3.4 Static Metadata Extraction

Metadata is often defined as ‘data about data’ and provides additional meaning or information that might otherwise be difficult or impossible to determine by casual observation. There are generally two different types of metadata associated with text, the most prominent being embedded or static information that is explicitly supplied as part of, or in addition to, each element of the data. This static information can take many forms and is either intrinsic to the actual text composition, like structural or navigational tags, or is appended separately from the main content, such as titles, author details or supplemental keywords [71].

3.3.4.1 Synsets

A common form of metadata and also known as a ‘synonym ring’ or ‘cohesion’, a synset is a collection of synonyms and semantic definitions that attempt to conceptualise the intrinsic meaning of words or word phrases. Their intension is to allow text with similar meaning, but different vocabulary, to be grouped together in a manner that makes classification and similar text related tasks easier.

While dynamic generation of synsets is possible [31], with the most popular resource for such tasks being WordNet [107], they are often incorporated as a static part of the document content. This is especially the case with hypertext, as a great deal of research has been conducted on enhanced synset annotation during development of the semantic web [111], which aspires to increase the interaction between machines and humans. Though the evolution of hypertext is still in early stages, several ontology languages have been defined that all have the common aim of implying additional information about the data they relate to. Acknowledged by the World Wide Web Consortium (W3C), the most widely accepted of these ontology languages are ‘Extensible Markup Language’ (XML) [112], ‘Resource Description Framework’ (RDF) [113] and ‘Web Ontology Language’ (OWL) [114].

3.3.4.2 Folksonomy

Often referred to as ‘collaborative tagging’, ‘social tagging’, ‘social classification’ and ‘social indexing’, folksonomy forgoes the assumption of external metadata not being available and instead relies on definition by sources outside the original data scope. Comprised entirely of embedded consumer supplied information is has been used extensively with Internet based hypertext, making many consider it one of the first steps toward realising the notion of the semantic web. Due to the

fundamental concept it is also predominantly used in policy based classification, where a bias towards the intended audience rather than a globally acknowledged criteria is preferred, deliberately taking user request preferences and context into account.

Though little formal research was initially conducted in the area [115], as the number of applications employing it grew the intensity of research increased, with multiple studies focusing on statistical trend analysis [116]. These investigations reveal that, given enough time, user generated meta-labels tend to converge towards a stable taxonomy [117], which shows that categorisation in a standardised form is possible without conventional classification processing.

Despite the conclusive evidence there are still concerns about the relevancy of purely consumer based classification, as lack of official standards and inconsistency ensuing from diverse backgrounds and knowledge can yield excessive obscure meta-labels. As a result future directions may involve the integration of controlled dictionaries [118] or ontologies [119], imposing boundaries on the available meta-label vocabulary, but also detracting from the simplistic nature of the technique.

3.3.5 Dynamic Metadata Generation

Traditionally less prominent than static metadata, with regard to text classification, is dynamic metadata that is derived directly from the document text, normally by means of statistical or contextual analysis. This type of information is generally taken for granted during manual classification, but neglected in automated systems that are only able to process the data explicitly supplied to them.

3.3.5.1 Statistical Content Analysis

Explicitly including basic statistical information as part of the document data, provides a mechanism to retain some of the details routinely lost during the various stages of a categorisation system. As an example, the common pre-processing steps of stopword elimination and stemming can alter the number of unique words and length of a document, while employing the popular bag-of-words representation causes the loss of all aspects relating to document, paragraph and sentence structure. Further combining these with moderately aggressive dimension reduction and a normalised index weighting metric, as is regularly done, yields a final format that bears little resemblance to the original and offers considerably less information.

While the loss of this core data proves inconsequential in many classification problems [120], there are some that experience increased performance by their inclusion, such as those heavily reliant on stylistic properties like authorship identification [11].

3.3.5.2 Named Entity Recognition

Frequently textual data contains highly informative rare terms that can lose much of their worth if normalised, or risk be removed completely during dimensional reduction. Examples are terms that rely on embedded punctuation or capitalisation, such as acronyms or subject and object identification, the descriptive ability of which greatly decreases if the original format is altered. To prevent modification of these terms they are first discovered, usually by means of a fixed dictionary, gazetteer or a defined set of rules [32], and then marked as immutable so other processes do not change them.

Named Entity Recognition (NER), also known as 'entity identification' and entity extraction', is considered a principal component of several text related tasks and can have significant positive impact on results [121]. However manual recognition techniques necessitate extensive domain expertise and automated systems, while achieving comparable accuracy, are also resource intensive and often domain dependent [122].

3.3.5.3 Part-of-Speech Tagging

Part-of-Speech tagging (POS or POST), also referred to as 'grammatical tagging' or 'word-category disambiguation', is the process of denoting the linguistic category of a word based on its syntactic and morphological behaviour. There are two primary groups of algorithms; rule-based [33], which exploit characteristics of test data to define explicit rules, and stochastic [123] [124], which employ probability distributions calculated from sample data.

In general rule-based algorithms tend to be less accurate than stochastic methods, but generate much smaller and simpler models, making them appreciably faster to construct and execute. However, there are rule-based techniques available that automatically extract and dynamically improve their rule set, providing levels of accuracy comparable to the best performing stochastic algorithms [33].

When performing part-of-speech tagging the principal concern is word sense disambiguation, in particular the problem of polysemy, as many words can be used in either noun or verb forms without specific morphological alteration. Normally this can be overcome by using the context of a word to distinguish its type, though ambiguous cases do exist where a correct result is impossible to guarantee. The procedure is also language specific, as different collections of linguistic categories apply to each language and writing system, with a large variation in size and complexity. However despite these issues, part-of-speech tagging is frequently a precursor to many other processes, including statistical content analysis [11], named entity recognition [120], selective stopword elimination [125] and some of the more complex stemmers [108].

3.4 Data Representation

Data representation or ‘tokenisation’ is required prior to most steps included in a classification solution and involves the conversion of data into a suitable format for the process about to be applied. In general this involves breaking down each document into a set of fundamental components, referred to as ‘terms’, ‘tokens’ or ‘symbols’, that each uniquely relates to a globally defined characteristic or identifying property found within that document. The definition used can considerably affect the overall solution performance and should convey as much information as possible that relates to the particular problem under investigation.

A substantial number of definitions exist, with most taken directly from text based Information Retrieval. They range from basic terms obtained directly from the text content, such as words or phrases, to complex multipart metadata extracted by analysis of the document structure or writing style, such as paragraph or punctuation statistics. Research carried out, by both Information Retrieval and Text Categorisation circles, also strongly suggests that no single data representation is exclusively superior to all others for every encountered problem type [13]. Furthermore, result trends imply that the more intricate terms, which are notably more resource intensive to derive, seldom give rise to performance gains deemed worthy of the additional overheads [1]. However, it ultimately depends on the characteristics of the problem being examined, for instance it is unlikely that basic terms would be sufficient to classify a corpus containing extremely high levels of homonymy, polysemy and synonymy.

3.4.1 Bag-of-Words (BOW)

By far the most common representation of text documents is to simply consider them as a big collection of words [16], sometimes called a ‘sparse Bag of Words’ (sBOW), with each word being treated as an individual term. This is considered the most basic representation, preserving minimal information and retaining no word ordering or contextual meaning from the original text, which has led to questions about its usefulness for some problems [126][127]. Often it will also result in a large number of derived distinct terms, frequently reaching thousands or tens of thousands depending on the specific corpus investigated, which sparsely relate to individual documents [80]. However despite these drawbacks it has proved a useful representation for many experiments, retaining enough statistical data necessary for popular indexing techniques whilst having a relatively minimal computational overhead. It is no doubt because of these reasons and its easily interpreted nature, that bag-of-words has been regarded as the superior representation for a substantial time, with similar interpretations published in relation to the first mechanised text indexers dating from the 1960’s and earlier.

3.4.2 Term Pairs/Phrases/Groups

A simplistic representation is that of word pairs and short word phrases, which employs the bag-of-words representation in order to extract elements that are contained within a term group. The benefit over individual terms is that some, though generally very limited, contextual information can be retained from the original text, which used in conjunction with statistical data may afford greater overall classification accuracy. Another advantage of using this technique is that it often provides an enhanced human understanding of the data, allowing increased interaction and minor adjustments as desired before the actual classification stage takes place. One area where this is particularly helpful is that of dealing with word sense disambiguation, as the presence of surrounding text often provides enough contextual information to greatly improve the determination of ambiguous words. For example the word “bat” by itself it could have a variety of meanings but use of a word pair, such as “cricket bat”, or a short phrase, like “bat uses sonar”, make the meaning virtually unquestionable.

Unfortunately with the increase in representation completeness there are a number of associated disadvantages, the principal one being a rise in computational effort as the term groupings are often derived after a bag-of-words has already been generated. Expansion in the number of generated problem dimensions is another possibility, though this is reliant on the structure of each grouping and it is possible that the number of features may actually be reduced. However, even if only sequential word pairs were extracted, though the dimensions would be one less than standard bag-of-words, the memory storage needed for each document would almost double in size. By contrast, if specific non-overlapping phrases were desired a reduction in the number of terms and memory requirement would be observed, but computational complexity would rise and suitable domain knowledge for phrase definition would likely be necessary.

Historically, use of this technique in both text categorisation and information retrieval research has afforded little improvement to application performance and where gains have been achieved they are rarely considered satisfactory in comparison to the drawbacks experienced. There are a few cases, however, where they have been shown to provide improved results that do warrant the additional overheads [37].

3.4.3 N-Grams

In relation to textual data n-grams are comprised of a linear combination of characters, words or phrases that form overlapping sequences of a fixed ‘n’ length derived from a larger sequence. The concept is similar to word pairs or phrases, in that it retains semantic information from the original data, but can generate completely abstract sequential character or word patterns that sometimes expose hidden relationships. N-grams have proved effective for certain text categorisation problems,

such as language classification [54], as well as many other research areas, including DNA sequence analysis for genetics.

The 'n' value used during term creation is a parameter chosen to maximise performance for the specific application undertaken and is usually optimised by empirical means. When elements comprise individual words the use of one–grams (uni-grams) form a bag-of-words representation, two-grams (bi-grams) are equivalent to sequential word pairs and three-grams (tri-grams) or above are analogous to sequential word phrases. However, if the elements are represented by a unit other than words, single characters for example, the problem characteristics become similar to that of fixed length pattern matching.

this is a good sample string		
Bi-gram (Words)	Tri-gram (Words)	15-gram (Characters)
"this is" "is a" "a good" "good sample" "sample string"	"this is a" "is a good" "a good sample" "good sample string"	"this is a good " "his is a good s" … "od sample strin" "d sample string"

Table 3.3 : Possible n-Gram Representations of a Sample String

As the length parameter is adjustable, limited control over the memory and computation requirements is possible and allows the representation to be tailored to specific tasks. This flexibility is useful as many n-gram arrangements produce excessive unique terms, especially with regard to text documents, often necessitating aggressive dimensionality reduction. In addition, depending on the element type and length value chosen, a bias can occur towards either highly frequent or infrequent aspects present within the data. While this is an issue for any representation format it can be greatly exaggerated by use of n-grams and the only way to limit it is to perform data smoothing, usually by applying filtering techniques that remove or supplement extreme occurrences of aspects to reduce their effect. Another potential area of bias are the elements at the start and end of textual data, as these must either be padded by 'null' elements to make them equal to the fixed length or only be included in full length terms, making them seem less frequent than the more centralised content.

It has also been stated that the inherent concept behind n-grams is flawed, as it suffers from a lack of ability to represent long range relationships or dependencies existing in the data. However, this is not a concern with the majority of text categorisation applications, as they typically make the assumption of complete term independence and this is enforced by most representation techniques employed.

3.4.4 Metadata

Many forms of metadata are available for text documents, including those that form part of the original data, for example folksonomy, semantic web synset tags or training set category labels, or that are dynamically generated as part of a processing step, such as generalised content statistics, named entity recognition or part-of-speech tagging. While most applications rely on metadata generated explicitly from term occurrence or absence in the main body of the textual content, it is uncommon for them to include other abstract forms of metadata despite the very nature of it potentially increasing classification performance [128]. It is also rare for a representation to be comprised purely of terms derived solely from metadata, with the preference being to use them as a compliment to the standard terms in order to guide the classification process in a subtle manner.

Traditionally it is acknowledged that use of metadata for text document retrieval provides trivial gains in performance and as such little research has been conducted using it in within text categorisation, with preliminary experiments reaching the same conclusion [120]. However growing interest in the semantic web and data traversal capabilities via navigational hyperlinks permits many forms of metadata not possible within information retrieval and these have shown promising classification results [129]. Popular data mining frameworks have also not ruled out the significance of metadata, with some allowing certain types to be marked immutable to prevent alteration or removal during future stages of the classification process [130]. This is a particularly useful feature for low frequency data, such as specific named entities, which might be extremely rare but may contain high levels of information about the data it refers to.

3.4.5 Hybrid Representations

	this is also quite a good sample string
Non-Overlapping Word Pairs	(this is), ~~(is also)~~, (also quite), ~~(quite a)~~, (a good), ~~(good sample)~~, (sample string)
Word Bi-Gram	[this is], [is also], [also quite], [quite a], [a good], [good sample], [sample string]
Combination	[(this is) (also quite)], [(also quite) (a good)], [(a good) (sample string)]

Table 3.4 : Combining Approaches to Exploit Their Strengths

Hybrids of any representations are also viable and can be accumulative, where terms of various representations combine to form a single set by applying them in sequence, with the output of one becoming the input of the next. These amalgamations provide complete flexibility and may counteract

the weaknesses of individual methods and capitalise on their benefits. For example a combination of non-overlapping word groups with n-grams could potentially represent semantic relationships over larger element collections than grouping alone, whilst demonstrating lower dimensionality and less memory consumption than just n-grams.

It is difficult to find research that directly tests hybrid representations, though many publications inadvertently make use of it, such as those that combine the bag-of-words representation with term grouping to obtain word pairs or add enhancements by means of supplemental metadata.

3.5 Data Indexing

Prior to most processing tasks textual data must be converted into a practical format that is suitable for machine consumption and this is achieved by the determination of two factors. Firstly the individual units of data must be appropriately symbolised, as is handled by the data representation stage, with each unit of data being referred to by an extracted term. Secondly every term must be assigned a quantifiable weight or set of values for each document it relates to, indicating the importance between that term and the particular document. This weighting or relational quantifier is usually numerical, to make direct comparison between terms and documents more feasible, and the resultant term and document associations form the foundation of all machine readable data indexing techniques.

Calculation of the index weightings or values can often be a computationally expensive part of a categorisation solution, but is frequently a key factor in the overall performance [59][131]. Though the measurements may include global or category scoped characteristics each index is intentionally document specific, in order to maximise the difference between individual documents and make comparative assessment more discriminative. It is also possible to have multiple metrics per representational unit, for example indexing techniques may assign several different weightings or retain the location of terms within each document, though this inevitably makes the process more complicated.

3.5.1 Standard Boolean Model

One of the earliest and simplest models used in automated text processing is the Boolean Model of Information Retrieval (BIR) [132], which employs concepts based on set theory and Boolean logic. Despite the simplistic nature it is a successful technique for many information retrieval related tasks and is still widely used, especially in systems with large dimensions that require efficient processing, such as large scale search engines.

However in the context of text categorisation logical query matching does not occur, instead being replaced by a variety of potential classification algorithms that consider only complete documents. This means only the document vector representations initially constructed by the model are utilised, making this equivalent to the Vector Space Model with a basic Boolean weighting function.

In respect to document vector creation, each document is converted into a bit vector that represents information by recording the presence or absence of terms in the text content. This can be defined as follows, making the assumption that all references are only to the data that remains after completion of any previous categorisation stages:

Let D signify the set of all documents in a corpus, such that $D = \{D_1, \ldots, D_j, \ldots, D_{|D|}\}$, where $|D|$ denotes the total number of documents and D_j represents a particular document.

Let T signify the finite set of unique terms extracted from the corpus, such that $T = \{T_1, \ldots, T_k, \ldots, T_{|T|}\}$, where $|T|$ denotes the total number of terms and T_k represents a particular term.

Let w_{kj} signify the presence '1' or absence '0' of a particular term k within a particular document j and w_{Tj} signify the overall absence or presence of each term from the entire set of terms T for the particular document j.

Then a document vector for a particular document j can be designated using the standard Boolean model as w_{Tj}, which takes the form $w_{Tj} = \{w_{1j}, w_{2j}, \ldots, w_{kj}, \ldots, w_{|T|j}\}$.

Additions to the standard model have also been implemented in the field of information retrieval, producing the 'Extended Boolean Model' [133], which permits the use of indexing values other than simple Boolean weighting. This extension effectively converts the document vectors to those generated by the Vector Space Model, with the definition w_{kj} changing to signify the weighting value for a particular term k within a particular document j given some weighting function w.

3.5.2 Vector Space Model (VSM)

First introduced in the 1960's as part of the 'Salton's Magic Automatic Retriever of Text' (SMART) system and later published in its own right [134], text categorisation solutions almost exclusively use the indexing techniques of the Vector Space Model. The basic concept involves the production of document vectors in a similar manner to the standard Boolean Model, with the ability to record more than just term presence or absence information. This is achieved by mapping individual documents to all extracted terms by means of a numerical value, calculated by the application of a weighting function to term occurrence statistics. Numerous weighting functions are possible and employ a mix of global, category and document based statistics, though each has associated benefits and drawbacks.

Combining the generated document vectors together produces a term-by-document matrix, which gives an overview of the assigned weight between every possible term and document pairing. Each entry in the matrix refers to a specific weight w_{kj} of a particular document D_j and a particular term T_k, for the weighting function w, with columns representing document vectors $w_{Tj} = \{w_{1j}, w_{2j}, \ldots, w_{kj}, \ldots, w_{|T|j}\}$ and rows representing term weighting throughout the entire dataset $w_{kD} = \{w_{k1}, w_{k2}, \ldots, w_{kj}, \ldots, w_{k|D|}\}$. This allows analysis on a global level, though is rarely efficient if the matrix is sparsely populated due to a high volume of terms not being present in all documents, which is a tendency with textual data [2][17].

	D_1	D_2	...	D_j	...	$D_{\|D\|}$
T_1	w_{11}	w_{12}	...	w_{1j}	...	$w_{1\|D\|}$
T_2	w_{21}	w_{22}	...	w_{2j}	...	$w_{2\|D\|}$
...	...	...	...	...	...	...
T_k	w_{k1}	w_{k3}	...	w_{kj}	...	$w_{k\|D\|}$
...	...	...	...	...	...	...
$T_{\|T\|}$	$w_{\|T\|1}$	$w_{\|T\|2}$	...	$w_{\|T\|j}$	...	$w_{\|T\|\|D\|}$

Table 3.5 : Vector Space Model term-by-document Matrix Example

Aside from the potentially large memory requirements there are also other limitations, such as the loss of term ordering and similar meta-information that is not recorded by the standard index. Another possible source of weakness is the chosen weighting function, as being the only data available to indicate the term and document associations make it a crucial part of the categorisation solution. If unsuitable functions are employed the overall performance can suffer, for example when documents vary in size the use of a non-normalised weighting can show significant bias towards those with longer content. Often the intrinsic assumption of statistical term independence and semantic sensitivity are also considered issues, though these are present in many indexing methods and appropriate data representation and weighting can overcome them to a degree [135].

Despite the shortcomings, the term-by-document matrix offers advantages over other document vector representations, even though they might be more compact or simplistic. By providing a global overview in a relatively intuitive manner, it is easy to comprehend and allows rapid determination of many potentially informative statistics that are central to many text processing algorithms. This includes the quick discovery of term occurrence and importance within multiple scopes, which is crucial for term threshold, ranking and probability metrics [18] and the examination of negative term characteristics [136].

3.5.3 Inverted Index

Also known as a ‘postings file’ or ‘inverted file’ this is one of the most popular indexing techniques employed in document retrieval [137] and is utilised by a variety of different text processing systems. There are two main variants that both record characteristics along with their associated locations throughout the data, giving the potential to significantly speed up query discovery at the expense of increased storage and initial computation. The standard ‘inverted index’, or ‘record level inverted index’, maps each term to a list of the documents that contain it, while the more complex ‘full inverted

index', often referred to as a 'word level inverted index', also tracks individual term positions within each document.

A full inverted index takes more effort to initially create and requires more memory to store but also offers increased functionality, including the ability to represent phrases and potentially convert an entire dataset into a compressed format without any information loss. This is because each entry is associated with all the documents that contain it in combination with positional references that relate to term occurrence within the content, with multiple different entries added if the term is present more than once in any specific document.

Sample Documents:

D_1 = "this string is a sample string"
D_2 = "this is another sample string"
D_3 = "this is a good sample"

Sample Term	Standard Inverted Index	Full Inverted Index
"a"	{ D_1, D_3 }	{ (D_1, 3), (D_3, 2) }
"another"	{ D_2 }	{ (D_2, 2) }
"good"	{ D_3 }	{ (D_3, 3) }
"is"	{ D_1, D_2, D_3 }	{ (D_1, 2), (D_2, 1), (D_3, 1) }
"sample"	{ D_1, D_2, D_3 }	{ (D_1, 4), (D_2, 3), (D_3, 4) }
"string"	{ D_1, D_2 }	{ (D_1, 1), (D_1, 5), (D_2, 4) }
"this"	{ D_1, D_2, D_3 }	{ (D_1, 0), (D_2, 0), (D_3, 0) }

Table 3.6 : Standard and Full Inverted Index Examples

With simple additions to the basic format of either the standard or full inverted index it is trivial to add one or more weighting metrics, making this representation suitable for text categorisation tasks [138]. However a drawback of the structure is the strict term-to-document directed mappings, which can affect performance of any process requiring document-to-term orientated information. The only way to solve this issue and effectively handle queries in the reverse direction is to generate a secondary index, though this increases speed it is not particularly efficient and may negate any benefits gained from data compression.

Another disadvantage of the standard inverted index is the loss of term ordering, leading to greater storage requirements if this is necessary as the original data must then be retained along with the index. Use of a full inverted index avoids this problem, but also has larger computational overheads for initial creation, dynamic updates and navigation of elements for each entry, especially as the index grows.

In general as the average term size and number of documents in a dataset increase the potential for reducing memory consumption improves, though it still possible for an index to use more space than the original data. Research has also been conducted into more efficient data structures, like sorted arrays and prefix B-Trees [139], along with compression techniques, such as integer compression [140]. This has subsequently introduced several optimisations that allow adjustments to storage requirements, indexing speed and ease of maintenance.

3.5.4 Weighting Metrics

The purpose of a weighting metric is to provide an informative measure representing the relationship between a term and a document that contains it, with an overall aim of detecting associations between documents relative to that term. All of the metrics manipulate statistical values extracted from the dataset in an attempt to define the importance of a term, usually within the scope of individual documents but also sometimes in regard to the entire dataset. Extensive research has been conducted in this area giving rise to a variety of available techniques, with the majority focused around two empirical observations [59][2][1][141][13]:

1. The more frequently a term appears in a document the more likely it is to be relevant to the categories assigned to that document.

2. The more frequently a term appears throughout the entire corpus the less discriminative it is likely to be for specific documents and assigned categories.

Only the most common weightings are considered here and are described using the following definitions:

Let d denote an individual document from the overall set of documents D.

Let t denote an individual term from the overall set of terms T.

Let d_t represent a document d that contains the term t.

Let $F(d_t|D)$ represent the total number of documents containing term t from the overall set of documents D.

[1] Note that the equations given in this reference use n_i to refer to the number of documents within the corpus that a term i occurs in, but confusingly describes it as representing the total number of times word i occurs in the whole collection.

Let $F(t|d)$ represent the total occurrences of term t within document d.

Then for a particular weighting metric w the importance of term t with relation to document d is designated as w_{td}.

3.5.4.1 Boolean Weighting

Being the simplest weighting measure this has the fastest computation, but exploits very little of the available statistical information, merely recording the presence or absence of a term within the document by assigning a weight of 1 or 0 respectively. A beneficial side effect of this minimalist nature is that normalisation is not required as the document length is inherently ignored and therefore makes no difference to the derived values.

$$Boolean_{td} = 1 \text{ if } F(t|d) > 0, 0 \text{ otherwise}$$

3.5.4.2 Term Frequency (TF)

This is another basic method and is centred on the notion that frequent terms are more informative than those that occur less often, directly employing the number of occurrences of a particular term within the document as the weight. Since the values are unmodified there is a favourable bias towards longer documents and certain pre-processing or dimension reduction techniques might become more influential, primarily those that combine multiple terms into a single entity.

$$TF_{td} = F(t|d)$$

3.5.4.3 Normalised Term Frequency

Based on the same principle as Term Frequency this utilises identical statistical data but converts all values to a relative scale across the corpus [15], reducing the bias caused by inconsistent document lengths. As normalisation yields the proportion of a document that each term accounts for relative to the most common term it generates vectors that match semantic content regardless of size disparity, typically making it preferred to the non normalised version.

$$NormalisedTF_{td} = F(t|d) \ / \ max(F(T_k|d))) \text{ where } T_k \in T$$

3.5.4.4 Inverse Document Frequency (IDF)

Inverse Document Frequency is based around the concept that the scarcity of a term across the entire dataset is a factor when determining its importance, with those occurring less often providing enhanced discriminative information for differentiating documents. The measure is generally considered an heuristic with no strictly defined theoretical basis, though it has also been described as a probabilistic function and likened to Zipf's law, which states term occurrence within a corpus is inversely proportional to its frequency ranking [142].

$$IDF_{td} = \log (|D| / F(d_t|D))$$

Due to a lack of individual document details in the calculation, this method always assigns the same importance to a given term and consequently is regularly combined with or embedded in other document dependant weighting metrics.

3.5.4.5 Term Frequency-Inverse Document Frequency (TF-IDF)

Independently Term Frequency and Inverse Document Frequency both have defects that may affect performance with certain dataset characteristics, so TF-IDF combines them in an effort to overcome these issues. The principal weaknesses of Term Frequency are the bias towards document length, if not normalised, and the existence of common words that are present in many unrelated documents, whilst Inverse Document Frequency is unable to provide document specific values. By amalgamating the measures together a compromise is reached, making the most important terms those that repeatedly appear in individual documents whilst rarely being present throughout the majority of dataset.

$$TF\text{-}IDF_{td} = F(t|d) \cdot \log (|D| / F(d_t|D))$$

The mix of local and global scopes has proven superior for many problems and has made TF-IDF one of the most widespread weighting metrics [1], with multiple variations based on the same notion. Probably the most popular algorithm and the one that is generally considered the standard form is simply the product of the two basic components. Caution is needed though as Normalised Term Frequency is sometimes indiscriminately substituted in place of the raw frequency count, often without any notification despite the affect it can have on performance.

3.5.4.6 TFC Weighting

The standard TF-IDF metric does not typically use normalised components and to an extent can still suffer a prejudice with dissimilar document lengths, so the TFC adaption applies a form of cosine normalisation [59] and was first introduced as part of the SMART system [141].

$$TFC_{td} = (F(t|d) \cdot \log(|D| / F(d_t|D))) / \sqrt{(|T| \Sigma k=1 (F(T_k|d) \cdot \log(|D| / F(T_k|d)))^2)}$$

3.5.4.7 LTC Weighting

LTC is another modification of TF-IDF that uses the logarithm of term frequency to apply a smoothing effect, further reducing the impact of large variance in term occurrence statistics. Similar to the TFC weighting this also applies a form of cosine normalisation and was introduced as part of the SMART system [143], though is sometimes expressed using slightly different equations [59][2].

$$LTC_{td} = (\log(F(t|d) + 1.0) \cdot \log(|D| / F(d_t|D))) / \sqrt{(|T| \Sigma k=1 (\log(F(T_k|d) + 1.0) \cdot \log(|D| / F(T_k|d)))^2)}$$

3.5.4.8 Entropy Weighting

More sophisticated than most other measures this is employed in many different types of research and has been shown to give superior results [144], but with a relatively high computational requirement. Again it is comparable to TF-IDF being comprised of two distinct parts multiplied together, the locally scoped logarithm of term frequency and the globally scoped entropy of term appearance throughout the dataset.

$$Entropy_{td} = \log_{10}(F(t|d) + 1) \cdot (1 + ((1 / \log_{10}(|D|)) \cdot |D| \Sigma j=1 ((F(t|D_j) / F(d_t|D)) \cdot \log_{10}(F(t|D_j) / F(d_t|D)))))$$

The second element of the multiplication represents the entropy or average uncertainty of a particular term throughout the dataset [2]. It returns a value between 0 and 1 that represents the distribution documents containing the term, with 0 signifying that it is present in all documents and 1 meaning it occurs in only a single document.

3.6 Dimension Reduction

A principal barrier that must be overcome during text categorisation is the large dimensionality inherent with nearly all problems related to the area, which is ultimately dependent on many factors including both the data and solution composition. Even despite the initial data preparation, some of which can be considered a crude form of data reduction, a viable decrease in size is rarely achieved that would allow feasible resource consumption. Therefore dimension reduction, also known as 'feature reduction' or 'term space reduction', is usually a critical process and is often included in a strategy multiple times, either as a distinct step or integrated with other stages.

Frequently the problem dimensions encountered are considerable, reaching thousands or tens of thousands of terms [17][16]. This consequently causes the majority of purely academic research to be conducted on strictly controlled corpora, consisting of purpose made or heavily tailored industrial data. Leading to solutions that may become greatly inhibited as datasets expand and demonstrate less rigid structuring, severely limiting their practical application for most real world situations. Therefore it is desirable to reduce the feature space and accomplish a suitable degree of scalability, preferably in an automatic and computationally efficient manner.

A great deal of effort has been devoted to the reduction of problem dimensions by many different communities and a large number of techniques have been adopted into the field of text categorisation [57]. Some studies also present comprehensive evidence that shows highly aggressive data reduction can actually improve solution accuracy and not just scalability [72]. Though this is only apparent for certain methods and is primarily due to them generalising the constructed classifiers and helping to avoid over-fitting the training data [16][57][136].

All text categorisation dimension reduction techniques can be broadly defined under two main categories; 'feature selection', which attempts to select the most informative terms, and 'feature extraction', where existing terms are adapted to construct new ones. Both types are generally determined on a per category basis, rather than per document like index weighting methods or globally scoped as is common during pre-processing. To avoid affecting performance in an overly negative way it is important to ensure that only low informative terms are removed or modified, especially as more aggressive methods are applied.

3.6.1 Aggressivity

In order to quantify the reduction achieved for a set of terms there are two calculations noted in text categorisation literature, which are known as the 'aggressivity' [136] and 'reduction factor' [59].

Although published as separate entities both calculations are actually equivalent[1], with each merely being a different form of the other. This can be shown if the equations are multiplied by $|T|$, where $|T|$ is the magnitude of the original set of terms T and the reduced set of terms T' is of size $|T'|$.

$$\text{Aggressivity} = 1 - (|T'| / |T|)$$

$$\text{ReductionFactor} = (|T| - |T'|) / |T|$$

The output is in the range of 0 to 1, with higher values representing a greater decrease in the number of features. Definitions may also include the stipulation that $|T'| \ll |T|$ but this is not necessarily a correct assumption, as some algorithms might not always reduce the term set. Another misconception normally present is the condition that $|T'| > 0$, which is intended to prevent creation of the empty set, though again in a few exceptional circumstances this is also possible.

While these are usually applied to feature selection techniques, as T' is meant to be a subset of T, they could also be employed to give an idea of the influence made by feature extraction. Just minor modification is needed to substitute T' for the newly generated set, though if both the original and composite terms are kept the size could increase and the result would become a negative value. As yet, no substantial correlations between this measure and solution efficiency or effectiveness have been determined [136], so it is currently more informational than practical.

3.6.2 Feature Selection

Also referred to as 'subset selection', this group of methods focus either on the identification and selection of informative and relevant terms or the removal of non-informative and irrelevant terms, which were not eliminated during previous stages. In general they are univariate as they consider only individual features, are moderately simple to integrate within solutions and typically have a relatively low computational overhead. However, while they all tend to choose terms based on their occurrence throughout the dataset, the effect on performance varies notably between them as does the core algorithm composition.

There are three types of feature selection [145], with filtering being the main one used within text categorisation due to its fast computation time and the intrinsic high dimensionality of the problem [146]. This technique applies a scoring metric to each term and prunes any that do not meet a specified limit, normally determined experimentally via cross validation. Wrapper and embedded based methods are less common, as they can over-fit the training data and require more resources to recursively construct and test predictive models for discovery of appropriate feature sets.

[1] Note that the cited references share a common author, which may cause uncertainty about the calculation equivalence and differing names.

Pruning is achieved by ranking terms in order of relevance, as established by the chosen scoring measure, and then a cut off point is identified to distinguish those that should be kept or removed. The values are assigned based on probable or actual occurrence statistics derived from the training documents and the threshold parameters are simply upper or lower bounds that signify an acceptable maximum or minimum. Usually a fixed percentage or number of terms is retained and all test documents are filtered prior to classification according to the final set produced.

Nearly all scoring calculations employ term occurrence statistics derived from documents that only belong to a particular category, leading to localised associations that give relevance in a category dependant manner. However it is often desirable to use a globalised average of these measures, able to represent the relationship of terms in connection with the entire dataset, otherwise every test document would need to be independently filtered and classified for each category.

The most popular globalisation methods in text categorisation are the 'maximum average', which takes the maximum category score for each term, and the 'weighted average'[1], which computes the mean score biased by the number of documents belonging to each category [136][18]. There are also several other possible calculations, such as the average or total sum [59], but they are seldom utilised and there is little evidence to be found regarding their suitability.

$$\text{MaximumAverage}_t = \max(\text{CategoryScore}_{ct}) \quad \textit{where } \{c : c \in C\}$$

$$\text{WeightedAverage}_t = |C| \sum i=1 \; (\textit{CategoryScore}_{it} \cdot P(d_i)) \quad \text{where } \{i : i = C_i \text{ and } C_i \in C\}$$

$$\text{Average}_t = (|C| \sum i=1 \; (\textit{CategoryScore}_{it})) / |C| \quad \text{where } \{i : i = C_i \text{ and } C_i \in C\}$$

$$\text{TotalSum}_t = |C| \sum i=1 \; (\textit{CategoryScore}_{it}) \quad \text{where } \{i : i = C_i \text{ and } C_i \in C\}$$

Only techniques encountered regularly are considered in this section, though even these can be a potential cause of confusion as they tend to exhibit a large variation of definitions throughout literature. In addition while a few publications seem to use bespoke formats [18] many simply repeat the notation and equations found in the original source, duplicating any errors or complexities that might be present [16] [57] [148][2][149]. Consequently all of the equations given have been verified and universally restructured to use the following descriptions:

Let d denote an individual document from the overall set of documents D.

Let t denote an individual term from the overall set of terms T.

[1] Note this is also sometimes referred to as the 'weighted sum'.

Let c denote an individual category from the overall set of categories C.

Then for a particular feature selection measure f the relevance of term t in relation to category c is designated as f_{tc}.

Furthermore;

Let a_b represent all a associated with b.

Let $a_{\bar{b}}$ represent all a not associated with b.

Let $\{a:b\}$ represent the set of a that conforms to condition b.

Let $F(a|b)$ represent the total number of occurrences of a within b., such that $F(a|b) = |\{a:b\}|$.

Let $P(a)$ represent the probability of a occurring.

Let $P(a|b)$ represent the conditional probability of a given that b has already occurred.

Let $P(a\&b)$ represent the conjunctional probability that b occurs followed by a.

Generally the most prevalent definition is supplied, except where multiple equivalent forms are common in which case these are included as well. It is also worth noting that the same basic statistical data is reused throughout the majority of techniques, so combining several different calculations has relatively trivial overhead once the initial probabilities are deduced.

3.6.2.1 Outlier Count

Not strictly a means for reducing the dimensional space of a problem, the Outlier Count was introduced to predict the effectiveness of performing feature selection on a dataset by examination of the information distribution [16]. It is based on ascertaining the rate of decline in term Information Gain values by the analysis of outliers in relation to the standard deviation and average Information Gain, as calculated using statistics derived from the entire dataset.

It merely involves calculating the Information Gain measure for every term, then recording the outlier count as the number that have a value greater than three standard deviations σ_{InfoGain} above the mean μ_{InfoGain}. Generally datasets that exhibit higher counts are the most likely to benefit from extensive feature selection, as it is more probable that they contain a small subset of terms with the main characteristics required to discriminate between categories.

$OutlierCount(D, T) = |\{t: InformationGain(t) > \mu_{InfoGain} + 3 \cdot \sigma_{InfoGain}\}|$

Despite the incorrect assumption that the underlying distribution of Information Gain values is always normal, the limited empirical results gathered do suggest that this provides reasonable estimations. Consequently it is potentially a reliable method for implying how aggressive the level of term filtering should be for a particular dataset in order to achieve the maximum benefit. However, one negative aspect is the additional resources needed to determine the initial values, especially when Information Gain is not the principle measure being employed, there are also no results available for its use with other feature selection metrics.

3.6.2.2 Document Frequency

Almost certainly the simplest feature selection technique, this involves calculating the number or percentage of documents throughout a dataset in which each term occurs and removing those that do not meet a predefined threshold. The basic principle is that terms rarely encountered add little or no significant worth to the categorisation process or are a cause of noise, so can feasibly be ignored in order to reduce the feature space. Potentially a threshold could also be identified for pruning exceedingly common terms, though this is unusual and generally dealt with during pre-processing via stopword elimination, which has a similar but opposed theory [148].

Being easy to implement minimal resources are required, yet studies have consistently shown that superior results can be achieved which are often comparable to methods involving a much higher level of complexity. Due to this it is a popular technique and capable of being scaled to handle considerably large problems, with a computational time complexity of $O(n)$ which is approximately linear to the number of documents in the corpora under investigation.

$\mathrm{DocumentFrequency}_t = P(d_t)$ (percentage)

$\mathrm{DocumentFrequency}_t = |\{d_t : d_t \in D\}| = F(d_t|D)$ (volume)

Typically the calculations are based on all documents regardless of category associations, though it is also possible and sometimes convenient to establish the values in a category dependent manner [136]. This distinction is important as the type of equation can give a notably different outcome because of the globalisation mechanisms employed, which in turn affects the ability to duplicate any categorisation systems using this reduction measure.

$\mathrm{DocumentFrequency}_{ct} = P(d_{ct}|d_c)$ (percentage)

$\text{DocumentFrequency}_{ct} = |\{d_{ct}: d_{ct} \in D_c\}| = F(d_{ct}|D_c)$ (volume)

Conceptually the removal of rare terms on the premise that they are less informative than more common ones and therefore do not notably influence overall performance, is a contradiction to the conventional Information Retrieval belief. Traditionally it was thought that rare terms were always informative and only to be pruned to increase computation efficiency, not for the purpose of enhancing predictive accuracy. However, experimental results suggest that the contradiction is well founded, as substantial feature space reduction with this technique has been shown to improve performance in several solution strategies [18].

3.6.2.3 Odds Ratio and Relative Risk

Commonly used in medical areas, Odds Ratio measures the magnitude of increase in likelihood that an event will occur for one group over another [150]. Using the standard implementation an outcome of 1 implies that the event is likely to happen equally in both groups, with no discrimination between them. Similarly, any other value suggests the event is more likely to occur in one or the other, with values larger than 1 giving priority to the first group and lower meaning the second has precedence.

Although this general concept is quite simple the range of possible values is a common cause of confusion, as an outcome of infinity means the event will occur solely in the first group and 0 denotes it is exclusive to the second. Due to this and it expressing the difference between two sets of odds, it is difficult to gauge how the deduced values relate to the actual event chance, but if used correctly it can provide reliable predictions of event likelihood between the subjects being compared [16] [76][1].

In some research the more intuitive Relative Risk or 'Relative Ratio' is used as a direct alternative to Odds Ratio as, though it has many similarities and portrays the same information, its computation is based on normalised percentile values in place of the raw data. This allows an outcome representing the chance of an event happening in direct relation between the groups being compared, rather than in relation to the respective odds of the event taking place for each. As an example, if an event has a 75% chance (3:1 odds) of happening to group α and a 50% chance (1:1 odds) of happening to group β, then Relative Risk determines that the event is 1.5 times more likely to occur to group α than group β. However, using Odds Ratio the odds associated with group α are determined to be 3 times the magnitude than those associated with group β.

While the mechanics of Relative Risk are easier to understand it does have disadvantages [150], the largest being difficulty in handling covariant adjustment, where a scenario has additional factors that might affect the two subjects being compared. It is also unable to handle missing data and can often be confusing when exploring a small relative change in one probability and the corresponding change

[1] Note that several variants of the standard Odds Ratio algorithm are employed in this publication.

in the opposing probability. To emphasise this consider that group α has a probability of 1% (1 chance in every 100), for the occurrence of an event and that group β has 99% (99 chances in every 100). In order to increase the probability of group α to 2% (1 chance in every 50), group β is reduced to 98% (49 chances in every 50), so the same chance as before for group α has meant the relative chance of group β has nearly halved to 49 instead of 99.

$$\mathrm{OddsRatio}_{ct} = (P(d_{ct}) \cdot P(d_{\bar{c}\bar{t}})) / (P(d_{c\bar{t}}) \cdot P(d_{\bar{c}t}))$$

$$\mathrm{RelativeRisk}_{ct} = P(d_{ct}) / P(d_{\bar{c}t})$$

For use of Odds Ratio and Relative Risk in the field of text categorisation the event considered is the probability of occurrence for a specific term, with one subject taken as a particular category from those present in the training data and the comparative subject consisting of all remaining categories. This effectively makes the calculation a reflection of how likely the term is to appear in the chosen category in relation to its potential occurrences in all the others, which indicates its discriminative ability for that category.

3.6.2.4 Mutual Information

Occasionally referred to as 'Transinformation' this is frequently employed for statistical language modelling of word associations, as the output measures the characteristics shared between two variables, which in the case of text categorisation are the terms and categories. These common traits subsequently allow an estimation of how much can be determined about one variable when given the other, making it possible to formulate predictions regarding the association between a particular term and a specific category.

If the calculated Mutual Information between a pair of variables is 0 they are considered completely independent with no familiar attributes, which generally infers that they will be mutually exclusive throughout the entire dataset. Conversely, the higher the result is above 0 the more probable it is that both variables provide exact details of each other, signifying they are always likely be encountered together. Similar to some other measures the required time complexity is $O(n)$ and is approximately linear to the number of documents and terms under scrutiny [18].

$$\mathrm{MutualInformation}_{ct} = \log(P(d_{ct}\&d_c) / (P(d_t) \cdot P(d_c)))$$

The main weakness of Mutual Information is its tendency to assign disparate values to terms that substantially vary in frequency throughout the dataset, even if they have similar category associations [18]. Altering the standard equation to an equivalent form emphasises this behaviour, making it apparent that the denominator will be greater for common terms, while the numerator is not affected

by their presence among documents unrelated to the category. This effectively assigns a value proportional to the amount of information a term provides adjusted by its total overall occurrence, which exhibits a natural bias in favour of rarer terms [148]. It also demonstrates heightened sensitivity to inaccurate probability estimates that is compounded further by scarce terms directly due to this bias.

$$\text{MutualInformation}_{ct} = \log((P(d_{ct}|d_c) \cdot P(d_c)) / (P(d_t) \cdot P(d_c)))$$

$$\text{MutualInformation}_{ct} = \log(P(d_{ct}|d_c) / P(d_t))$$

Often deemed a disadvantage this limitation is widely acknowledged as the reason for lower quality results compared to other measures, though it can also prove a positive aspect if appropriate steps are taken to prepare the data so that only significantly informative rare terms remain.

3.6.2.5 Information Gain

Information Gain is an alternative name for 'Kullback-Leibler Divergence', but is also known as 'Kullback-Leibler Information Criterion' (KLIC), 'Information Divergence', 'Relative Entropy' and 'Expected Mutual Information' [149]. A diversity that is probably due to its regular use as a variable goodness criterion in a range of disparate areas, including Machine Learning, Quantum Physics and Applied Experimental Psychology. It utilises the same basic principle as Mutual Information, analysing common information to make predictions about one variable based on knowledge of another, though experiments generally conclude that it yields superior results [18][16][146].

$$\text{InformationGain}_{ct} = P(d_{ct}\&d_c) \cdot \log(P(d_{ct}\&d_c) / (P(d_t) \cdot P(d_c))) + P(d_{ct}\&d_c) \cdot \log(P(d_{ct}\&d_c) / (P(d_t) \cdot P(d_c)))$$

$$\text{InformationGain}_t = -(|C| \sum i=1 (P(C_i) \cdot \log(P(C_i)))) + P(t) \cdot (|C| \sum i=1 (P(C_i|t) \cdot \log(P(C_i|t)))) + P(\bar{t}) \cdot (|C| \sum i=1 (P(C_i|\bar{t}) \cdot \log(P(C_i|\bar{t}))))$$

There are two different equations typically stated for determining the value, though they relate separately to single-class [1][59][16] and multi-class problems [2][18], with one being a globalisation technique specific to Information Gain. From inspecting the definitions it is apparent that there is a distinct resemblance to Mutual Information, which actually forms part of the calculation, though the measures are clearly not equivalent as stated in some publications [59]. However, they do involve many of the same initial category and term probability estimations and share similar computational time complexity, which is approximately linear to the number of documents and unique terms [18].

Despite effectively being an extension of standard Mutual Information, there are a couple of fundamental differences that significantly contribute towards the improved assessment of term

relevance and discriminative capacity. These are the addition of negated information, accounting for the absence of a term within a document, and a type of normalisation, which uses the term and category conjunctional probability to weight the computed values. In combination these factors help to offset the negative impact experienced by both the bias associated with rare terms and also the potentially high sensitivity to inaccurate probability estimations.

3.6.2.6 Chi-Squared Statistic

Widely accepted as one of the most effective text categorisation dimensional reduction measures [146][18], this is often referred to as 'Chi-Squared', 'Chi-Squared Test' or just symbolised by χ^2. It determines the expected probability of multiple element combinations by analysing their trend of occurrences both individually and relative to each other throughout the dataset. In this case the elements considered are a particular term and a specific category, while the calculated values correspond to the lack of independence between them, with an output of 0 showing they are mutually exclusive.

$$\text{ChiSquared}_{ct} = (P(d_{ct}) - (P(d_c) \cdot P(d_t)))^2 / (P(d_c) \cdot P(d_t)) + (P(d_{c\bar{t}}) - (P(d_c) \cdot P(d_{\bar{t}})))^2 / (P(d_c) \cdot P(d_{\bar{t}})) + (P(d_{\bar{c}t}) - (P(d_{\bar{c}}) \cdot P(d_t)))^2 / (P(d_{\bar{c}}) \cdot P(d_t)) + (P(d_{\bar{c}\bar{t}}) - (P(d_{\bar{c}}) \cdot P(d_{\bar{t}})))^2 / (P(d_{\bar{c}}) \cdot P(d_{\bar{t}}))$$

When the training set is able to provide a sufficient representation of the test documents, the generated chi-square values can be directly measured relative to the degrees of freedom in the problem in order to determine their p-value and significance. The degrees of freedom refers to the number of variables that may vary within a particular combination of elements, so for the pairing between a single category and each individual term it is one, as the term changes but the category is kept the same.

One Degree of Freedom											
Chi-Squared Value	0.001	0.02	0.06	0.15	0.46	1.07	1.64	2.71	3.84	6.64	10.83
P-Value	0.95	0.90	0.80	0.70	0.50	0.30	0.20	0.10	0.05	0.01	0.001
Significance	Insignificant								Significant		

Table 3.7 : Mapping of Chi-Square, P-Value and Significance for One Degree of Freedom

A p-value establishes how notable the connection is between the elements being tested and is determined by comparing obtained results against the chi-squared statistical distribution, indicating the probability that the expected results are at least as extreme as those observed. There are several prominent p-values but 0.001, 0.01 or 0.05 are common for denoting the level of significance, therefore term and category combinations that yield chi-square measures equal to or higher than their corresponding values are deemed to be significantly lacking independence [151].

There are several different equations based on the chi-squared distribution with the most common being 'Pearson's Chi-Squared Test'[1], though this is regularly used in a modified form [18][2][2][16][3] modChiSquared that is only applicable for two-by-two contingency tables containing one degree of freedom [152]. Another popular version is 'Yate's Chi-Squared Test' or 'Yate's Correction for Continuity', which attempts to prevent overestimation of statistical significance for small data observations by decreasing the final output and effectively raising the p-value [153]. However this can over correct and means element dependency is sometimes ignored when perhaps it shouldn't be, leading to a belief that the alteration is not entirely necessary.

$$modChiSquared_{ct} = (|D| \cdot ((P(d_{ct}) \cdot P(d_{\bar{e}\bar{t}})) - (P(d_{c\bar{t}}) \cdot P(d_{\bar{e}t})))^2) / ((P(d_{ct}) + P(d_{c\bar{t}})) \cdot (P(d_{\bar{e}t}) + P(d_{\bar{e}\bar{t}})) \cdot (P(d_{ct}) + P(d_{\bar{e}t})) \cdot (P(d_{c\bar{t}}) + P(d_{\bar{e}\bar{t}})))$$

$$modChiSquared_{ct} = (|D| \cdot ((P(d_{ct}) \cdot P(d_{\bar{e}\bar{t}})) - (P(d_{c\bar{t}}) \cdot P(d_{\bar{e}t})))^2) / (P(d_c) \cdot P(d_{\bar{e}}) \cdot P(d_t) \cdot P(d_{\bar{t}}))$$

$$YatesChiSquared_{ct} = ((P(d_{ct}) - (P(d_c) \cdot P(d_t))) - 0.5)^2 / (P(d_c) \cdot P(d_t)) + ((P(d_{c\bar{t}}) - (P(d_c) \cdot P(d_{\bar{t}}))) - 0.5)^2 / (P(d_c) \cdot P(d_{\bar{t}})) + ((P(d_{\bar{e}t}) - (P(d_{\bar{e}}) \cdot P(d_t))) - 0.5)^2 / (P(d_{\bar{e}}) \cdot P(d_t)) + ((P(d_{\bar{e}\bar{t}}) - (P(d_{\bar{e}}) \cdot P(d_{\bar{t}}))) - 0.5)^2 / (P(d_{\bar{e}}) \cdot P(d_{\bar{t}}))$$

$$YatesChiSquared_{ct} = (|D| \cdot ([(P(d_{ct}) \cdot P(d_{\bar{e}\bar{t}})) - (P(d_{c\bar{t}}) \cdot P(d_{\bar{e}t}))] - (|D|/2))^2) / (P(d_c) \cdot P(d_{\bar{e}}) \cdot P(d_t) \cdot P(d_{\bar{t}}))$$ where $[(P(d_{ct}) \cdot P(d_{\bar{e}\bar{t}})) - (P(d_{c\bar{t}}) \cdot P(d_{\bar{e}t}))]$ represents an absolute value

Unlike some other dimension reduction techniques the chi-squared metric automatically provides a normalised value that can be directly compared across term and category combinations, making the application of universal threshold strategies easier. However these normalised values cannot be accurately compared to the chi-squared distribution for determination of realistic p-values when the training data is sparsely populated, leading to unreliable probability estimations for rarer terms. This weakness may be partially resolved by employing a possible variation of the measure, but this might cause the opposite issue where terms are considered too significant.

3.6.2.7 Correlation Coefficient

Correlation Coefficient is very similar to the chi-squared statistic, but does not generate an absolute independence value for each category and term combination, instead producing an outcome in the range of -1 to +1. It effectively calculates the square root of the modified Pearson's equation [1] and provides a way to determine both positive and negative association between elements. This makes it

[1] While the most commonly cited and given here as the prominent equation, it is unusual for publications to explicitly state which particular format is being used, potentially causing confusion.
[2] Note this reference equation is correct, but its description of notation is erroneous as D actually relates to the number of documents not belonging to class c_j and not containing word w.
[3] Note the equation in this reference does not square the numerator, which is likely a typographical error as it is squared in the corresponding reference quoted by the authors.

possible to focus exclusively on those that indicate an affirmative correlation, where the presence of one suggests the presence of the other, which is not achievable with standard chi-squared.

$$\text{CorrelationCoefficient}_{ct} = ((\sqrt{|D|}) \cdot ((P(d_{ct}) \cdot P(d_{\bar{c}\bar{t}})) - (P(d_{c\bar{t}}) \cdot P(d_{\bar{c}t})))) / \sqrt{(P(d_c) \cdot P(d_{\bar{c}}) \cdot P(d_t) \cdot P(d_{\bar{t}}))}$$

Empirical testing suggests that Correlation Coefficient has superior performance to chi-square alone, however the findings only reflect the use of localised per category threshold strategies [154]. It has also proved to be of interest for solutions that employ negated terms, which are those that imply a lack of association to a category when present in a document, though primarily as the foundation for another reduction technique [136]. Additionally due to the altered equation this cannot be directly mapped to p-values in the same manner as normal chi-squared statistic values, which holds true for all methods based on the chi-squared distribution.

3.6.2.8 Simple Chi-Squared

Another variant of the chi-squared statistic this version removes certain parts of the algorithm that are seen as unintuitive, with experiments showing it to have inferior performance unless aggressive dimension reduction is applied [136]. However, it does reduce computational requirements in comparison to the standard equation, making it appropriate for large datasets and solutions that do not scale particularly well. Although owing to the nature of its simplification it is not viable for mapping to p-values or any similar significance and chi-squared distribution based functions.

Originally derived from Correlation Coefficient, it retains the ability to distinguish between positive and negative element associations, but further condenses the equation according to three observations. The constant value of the numerator $\sqrt{|D|}$ has been eliminated, as it is identical for all term and category combinations and therefore redundant. Whereas the denominator has been removed entirely, reasoning that part of it $\sqrt{(P(d_t) \cdot P(d_{\bar{t}}))}$ emphasises rare terms, which have previously been shown to contain trivial discriminative ability, and the remainder $\sqrt{(P(d_c) \cdot P(d_{\bar{c}}))}$ favours rare categories, which bias certain evaluation measures. The resultant formula can be quickly estimated for an individual term and category by calculating the determinant of their contingency matrix and dividing the outcome by the total number of documents.

$$\text{SimpleChiSquared}_{ct} = (P(d_{ct}) \cdot P(d_{\bar{c}\bar{t}})) - (P(d_{c\bar{t}}) \cdot P(d_{\bar{c}t}))$$

Studies demonstrate that using the maximum average global weighting mechanism tends to give the best performance and have discovered that simple chi-squared begins to surpass the normal version at levels of around 98% feature reduction [136]. They also found that it exceeds Document Frequency at about 95% reduction by a sizeable margin, but it is also more complex and has slightly increased computation. However, the practicality of these findings is debatable, as the effectiveness of many

systems is likely to decline sharply by this point due to impairment from the poor distinguishing power of so few features.

3.6.3 Feature Extraction

Feature extraction or ‘re-parameterisation’ is a multivariate process that works on several features at once, attempting to merge them in order to synthesise a new term that incorporates all the information of the original components in a compact form. The generated terms are deliberately produced orthogonal to each other to minimise dimensional space, whilst keeping them distinct so they retain any discriminative ability. Usually this is accomplished by considering elements that display polysemy, homonymy or synonymy characteristics, but can also be generalised by employing clustering techniques to determine potential category specific combinations.

Ideally algorithms should fabricate a single term for each discrete group of features that uniquely identifies a category, though this is difficult due to the overlapping nature of most textual elements. However, there are multiple advantages to this approach and the main benefit over feature selection is that all associated terms are able to contribute towards the assignment process, rather than only those believed to be important. This means even documents that contain loosely related features may still be classified correctly, based on the collective relevance of their entire content, whereas selection techniques would possibly ignore them completely [1].

Experimental studies have shown that some feature extraction methods give improved performance [1][86][80] but there are drawbacks, with the prominent one being computational complexity, which is significantly greater than the majority of feature selection approaches. Another disadvantage is the elimination of highly descriptive individual features that get absorbed into the derived terms, causing the loss of essential information despite a possible rise in perceived effectiveness [1]. While this can actually be positive, as the generalisation could help to avoid over-fitting the training data, it is likely to be part of the reason why feature extraction is less common than feature selection.

In addition when a feature combination and the low-level components it consists of are all considered important it is viable to retain them both, but this is rare and often counterintuitive as it increases problem dimensions. Though it might prove a useful strategy for solving many difficult issues, such as those posed by word sense disambiguation, where a single term may contain equally significant information by itself and also as part of a constructed group.

3.7 Classification Algorithms

Many various types of classification algorithm have been employed for the task of text categorisation, with the majority being based on modified concepts ported from the areas of Information Retrieval and Machine Learning. They also include a mixture of both parametric and non-parametric algorithms, with the former built from estimations or statistics derived from the underlying data structures and feature densities, while the latter relies on either individual category profiling or associations to similar training examples [4] [1].

It is widely recognised that the choice of classifier plays a vital role in the overall performance of a categorisation solution and consequently this particular stage seems to receive the most attention. Though due to regular over simplification of the text categorisation process, some steps that occur prior to classification are frequently neglected or end up being merged into the algorithm definition. However it is beneficial for all the involved parts to be separately described and investigated, as clear explanations about what relates to data preparation or actual classification would remove much of the confusion caused by current approaches. It would make different combinations of stages more accessible, expanding the potential for exploring varying solution compositions.

Nearly all of these existing methods also suffer from an inability to effectively cope with multi-class or multi-label problems, typically having to split them up and attempt solutions on a per-category basis that are then pieced together to obtain the final result [28]. This often increases construction and execution times significantly, usually directly in proportion to the number of categories, as distinct classifiers must be trained for each and then individually tested on every unseen document before the outputs are amalgamated. Separation of the stages is advantageous during this procedure as any processes associated to data preparation only need to be completed once and not for all iterations of training and testing.

Throughout literature it is widely acknowledged that both K-Nearest Neighbour (KNN) and Support Vector Machines (SVM) are consistently the best performing algorithms for text classification [42][16][17], while Naive Bayes (NB) seems a popular choice for use in comparisons [42][17]. They are also well established techniques originating from other research areas and are clearly defined with minimal overlap from other categorisation stages [15], meaning either of them would be a viable comparison benchmarks as opposed to the array of alternative classifiers available.

3.7.1 K-Nearest Neighbour

K-Nearest Neighbour (KNN) is a non-parametric and non-linear statistical based approach that has been used in pattern recognition for over four decades and was one of the first techniques applied to the task automated text categorisation [42]. It is comparatively fast compared to other methods as it

employs a lazy learning approach that does not require the training of a predictive model and is also able to achieve full potential due to parameters that can be automatically optimised, though at the cost of increased computation.

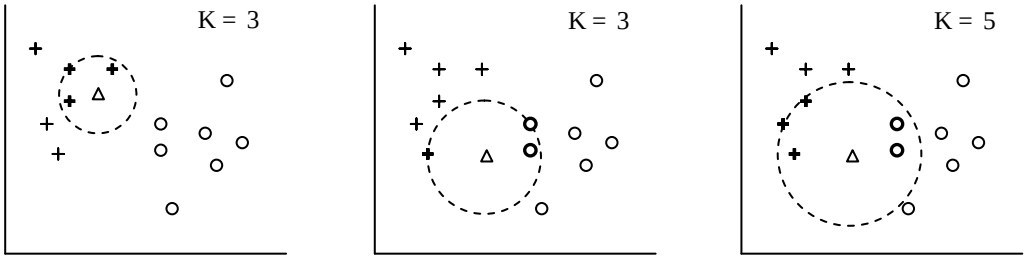


Figure 3.4 : Nearest Neighbour Determination

During classification every test document is checked against the 'k' most similar training documents, as determined by a chosen similarity measure, and a prioritised list of candidate categories is created by ranking those assigned to each of these nearest neighbours according to predefined criteria. Finally either a global or localised per category threshold strategy is applied to ascertain what elements in the candidate list should be assigned to each particular item, using some form of tie-breaker if multiple identical similarities are present.

A wide array of similarity measures, ranking criteria, threshold strategies and tie-break mechanisms are available, making KNN an extremely versatile algorithm, but also introducing a number of distinct components that often require empirical trials to adequately determine. This process is not always straightforward either, as all of them contain a range of possible options that vary in complexity, such as prioritising candidates by the quantity of assignments among the neighbours or by a function based on manipulating the similarity scores [42].

Several parameters also need to be defined and optimised, including any related to the various algorithm parts and importantly the choice of k, which represents the number of neighbours to select and typically has a significant effect on performance. Most are generally optimised automatically by repetitive experimentation on a validation set, aiming to satisfy the prerequisites of the specific categorisation problem or maximise particular evaluation metrics [35]. Additional variables can also be incorporated to further increase flexibility, for example a limit that restricts the scope of the neighbourhood being search or that allows a fuzzy k value in certain situations [15], though these involve more intricacies.

The main weakness of the approach is the numerous component and parameter options that must be combined and optimised to suit the task being undertaken. While providing considerable adaptability they also affect the classification accuracy and computation required to discover the nearest training

examples, so need to be determined appropriately [1][42]. Although the majority of them can be automatically optimised to give reasonable performance, provided enough training data is available and that it sufficiently portrays the test samples.

Another issue is the potential for highly frequent categories to skew the distribution of assignments throughout the dataset, as the selected nearest neighbours will tend to be dominated by the more common labels. To an extent this can be overcome by weighting candidate rank according to category density or using the average similarity score to prioritise the list, meaning rarer and closer elements become more influential in exchange for increased computation during classifier execution. Noise and irrelevant features are also factors that affect the quality of results, making the application of suitable data preparation essential to minimise such characteristics.

3.7.2 Support Vector Machines

Support Vector Machines are non-parametric and non-probabilistic, with the present format first introduced in 1995 and originally referred to as 'Support-Vector Networks' [155]. They employ an integrated kernel to create vectors from the features of unclassified documents, which it then compares to prototype vectors previously derived from training data. A single prototype vector is generated for each individual category and is formed by finding the maximal margin hyperplane that separates all features belonging to the training documents assigned to the category from all of those that are not.

This process is easiest to visualise when the positive and negative instances of a category are linearly separable and contain only two dimensions, which relate to the particular features used to distinguish the instances. Support vectors are first discovered from the training data to define boundaries of the two distinct groups that represent positive and negative correlation with the category, these are in turn utilised to create a prototype vector centred equidistant between them. Test instances are then evaluated relative to this central decision surface and their category association is decided according to which side they are located.

The aim is to determine those support vectors that maximise the distance or 'margin' between the boundaries and their resultant prototype vector, hypothetically producing the greatest separation of the opposing training samples and increasing classification accuracy. Extension of the basic concept to include any number of features is also possible, in which case the decision surface takes the form of a multidimensional hyperplane. It is also not restricted to just linearly separable data and requires no manual input, with all parameters capable of being automatically tuned [17], though the actual quantity and type of variables depend on the specific components employed.

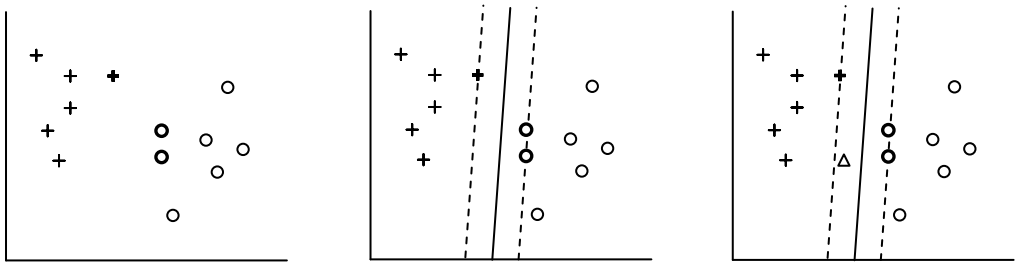


Figure 3.5 : Two Dimensional Support Vector Determination for Non-Maximal Linear Hyperplane

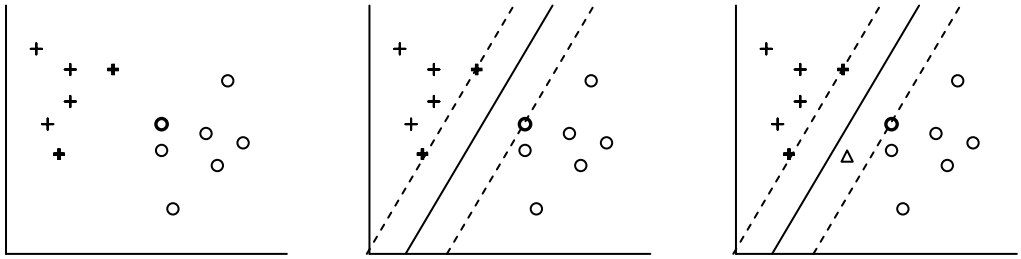


Figure 3.6 : Two Dimensional Support Vector Determination for Maximal Linear Hyperplane

A variety of kernels can be integrated into the algorithm in order to transform the features and map them into a new problem space during the creation of prototype and unlabelled test vectors. They allow the effective handling of nonlinear sample distributions and can be relatively simple or decidedly complex functions, including those that take polynomial, Gaussian or hyperbolic forms. Selection of an appropriate kernel is usually a significant factor in the performance of an SVM and depends heavily on the characteristics of the data being processed, though basic linear equations have been shown to work well for most text categorisation tasks [42][17].

The primary drawback of this approach is its inability to efficiently deal with multi-class problems, having to build and evaluate a different binary classifier for every category, which typically increases the resources required for training and testing. Each predictive model also naturally produces a hard categorisation decision, so a separate standardised mechanism must be developed if any form of global candidate ranking is desired. Another possible issue is that the process of attaining assignments is completely automated and sometimes the logic behind a decision or tuned parameter is not easily understood, making manual adjustment difficult.

However despite the disadvantages it has potential as a scalable solution, as the generated predictive models are able to individually classify test data and then merge the unconnected results together without negative impact. Incremental training is also feasible because only the support vectors determine the decision surface, meaning a model does not need to be rebuilt when training instances are added or removed unless they are an existing support vector or closer to the hyperplane than the

current boundaries. It is further viable to extract support vectors independently from several distinct subsets of the training data and then combine them together to derive the final classifier, reducing limitations due to memory and computational demands as the dataset grows [2].

3.7.3 Similarity Metrics

A considerable variety of similarity metrics are possible [156], which all attempt to determine the likeness between two samples, in this case document vectors containing an array of terms that each have an associated weight. Note that it is important not to confuse similarity with distance, which is a measure of the difference or dissimilarity between two samples.

The following definitions apply when given the vectors of two documents that need to be compared:

Let D signify the set of all documents that are being compared, such that $D = \{D_1, \ldots, D_j, \ldots, D_{|D|}\}$, where D_j represents a particular document.

Let T signify the finite set of unique terms extracted from the combined documents vectors, such that $T = \{T_1, \ldots, T_k, \ldots, T_{|T|}\}$, where $|T|$ denotes the total number of terms and T_k represents a particular term.

Let w_{kj} signify the weighting of a particular term k within a particular document j and w_{Tj} signify the weightings related to each term from the entire set of terms T for the particular document j.

Let the vector containing weights for all of the terms for a particular document j be designated using w_{Tj}, which takes the form $w_{Tj} = \{w_{1j}, w_{2j}, \ldots, w_{kj}, \ldots, w_{|T|j}\}$.

Let w_{T1} represent the vector of the first document being compared and w_{T2} represent that of the second document being compared.

Then for a particular similarity metric s the measure of likeness between the vector w_{T1} of the first document D_1 and the vector w_{T2} of the second document D_2 is designated as $s(D_1, D_2)$.

3.7.3.1 Euclidean

Euclidean distance is a simple metric that measures the direct diversity between two samples, in the case of vectors it represents the straight line distance between the vectors and is sensitive to the magnitude of the weightings. It is one of the simplest and most commonly used but requires

modification before it can be employed in the form of a similarity metric, which may be done in a number of different ways.

$$\text{EuclideanDistance}(D_1, D_2) = \sqrt{(|T| \sum k=1 ((w_{k1} - w_{k2})^2))}$$

$$\text{AdjustedEuclideanSimilarity}(D_1, D_2) = 1 / (1 + \sqrt{(|T| \sum k=1 ((w_{k1} - w_{k2})^2)))}$$

The adjusted Euclidean Similarity measure takes the reciprocal of the standard Euclidean distance with the value of one added to account for the possibility of empty or zero weighted entries in a vector.

3.7.3.2 Cosine

By calculating the angle between vectors cosine similarity, also known as the 'angular metric', determines relative likeness based on direction rather than magnitude, making it insensitive to variation in length.

$$\text{CosineSimilarity}(D_1, D_2) = (|T| \sum k=1 (w_{k1} \cdot w_{k2})) / (\sqrt{(|T| \sum k=1 (w_{k1}^2))} \cdot \sqrt{(|T| \sum k=1 (w_{k2}^2))})$$

3.7.3.3 Jaccard/Tanimoto

The standard Jaccard coefficient is suited only for binary vectors and despite widespread use there is much confusion when compared to the Tanimoto coefficient, though one consensus is that the latter is a more generalised form which is equivalent with binary vector weights [156][157]. Often Tanimoto is also referred to as 'Extended Jaccard' or the two names are concatenated to 'Jaccard-Tanimoto', however the distance measures that correspond to each of them are not the same, which may be an additional factor in the confusion.

$$\text{TanimotoSimilarity}(D_1, D_2) = (|T| \sum k=1 (w_{k1} \cdot w_{k2})) / ((|T| \sum k=1 (w_{k1}^2)) + (|T| \sum k=1 (w_{k2}^2)) - (|T| \sum k=1 (w_{k1} \cdot w_{k2})))$$

3.7.3.4 Sørensen-Dice

Also known as 'Sørensen index', 'Dice's Coefficient', 'Czekannowski index', and multitude of other variations, it is quite similar to the Jaccard similarity and produces a measure based on both the magnitude and items present within each vector.

$$\text{DiceSimilarity}(D_1, D_2) = (2 \cdot (|T| \Sigma k=1 (w_{k1} \cdot w_{k2}))) / ((|T| \Sigma k=1 (w_{k1}^2)) + (|T| \Sigma k=1 (w_{k2}^2)))$$

3.7.4 Threshold Techniques

Classifiers that generate a list of candidate categories for each document in the test set, ranking them by means of some associated relevancy score, must employ a threshold strategy to determine which will be assigned. This is vital whenever binary classification is necessary, including many real-world situations, and the choice of technique often has a considerable affect on performance, with different solutions yielding varying results that depend on the characteristics of the particular application [158][4]. Despite this it is frequently overlooked and considered a trivial step in the overall categorisation process, though it becomes progressively harder to effectively apply as problem dimensions grow and more noise is present in the dataset [158].

Only a small number of techniques are commonly utilised, but there are also variants that combine favourable aspects in an effort to minimise the flaws and some evidence shows these hybrids can prove superior to the individual methods they are derived from [158]. In general each type has user defined variables and may be applied either per category or per document, with both having specific benefits and disadvantages that must be carefully balanced according to the task being undertaken. However in most research environments the decision of which to use is less important, as many popular evaluation metrics tend to require automatic tuning of threshold parameters to achieve precise values, therefore making it almost irrelevant.

The parameters themselves are usually quite simple but can be difficult to optimise, normally being determined experimentally through empirical testing on a validation set. Though where classifiers output information that allows theoretical threshold computation, such as those that produce assignment probability estimates, it is also possible to analytically derive optimised values by means of the probability thresholding principal [159]. In these cases an effectiveness function is maximised and will always provide the best expected effectiveness for the given classifier on the particular set of training data being investigated.

3.7.4.1 Scored

Referred to as ‘SCut’ or ‘CSV’ threshold this directly uses the information that shows how relevant a candidate category is to a test document, which is known as the categorisation status value [4]. It involves the use of a validation set to associate a score with each individual category and document pairing and only those that meet a minimum requirement are retained, which allows any number of assignments providing they are deemed appropriate. This may be applied on a per category basis,

where each category has a separate limit that must be achieved in order for it to be assigned, or conversely from a per document perspective, where each document has a score that must be met.

All category or document cut-off values need to be explicitly defined for this method and, while possible to utilise a single global limit, it is challenging to manually derive suitable parameters so tuning is typically automated. One advantage of this is the ability to optimise against customised criteria that exhibit a bias matched to the particular application, which permits a high level of fine tuning at the cost of potentially over fitting the data [158]. Quality of the training and validation data is also crucial, as the overall performance is heavily reliant on it being a true representation of the test data.

3.7.4.2 Proportional

Also known as 'PCut' this assumes that the same ratio of document to category assignments observed on a training validation set will be present in the test data, therefore deriving the percentage of documents that should be associated with each category. Due to the nature of the technique it is only viable for generating a category based threshold, as it requires that all test documents are first scored in respect to every category so that those most relevant can be assigned until the desired proportions are attained.

Possible adjustments are restricted but a multiplier can be exploited to modify the allocations for the distinct categories, giving the capacity to over or under fit results according to the specific application. In effect the percentage of assignments to make per category becomes the prior probability that a document will be assigned altered by the multiplier, so when the multiplier is zero the classifier acts like a trivial rejecter and as it increases the behaviour becomes similar to a trivial acceptor [158]. A feasible balance can be accomplished by automatically tuning the multiplier against a validation set, though to maximise performance it must be a realistic simulation the test data.

A potential drawback is that the necessary assignment densities can only be achieved properly when the entire test set is categorised simultaneously, meaning it is impractical for document pivoted tasks where data is made available in fragments. The notion that the assignment ratios will remain constant is another constraint, as there are a several problems where this is definitely not the case [158], however the flexibility to easily revise independent category thresholds can counteract this to an extent.

3.7.4.3 Ranked

Alternatively named ‘Fixed’ threshold or ‘RCut’ the candidate categories or documents are ranked by score and a globally scoped parameter is used to determine how many of the top items will be assigned. The parameter must be between one and the total number of categories or documents, depending on the classification perspective, and can either be manually or automatically chosen with an appropriate validation set. A value of one means only the top scoring item is assigned and is equivalent to treating the task as a single label problem, whereas as higher values progressively make the classifier behave like a trivial acceptor.

Generally this proves inferior to other techniques in the majority of cases, as the fixed number of assignments is unable to take into account how relevant they are and being hard to fine tune it cannot accommodate intricate levels of optimisation [158]. This is especially apparent when there is a broad range of assignment densities, as the results can either be over fitted to satisfy those highly ranked, under fitted in an attempt to handle all candidates, or kept in the middle with the benefit and detriment of both extremes. Despite these failings it is a simple strategy that is well suited to fragmented test data and all applications where the document and category assignment concentrations are always similar.

3.8 Evaluation Measures

Nearly all evaluation measures are adapted from other general areas of Information Retrieval, with a broad range available that each attempts to embody the performance of a given categorisation solution in a simple format. They are also based on empirical means rather than logical analytics, due to the inherently subjective nature of text categorisation and an inability to define a formal problem specification [4]. Additionally classifiers can provide either a ranked list of candidate categories, assigned by means of a threshold or similar parameter, or just a fixed set of automated category assignments [35]. So if a particular method needs the flexibility to regulate certain aspects it is unlikely to be suitable for the latter type of classifier, which cannot easily vary its output, and will be more appropriate for estimating candidate ranking performance.

Two main factors are relevant to evaluation; effectiveness, which relates to the overall accuracy and completeness of the assignments made, and efficiency, which relates to the overall speed of solution construction and execution [1]. Despite the palpable worth of both factors there is an established belief that effectiveness is imperative and should be the primary concern, whereas efficiency is usually considered a negligible secondary criterion. This is because processing time mostly depends on software and hardware constraints, leading to volatility and the notion that technical improvements diminish its significance [13], though the current exponential rate of data generation might escalate its importance.

Currently available techniques have several limitations that are prominent in experimental investigations, as standardised comparison between various measures is impossible and the aptitude of many is doubtful [44], which is compounded by automatic parameter optimisation. They are also highly restrictive outside of research environments, when required for realistic appraisal of categorisation solutions, as while performance may be sufficiently approximated, none have the capacity to provide any insight into application complexity or scalability.

3.8.1 2-Way Contingency Table/Confusion Matrix

While the terms confusion matrix and contingency table can each have a distinct meaning in other areas of research, in the fields of information retrieval and text categorisation the two are often intermixed [3][35] [53] [45]. They are the most common method for determining the performance of a given categorisation solution, using the proportion of correct and incorrect category assignments to calculate a variety of meaningful statistics [81]. The raw figures are useful for evaluating the effectiveness when applied to single-class problems, while those containing multi-class characteristics require a combination of the values to achieve a meaningful appraisal.

		Actual		Predicted Totals
		Yes	No	
Predicted	Yes	a	b	a + b
	No	c	d	c + d
Actual Totals		a + c	b + d	a + b + c + d

Table 3.8: Contingency Table Overview

a = True Positive (TP) - classifier correctly predicted that category should actually be assigned.

b = False Positive (FP) - classifier incorrectly predicted that category should be assigned, also known as a 'Type I Error'.

c = False Negative (FN) - classifier incorrectly predicted that category should not be assigned, also known as a 'Type II Error'.[1]

d = True Negative (TN) - classifier correctly predicted that category should not be assigned.

A wide variety of subjective probabilities can be derived from the values present in a contingency table [159], each of which allows a slightly different synopsis on the expectation of performance. However due to the traits normally present in most text categorisation problems only a select few of these calculations are regularly employed. It is also necessary to consider those that are used in combination to ensure that a balanced and meaningful analysis is made, as any one particular measure can usually be optimised at the detriment of another.

One example of this is the popular amalgamation of precision and recall, the values of which are relative to each other [13]. Perfect precision is accomplished by means of a trivial rejecter, which is a classifier that merely rejects all category assignments, as this would prevent any false positives by ensuring no positive predictions are made. Likewise perfect recall is possible by the use of a trivial acceptor, which simply accepts every assignment, as this always guarantees there are no false negatives by not permitting any negative predictions.

[1] Note that contrary to other mathematical notation used throughout this document, in this particular section c does not specifically relate to a category.

3.8.1.1 Accuracy

Representing the percentage of all correctly predicted assignments and rejections, this is probably the most instinctive method for evaluating performance and allows for direct comparison between solutions. However despite being common throughout Machine Learning it is rare in text categorisation, as it only provides meaningful analysis for single-label problems [16]. This is because of a high perceived accuracy when classifiers displaying certain attributes are employed to solve multi-label problems that have significantly unbalanced category density or ratio of positive to negative examples.

This is apparent for a classifier that assigns the most frequent category to everything, as it will have a superficial accuracy proportional to the occurrence of that particular category in relation to all of the others. Similarly when there are considerably more negative examples than positive a trivial rejecter will achieve an elevated level of accuracy, which will increase in line with the total volume of documents. Neither of these qualities is desirable, but when used as the primary measure for automatic parameter tuning on a validation set, the parameters will tend toward values that emulate these behaviours [4].

$$\text{Accuracy} = (a + d) / (a + b + c + d) \quad \text{if } (a + b + c + d) > 0, \text{ otherwise } 1$$

3.8.1.2 Error

Referred to as ‘error rate’ [159], though not to be confused with ‘error rate’ when used to define the inverse of precision [160], this is the opposite of accuracy and represents the percentage of predictions that are erroneously assigned and rejected. Like accuracy it is also a poor evaluation metric for anything other than single-label problems, as the same combination of circumstances and classifier types give rise to a perceived low error value.

$$\text{Error} = (b + c) / (a + b + c + d) \quad \text{if } (a + b + c + d) > 0, \text{ otherwise } 0$$

$$\text{Error} = 1 - \text{Accuracy}$$

3.8.1.3 Precision

Precision has a specific meaning in text categorisation that is not always analogous to other research areas, as it refers to the ‘positive predictive value’, ‘correct positive predictions’ or ‘the degree of soundness’ [1]. In a practical sense it denotes the number of correctly assigned categories as a

percentage of the total assignments made, showing how many are right irrespective of whether all required category associations have actually been allocated. This effectively makes it the conditional probability that each category assigned to a random document will be right, so higher values are better and an ideal solution that always yields correct predictions would have a perfect score of 1.

$$\text{Precision} = a / (a + b) \quad \text{if } (a + b) > 0, \text{ otherwise} = 1$$

3.8.1.4 Recall

Also known as ‘sensitivity’, ‘true positive rate’, ‘actual correct positives’ and referred to as the ‘degree of completeness’ [1], this represents the number of correctly predicted assignments as a percentage of the total authentic assignments. In effect it measures how many of the categories that should be assigned actually are, regardless of any other spurious predictions made. This makes it the conditional probability that every required association for a random document will be found, so higher values are better and an ideal solution that never fails to predict all of the right categories would have a perfect score of 1.

$$\text{Recall} = a / (a + c) \quad \text{if } (a + c) > 0, \text{ otherwise} = 1$$

3.8.1.5 Fallout

This is a rarely used alternative to precision [159], which signifies the proportion of incorrect assignments as a percentage of the total number of categories that should have been rejected or left unassigned. Also known as the ‘false positive rate’ it measures how many categories are wrongly assigned, regardless of predictions about those that are not, and is closely related to the specificity or true negative rate, which indicates the appropriately rejected negative cases. It is the conditional probability that a random document will have an unrelated category assigned, so lower scores are better and an ideal solution that never assigns an irrelevant category would have a perfect score of 0.

$$\text{Fallout} = b / (b + d) \quad \text{if } (b + d) > 0, \text{ otherwise } 0$$

$$\text{Specificity} = d / (b + d) \quad \text{if } (b + d) > 0, \text{ otherwise } 1$$

$$\text{Fallout} = 1 - \text{Specificity}$$

3.8.2 Global Averages

The nature of a contingency table means that it relates only to a single document or an individual category, so in order for analysis on a global level all of the respective document or category statistics must be combined. There are two ways to achieve this, know as micro and macro averaging, which both use the same predicted and actual category distributions but total them in a different way to generate results with a particular bias.

One method is not necessarily more advantageous than the other, as it depends on the specific evaluation requirements, and there is evidence to suggest that the inclusion of both together is beneficial wherever possible [42]. Despite this some believe that macro averaging should be the method if choice [13], since it assists in dealing with rare categories, though this contradicts the notion that a general agreement cannot been reached on which is the better measure [1][1]. Certainly throughout literature it seems that micro-averaging is notably more common [136], possibly because it usually generates larger figures [42][149][71], with only a few cases where macro-averaging is utilised exclusively [140].

In the following definitions N represents the total number of distinct contingency tables and x_i denotes one of the four possible values that is present in a particular contingency table i. Only the examples of precision and recall have been shown, but the same techniques are relevant to any type of calculation that can be applied to the data.

3.8.2.1 Micro-Averaging

This essentially treats every aspect of the entire dataset as a single entity, amalgamating all individual contingency tables together and performing calculations using the data from the resultant globally scoped table. This has the effect of making the actual and predicted assignments reflect those present throughout the whole classification space and every measure derived from them is similarly scoped. Due to the raw values being summed a higher weighting is given to common categories, proportional to the number of positive examples in the test data [13], meaning this technique favours solutions that are proficient at classifying frequent categories.

$$\text{Micro-}Precision = (N \Sigma i=1 (a_i)) / (N \Sigma i=1 (a_i + b_i))$$

$$\text{Micro-}Recall = (N \Sigma i=1 (a_i)) / (N \Sigma i=1 (a_i + c_i))$$

3.8.2.2 Macro-Averaging

[1] Note that these contradictory statements are made by the same author, though over a span of several years, yet neither really corresponds with apparent trends in published material.

When macro-averaging is applied all calculations are first performed using the values of each individual contingency table and the global measures are then determined by computing the average of these localised results. As each separate contingency table has an equal weighting toward the overall outcome both common and rare assignments have the same potential to influence the final evaluation, which means solutions need to be adept at classifying all category densities.

$$\text{Macro-}Precision = (N \Sigma i=1 (a_i / (a_i + b_i))) / N$$

$$\text{Macro-}Recall = (N \Sigma i=1 (a_i / (a_i + c_i))) / N$$

3.8.3 F-Measure

No single measure derived from a contingency table is adequate at providing a meaningful analysis of performance and therefore they are typically combined in some manner. The F-Measure or 'F-Score' is based on an effectiveness measure [161] and is one of the most common methods of doing this, consisting of a weighted combination of the precision and recall values. It permits a bias toward either of the measures in order to reflect their importance with respect to the particular classification task taking place, for instance emphasis on precision will penalise incorrect assignments while a focus on recall will penalise too few assignments.

$$F\beta = ((1 + \beta^2) \cdot Precision \cdot Recall) / ((\beta^2 \cdot Precision) + Recall) \quad where\ 0 \leq \beta \leq \infty$$

In the generalised form [159][160] the f-value β indicates the weighting applied to the measure, where $0 \leq \beta < 1$ gives more emphasis to precision and $1 < \beta \leq \infty$ places increased significance to recall [13][1]. This effectively prevents issues that affect the precision and recall metrics when used on their own, as a trivial rejecter with a precision of 1 and recall of 0 and a trivial acceptor with a recall of 1 and precision that equals the average category density per test document [1] will both yield an F-Measure of 0 [13][2].

$$F1 = (2 \cdot Precision \cdot Recall) / (Precision + Recall)$$

When the weighting is balanced, that is β is equal to 1, the equation becomes the special case known as '$F1$', which reduces to a simpler form as it is equivalent to the harmonic mean of precision and recall. This is the favoured f-value [44] though others are also sometimes used, most notably '$F2$' where β is set to 2 and recall is deemed twice as important and '$F0.5$' where β is set to 0.5 and precision has double the significance.

[1] Note there is a minor error in the referenced publication that has been corrected here.
[2] Note that a precision of 0 as stated in the referenced publication is only viable in the unlikely situation where no categories should actually be assigned.

Occasionally and alternative version of the generalised equation is also applied [1], though it can be confusing as the f-value is substituted by a formula that requires a variable scale between 0 and 1. While this provides an input that is related to the ratio of precision against recall, it is not always an intuitive indicator of the f-value being employed, for instance 'F1' requires $\alpha = 0.5$, 'F2' needs $\alpha = 0.2$ and 'F0.5' uses $\alpha = 0.8$.

$$F\beta = 1 / ((\alpha \cdot (1 / Precision)) + ((1 - \alpha) \cdot (1 / Recall))) \quad where\ 0 \leq \alpha \leq 1\ and\ \beta^2 = (1 - \alpha) / \alpha$$

3.8.4 Break-Even Point (BEP)

An popular evaluation metric, the break-even point occurs when classifier parameters are tuned to produce precision and recall that both match a single figure. In theory this value represents the maximum performance of a categorisation solution with respect to both measures and is actually a specific form of the 'F1' measure. Often the break-even point is close to the optimal 'F1' score but is not necessarily equivalent, as precision and recall are not always possible to match in the optimal case.

Sometimes the break-even point might utilise a different assignment threshold to the optimal 'F1' score, which can make it yield a lower value [35][1], though if both apply the same threshold it is always equal or greater. This relationship is evident when restructuring the equation under the assumption that precision and recall are the same, as the arithmetic mean of two variables is always larger than or equal to their harmonic mean. When comparing different systems with the same assignment threshold this means that one with an optimal 'F1' score higher than the break-even point of another is definitely the superior classifier, however the converse may not be true unless the precision and recall values are balanced.

$$\begin{aligned} \text{Break-Even Point} &= F1 \quad when\ Precision = Recall = \alpha \\ &= (2 \cdot \text{Precision} \cdot \text{Recall}) / (\text{Precision} + \text{Recall}) \\ &= (2 \cdot \alpha \cdot \alpha) / (\alpha + \alpha) \\ &= (2 \cdot \alpha^2) / (2 \cdot \alpha) \\ &= \alpha = Precision = Recall \end{aligned}$$

Despite being a prevalent technique the break-even point has numerous flaws, the most significant of which is the determination of identical precision and recall values, as this is seldom trivial by empirical means. Often classifier tuning is insufficiently fine-grained to achieve the exact information and an interpolated break-even point is estimated instead. This is done by averaging the precision and recall

[1] Note that the relationship stated in the referenced publication is always incorrect when the same category assignment threshold is used for both measures.

of the available data point nearest to the desired criteria, but becomes more imprecise as the values are further apart [35].

Further issues are that the metric represents a solution at virtually optimal performance, when test data is utilised to adjust acceptance thresholds and gain the required values, which is impossible with real categorisation problems. This in combination with the lack of detail about the complexity and quantity of parameters and their robustness to change or differing dataset characteristics, makes it impractical for evaluating systems outside of research environments. A final major drawback is that no time related data can be recorded, either in reference to the construction or execution stages, which is neglected by all evaluation methods but grows increasingly significant as data volumes increase.

3.8.5 Interpolated 11-Point Precision

Though not directly comparable with the other metrics, this also represents the effectiveness of a categorisation solution with only a single value and is similarly derived through a combination of the precision and recall measures. It is often employed on a per document basis to evaluate multi-label classifiers [35], which determine a ranked list of candidate assignments for each document, however it can also be done from a per category perspective to appraise others systems capable of varying recall [1].

1. For each interval between the eleven recall thresholds of 0.0, 0.1, ... , 0.9, 1.0 find the maximum precision value and use it as the "representative" precision for the lower boundary of the interval.

2. For the recall threshold of 1.0 the "representative" precision is either the exact precision value if possible, the precision of the closest recall point in the 0.9 to 1.0 interval or the default value of 0 if that interval is empty.

3. Interpolation: For each threshold replace the "representative" precision with the maximum "representative" precision value from this or any higher thresholds.

4. Per-interval averaging: Average the values for each specific interval over all test data to generate eleven globally scoped interval precision scores.

5. Global averaging: Average the globally scoped interval precision scores to obtain a single number that represents the final performance measure.

Figure 3.7 : Steps for Calculating 11 Point Interpolated Precision

Calculation involves altering the parameters that govern category acceptance to achieve eleven set values of recall, equal to the inclusive range of 0.0 to 1.0 with 0.1 increments. Then using the assignments established for each of the eleven matches, the maximum corresponding precision values are computed and averaged to obtain the performance measure. For multiple entities either

macro or micro averaging principles could be applied, though the original methodology seems to infer micro averaging [35], with the mean of each interval first producing eleven global points that are subsequently averaged to generate the final result.

As there is only a single value that represents effectiveness, this technique suffers from many of the same drawbacks encountered with other metrics, including a lack of parameter complexity and timing information. In addition while the use of intervals reduces the need for exact recall determination, which is not always possible, it still attempts to gather eleven data points per entity which can give rise to many empty intervals, especially if done per document with a low category density.

It could be argued that the individual points prior to averaging provide a limited insight into parameter robustness, though this is only the case if they naturally decrease at each threshold. This is due to the reliance on maximal precision, which can conceal the true relationship between recall and precision and may artificially inflate the perceived performance [162]. Other issues might also occur as some literature refers to interpolated 11-point precision as ’average precision’ or ’11-point average precision’ [18][163], which have inconsistent formal equations that are not always maximised and yield different measures [164].

3.9 Chapter Summary

The proposed solution architecture has been examined further in this chapter, studying the purpose of its stages in greater detail and mapping a selection of existing techniques to each, indicating the type of processes they perform and demonstrating how current methods might fit the new structure. It also shows that a clear distinction can be made between them, both in respect to the style of algorithms they involve and the functional roles they fulfil, confirming their independence and revealing that text categorisation is evidently more complex than generally acknowledged at present.

A diversity of possible techniques and resources is already available for every stage and though some seem to receive more attention than others, for instance dimensional reduction and classification, it is apparent from the survey that they are all substantial and independent research topics. However despite several of them originating from well established and prominent circles, such as those of Information Retrieval and Machine Learning, there is a persistent level of confusion being caused by inconsistent definitions supplied throughout the literature. While the previous sections attempt to resolve some of this uncertainty, only a limited collection of the most frequently encountered aspects are covered, dealing with just a small part of an issue that relates to a much larger range of subject matter.

Unfortunately regardless of the doubt eliminated there will always be areas open to interpretation, with many algorithms lacking precise explanation of operational details that may have multiple alternative options or otherwise be difficult to determine. Classification algorithms often exhibit this deficiency but it applies to elements from every stage, for example the mechanism for dealing with unlabelled documents might not be mentioned during the creation of a data subset or a performance calculation. There are also circumstances where current methods are inadequate and appropriate substitutes are yet to be discovered, for instance the inability of evaluation measures to sufficiently portray the solution complexity or accurate computational requirements of time critical real world problems.

4 Existing Framework Review

4.1 Chapter Contributions

A selection of popular and established utilities, algorithm libraries and applications that are related to or capable of performing some degree of text processing are looked at in this chapter. It describes what constitutes a framework for the purpose of this investigation and defines a set of universally desirable characteristics that should apply to all those employed within a research environment where the prototyping and analysis of novel algorithms is required. Several leading open source frameworks are also comprehensively reviewed, with notes made of their innovative features, advantages, disadvantages and most importantly their ability to handle textual data, specifically focusing on the techniques used during categorisation.

As part of each framework review any components associated with text manipulation or classification have been surveyed, with those found being assessed for compatibility against the previously outlined stages and their capacity for adaptation, while the level of active support is also checked. Emphasis has been placed on the aptitude for rapid and modular expansion in a manner that can accommodate the proposed text categorisation architecture, though the currently available options for immediate production of practical solutions are also considered. The findings are compiled into a final summary that details the main benefits and drawbacks of the frameworks from a research perspective, while attempts are also made to identify the root cause of major issues and recommend ways that they might be reduced or eliminated.

4.2 Desirable Framework Attributes

In context a framework is referred to as a tool that allows the combination of multiple components to assist in the partial or entire completion of a process, where the particular process being considered in this instance is the categorisation of textual data. They may take the form of a specially adapted programming language, a code library and application programming interface (API), a small standalone software package or an entire suite of related software applications. Regardless of magnitude and functional scope there are a number of desirable traits applicable to any framework for it to effectively perform research related data mining tasks that involve the investigation of novel concepts.

4.2.1 Availability

It is essential for anything to be available to its anticipated consumers, though software intended to be modified and extended must be accessible in a number of different ways and a variety of factors need to accounted for. All core components ought to be easily obtainable and new developments should be straightforward to locate, install and keep updated, which normally means the use of either centralised content or appropriately synchronised dispersed resources. The same requirements also hold for any additional native or third party modules and preferably a regulated mechanism would allow the safe publication and distribution of these customisations.

Licensing is something else that warrants consideration and specifically what permissions affect the different portions of the code. It is usually accepted that central functionality is not expected to change and might be protected as a corporate asset, but when graphical interfaces and add-ons are involved it is beneficial that they be more flexible and able to accommodate bespoke adaptations. This is an area where commercial products tend to suffer, as much of their content is deliberately locked down or requires alterations to follow strict protocols before being formally endorsed. The level of support can potentially be negatively affected in these circumstances as well, because companies naturally concentrate their main attention on profitable customers and treat others with a lower priority.

4.2.2 Usability

Adoption of a system by the intended users relies on initial ease of operation and rapid productivity, in combination with readily obtainable information and an intuitive set of basic concepts. Simplistic interaction with the application via a suitable user environment is also important, whilst accessible documentation and tutorials should be present to cover any advanced features or complicated aspects. With the requirement for specialist knowledge or a reliance on resources not generally obtainable by a typical user being kept to an absolute minimum and avoided where possible. In

addition active maintenance and regular updates should be available and ideally supported by an appropriate community that is able to assist with any problems that arise.

4.2.3 Functionality

A framework should be implemented in a modular and generic manner and be able to handle multiple situations, capable of not just the original intended purpose but a variety of approaches as dictated by the explorative nature of the research conducted. It should also demonstrate advanced functionality and reliability, whilst containing the potential for simple incorporation of customised components and extended features that go beyond those required for basic tasks.

The ability for 'black box' functionality of individual and grouped aspects is also desirable, reducing the development effort and increasing practicality by permitting areas not under scrutiny to be processed without explicit knowledge of internal mechanisms. While the level of abstraction should be balanced in a manner that does not cause a negative impact to effectiveness by being overly restrictive and preventing an algorithm from achieving its full potential.

4.2.4 Practicality

Imperative for a tool used in research is the ability to generate results from datasets that are taken directly from real world problems or that give significantly realistic simulations that are of practical application. Part of this means they should be able to keep pace with advancements in technology and have a scalable resource capacity that is capable of handling the volumes of data expected in the foreseeable future. The output also needs to contain, or at least include enough information to derive, all of the statistical metadata required to complete any analysis criteria and provide cohesion with the next components in the processing chain.

Another essential feature is the consistent generation of reproducible output, meaning that a specific solution using an identical set of input parameters and data must always provide exactly the same results. This duplication must occur every time and apply to each component, regardless of the executing platform or any 'randomised' aspects, so utilities such as seeded random number generators need to be employed. Without this functionality the research benefits are severely limited, as accurate benchmark comparisons would become unfeasible and remove much of the capacity to validate assertions about any experiments performed.

4.2.5 Compatibility

It is crucial that the output produced by all components is formatted, or can be converted into a form, suitable for the following processing step in a solution or where appropriate for use as the input to another application. Bespoke schemas, obscure variables and unconventional formatting should be avoided where possible as they may be prone to errors or necessitate additional transformations that increase the required development effort. Native conversion tools for acknowledged formats and use of standardised schema definition languages, such as data tables or XML, are preferable as they reduce the potential for compatibility issues.

Backwards compatibility between the central framework and components should be maintained by future versions and updates, ensuring that previously created solutions remain viable and continue to supply reliable results. This is especially important when solutions are reused or their output is depended upon by other systems, as investigations completed under a different version might be negated and become impossible to validate. Despite these constraints the capacity and prospective scalability should not be impeded, as data mining tasks involve the processing of sizable datasets which are likely to keep expanding in the future.

4.3 Existing Frameworks

Many data mining applications exist, though the majority have limited scope with regard to text mining and few can handle the entire process or intricacies of text categorisation, normally consisting of basic algorithms that are treated as minor extensions to the primary content. The most popular generalised and specialised text based frameworks have been reviewed, focusing on their usability, adaptability and capacity to deal with the various stages of text categorisation, along with any notable constructs or mechanisms employed.

While there are abundant products that can be purchased, some even focused exclusively on text mining, they typically have confidential architecture and code that means they are only useful as fully independent packages or static embedded libraries. This restricts suitability in research environments as they are not subject to modification or extension, therefore commercial solutions are deliberately not considered. Instead the concentration is on open source frameworks with licensing that permits public alteration, actively encouraging additions and the exploration of new techniques.

A survey was published in 2012 by the data mining resource KDNuggets that lists the most popular frameworks employed for real world data mining and analytics [165], though the entries are not specific to text categorisation many do contain modules aimed at solving the problem. Consequently the top ranking open source solutions have also been investigated, with an emphasis on their level of support for text processing, ease of use, available learning material, scalability, compatibility and capacity for modification.

Framework	Reference	Overall Rank
R-Project (R)	[166]	1st
Rapid-I: RapidMiner	[167]	3rd
KNIME	[168]	4th
WEKA/Pentaho	[169]	5th
Rapid-I: RapidAnalytics	[167]	8th
Orange	[170]	13th
Other free analytics/data mining software	n/a	15th

Table 4.1 : KDNuggets 2012 Popularity Poll of Data Mining Frameworks (Open Source)

A variety of other resources were also examined, under the stipulation that they provide a minimum level of functionality to enable the completion of a basic step in one of the core text categorisation stages, either performing the whole procedure or being easily integrated into another system. This included graphical and console applications along with a number of text algorithm libraries

[171][172][173][174], however due to the volume of minor utilities available only those that are relatively extensive have been summarised here.

Most of the other libraries reviewed made assumptions that constrained their ability to adequately deal with several stages, imposing particular representations or pre-processing steps and allowing only simplistic indexing metrics. Some of the specialised tools were more suitable and tended to be very flexible when interfacing with other components, probably because they were purposely created to be part of a larger process, though their small operational scope is a limiting factor in itself.

4.3.1 R-Project (R)

Possibly the most popular framework for generic data mining tasks, 'R' [166] was introduced prior to 1997 by Robert Gentleman and Ross Ihaka, who are both based in the University of Auckland statistics department. It is primarily used in academia but has grown substantially over the past decade and is now present in commercial environments, including some of the world's largest organisations. Although based on the 'S' statistical programming language, computationally intensive processes can be implemented in other languages, such as C, Java or Fortran, and then linked to and executed at run time. Similarly any objects created in R can potentially be manipulated via C or Java code, though this is generally considered an advanced level of programming.

4.3.1.1 User Interface

While predominantly console based there are several developer utilities that assist in rapid script production, the majority being free of charge and available for a range of platforms. Some are also designed to emulate the characteristics of or be embedded in other prominent software, for example 'Tinn-R' and 'Deducer' share similarities with MATLAB and IBM SPSS Statistics respectively, whereas 'RExcel' is a Microsoft Excel add-on. Another notable variant is 'R-Enterprise' by Revolution Analytics [175], which provides improved parallel processing, the ability to handle large datasets, a visual development environment, dedicated support and a selection of interfaces and libraries for common languages. A web based interface is also currently planned, but while the GNU General Public License (GPL) covers the core R libraries all advanced Revolution Analytics products are presently only free to academics, with just an optimised version of the original language not requiring commercial licenses.

Over 5300 expansion packages are freely obtainable from multiple online repositories, with the biggest being 'The Comprehensive R Archive Network' (CRAN) which encompasses a network of FTP mirrors. A number of these are also related to processing textual data [176] and many encapsulate existing libraries or provide mechanisms for interfacing with other frameworks. New

packages can be submitted to the CRAN repository by anyone but must follow a strict procedure and undergo peer review by a team of volunteers [177], despite this the sheer diversity of additional content is probably the reason for it being a dominant data mining solution.

4.3.1.2 Text Related Components

Two notable text processing packages are 'tm' [178] and 'RTextTools' [179], the first of which is described as a framework and contains utilities for carrying out several stages, including pre-processing, tokenisation and indexing. There is also the possibility to use it for defining a corpus, creating term by document matrices and even generating models from training data for the classification of a test set. However it does have limitations and relies on numerous external packages to provide most of the advanced features, such as coping with different data formats and the actual classification algorithms.

By comparison "RTextTools" is a standalone package based around the core R libraries with a beta version first released in mid 2011 and an official stable release available from the CRAN repository, while a further developer version is also hosted by the 'GitHub' online code repository. It presently includes nine algorithms for automated text classification and has a variety of analytical capabilities, with the latest release being 1.4.1 which is dependent on the core R language version 2.15.0 or above. Unfortunately at the time of writing the dedicated webpage and GitHub project have not been updated for over half a year so the current state of development is unknown, though it has been utilised by an impressive list of universities and forms part of the tm package.

4.3.1.3 Notable Limitations

One problem with the R language is that it has a steep learning curve for those not familiar with the syntax, especially if lacking suitable experience with similar scripting languages. There are also a substantial amount of packages that are not always independent, as a sizeable quantity relies on specific versions of the core library or other packages, which may in turn contain further dependencies. This can complicate the import and update process, although simplified installation tools are available that alleviate the issue.

Another limitation is that all data being processed is stored within the main system memory, unless a volatile memory mapping is emulated using an alternative storage device. Consequently there are scalability issues and large datasets need to be divided into several manageable portions, which becomes especially difficult when dealing with high dimensional sparse data like text documents. This is a recognised concern however and packages are already being produced that allow parallel computation and stabilised memory maps to secondary storage locations.

4.3.1.4 Development Information

The R-project has a closed group of dedicated members that are exclusively responsible for the improvement and review of updates to the central code, though anyone can submit extensions and additional libraries in the form of supplemental packages. Bug tracking is powered by 'Bugzilla' and is very active with the latest version at the time of writing being 3.0.1, named "Good Sport", which was released on 16/05/2013, while the previous version 2.15.3, called "Security Blanket", was only completed on 01/03/2013.

4.3.2 WEKA

Development on the Waikato Environment for Knowledge Analysis (WEKA) first began in 1992 at the University of Waikato in New Zealand as a government funded project. It originally consisted of a central interfacing component designed in Tool Command Language (TCL), which linked together and directed several different modules and utilities that were written in various programming and scripting languages. Then in 1997 the entire application was completely redeveloped in Java, including all of the major linked algorithms and processes, becoming the foundation for the current version.

It was awarded the SIGKDD Data Mining and Knowledge Discovery Service Award in 2005 and the following year an exclusive license for use in business intelligence was acquired by the Pentaho Corporation. Consequently it presently controls the data mining and predictive analysis requirements of the Pentaho business intelligence suite, though the software itself is still subject to the GNU General Public License version 2.0 (GPLv2). This means only commercial applications or those not also governed by the GPL must specifically purchase a license, which is one of the contributing factors for its widespread success [180], having more than four million downloads since being hosted by the Sourceforge.net online code repository.

4.3.2.1 User Interface

While starting out as a command line only application, three distinct graphical user interfaces were added between 1999 and 2003, as the major stable versions of 3.0 to 3.4 were officially released. They are known as the 'Explorer', the 'Knowledge-Flow' and the 'Experimentor' and along with the basic command console all of them can be used to manipulate data, though they are separate and each serve a different purpose [169].

The Explorer workbench is considered the primary interface and is the most involved, designed for batch processing and containing six panels that represent the possible data mining tasks. 'Preprocess' is for importing data and transforming it into the desired format by means of filtering algorithms. 'Classify' is for applying classification and regression techniques, which employs cross validation by default and displays the constructed models and their estimated accuracy. 'Cluster' provides access to a number of clustering approaches. 'Associate' allows discovery of relationships between the various attributes via association rule learners, however, it only has a comparatively small selection. 'Select attributes' is a way to distinguish attributes that are likely to have the best discriminative ability. 'Visualize' displays a colour coded scatter plot of the data and permits the selection and filtering of individual points.

Knowledge-Flow offers a global view of the entire process taking place, sharing many similarities to the graphical layout of other frameworks and sometimes becoming difficult to manage due to the volume of available modules. Despite being less focused on a particular task it is also has the same capabilities as Explorer but without the need to preload the entire dataset, enabling it to perform incremental updates and handle larger content.

In contrast to the previous interfaces the Experimentor is quite restrictive, being designed to compare multiple algorithms at once in order to determine which achieve the most appropriate predictions. The evaluations can be performed over a collection of several datasets simultaneously and be spread across a network of distributed machines to make it highly scalable, while results can be compiled and stored in centralised locations.

4.3.2.2 Internal Data Structures

Unable to easily cope with multi-relational data, where information is stored in numerous interconnected tables, WEKA internally uses a single flat storage structure and separate utilities are normally required to convert external sources into a suitable format. The main file type natively supported is the 'Attribute Relationship File Format' (ARFF), which is ASCII encoded text comprised of two sections that outline the data organisation and then define the actual instances.

A header first specifies a 'relation' and then lists the individual 'attributes' that belong to it, denoting their name and type which can be either numeric, string, date, nominal or relational. Date attributes are followed by their expected format and nominal declarations include their case sensitive possible values, while relational allows the nesting of attributes and must be used in conjunction with the 'end' keyword. All inputs are placed on new lines and preceded by a '@' symbol, except comments that are signified by '%'.

```
% Simple example ARFF layout comment

@relation document
@attribute content string
@attribute published date [“yyyy-MM-dd”]
@attribute class {class1, class2}

@data
“Content 1”, 1979-11-27, class1
...
“Content 2”, 1981-10-10, class2
```

Figure 4.1 : Basic ARFF Sample Layout

The ‘data’ statement indicates the end of the header and the addition of samples, which are each represented on a single line by comma separated values in the same order as their corresponding attribute declarations. Any missing details are replaced by a ‘?’ and those containing spaces, reserved symbols or nested relational data must be surrounded by quotes. By default the last value on a line is treated as the class in supervised problems and an optional weighting can also be added to the end of a row to show the importance of that particular instance.

Another native file type is ‘SparseARFF’, which only needs non zero or missing values to be explicitly defined in the data section along with their relative position, making it more compact for sparse samples like a document term matrix. However they had a known bug when using string attributes, as strings and nominal options are internally stored as integer index locations of a zero-based array containing all potential attribute values. While efficient this caused an issue because SparseARFF set undefined values to 0 by default, which is considered void during replication, so whenever duplication occurred the first indexed string was lost and the others became incorrectly referenced. The bug was reported fixed quite recently, 2013-01-29, but previous versions must still avoid it by inserting a dummy string into the zero index location if this input type is used.

Newer extensions to the standard ARFF files also exist, referred to as Extensible Attribute-Relation File Format (XRFF), which describe the same features with an XML based document definition. Although less efficient they provide an intuitive and understandable representation that permits greater flexibility when creating and consuming data. For example they have the ability to explicitly define attribute and instance weightings in addition to the class variables by attaching meta-data directly the relevant elements, removing much of the intricacy involved with conversions between formats.

```
<dataset name="corpus" version="1.0">
  <header>
    <attributes>
      <attribute name="content" type="string"/>
      <attribute class="yes" name="class" type="nominal"/>
        <labels>
          <label>class1</label>
          <label>class2</label>
        </labels>
      </attribute>
      <attribute name="published" type="date [yyyy-MM-dd]"/>
    </attributes>
  </header>
  <body>
    <instances>
      <instance>
        <value>”Content 1”</value>
        <value>class1</value>
        <value>2013-05-01</value>
      </instance>
      ...
    </instances>
  </body>
</dataset>
```

Figure 4.2 : Basic XRFF Sample Layout

It is also possible to produce SparseXRFF notation by setting the type of the particular instance element to ‘sparse’ and assigning an index to each of its values, denoting their position relative to the attribute order. Even despite the reductions this affords it is still acknowledged that the XML structure is likely to generate sizeable files, therefore the framework recognises and automatically processes data that has been compressed with gzip and has a filename ending in ‘gz’. It is further capable of compressing output as it is saved and applies to both XRFF and ARFF formats, giving the naming conventions of ‘xrff.gz‘ and ‘arff.gz’ respectively.

4.3.2.3 Text Related Components

Since its release a multitude of applications have been built to enhance WEKA or that use it as a core component [181], including many other frameworks that maintain interfaces to access its functionality, but few are aimed at solving the issue of text categorisation or general text processing. WEKA itself also has only a very small selection of text mining compatible utilities available and there is little mention of them throughout any of the documentation [169][182]. Among these there are just three modules for converting documents into the ARFF format, two simple directory based methods and

one that reads a single Comma Separated Values (CSV) file, which are all too limited to handle the majority of text resources in their standard format.

The first directory based module, called ‘TextDirectoryToARFF’, is only applicable to WEKA versions 3.4.0 up to and including version 3.5.3 and requires that the text files be in a single directory with an explicit filename ending of “.txt”. Every data instance must be a separate file and it is restricted to storing just the filename and content as discrete attributes, while any other information must be manually appended after the conversion process.

Applicable to WEKA version 3.5.3 and upwards the second directory based method is an extension of TextDirectoryToARFF and is referred to as ‘TextDirectoryLoader’. Although subject to most of the same restrictions, this has the optional ability to read documents from multiple subdirectories provided they are located inside a single parent folder and are not structured more than one level deep. In addition to creating attributes from the filename and content it can also associate each instance with a class label according to the name of the subfolder it came from.

The CSV alternative exploits the general ‘CSVLoader’ module that is not strictly text specific and can be employed for any type of data, establishing attribute mappings from the order of the contained information. However the import procedure turns all non numerical attributes into nominal types by default, so the ‘NominalToString’ filtering module must also be applied in order to convert the document content into a proper string. After filtering the output needs to be saved as an ARFF or XRFF file to retain the updated type definitions, which should now be in a format suitable for the other text related modules.

Once the textual data has been imported there are very few tools for manipulating it, with only the solitary ‘StringToWordVector’ filter being available to deal with the majority of stages prior to classification. This is must be applied to convert the string content into vectors that can be consumed by most classification algorithms, but while it has a number of definable variables it does not allow much flexibility. This is because it attempts to encapsulate pre-processing (stopword elimination, stemming and capitalisation), representation (bag of terms), dimension reduction (frequency pruning) and indexing (binary and TF-IDF) into just one step, which not only makes it restrictive but prevents the creation of more sophisticated approaches.

A third-party utility ‘TagHelper Tools’ is also briefly referenced in the literature, which is built on top of WEKA and includes a selection of pre-processing techniques that generate vectors containing bigrams and Part-of-Speech tagging. However it is uncertain if the software is actively maintained, as the main information page still promotes an ‘up and coming’ event that happened in July 2008 [183] and there appears to be no related publications beyond 2007 [184].

4.3.2.4 Capacity for Expansion

Development of custom modules and extensions is a difficult task, as they need to utilise the existing libraries and underlying data structures in order to integrate with the framework, whilst also requiring validation against a set of rules enforced by the graphical interfaces. After these conditions have been met the result can be compiled into a 'package' that may be sent for further testing so that it might appear in the official repository, though the original creator will always have to maintain and host the downloadable file. Making this harder is a lack of readily available documentation that adequately describes these procedures, with searches for online or published resources revealing more sources asking for guidance than providing it.

4.3.2.5 Notable Limitations

Several caveats are involved when installing and running the application, as there are known bugs that require workarounds and the information development roadmap means some modules are not backward compatible with previous versions. The Java Runtime Environment (JRE) must also first be installed and configured with appropriate memory and hardware settings before the program can be executed or have expansions added. This is especially problematic on the increasing number of platforms that do not natively support Java, though these issues are similar for all Java based applications and are alleviated to a degree when installation packages are present to simplify the process.

4.3.2.6 Development Information

WEKA is maintained in parallel with the major editions of the book entitled 'Data Mining: Practical Machine Learning Tools and Techniques', with a stable release of the software being produced to coincide with each new edition. It also follows the Linux pattern of releases, with each major even number relating to a stable version and odd numbers relating to a developer or beta version. At the moment the currently active developer version is 3.7.9 and the last major stable version, which relates to the 3rd edition of the book, is 3.6 with a minor version of 3.6.9 that is still subject to bug fixes.

4.3.3 RapidMiner

The RapidMiner application is written purely in Java and was started in 2001 at the Artificial Intelligence Unit of the Dortmund University of Technology, though it was originally called 'Yet Another Learning Environment' (YALE) [185]. Officially it became known as RapidMiner during the move from Yale version 3.4 to 4.0 beta on the 29th May 2007 due to legal issues, with the change also

included several significant updates. Among these were enhancements to the graphical interface, the addition of eighty new processing operators, performance increases of up to 60%, standardisation of underlying formats, functionality improvements and multiple bug fixes.

In an annual poll run by the KDnuggets data mining resource it ranked second as the most popular data mining and analytic tool used for real projects in 2009, but was first in both 2010 and 2011 before being overtaken by the statistical programming language R and a commercial alternative. Part of this success is no doubt because of the ability to incorporate many of the learning schemes and attribute evaluators from the WEKA framework, along with the statistical modelling features of R-Project. The community edition of the software is also freely accessible under the GNU Affero General Public License (AGPL) version 3, while the enterprise edition requires a commercial license but is directly supported by the product developers and can be embedded in proprietary applications.

4.3.3.1 User Interface

RapidMiner is based around ‘operators’ that each read data from an external source or accept it via one or more ‘input ports’, then apply some form of transformation before sending the results to one or more ‘output ports’. Operators may be nested inside each other and their outputs can be connected to any number of compatible inputs, effectively representing a sequence of analytical processes that is known as the ‘operator tree’. Internally this is stored as an XML structure and can be created manually, by code embedded in external applications, with console commands or via a graphical interface, which has evolved to become relatively simple yet functional.

4.3.3.2 Internal Data Structures

Imported data is held in a central table and accessed by specialised reference collections, but while these have mechanisms to assist development they can still be complicated because they are abstracted in an attempt to trade between efficient computation and memory consumption [186]. The main table is known as the ‘ExampleTable’ with each column heading being an ‘Attribute’, while the individual rows are referred to as ‘Examples’. An ‘ExampleSet’ can also be derived from all or a subset of the Examples to permit access to selected items and associated information, without the need to directly contain or copy the actual data.

Every ExampleSet stores a ‘Partition’ which maintains a list of pointers to the specific items it relates to and provides an ‘ExampleReader’ for iterating over them. By only storing references to the data it means certain portions can be imitated without significantly increasing the memory requirements, though any changes are propagated to all the structures that reference it. This approach favours speed and memory conservation over the ability to modify elements in a confined localised manner,

but other implementations are available with customisations also possible for tailoring functionality to any application [187].

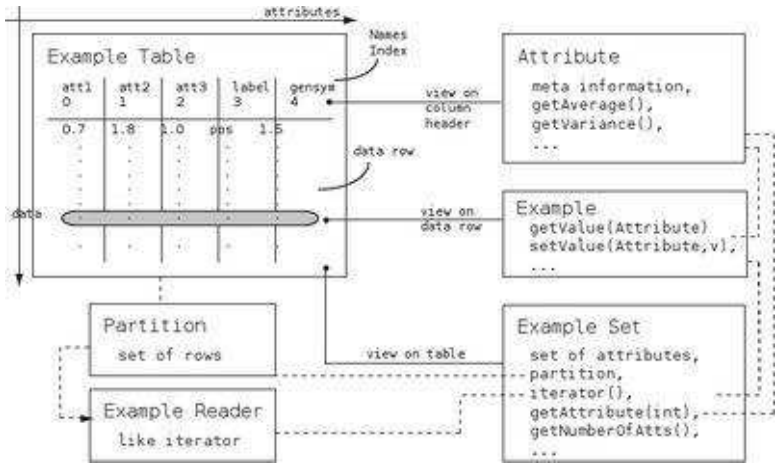


Figure 4.3 : RapidMiner Internal Data Structure

4.3.3.3 Text Related Components

There is a dedicated module based around the 'Word Vector Tool' library [188] that covers all aspects of textual data mining and analytics, which may seem complicated to those just concerned with a particular area. As a general text processing library the Word Vector Tool can handle many different tasks, however it contains only the most common algorithms and does not include all stages of a categorisation solution, though it does manage to keep them reasonably separate. While this gives rise to a number of useful basic operators they must be nested within a special 'ProcessDocuments' operator, which combines multiple algorithms together and blurs the distinction between stages.

The ProcessDocuments operator is employed as a component inside a standard OperatorTree, meaning additional knowledge of the various generic data mining processes is needed. It works on the principle that each individual example is subject to the transforms it contains before they are recompiled back into a single list that can be passed on to the next operation. This is suitable for basic algorithms but limits the use of calculations based on global dataset information and restricts the choice of indexing structure and weighting metrics to the few that are embedded within the actual ProcessDocuments operator.

As a consequence of the imposed constraints the text module is adequate for undertaking average classification and analysis tasks, but lacks sufficient detail or flexibility to cope with bespoke research solutions beyond the addition of simple pre-processing. Further examination gives the impression that this might be deliberate, due to the existence of 'RapidDoc', which is a separate propriety document

management and categorisation system available exclusively to business. The official documentation of which leaves no doubt about it being geared toward commercial aspirations, stating it is maintained by internal staff and that no configuration is necessary to generate instant results [189].

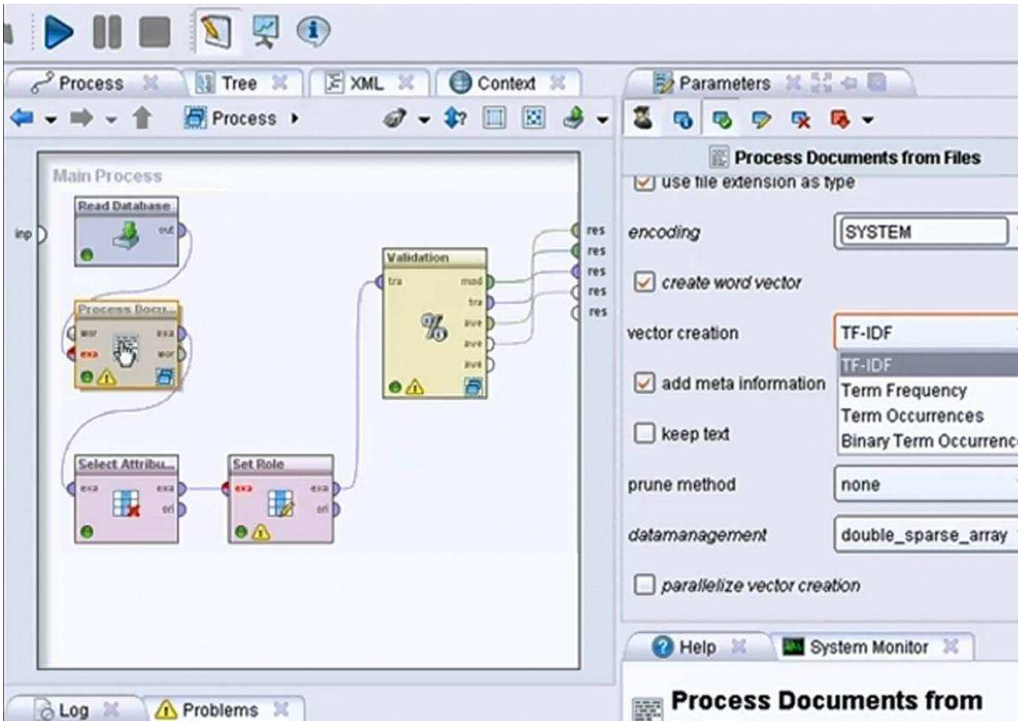


Figure 4.4 : RapidMiner Process Documents Operator

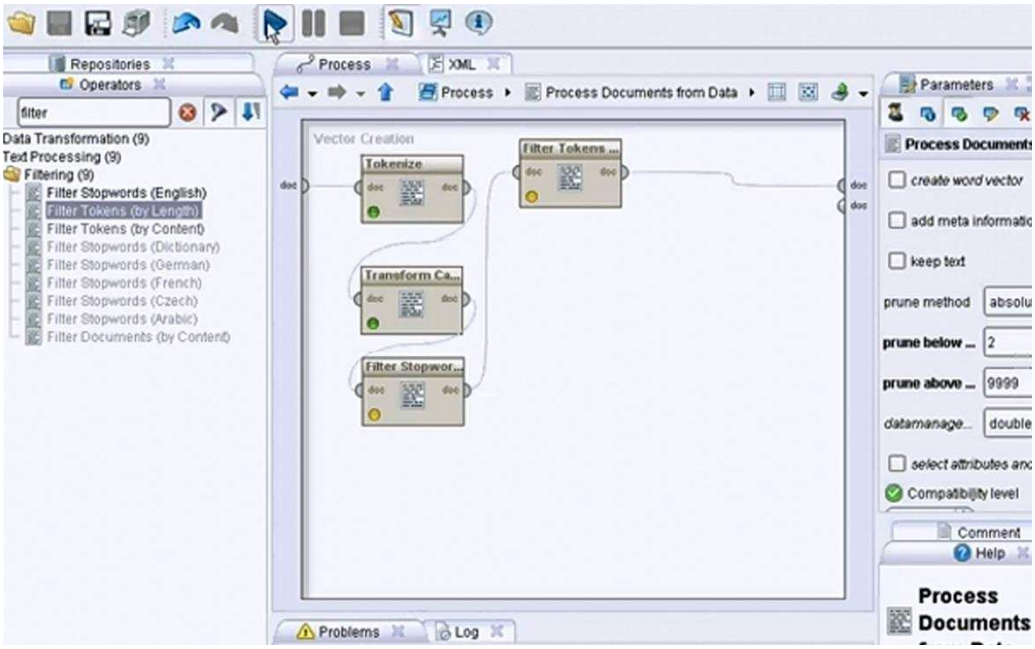


Figure 4.5 : RapidMiner Process Documents Internal Flow

4.3.3.4 Capacity for Expansion

Creation of custom components involves a steep learning curve and significant level of programming ability, though almost any part of the framework can be extended including the graphical interface. Operators in particular can be produced in two ways, with the fastest being to exploit a built-in operator that allows runtime execution of the 'Groovy' scripting language, but this is considered temporary and described as "a quick and dirty hack" in the official white paper [187]. The other option is to build a full extension in Java, which is acknowledged as being more difficult because it must interact with the core data structures and satisfy requirements imposed by the graphical interface.

The benefits of putting in the effort to make full extensions are highlighted in the documentation, which emphasises that they are of great value to the community. New components and updates can also be freely submitted by attaching a subversion patch file to an online feature request, though the exact procedure that occurs after this is unclear. Any items submitted in this manner need to follow a strict set of rules and coding conventions, whilst a form must also be completed that grants joint universal copyright permission of all material to Rapid-I and the original contributor.

4.3.3.5 Development Information

Being under development for over a decade and having a significant commercial aspect, there are a number of official reference manuals and guidelines, along with an active online community and a data mining book that uses RapidMiner for demonstrations [190]. The latest stable version of the framework is 5.3 that was released in late January 2013, though due to changes in the integration mechanism it is not backward compatible with any operators or extensions built prior to version 5.0 [187]. All versions and related software are anonymously obtainable as free community editions from a 'Subversion' code repository hosted on 'SourceForge' since 2004, while bug tracking and feature requests or submissions are handled by 'Bugzilla'.

4.3.4 KNIME

Development of the Konstanz Information Miner (KNIME) started in January 2004 at the University of Konstanz and the first official release occurred in 2006, with a clear intention of being competitive against commercial products when dealing with large scale projects [191]. Originally it was employed mainly for pharmaceutical data mining tasks but gradually expanded into other areas, though there are still several integrated core modules and installable extensions specifically dedicated to chemistry related operations and formats.

It is composed entirely in Java and the graphical interface uses the ‘Eclipse’ environment, however several extensions are available that increase functionality by wrapping aspects of other frameworks and languages, such as WEKA and R-Project. As of version 2.1 the core software is generally covered by the GNU General Public Licence (GPL) version 3.0, with minor amendments that stipulate the independence of extensions and material related to Eclipse as separate applications [192]. Consequently despite being bundled as part of the overall KNIME package, the Eclipse software is under the ‘Eclipse Public License’ (EPL) and individual extensions are licensed according to the desires of their original contributors.

Though only a single version of the core framework exists it has a strong commercial side, with the option to purchase consulting, training and coaching packages or licences that guarantee priority support, feature requests and customised installations [193]. Being specifically developed as enterprise grade software there are multiple levels of licensing, which allow for the possibility of being mixed in different combinations or restructured in order to tailor the product for individual needs.

Supporting this commercialisation is a substantial partner program that is aimed at prospective resale channels, support coordinators and developers of extensions or embedded systems. Exclusive commercial expansions of the basic application are also available, including team resource sharing, remote or scheduled execution, infrastructure integration, user control access, online portals and distributed processing over machine clusters. Clothing merchandise displaying various designs of the KNIME logo can also be purchased via a dedicated Internet retail outlet.

4.3.4.1 User Interface

Predominantly using a graphical interface the main area is referred to as the ‘workbench’ and depicts the current project or ‘workflow’ being employed, which in turn consists of modular transformations known as ‘nodes’. While the general layout shares many similarities with other frameworks it offers a notable selection of data visualisations, taking the form of ‘visualization nodes’ that permit a wide variety of chart, graph and tabular formats. The majority of these are also interactive and may be filtered via a ‘Color Manager’ node or the ‘Hiliting’ function, which both assist information analysis by providing intuitive representations of the inputs and generated results.

The workbench consists of multiple sections, including displays containing descriptions of selected nodes, lists of recently used and favourite components, an outline of the current workflow and a console that reports any system messages. To add nodes to the workflow they are simply dragged into the flow from the relevant list and then connected as required to the compatible inputs and outputs of previously inserted nodes. Each node present in the workflow shows its current status, which can be unprepared, configured or executed, and represents inputs on its left side and outputs

on the right, with distinct symbols and colours to signify different data types and make compatibility quicker to determine.

Extensions can be installed in several ways but doing it from within the application is easiest, though this requires specific software and might necessitate updates to the core system or occasional manual configuration. It is also possible to integrate individual nodes by simply copying a 'Java Archive' (JAR) file containing their compiled binary code into a particular folder and restarting the application, however dependencies between nodes tend to complicate this approach. Regardless of how they are included extensions provide considerable benefits, giving support for additional visualisations, data sources, transformations, reporting features and interfaces to the functions of other frameworks.

4.3.4.2 Text Related Components

A basic text processing extension is available called 'KNIME Text Processing' [130][194], the current version of which is 2.7.1 and new releases seem to coincide with major updates to the core system. It is capable of dealing with all tasks prior to actual classification and aims to either extract a collection of keywords or create a document vector, which is compatible with the standard data mining nodes that generate and employ predictive models. While there are only a limited selection of nodes representing the most popular metrics and algorithms, they are kept independent and can be organised to crudely approximate the proposed stages of a text categorisation solution.

Data input is handled by 'Parsing' or 'IO' nodes that include a 'Document Grabber' for querying the PubMed database of medical journals, a 'Flat File Document Parser' that creates items from individual files and two nodes that construct and interpret the internal XML storage format. Pre-processing uses a mixture of 'Enrichment', 'Pre-Processing' and a subset of 'Transformation' nodes, which apply Named Entity Recognition, Part-of-Speech Tagging, basic morphological operations and global filtering. Data representation has just a single option, the 'BoW Creator' transformation, which converts text into a simple bag-of-words. Dimension Reduction is done by the 'Frequency Filter' and 'Keyword Extractors', which all remove or select terms based on their estimated relevance. Finally indexing is accomplished with the 'Document Vector' transformation, while classification and evaluation are completed via the generic predictive model and scoring nodes.

Specific data structures are employed for the text processing nodes, with three main formats that represent the initial documents as input from a source, a list of their derived terms with optional numerical mappings and a layout similar to a conventional term-by-document matrix. A term can signify either a single or group of words and may have additional informational tags placed after it enclosed by square brackets, while a document refers to a 'DocumentBlob' or 'DocumentCell' object which record the full details of a document and are identified by its title.

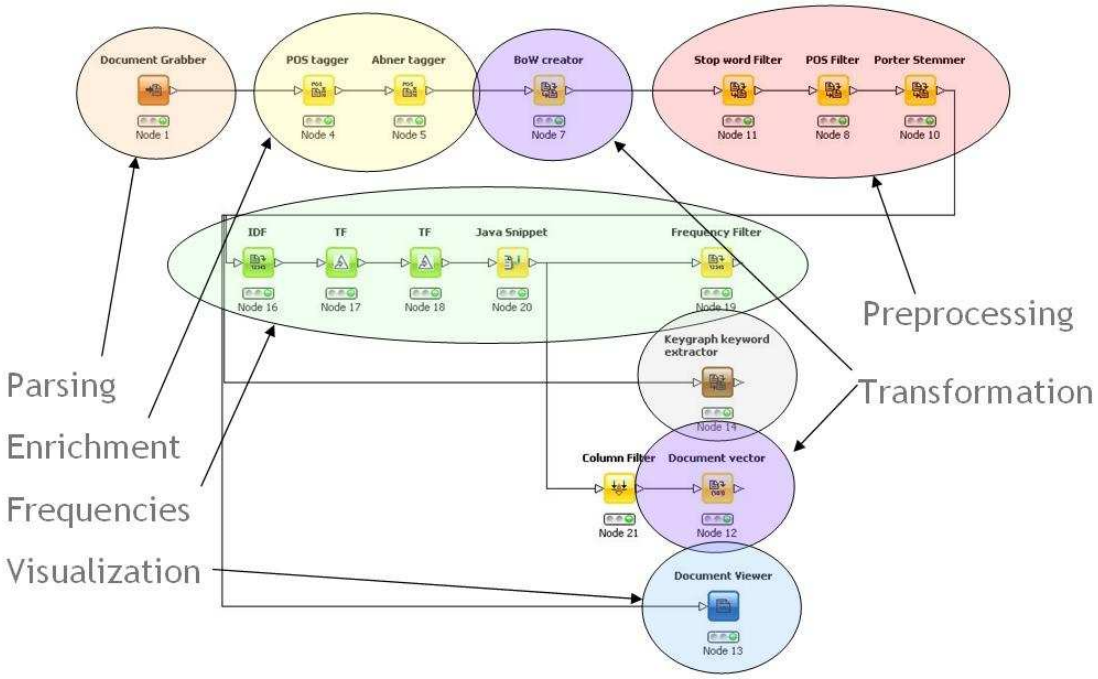


Figure 4.6 : KNIME Text Processing Nodes

The first structure only contains a list of document objects and is exclusive to the IO nodes that read text data into the system at the beginning of a workflow. Conversely the matrix format is produced by the Document Vector transformation and is comprised of generic vectors that can be passed to standard data mining nodes for further processing. All other operations utilise a construct based on a list of the explicit term and document pairings, which is initially built by applying the BoW Creator transformation to the output of an IO node. These map terms to references of the document objects rather than storing multiple copies and can have an unlimited amount of metadata attached to each pairing.

It is unclear how efficient these structures are or what happens when documents with different content have the same identifying title, but using object references considerably reduces the memory consumption at the cost of increased computation. This is often beneficial even though KNIME is able to exploit any available physical storage devices and does not have the constraint of many other frameworks where information is only stored in main memory. Combined with the modularity of text processing operations this affords notable flexibility, however keeping formats in line with generic nodes introduces complexity and there is also a general lack of algorithm diversity.

4.3.4.3 Capacity for Expansion

The standard desktop application is intended for immediate use of existing components, but there is also a KNIME Software Development Kit (SDK) that includes the Eclipse development environment and a 'New Node Wizard' [195] to assist in development of custom nodes. Despite this feature the creation of bespoke nodes is not trivial, though it does allow rapid generation of a skeletal class definition containing the majority of elements needed for their construction. These consist of the 'NodeFactory', 'NodeModel', 'NodeDialog', 'NodeView' and an XML snippet that outlines the item description, which must all adhere to strict conventions in order to maintain compatibility with the core mechanisms and graphical interface [196].

'NodeFactory' encapsulates all other components and produces an instance of a node object when it is added to a workflow.

'NodeDialog' defines the graphical interface that allows the input of parameters and customisable settings. It must be constructed with the Java Swing library but there are a number of default dialog elements available that aid creation and avoid what the official documentation describes as 'the pain of working with swing components and layouts' [197]. Each dialog element also requires a specific 'SettingsModel' object that is responsible for storing, loading, saving and validating any variables attached to it and mapping them to the 'NodeModel'.

'NodeModel' represents the main functionality of a node and when executed it accepts data from the input ports, manipulates or transforms it and then supplies the result to the output ports. It is responsible for checking the validity of the input data specifications, generating corresponding specifications for each output port and saving or loading the internal data model it builds to an external source. Due to tight integration with the graphical interface it must also validate, load and save input settings, supply display warnings and send notification when a view needs updating to reflect any modifications.

'NodeView' shows the data related to a node at various stages of being processed and may be updated at any time if the NodeModel sends notification that a change has occurred. They are created from Java components, such as those from the Swing library, or drawn using Graphics objects. It is also possible to have several types per NodeModel, while multiple instances can be opened simultaneously for the same or different nodes present in a workflow.

4.3.4.4 Development Information

A separate Internet site "KNIMEtech" [198] is dedicated to supporting the user and developer community, with links to online documentation, forums and previews of key features due to be released. It also contains sample code for the creation of custom nodes, detailing how they might be integrated into particular installations, and highlights a book written by the developers that employs

the framework for various demonstrations [199]. While some of this material requires registration, for example asking questions or posting on the forums, most of the content is directly accessible.

The latest stable version 2.7.4 was released on 19th April 2013, using Eclipse version 3.7 and Java version 7, but this is not backward compatible with installations prior to 2.7.0. A list of known issues that affect the current and previous releases of the framework is also available [200], including problems with the core functionality, individual or groups of nodes and any specific software platforms. Developers can obtain the source code from a 'Subversion' repository, though a migration to 'Git' is planned, but registration to the forums is required as valid credentials are needed to gain read access.

4.3.5 ORANGE

Orange was started in 1997 by the two members of the Artificial Intelligence Laboratory at the University of Ljubljana and is now maintained as part of the Bioinformatics Laboratory. Created originally as a library of C++ components and utilities that ran via a command console, it is now written mostly in python, because it is deemed better suited to the type of data exploration and rapid prototyping tasks being undertaken. This conversion process is mainly evident in the core software that is written in a mix of both languages, which is possibly the reason why the framework is considered by some as a work in progress.

Easy analysis of the information being processed is deliberately focused on rather than an extensive library of Machine Learning techniques, as the developers feel there are already sufficient third party extensions that cover this aspect [170]. As a result the default algorithms integrated into the standard installation represent only those most commonly employed in their respective areas, while there is a substantial array of static and interactive data visualisations.

Several major updates to the framework are currently scheduled, including a full redesign of the graphical environment known as the 'Orange Canvas', which is due to be released as both a local and online application in 2013. Conversion of all C++ core components into a pure Python solution is also planned, based around multiple popular machine learning, scientific and mathematic libraries like 'NumPy', 'SciPy' and 'scikit-learn'. An interesting characteristic of this comprehensive use of third party extensions is increased development flexibility, as a practical separation of algorithm and graphical related content is enforced.

4.3.5.1 User Interface

The interface is similar to many other systems, comprising of a central section where the data mining pipeline is constructed by connecting individual ‘Widgets’ that represent certain transformations. Each widget also has a set of identifiable inputs and outputs, referred to as ‘signals’ or ‘communication channels’, which receive or supply a ‘token’ representing the data being processed. However unlike most systems there are multiple distinctive layout options for displaying available components, all of which tend to have a minimalistic appearance.

Arguably the views might too simplified as the input and output channels for each widget are grouped together, so it is not immediately obvious what data formats are accepted or provided without checking the description. When multiple options are available it is also not intuitive which connections are being made if defaults have been defined, requiring that a separate view be displayed to allow their inspection and modification. Likewise if no defaults are determined this view is automatically shown and must be manually configured for every link established, potentially detracting from the rapid prototyping experience.

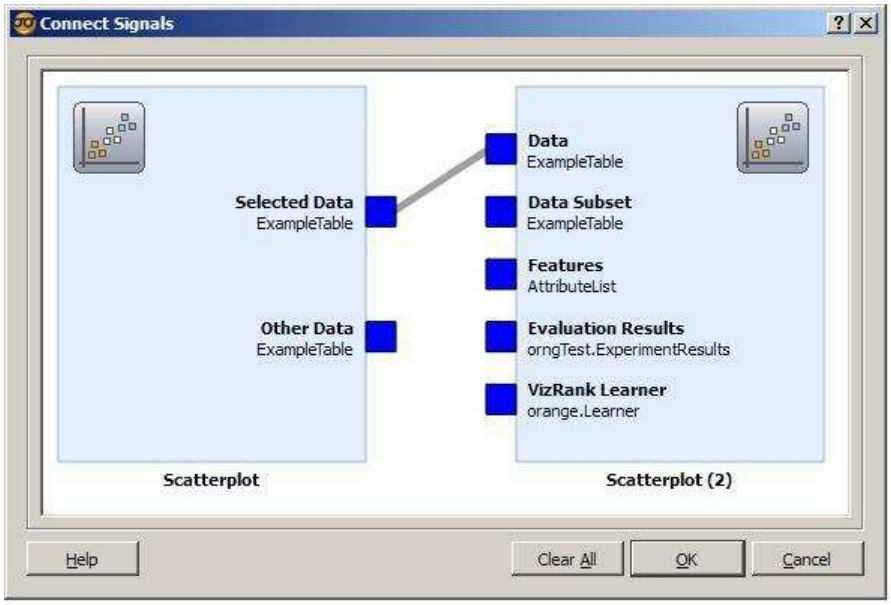


Figure 4.7 : Orange Connect Signals View

Some widgets also seem to combine a lot of functionality, allowing the input of several distinct elements and outputting only a single structure that contains multiple independent results. The ‘Test Learners’, ‘Predictions’ and a selection of visualisation widgets are examples of this and can make it difficult to determine the exact processes that are taking place. Though despite the possible confusion they do permit construction of straightforward pipelines and have the ability to directly compare and simultaneously illustrate numerous result sets.

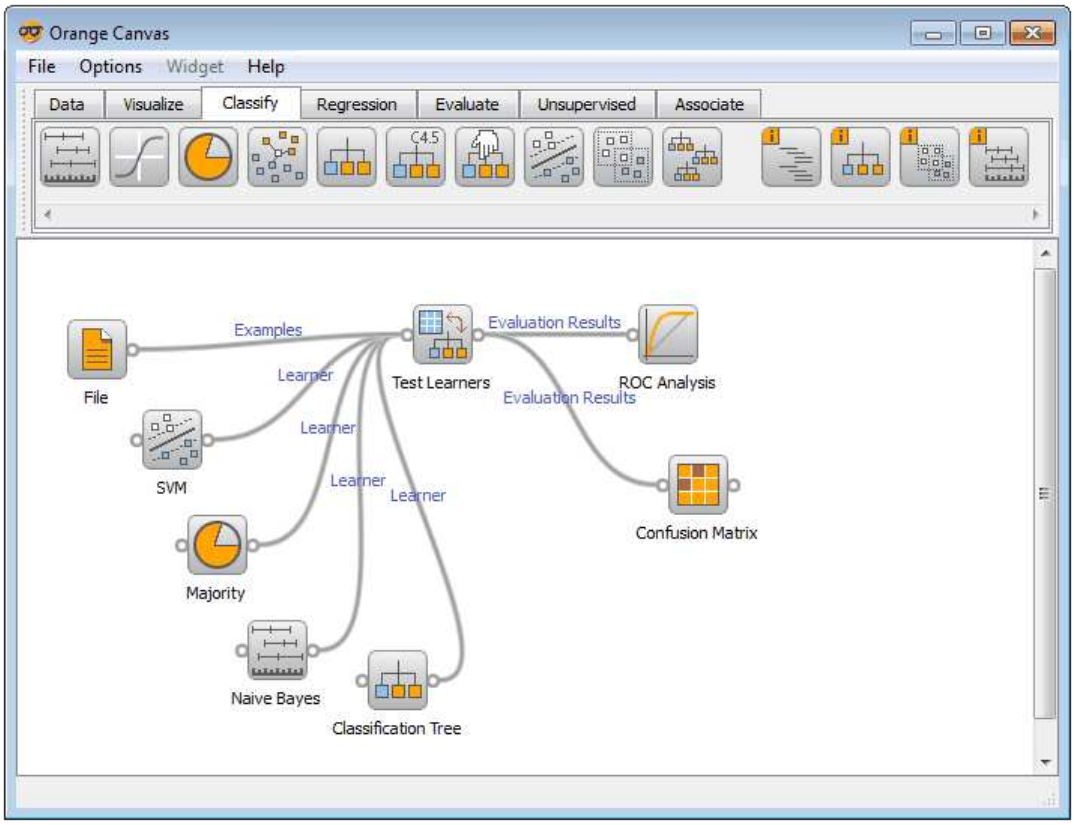


Figure 4.8 : Orange Test Learners Widget

4.3.5.2 Text Related Components

A small add-on is available for text processing [201] but it has insufficient documentation and a very narrow range of algorithms, implementing just a small fraction of those commonly encountered. Development seems sporadic as well, with several months between updates and no significant changes being added for more a year. Many of the categorisation stages and individual steps they involve are also mixed together in an unproductive way, causing conflict between the different widgets and making it unclear how they might be joined to form a modular solution.

'TextFile' is the sole option for initially loading documents into a system, reading from a single file formatted in either XML or SGM, though exact details of the required structure are hard to find.

'Preprocess' consists of basic stopword elimination, conversion to lowercase and stemming, which can all be applied in a variety of different languages, however the precise list of words removed and algorithms used are not specified. Certain aspects also have reduced flexibility as the order of execution is fixed and it automatically tokenises and reconstructs text in the background so that the input and output formats are similar.

‘BagOfWords’ creates a standard bag-of-words representation of the document content and indexes it based on either term frequency or TF-IDF weightings, though the actual TF-IDF calculation employed is uncertain as it is described simply as ‘log(1/f)’. There are also two optional methods for normalising the raw values, but again the calculations are uncertain due to the vague labels and accompanying information.

‘LetterNGram’ determines character n-grams with lengths of two, three or four and then automatically indexes them based on their frequency of occurrence. It does not seem possible to selectively filter the data or choose an alternative weighting metric when running this widget.

‘WordNGram’ merges aspects of pre-processing, representation, indexing and dimensional reduction into one widget. It generates n-grams of two, three or four consecutive words that are not split by punctuation and has the option to extract potential Named Entities by simplistic examination of word capitalisation. Indexing is done automatically with term frequency as the weighting, while dimension reduction uses a global threshold in conjunction with a choice of five different feature selection measures to remove any terms that do not meet a minimum value. It also has the ability to remove a custom list of stopwords, though it is uncertain when this is occurs and whether it affects individual words or the derived n-grams.

‘TextFeatureSelection’ is purely for dimensional reduction but has the capacity to remove entire documents from the dataset and not just discrete terms. Both types employ a minimum or maximum threshold of a fixed value or percentage, with feature selection being based on random choice, global frequency or the total quantity of documents containing the term and document selection using the content length or number of unique terms.

‘TextDistance’ is mainly for visualising disparity between the documents, producing a document-by-document matrix that records either the similarity or distance between the various pairings. This has little use in a text categorisation solution, but may give some indication of the relative hardness posed by a dataset, even though it is unclear how the values are determined.

4.3.5.3 Capacity for Expansion

Despite a steep learning curve new widgets can be developed relatively quickly as only simple code structures must be adhered to and the graphical interface enforces minimal requirements, with an XML style header in the Python script denoting the majority of details and visualisation options. Two customisable areas are also attached to each widget for display purposes, with the ‘ControlArea’ generally for showing configurable variables and basic information, while the ‘MainArea’ is typically used to reiterate these settings and present current input data in an appropriate manner.

Input and output types can be any format but should be chosen to match those that are expected before and after the widget in a pipeline, as the compatibility of a connection is automatically resolved by testing these for equivalence. Mechanisms are also needed for handling the incoming data, outgoing data and any additional display aspects, with a number of optional bespoke components available for enhancing appearance, such as progress bars and utilities for creating graphs or scatter plots.

Once completed a widget is easily added to the environment by putting it in a particular subfolder within the installation files, the location of which determines where it is shown in the repository layout, while a priority value set in the script XML header dictates its order relative to other items. Testing can be done via the full Orange Canvas or by a test script included at the end of the main code block, however if several data transformations are required prior to the process being checked this script could become complex.

4.3.5.4 Notable Limitations

When loaded into the system all information is stored in a table object that defines the individual samples along with a 'domain' description, which is a set of variables and meta attributes that describe characteristics of the data, for example feature names, class labels, types and categorical values. Although this provides a centralised method for managing the data it requires knowledge of specific functions for accessing and manipulating the data and relies on string definitions for the selection of desired characteristics. It also means that all data being processed is held in main memory, so any structures that grow beyond this capacity will fail, though it is possible to load and save data instances to text files using a native tab delimited format.

Another trait that might prove frustrating, especially when constructing extensive or continually modified pipelines while investigating big datasets, is that by default all processing is intentionally done on-the-fly as connections are created between widgets or new settings are applied. This can be made optional during production of a widget, which becoming the standard in newer components, but the developer is responsible for including this feature and it is not present in many existing components.

Orange shows potential for a rapid prototyping due to the relatively simple nature of fabricating basic widgets, though there is currently a significant lack of text processing capabilities. It has comprehensive documentation and code examples that explain how all areas of the application can be customised, along with an active forum where the core developers seem enthusiastic about answering questions and listening to suggestions. However advanced techniques do require a considerable number of specialist functions and can become overly complicated, as can the

resources that describe them, which would also benefit from reorganisation as exact information is often difficult to find. Aside from the forum, where discussions and feature requests are handled, there is also an online blog that works alongside the formal documentation to aid both general users and developers.

4.3.5.5 Development Information

The latest version of the core framework is 2.7, which was released recently and includes major updates to the Orange Canvas and visual programming interfaces. The primary code and all prominent add-ons use 'Mercurial' repositories that are hosted on 'BitBucket', with bug tracking provided by 'Trac'. A Wiki and ticketing system is attached to the bug tracking and those that have registered can edit content or start new tickets to request features or report bugs. Developers can also access a 'Skype' chat and contribute code, but the guidelines are minimal and refer to fixing existing tickets rather than submitting new widgets or add-ons. The submission and review process is also unclear, though all developers must to agree to a 'Contributor License Agreement' (CLA) that permits irrevocable and perpetual joint ownership of their code with Orange.

4.3.6 GATE

Developed at Sheffield University and started in January 1995 the 'General Architecture for Text Engineering' (GATE) is one of the few text specific open source projects and was first officially released in 1996, but subsequently re-released with a new code base in 2002. It is written in a combination of Java, Groovy and other languages compatible with the Java Virtual Machine (JVM), with the current version requiring Java version 1.6.0_10 or above and only updates or bug fixes compatible with version 6 being accepted. The core components are licensed under version 3.0 of the 'GNU Lesser General Public License' (LGPL) with minor revisions[1], while third party plug-ins use mix of licenses, including the 'GNU Affero General Public License' (AGPL) version 3.0 and the 'Apache Licence' version 2.0.

Much of the online information describing GATE is geared toward commercialisation of the system as an enterprise solution, though some is aimed at individual tutors and researchers with example case studies from business, education and research settings. Details of bespoke services for installations in corporate environments, training courses and the potential to gain various levels of certification are presented as well. Sections are also available regarding sponsorship [202] and partner programs [203], which offer the benefits of discounted training, advanced access to new releases, premium support, feature requests and priority maintenance.

[1] At present the main GATE website cites LGPLv3, while the code repository hosted on SourceForge cites LGPLv2.

4.3.6.1 User Interface

GATE is actually a family of products based around a collection of text processing related applications that provide a range of basic functionality, semantic annotation, named entity recognition and information extraction. For the purpose of research two products are likely to be of interest, the first is 'GATE Embedded' which encompasses the core libraries that form the hub of all incorporated utilities and that are intentionally designed for simple integration with external systems. The second is the graphical interface that runs on top of these libraries, called 'GATE Developer', which allows easier interaction and enhanced visualisations when creating text mining processes and analysing the results they generate.

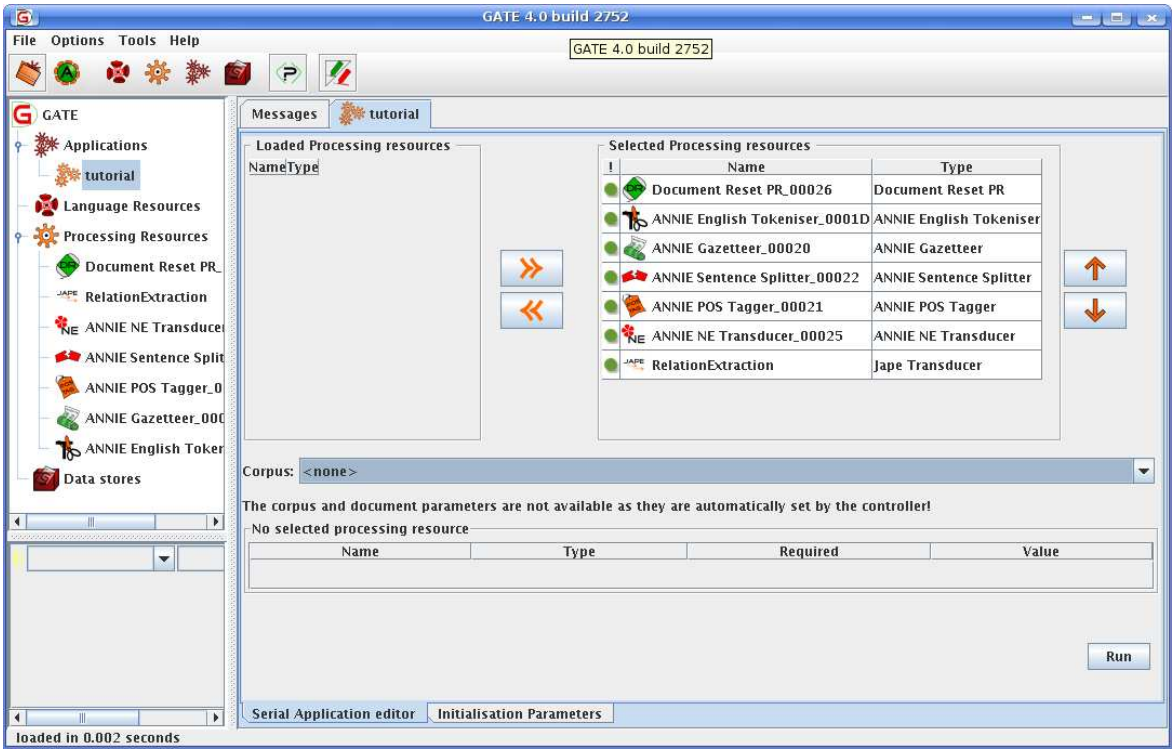


Figure 4.9 : GATE Pipeline View

The framework is built around 'Language Resources', 'Visual Resources' and 'Process Resources', which are collectively known as 'a Collection of Reusable Objects for Language Engineering' (CREOLE) and are combined to make an 'Application' containing a 'Pipeline' of tasks. A Language Resource refers to the textual data being worked on and is able to load a single document or an entire corpus into the system using a variety of components that support a number of common formats. Visual Resources are attached to the other resource types and are responsible for displaying the data and any inputs at different points during execution. Process Resources handle the actual

transformations that take place and are the main elements of a pipeline, with the output of one becoming the input of the next. Only a linear progression of process resources is possible and each is initialised with defined parameters that can be manually adjusted at runtime.

4.3.6.2 Text Related Components

Many third party text processing software libraries have been partially or wholly embedded within the framework, typically in the form of a 'plug-in' consisting of extra Processing Resources that permit additional functionality [204]. Often these supplementary utilities are the only means to perform certain tasks, such as text categorisation, because the core architecture is designed mainly for advanced term or phrase tagging and the structured annotation of documents. Some of the more prominent packages presently integrated include; LingPipe [205], Snowball [173], MALLET [172], Lucene [174], OpenNLP [206], Wordnet [107], SVMlight [207] and a selection of machine learning modules from WEKA.

4.3.6.3 Capacity for Expansion

Despite a reliance on external solutions the creation of new Resources is not trivial due to the event listeners and required for the user interface, though development is not as complicated as some of the other frameworks employing extensive graphics. Custom plug-ins must all follow strict regulations for compatibility, but can be submitted to the central build by anyone or hosted from individual disparate repositories that are straightforward to configure via simple XML. Once installed plug-ins are merely listed alphabetically within the application, so the name determines their general location and should be specified according to the functionality provided and formatted in a particular way to make them easier to find.

4.3.6.4 Development Information

Little information is obtainable about the internal workings of the core framework or its full range of capabilities, however there is comprehensive documentation regarding most of the plug-ins. As would be expected of a commercialised product the software is actively maintained, but there are large gaps of several months between each major release. Certain information also does not appear to be regularly updated, for example the details of upcoming events and release roadmap on the main website are presently two years out of date, while the most recent news on the site hosting the code repository is over four years old.

Aside from a public Wiki containing a detailed FAQ section there are also several online tutorials, a mailing list, user guides and a small number of purchasable books, all of which explain the basics of GATE and the extensions distributed with the standard package. An official book ‘Text Processing with GATE’ [160] is also available to buy, though is very similar to the freely accessible online manual [208] but with additional content related to the graphical interface. Specifically for those wanting to produce new components or modify the existing ones there are a variety of general guidelines that are accompanied by code definitions and example templates for common tasks.

The current stable version is 7.1 and was released in November 2012, though the developer version is built nightly and is easily obtainable as are archives of previous stable releases. Code for the core application and the plug-ins distributed with it are located in a ‘Subversion’ repository hosted on the ‘SourceForge’ website, which is also responsible for bug tracking. Anyone can report bugs or send patches to fix a particular issue, however these must be submitted via the general mailing list or through direct contact with the central development team who review everything before applying it to the official build.

4.4 Comparative Overview

This section provides an overview of the information gathered for each reviewed framework, breaking it down into a comparative summary according to the desirable criteria supplied at the beginning of this chapter. Note that whilst care has been taken to keep every detail factual, there are some areas that are unavoidably subjective and in these cases the conclusions reached were based on evidence gathered during the previous section.

4.4.1 Availability

Core packages and source code are easy to obtain for every framework, with only KNIME requesting optional user registration as part of the process, though all except R also require appropriate runtime environments to be installed on the platform being used. Python is required for Orange, while the rest need a compatible version of the JAVA Virtual Machine (JVM), which may additionally involve manual configuration of environmental variables in order to allocate enough system resources. In general they all appear to be actively maintained and have very regular updates and releases, aside from GATE which has a markedly longer release cycle. Extensions for the core applications are readily available as well, but some do suffer from complicated installation due to dependencies on specific versions of other external packages, something that especially seems to be the case for R and KNIME.

Overall there is a large diversity of available add-ons and different mechanisms are employed to publish content from third party contributors that is not natively embedded. RapidMiner, Orange, KNIME and GATE offer the option to host independent repositories or allow submission to official centralised ones, subject to meeting strict development standards and insisting on joint copyright or ownership. WEKA has a similar arrangement too, but contributors remain responsible for hosting their own downloadable files. By contrast, R is a language and environment rather than an application so does not have extensions in the same way, though several entities, including the project originators, host repositories of code libraries that each has their own submission procedures and rules.

The reviews were deliberately focused only on open-source frameworks, so the core components are freely available to a certain degree and most fall under one of the GNU General Public Licence (GPL) versions. However, there are discrepancies, for instance GATE has numerous licenses covering the native embedded modules and KNIME and Orange have different licenses for their graphic interfaces, while their modules and those of R are subject to contributor restrictions. Only WEKA has a standard license covering its whole core content and official modules, but does offer a commercial alternative to any customers wishing to embed it inside non compliant applications.

A considerable amount of commercialisation is also evident, despite the open source criteria, with the biggest provider of content related to R requiring commercial licenses for non-academics and a

variety of tailored licensing structures being supplied by the rest to meet specific needs. RapidMiner, KNIME and GATE in particular come across as very profit orientated, as each offers priority support and enterprise deployments at a cost, while RapidMiner also has dedicated products only available for purchase and GATE is aimed almost entirely at paying consumers.

4.4.2 Usability

The graphical interfaces greatly increase usability for the average user, but none of them are perfect and R in particular has a very steep learning curve due to being a statistical based language. WEKA, RapidMiner and KNIME also have issues, as though they are intuitive and very user orientated, the sheer volume of available packages can often make it difficult to locate specific components. Having fewer extensions this problem is not as prominent in Orange and instead it is let down by the overly minimalistic appearance, with awkward aspects like multiple connect signals and widgets that attempt too much at once. Conversely GATE has a less intuitive layout that often seems cluttered, which is not helped by the alphabetical ordering of add-ons and use of unusual terminology and acronyms.

Interaction with the frameworks is mostly intuitive and the built in components enable beginners to achieve a lot in a short time, R is the only exception as the lack of an extensive graphical interface means it requires a good working knowledge to be productive. They all have the equivalent of a console input and, apart from R and GATE, contain views that share many common traits and similar nonlinear work flows, but there are novel elements that distinguish between them. WEKA has three versatile views for distinct purposes, Orange has a number of possible screen arrangements and KNIME has excellent native data visualisations, which leaves GATE appearing quite restrictive by comparison.

Comprehensive documentation and guidance is available for the core content of everything reviewed, mainly taking the form of online tutorials and active community forums, though books and other types of publications are also widely obtainable. This is unsurprising giving the popularity of the products and has been augmented in most cases by their commercialisation, with options to purchase priority support and training directly for KNIME, RapidMiner and GATE.

4.4.3 Functionality

Being specifically aimed at text processing GATE has the best selection of relevant algorithms, but even so it still relies on embedded third party libraries to provide the majority of its features, including categorisation. Surprisingly all the other frameworks are currently lacking in text mining content and rely on external modules to supply only the most basic tasks, while employing generic classifiers that require the data to be converted into appropriate formats. Of those reviewed the KNIME text facilities

are perhaps the closest competition for GATE because they are reasonably atomic processes, though they utilise complex internal data structures and, like the rest, do not offer advanced functionality.

Unfortunately none of the available packages are complete or modular enough to be applied directly to the architecture proposed in chapter two, with WEKA and Orange being the worst for grouping stages together in single modules and enforcing too many assumptions. Text related documentation is notably poor as well, containing gaps in the information about configuring settings properly or describing the exact transformations that are applied by particular components. RapidMiner does show some potential but is let down by its limited capability and the need for special nested operators, which might be deliberate as the company that owns it produces a suite of text specific commercial applications.

All of the frameworks with an extensive graphical interface require substantial effort when developing new components, primarily due to the custom event listeners and manual data validation that takes place. Each has specialised code libraries that must be utilised when binding customisations to the user environments and are overly complicated because of the elaborate storage mechanisms and diversity of interactions that can occur. However, both GATE and Orange do seem to have slightly simpler procedures, likely due to them relying heavily on third parties to provide the bulk of their data transformations. Obviously R does not necessarily share these issues, being an actual programming language, but the required learning curve and inadequacy of existing text processing libraries are a severe drawback nonetheless.

Guidance for those producing customisations is moderate and there are plenty of online tutorials and actively encouraged developer forums, probably because community contributions are an economical method of expansion. KNIME in particular has thorough documentation and a Software Development Kit (SDK) that contains facilities to help generate extensions, while the creators of RapidMiner go a step further and offer direct developer support to those that hold a commercial license. In regard to programming languages employed, the only limitations are those imposed by the underlying run time environments; R is a language in itself, Orange is presently a mix of Python and C++ and the others are written predominantly in JAVA.

4.4.4 Practicality

Orange is perhaps the worst at handling sizeable real world data, as everything is stored in system memory of the machine it runs on and a default setting means that data transformations immediately execute as they are created. In standard form both WEKA and R also suffer from similar restrictions, but several extensions are available for them that permit parallel processing using either machine clusters or grid computing. The rest claim enterprise level scalability, which is standard in KNIME and part of the commercial packages for RapidMiner and GATE, though while the first two allow cluster

and grid computing the GATE documentation does not elaborate on how it is achieved or what constraints are involved.

Another point worth noting is that WEKA requires input to be in a highly specialised format and a lack of modules for reading textual data means that a custom step will probably be necessary to perform the conversion. RapidMiner has issues due to the way it handles data too because it was designed with efficiency in mind, causing simple mechanisms for accessing internal storage to be sacrificed and subsequently making it harder to develop customisations. These shortcomings suggest that, when it comes to dealing with practical simulations and problem scalability, KNIME is almost certainly the best choice from an open-source standpoint.

In respect to output, all the frameworks are theoretically able to generate reproducible work flows and results, making them viable to form the basis of accurate comparative benchmarks, though in reality much depends on the quality of the individual modules and customisations employed. This poses a considerable problem since the current text processing packages are inadequate when considering their granularity and capacity to report information after each stage undertaken during categorisation. Consequently, despite a selection of data visualisations and reports being available across the board, either GATE or KNIME would be the dominant option in this case, purely because they have the most focused and atomic text related components.

4.4.5 Compatibility

Despite being competing products the interoperability between R, WEKA, KNIME and RapidMiner is surprisingly good, with easily obtainable extensions that allow all of them to directly support modules built for WEKA and R. Unfortunately this does not appear to hold the other way around for RapidMiner or KNIME components and neither GATE nor Orange exhibit the same compatibility, though GATE does have a limited interface for WEKA and there is an Orange add-on that reads WEKA specific data formats. Except for R, linking any of them to external applications is also not a simple task due to the generally poor separation of algorithms and user environment, but embedded versions do exist and RapidMiner work flows are stored as basic XML so their manipulation is straightforward.

Internally flat files or two dimensional table structures are employed to store the data about to be processed and these are accessed in a variety of ways; R and Orange tend to use in-memory objects, GATE has a series of key value pairs and the rest build structures akin to relational data tables. None of these are universally recognised formats and must be exported in a standardised manner before they will be accepted by most applications. GATE excels at this and natively interprets and generates a number of common formats, whereas WEKA only accepts its own proprietary file types and the others rely on a mix of embedded and third party extensions to satisfy the particular requirements of each transformation taking place. In every case the information that can be output is entirely restricted

by the end result provided by the individual modules and any suitably verbose details they might generate during execution.

Unexpectedly when it comes to backward compatibility GATE alone does not seem to have significant problems, with just an imposed constraint of the JAVA edition newer versions must adhere to. All the rest have acknowledged issues with dependencies between their various packages and also specific versions of the core content, which can be so elaborate that installation advice is frequently provided and every framework has a dedicated package manager to assist with the procedure. KNIME and RapidMiner seem especially weak in this area, as both have previously released core updates that knowingly broke all the modules published prior to them, though this definitely does not occur on a regular basis.

4.5 Analysis of Suitability

Analysing the frameworks currently available it is apparent that, while each has a number of benefits, none are particularly suited for the investigation of novel approaches comprised of several distinct stages. Many provide exceptional intuitive user environments along with a high level of modularity and reusability, though these attributes also tend to be the reason for limited scalability and simple integration of new features. This is especially an issue for text categorisation as it generally involves large dimensionality, significant data and multipart solutions, for which existing tools are inadequate lacking the flexibility or capacity to handle more than the most basic text processing tasks.

When considering the research of bespoke algorithms the use of specialised language or presence of a comprehensive graphical interface repeatedly leads to a steep learning curve and variety of complex requirements that must be satisfied before attempting the addition of customised components. Several prevalent frameworks also appear to be heading towards large scale commercialisation, with a distinct focus on supporting business over the open community, which supplies necessary funding but can mean applications become heavily controlled and diminishes research priority.

Due to the difficulty and effort required for expansion and customisation of frameworks, many new algorithms are created through the use of disposable prototyping techniques as they tend to be much quicker for producing results. However these are inclined to be single use tools, normally designed with little flexibility and unable to incorporate developments beyond their intended function, which causes them to be unfeasible for extension or effective recycling. Typically they are also hard to precisely define and recreate, making exact duplication of algorithms or results problematic and giving rise to discrepancies when performing comparative analysis

In addition the nature of rapid development regularly means that aspects outside the main focus of investigation are implemented to a minimum level or ignored completely, potentially causing further inconsistencies if details get neglected and are not accurately defined. A variety of text processing utilities exist that are sometimes employed to deal with common categorisation tasks, predominantly those occurring in the initial stages, which removes the need to reproduce them for each solution and reduces much of the ambiguity. Though they are not always pertinent with many having a narrow scope, being highly task specific or enforcing certain assumptions, for example Snowball [173] is dedicated just to stemming, Lucene [174] is designed expressly for text search and the Simmetrics library [209] enforces limited data representations and index weighting metrics. Consequently despite the importance of reusable elements, particularly in an area like text categorisation that involves many easily overlooked stages, it can be detrimental to use them if they excessively restrict experimental variation.

It is clear that a solution is needed that provides the advantages of both an established framework and prototyping techniques, allowing the rapid development of new concepts in a flexible manner with reusable and portable components. Whilst keeping the process as simple as possible and avoiding the associated drawbacks of either approach, such as restricted capacity for expansion or complex integration of new features. There are also other considerations that need to be made to ensure that any derived product is fit for research purposes, while containing adequate functionality to be of practical application.

4.5.1 Research Orientated

Existing frameworks function reasonably well and where applicable have appealing visuals, but whether console or graphically based they seem to be, and some definitely are [170], designed with the user interface as the primary focus, which makes the process of adding new components quite convoluted. To be practical and flexible enough for research they really need to be centred on the fast production and easy integration of algorithms, in a manner that permits complete segregation and exploitation of the individual components as parts of other solutions.

Open licensing of the customisable components at the very least is also fundamental, allowing the sharing, reuse and modification of resources and giving the capability of exact result duplication and accurate comparative analysis. This was deliberately a prerequisite of the applications investigated so they all meet this requirement, though a few have started to demonstrate favouritism towards commercialisation and generation of profit over supporting the general community [160][210][193]. Public availability is another crucial factor, ideally with accessible repositories where items can be centrally stored or retrieved without a complicated registration or approval procedure and preferably a mechanism to distinguish those that are officially sanctioned. A number of frameworks have a feature for creating private repositories, but most make submissions to endorsed repositories challenging by imposing demanding criteria , formal review or only acknowledge specific partners [160][211][177], helping to ensure quality but involving considerable effort.

4.5.2 Ease of Use

Comprehensive documentation and guides are not necessarily required and should definitely not be a substitute for an intuitive application, but where they are needed it is important that they are concise and helpful. The majority of frameworks employ elaborate architectures and environments or attempt to handle so many different aspects that they have become overly complicated, necessitating all the intricacies to be explained in extensive reference material which is not always supplied. These traits are problematic for researchers, as excessive information takes time to digest and likely means an unreasonable learning curve, whereas sparse information can just prove too prohibitive.

Surplus functionality and caveats are another potential hindrance sometimes present, meaning extra steps are compulsory to achieve common tasks, which increases required knowledge and detracts from the effort spent on principle objectives. Instead a framework should actively assist the user, automating core processes and providing multiple ways to interact with a solution and accomplish goals, allowing work to be carried out in a manner that individually suits them. Generation of bespoke components should follow a similar pattern, entailing minimal specialised aspects and utilising generic techniques to accelerate development and simplify integration.

4.5.3 Scalability

It is vital for data mining applications to adequately deal with real world and 'big data', which can be a substantial feat for a problem like text categorisation has high dimensionality and sizeable corpora. Typically this means large storage capacity is critical for holding datasets and executing processes, yet nearly all the frameworks examined are limited to using only the main memory of the hardware they run on. A few are currently experimenting with third party utilities, particularly the Hadoop distributed processing framework [212], but generally are unable to handle any data larger than the capacity of main memory and seem to have no intention of doing so without reliance on external tools. Some do have the option to automatically archive data to a mass storage device when no longer needed, though the data being processed and subsequent transformations are exclusively held in their entirety within main memory.

Scalability of the data visualisations is a further issue as two dimensional tables are employed almost unanimously for representations and while these are an intuitive generic format, they are not feasible for conveying large volumes of records. It is also rare to find textual data stored in such a way and it is difficult to analyse raw data given this layout, so enhanced summaries and visualisations should ideally be used instead, but presently only a small number of applications natively provide them.

4.5.4 Customisation

This is a key area where all the evaluated systems tend to fall short, primarily due to the restrictions enforced by the internal data formats and the added complexity from the various integration requirements. Development effort of a custom feature is relative to the constraints imposed and the impact it has on their definition, which can be greatly minimised by automatically handling as many tasks as possible. Modern techniques are able to make this easier, but while the user environment of many frameworks is regularly updated most are still based around a central core of older technology, with some of them built over a decade ago [170][180].

It is imperative when researching new algorithms that creation and use of bespoke components is as straightforward as possible, even if this requires simplified user guides or added complexity in the core functionality. Despite this the frameworks have a varying degree of available resources to explain the necessary construction steps, with some offering very little or only community based material, while a rare few provide official design tools to generate basic code outlines [195]. All of them also rely on 'magic strings' to reference objects within internal data structures, which are inherently prone to runtime errors and negate a potential benefit of any strongly typed programming languages, especially when used in combination with intelligent development environments.

Another limiting factor is the lack of separation between the logic, integration and visualisation layers, which inhibits the separate development of each and makes it challenging for an individual without all of the appropriate skills to produce even small customisations. While the degree of overlap impedes the manufacture of reusable aspects that might be portable to other systems and, to a lesser extent, discourages the integration of third party libraries, which could increase the speed and ease of fabrication.

4.5.5 Modularity

All the frameworks considered have the potential to be highly modular, though the actual granularity is controlled by those who develop the individual components, as is the level of reusability. However many also have a significant amount of enforced overlap between their various layers and do not encourage any distinct separation, which leads to occasional backwards compatibility issues when any single part gets updated [187][211]. This trait also mean that compatibility with other frameworks is not always straightforward, often needing substantial effort to convert between the different formats, though some have managed to gain the functionality associated with selective components [191][167].

The possibility of integrating algorithms containing any level of complexity is available, with no imposed limitations or guidelines recommending what should constitute an individual component. While a definite advantage this means that some attempt to encompass too much and should probably be more atomic, which is a difficult issue to address without introducing design restrictions. An approach that supports complete segregation of algorithms and keeps integration trivial may discourage this behaviour, aiding in the reuse of logic and providing any desired level of granularity or solution architecture.

4.5.6 Flexibility

Conventional data mining is the principal objective for the majority of frameworks, which leads to the use of generic table structures and visualisations that are congruent with most data sources. However these generalisations are actually a hindrance for some problems, including text categorisation as the different information that requires tracking throughout the various stages does not always follow a standard format. For example category assignments, pre-processing metadata, feature selection metrics and similarity weightings all add extra dimensions that are hard to express in a two dimensional structure. This by itself may be the reason that few options presently exist for text processing and why those that do attempt to cover such a broad range of functionality within each component, making them unsuitable for the proper segmentation of text categorisation solutions.

Aside from the constraints of internal data structures there are also some in relation to the accepted format of external data, as some systems enforce the use of specialised datasets [169] that necessitate the creation of bespoke components to perform the appropriate conversion of source material. While never totally avoidable, as most data resources do not follow a single collection of strictly defined rules, this transformation can be laborious and is usually completed via an external application or disposable script because of the effort involved to incorporate it as a new component.

Specifically in regard to text categorisation the existing packages are notably inadequate, with most imposing very linear solutions due to merging processes together or enforcing assumptions about the interaction between stages prior to classification. The complexity and available solutions are also underestimated and confined by restrictions, such as the widespread fixation on the bag-of-words representation, frequency based feature selection and vector space weighting metrics. The text specific frameworks [160] are clearly more suitable for text categorisation than those for general data mining, but suffer from an inability to be customised and are still unable to handle the full range of stages.

Portability is one area that appears to be well supported, with created solutions and sometimes even the processed data having the possibility of being saved and reloaded for use on different hardware platforms, though typically only within the same application. The ability to include saved solutions as sub components of other solutions is also common and in certain situations another framework might be able to deploy them as a special element [191][167].

Ultimately the level of flexibility is related to three principle features, the incorporation of aspects from other sources, the ability to be incorporated by other sources and most importantly the capacity to readily add bespoke components. Unfortunately only a few of the frameworks investigated are currently able to embed or be embedded by others and none are particularly simple or quick to extend with customisations, which is a major obstacle when producing novel multi stage approaches.

4.6 Chapter Summary

Desirable traits for research applications were compiled in this chapter and used to guide a review of the foremost open source data mining frameworks and text processing utilities, with emphasis placed on their ability to handle the recommended categorisation architecture either in part or as a whole. While the majority performed adequately for standard data mining tasks, with many of them providing exceptional visual interfaces, a universal lack of sufficient text related functionality was exposed. This was due to none of the items examined offering more than basic algorithms and tending to merge those available into a single operation, which prevented segmentation of the proposed solution stages by combining multiple types together and imposing certain assumptions.

Support for the open source community seems to be reducing as well, with some frameworks plainly becoming geared toward monetary gain as they grow in notoriety. Although this may increase the general level of maintenance, it is likely to steer future development in a commercially beneficially direction, involving disproportionate enhancement to visual and corporate aspects over expansion of data processing libraries. Ultimately the above limitations in conjunction with the use of specialist languages, bespoke data formats and complicated presentation requirements led to all the considered frameworks being rejected as too cumbersome for the rapid prototyping of algorithms.

Similar to the full frameworks, the text related utilities are of little practical help, with each dedicated to only a small part of the overall categorisation procedure or being excessively restrictive. Typically they are also not designed to be expanded beyond their intended purpose, while several appear to have low development activity, perhaps due to abandonment in favour of larger frameworks. Consequently they are too inflexible for the construction of novel approaches through reuse of atomic components, though they could be useful references during the production of suitable alternatives.

5 New Framework

5.1 Chapter Contributions

Drawing on conclusions made in the preceding chapter the following section outlines a design for a generalised framework that permits the controlled execution of a structured sequence of processes. By enclosing the majority of complexity within core internal components it allows the benefits of rapid prototyping to be combined with the stability and scalability of a full framework, having the potential to reduce some of the problems evident in current applications while eliminating others completely. The processes also are highly customisable and not subject to unnecessary constraints typically imposed by specialised user interfaces or the mechanisms that encapsulate them, intentionally keeping their construction simple to ease development.

A trial implementation of a basic system that realises the conceived designs is also described, which has enough functionality to perform intricate experiments and undertake a comprehensive usability analysis. The characteristics and features of every key component are explained in detail, revealing the architecture employed and reasoning behind certain decisions so they might be easily reproduced and improved upon. Several perceived limitations and possible methods for overcoming them have been summarised as well, noting issues that may arise in particular situations or areas that would require enhancement prior to the framework becoming openly available

5.2 Perceived Benefits

After investigating existing frameworks and consideration of the issues present when employed in a research setting, it is clear that there are several areas that could be significantly improved. Possibly the biggest problem that requires attention from the perspective of novel algorithm research, which is common to practically all frameworks with a graphical interface, is the inability for rapid and easy customisation to accommodate new components. The main cause of this is the manner in which the user environments are tightly coupled with both data structures and algorithm definitions, requiring substantial effort when constructing bespoke elements to make them integrate and behave correctly with the enforced restrictions.

Note that the new framework does not do anything the existing ones are not technically capable of, provided that the modules were originally built to be adequately atomic and flexible enough to be readily customised. Unfortunately the current options lack sufficient modularity and enforce too many assumptions for them to be successfully mapped to the architecture recommended in chapter two, while deficiencies of the reviewed products mean reworking them would be a notably sizeable task. Therefore, the intention of the new framework is to make it possible to quickly develop a suitable text processing package, capable of satisfying the recommended text categorisation architecture, within a fraction of the time and effort normally required.

The proposed design is deliberately built with the creation of potentially complex multipart algorithms in mind, rather than focusing on the user experience and interface, as seems to be the case with most of the present alternatives. Consequently the emphasis is on the simple addition of new components and extensions to the framework, largely ignoring user environments and treating them only as a low priority secondary concern. As a result the end product is streamlined for those wishing to rapidly prototype original components and has very few requirements related to user interactions, with any that are present being entirely optional.

While this might lead to graphical interfaces that are perhaps not as feature rich as some others, it greatly simplifies and shortens the development process, permitting much faster prototyping while retaining the benefits and stability of a complete framework. However, this does not necessarily mean the user experience will be poor, as several events have been purposefully incorporated into the core functionality to allow inclusion of many desirable traits presently available in other products. The main difference is that the intricacies involved in their implementation have been abstracted away from the customisations, preventing unwanted pollution of the algorithms and data transformations. Ironically, this distinct separation of various framework parts also makes the creation of bespoke environments easier.

To increase development speed further most of the validation needed when linking multiple processes together is designed to be handled by the framework itself, rather than relying on the current trend of

manual attribute matching. Explicitly defined objects are employed to achieve this, which are no less flexible that other systems, but can be exploited to automatically determine if two links are compatible without requiring additional input. They also make storage of composite data structures much simpler, removing the need to convert them into two-dimensional table structures and generate complex data cubes from relation mappings. In turn this eliminates the confusing APIs associated with data access and manipulations and, in combination with the relocation of user interface aspects, greatly reduces the possibility of producing invalid components.

Although the framework design is highly generic and applicable to any processes, the initial reason for it is to facilitate the construction of a novel text categorisation package shaped around the architecture proposed in chapter two. Based on this, nonlinear interactions, atomic components and the ability to effortlessly substitute or insert modules were all key factors during the planning stages and form an inherent part of the core functionality. Though these are not unique traits, they give the flexibility that is essential to deal with the task at hand, something that is theoretically already available but so far ineffectively implemented and overly cumbersome to resolve using the current frameworks.

In regard to the proof of concept implementation, the design is not intended to be dependent on any particular programming language, but the decision to use one based on the .Net Common Language Runtime (CLR) was deliberate. This is because it offers a high level of compatibility with several other prominent languages, including those commonly used by the existing frameworks, and also provides a number of beneficial features that are fully explained later in this chapter.

5.3 Framework Design

To overcome the problem of complex interconnections between the various framework aspects the proposed design consists of distinct layers, following the computer science 'separation of concerns' principle. This involves breaking down the application into loosely coupled individual sections that only interact through a set of strictly defined contracts, allowing each to be implemented and maintained separately without the internal processes of one affecting the others. As such three layers have been determined; core functions, algorithm libraries and user interface, with each having a strict one-way interaction with only a single other layer and all containing dynamic aspects to permit modification and addition.

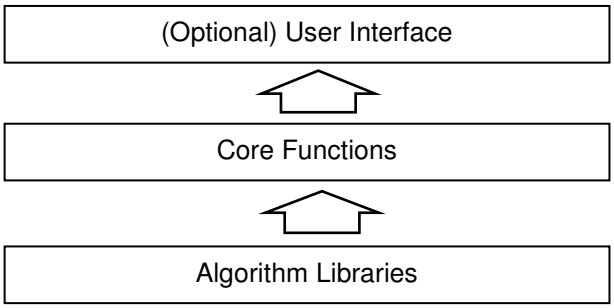


Figure 5.1 : Separation of Concerns - Framework Layers

The algorithm libraries contain the elements that perform the actual text processing and are deliberately kept independent from the other layers, putting focus on the creation of algorithms and not incorporation of the framework. Encapsulating the libraries are the core functions, which handle the flow of processes, error exceptions and data storage, whilst also providing simple mechanisms to dynamically load customisations and convert data into any required formats. Finally an optional user interface can be applied on top of the core layer to provide a tailored environment and user experience, with the one-way relationships and defined communication contracts preventing it from influencing the algorithm libraries or core functions.

5.3.1 Algorithm Libraries

Many available text processing utilities cannot really be considered complete frameworks as they are merely a library of discrete algorithms without any mechanisms linking them together, however some do provide interaction by means of sophisticated console or graphical based environments. Unfortunately these more involved utilities tend to contaminate the algorithms they define by merging them with functionality from the containing environment, meaning they become application specific, which increases complexity and makes reuse in other situations more difficult.

In response to these problems the proposed design keeps the algorithm libraries completely separate from all other parts of the framework, making them easier to implement and highly portable by including only the logic and parameters explicitly associated to the actual process taking place. Links to the libraries are also minimised, with direct references being restricted to only a small part of the core functions and prevented completely from the user interface layer. This makes the creation and integration of new and existing libraries as straightforward as possible, with the additional benefit of reducing compatibility issues when importing or exporting algorithms implemented in different programming languages.

5.3.2 Core Functions

The proposed framework is a very generic process ordering and execution system, which has been deliberately designed without any specific text document or data mining applications in mind. This is intentionally done to keep it as flexible as possible, avoiding any adverse restrictions to particular data formats or processing requirements that may inhibit some algorithms. Each core function has also been designed to allow the inclusion of bespoke components with only the most basic and minimal framework prerequisites, keeping customisation development as rapid and straightforward as possible. In order to make this achievable some parts of the core layer involve fairly intricate concepts that automatically handle functionality which must usually be incorporated with the custom components of other frameworks. While this makes the framework itself more complicated it simplifies many user defined areas, including data validation, exception handling, reporting of state and dealing with weakly typed data structures.

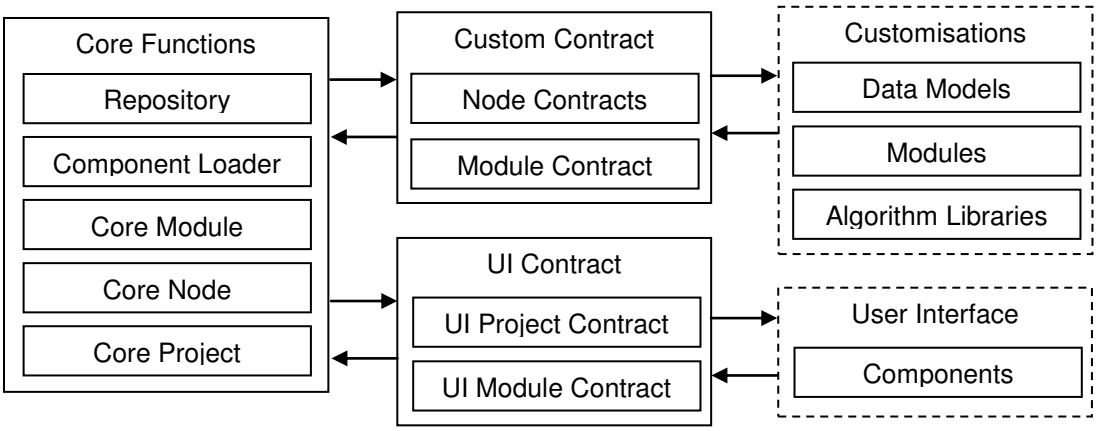


Figure 5.2 : Core Functions and Contracts

Further simplifying the creation of components are a selection of contracts that only allow access to the methods intended for producing customisations, effectively hiding and abstracting the complexity inherent to the remaining core functions. Different contracts also exist that are specific to the types of customisations that take place, such as the creation of a processing module and related data

structures or the implementation of a user interface to provide enhanced interaction. Supplying these contracts and ensuring they are dedicated to particular aspects streamlines addition and modification of the framework, reducing uncertainty about any necessary requirements and what methods are available to achieve them.

5.3.3 Repository

One of the prevalent issues encountered with many data mining tasks is the ability to handle large amounts of data, especially when sizable datasets or 'big data' are being processed and more so when the output from multiple stages need to be stored simultaneously. Many existing applications have trouble dealing with this as they are constrained to holding information only within the volatile runtime memory of the hardware they are executing on, limiting their practical usefulness and capacity to cope with many real-world tasks. To overcome this problem one feature within the core functions is a data repository, which is a mechanism for the storage and retrieval of generic data by means of a single contract that can be implemented in several ways, permitting access to different storage devices and locations.

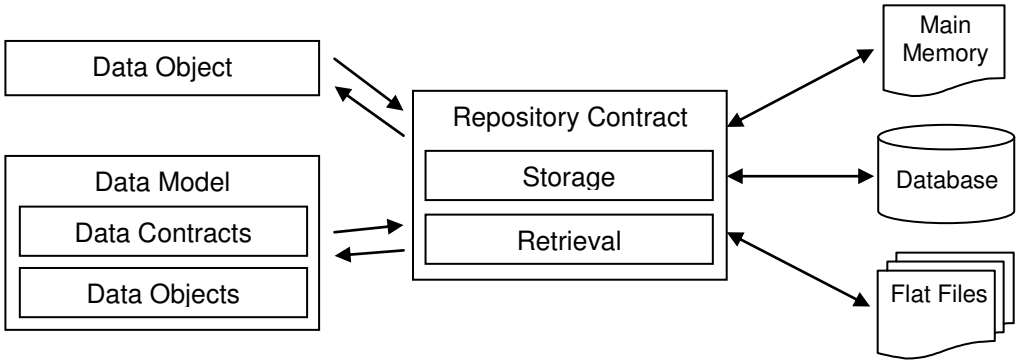


Figure 5.3 : Core Repository Overview

The standard entry point to get or set items from the repository always obeys the defined contract, though the actual implementation of the inner workings can be altered to utilise solitary or multiple different storage devices. Using this technique affords the framework great flexibility and the potential to exploit any combination of local or remote caches, providing unbounded expansion in the event that extremely large volumes of data are being investigated.

In order to attain the benefits offered by the repository all internal core features should be designed to automatically employ it wherever possible, without the need for any explicit configuration. Furthermore to maximise the potential for system scalability any custom components should also make use of the functionality, particularly if they involve long term storage of information or otherwise manage an excessive amount of data.

5.3.4 Component Loader

While easy creation of custom components is an essential part of the framework, the capacity to integrate those components in a straightforward manner is also important and this is the purpose of the component loader. This feature is able to load and initialise custom components at runtime in a simple and efficient manner, readying them for immediate use with minimal configuration or user input.

Conflicts that might arise when attempting to load an item multiple times are avoided by keeping track of previously loaded components, with a separate mechanism employed to handle references to any elements no longer available due to unanticipated removal. Additionally the loader is flexibility enough to deal with a variety of generic items and not just those defined by core contracts, as though customisations must adhere to a contract to permit framework integration, it is essential to load any associated libraries and dependencies as well.

5.3.5 Data Models

Data models are associated within a number of customised components and permit the storage and transport of data between the various processes taking place, they are also fully customisable to allow the implementation of any required data structure or format. Use of the repository when creating a data model can also increase the flexibility and scalability it is able to provide, making the storage and retrieval of significantly large volumes of data more feasible.

Compatibility of links between solution elements is controlled entirely by data models and is a crucial aspect of their design, as a connection between two processes can only be formed if they share a common model or contract. By ensuring an exact match between the strictly defined model or a supported contract of a process output and the desired input of all the linked processes, solution validity is guaranteed during initial design and disruptive unexpected errors at runtime are prevented.

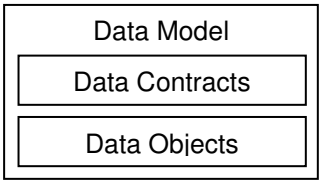


Figure 5.4 : Data Model Composition

5.3.6 Core Nodes

Customised modules can only send or retrieve information via the nodes attached to them, with each having an embedded data model that dictates which other nodes within a solution are compatible and therefore available to form links with. There are two main types of node available; output nodes, which form the start of a link and supply the outgoing data of a module, and input nodes, which form the end of a link and accept the incoming data of a module. An optional input node is also possible and is an extension to the basic input node, with all of the same core attributes but without the mandatory requirement of having associated connections.

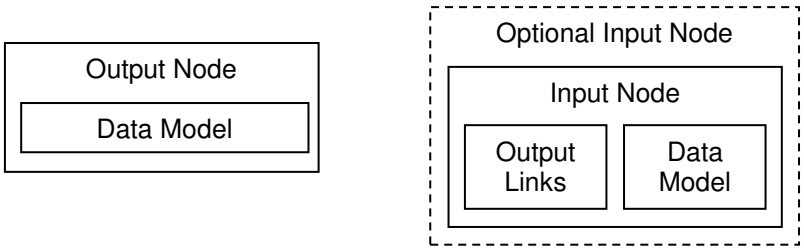


Figure 5.5 : Output and Input Node Composition

Output nodes are not aware of any links they might be associated with, merely broadcasting any change of condition, including when data becomes available, and supplying a copy of their data to anything that requests it. The actual output data is determined and set by the custom module that the node is attached to and for every request an exact copy is created and supplied, leaving the original in its initial state without having to repeat the module processing. It is important to use clones as the data may be passed to multiple distinct components or used for comparisons and visualisations, which could lead to unexpected results if anything already given the data makes alterations.

By contrast Input nodes are acutely aware of every link they are associated with, keeping a record of all output nodes they need to perform a request to and waiting for each to broadcast when their data is available. Once every connected output signals that their data is ready the input node is able to collect and assemble it into a single entity, preparing it for use by the custom module it is attached to. Optional input nodes work in an identical manner to standard input nodes but do not explicitly require connection to any outputs, however if a link is present the same mechanism of listening for available data and then collecting and preparing it for the module still takes place.

As links can only be created between nodes that have compatible embedded data models an output model or one of its contracts must match the connected input model, however an input model that is a list with elements equivalent to the output model is also considered compatible. In this situation when an input or optional input node retrieves data from its connections it will automatically compile it into an appropriate format, concatenating solitary items or existing item lists into a single structure. This is

done internally by the framework to make the linking of components easier and more intuitive, whilst also negating the additional steps and complex components typically required by other frameworks to achieve the same result.

5.3.7 Core Modules

Being the principal means to incorporate customised components into the framework, the core modules encapsulate data mappings that serve to initialise the embedded data models of their attached input and output nodes by employing the defined algorithm libraries. Each module is effectively a wrapper that automatically handles several fundamental tasks without the need for external input, providing all of the functionality required to fully integrate custom elements into the framework. They are comprised of one or more input nodes that receive incoming data from previous modules, a central data mapping component that converts and processes the retrieved information and one or more output nodes that store the modified data and supply a clone of it when requested.

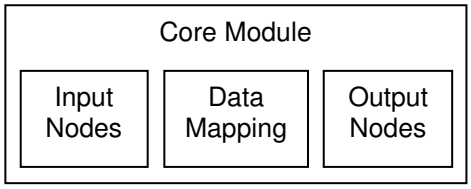


Figure 5.6 : Core Module Composition

All of the attached nodes can be accessed directly by the data mapping element encapsulated within a core module, with each input node automatically preparing the information it collects ready for processing and each output node waiting to be initialised once all processing is complete. The central mapping component transforms the prepared input data to a format suitable for the algorithms being employed, applies the algorithms and then transforms the results into the format required by the outputs and then pushes the data to them. Most of this is done internally as part of the framework and the only aspects that need to be defined are the specific input and output nodes attached to the core module, the data transformation mappings and the algorithms, if a library containing them is not readily available.

5.3.8 Core Projects

In essence a core project is a container that holds all of the core modules that make up a particular solution and is responsible for the execution of the individual components, in addition to tracking the status of all data as it is processed and transferred between them. It also provides a central location

where the majority of possible actions can be applied, including parameter configuration, adding or removing modules, making or breaking node links and starting or stopping the execution of a process. Aside from this they also fulfil the requirements to be treated as a core module, regarding any unlinked nodes as possible connections, meaning that individual projects can in turn be included as components inside other projects.

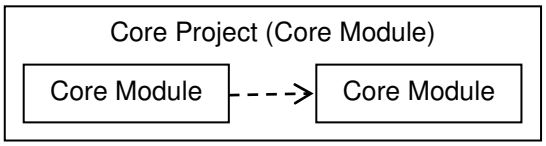


Figure 5.7 : Core Project Composition

Each solution can also be saved in a standardised and portable format, allowing any particular project to be easily shared and repeatedly reused, deployed on multiple systems simultaneously or integrated into other solutions. When saved only the details essential for reproduction of the solution structure are retained, such as the components, links and configured parameters, while any data that may have already been processed is removed. Not only does this make the information more compact and convenient, especially when dealing with large volumes of data, but it also removes the attachment to specific datasets and tries to encourage the use of open and publicly available resources.

5.3.9 User Interface (GUI)

With regards to research applications an extensive user interface can be beneficial, but is not necessary and should be a secondary consideration to achieving the principal research aims. As a result the framework layer that includes this functionality is intentionally kept separate from the core utilities and algorithm libraries, helping them to stay as simple and rapid to implement as possible. In order to interact with these only the UI contracts should be used as they provide access to the elements suitable for creation of any user environment, whilst concealing the superfluous ones.

The set of contracts allows interaction with core projects and by extension the production, manipulation and execution of the components they contain, along with the ability to independently query and retrieve status updates for all of them. This gives direct management of all user controlled aspects within the framework, including those related to modules, attached nodes, embedded data models and any links that can be formed between them.

By providing a broad scope of functionality and preventing intricacies of the user interface from influencing the other layers, any type of desired environment or user experience can be constructed, including those with a graphical foundation. However their production is more involved than most

other frameworks as default interactions must be defined, along with a mechanism for configuring any custom aspects, as the simplistic nature of components means they do not define their own interactions. Though potentially increasing user interface complexity this naivety also affords considerable flexibility, allowing fully tailored environments and easing construction of other components by not imposing rigid standards that must be followed.

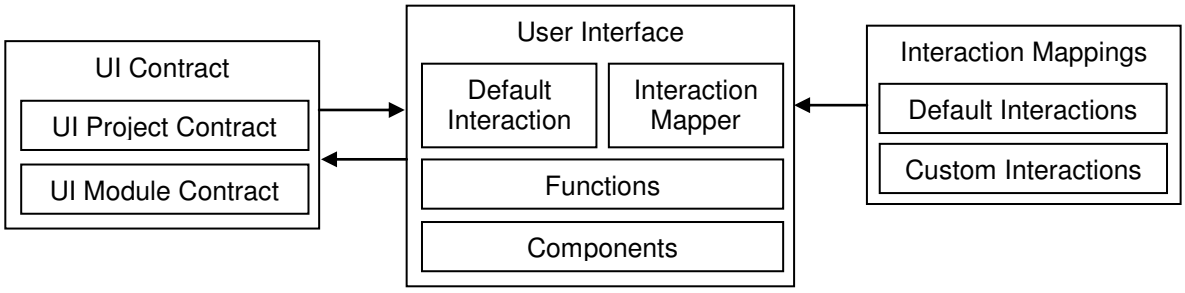


Figure 5.8 : Possible User Interface Overview

One possibility for providing custom components with their own unique interactions would be to provide a mechanism that maps distinct identifying characteristics of each component, for example its name or type, to a specific set of custom interactions. This would make it possible for each component to use the default interactions or have a dedicated alternative mapping that is interpreted in an appropriate manner by the user interface employed. Exactly the same approach can also be applied in a graphical environment, with a generic display being used that either refers to the default or component specific set of representations.

5.4 Framework Implementation

An implementation of the framework has been created based on the designs outlined, as both a proof of concept and to gather empirical results for investigating the theory of separating categorisation solutions into distinct stages. The general orientation of existing frameworks towards user interfaces makes them unsuitable for this type of study, with the imposed restrictions meaning significant resources are required for the expansion and development of new components.

5.4.1 Programming Language

For the core of the framework C# .Net version 4.0 programming language has been used, as it offers multiple advantages over the other languages commonly employed for this type of situation, such as Java and Python. It is a high-level programming language designed for enterprise applications and comprises industrial standards that are internationally recognised and backed by a large global corporation, providing a comprehensive support network and quickly resolving issues. For several years it has also been widely accessible for general use through the provision of free development environments and is easily obtainable from public sources, including installation packages from the Internet.

5.4.1.1 Compatibility

C# is part of the 'Microsoft Open Specification Promise' [213] and is designed to comply with the Common Language Infrastructure (CLI), which forms the core of the Microsoft .Net Framework and allows execution of multiple languages on different computing architectures and platforms. This means any other languages using the Common Language Infrastructure, or that can compile into the Common Intermediate Language (CIL), are directly compatible and can run on any system capable of deploying the Common Language Runtime (CLR).

In practical terms this gives the language a highly accessible licensing structure and means that, despite the core framework being written in C#, custom components can be written in a extensive collection of different languages, including VB.Net, J#, IronPython, IronRuby and many others [214]. Through the use of applications like IKVM.NET [215] it is also possible to use any languages based on the Java Virtual Machine (JVM) within the .Net Framework, including Java, Scala, and Groovy [216], further expanding development options. Additionally CS-Script [217] makes it possible to use C# in a manner similar to a scripting language, removing the need for explicit compilation and increasing flexibility.

Several open licence initiatives, for example Mono [218] and Portable.Net [219], also exist and are aimed at cross-platform compatibility, permitting the execution of any applications based on the Common Language Infrastructure and .Net Framework. These extend the range of operating systems and platforms capable of running CLI based applications, encompassing the majority of those commonly used throughout the world, including Microsoft Windows, Linux distributions, Mac OS X, Solaris, BSD, Android and most gaming consoles.

5.4.1.2 Similarity and Benefits

One of the advantages of high-level languages is that the vocabulary, structure and syntax are typically simple to understand and quick to learn, especially by comparison to low-level or specialist languages. Many also share a number of similarities and possess helpful attributes that make the basics relatively easy to pick up, even for those with little or no previous programming experience. C# is an excellent example with many common features present in other popular languages like Java, VB and C++, making the transition between any of them a moderately straightforward process. It also has a variety of beneficial aspects, such as automatic resource management, comprehensive development environments and extensive documentation, which are freely available as official or informal Application Programming Interface (API) specifications, guidelines and tutorials.

5.4.1.3 Architecture and Features

Like many popular languages C# is object-orientated and employs several fundamental concepts, including dynamic dispatch, encapsulation and polymorphism, which make it ideally suited for implementation of the designed framework. The various features provide the means to achieve effective segregation and abstraction of the core functions through the use of specialised contracts, whilst allowing addition of custom components by encompassing them within elements that manage their integration. It is also a strongly typed language, which actively speeds up development and reduces the potential for disruptive runtime errors by restricting invalid operations and assisting in the production of valid processes.

Only a single class inheritance hierarchy is possible, which prevents the issues referred to by the “diamond problem”, where ambiguity can arise if multiple entities are inherited and each override the same method of a common base entity in a different way. Multiple interface inheritance is allowed though, with an interface representing a contract that merely outlines the possible interactions of an entity without explicitly defining any actual functionality, permitting the use of composition over inheritance. This increases flexibility by removing the need for tight coupling between objects, so custom components can be substituted during runtime for any component that shares a common interface and can be used by any entity that employs that interface.

The notions of covariance and contravariance are also fully supported, further improving the ability to implement all practical aspects of the framework, including the use of controlled contracts to aid production of different forms of customisation. Covariance refers to the conversion of a specialised type to a more generalised type, for example the transformation of a distinct custom module into a module contract that is compatible with the framework. In contrast contravariance refers to the conversion of a general type into a more specialised type, for example the transformation of a node contract into a strictly defined input or output node.

Another benefit is the capacity for sophisticated generic programming, dynamic types and reflection, which all relate to the generation of common code to perform various operations on different types at runtime. When correctly applied they substitute a previously unknown at compile time for a valid entity during execution, which can then be modified in a predetermined manner or manipulated via predictable interactions to complete a set of processes. This functionality is fundamental to many of the core elements in the framework and is essential for the removal of complexities that would otherwise prevent simplification of custom components.

5.4.2 Data Repository

A basic interface is specified that provides simple repository methods to insert and retrieve any form of object, defining both nonspecific and generic parameters to permit a variety of type casting techniques. The actual operations that implement the interface use two levels of storage; a volatile short term cache, which temporarily puts data in the main memory of the system executing the application, and a long term store, which is held on a mass storage device. Using two different locations can improve data access as the cache offers fast retrieval of items being processed, while the long term storage allows expulsion of redundant objects from main memory and is able to handle much larger volumes of information.

Each time an object is sent to the repository it is put in both storage locations, the cache because of an assumption that it is likely to be accessed again soon and long term storage for stability in case it is expelled from main memory before it is needed. When an object is requested it is fetched primarily from the local cache or, if not found, from the long term storage, at which point it is replaced in the cache under the same assumption that repeated access is likely. Though in certain scenarios this redundancy may prove unnecessary, for example if all data fits within the main memory, it will generally increase flexibility and can shorten execution time. It is also a trivial task to disable or change either of the storage mechanisms or locations if desired, with the interface definition ensuring that no other components will be adversely affected.

All internal framework functions that potentially handle large volumes of data utilise the repository automatically, however it is also possible for custom components to employ it, though they must ensure that all long term storage is deleted when no longer required. Extensions to the standard interface and creation of specialised structures could abstract much of the inherent complexity, but despite data models being straightforward to upgrade it is not possible to enforce use of the repository.

5.4.3 Component Loading

Loading of custom components has been split into two distinct parts, the first of which is responsible for loading the actual assemblies that contain the compiled components, such as custom modules, data models and algorithm libraries. Currently only precompiled Dynamic-Link Libraries (DLL) are automatically discovered and loaded, but through the use of the many extensions available for the .NET framework it is possible to expand this to include numerous other formats. An interface is also defined for this feature and is incorporated within the publicly accessible UI contracts, giving any user environment the ability to dynamically search for and load elements at runtime.

Closely related to the loading of assemblies is the discovery and tracking of customised modules, along with the facility to automatically initialise them with all attached input and output nodes. Two factories have been created to handle this, which generate new instances of core modules and nodes as requested, with the module factory also scanning for and recording the various types obtainable from the currently loaded assemblies. When a core module is added to a solution the module factory is invoked with an identifying attribute of a custom component, for example the compiled class name, which it uses to verify the relevant type and creates a new instance. The returned object is then embedded into a new core module, which in turn extracts details from it via reflection and employs the node factory to initialise all the attached nodes.

As core modules and nodes are created they are assigned a unique identification label, which is immutable and only retained for the lifespan of that particular object, meaning it is not kept if the object is recreated for any reason. This is done to allow inclusion of several instances of the same module type within a single project and also so that a particular project can be deployed within another project multiple times. The main purpose of the identifiers is to aid user environments, which can exploit them to interact with projects and distinguish between components without the need to hold explicit object references in memory. Additionally they may also prove beneficial in parallel or grid computing applications, where identification of components and processes stored on physically remote systems is an essential feature.

5.4.4 Custom Data Models

Data models can be comprised of full objects, a set of interfaces or a mix of both and node compatibility is based on assignment using the principles of contravariance and subtype polymorphism instead of only exact matches. This makes it possible to link output nodes with embedded data models that inherit one or more elements to any input node that has one of those inherited elements embedded. Aside from the limitations present in the programming language no other restrictions are enforced, so objects with any feasible structure, format or inheritance chaining are possible. While this provides maximum flexibility care must be taken to prevent complicated node definitions that may unnecessarily reduce the compatibility of some custom components or conversely permit invalid connections.

The framework automatically detects suitable compatibility when connections are attempted, by checking for exact type matches or valid subtype assignments, and will not allow creation of invalid links. This means errors are discovered immediately during solution design and not at execution, where they may cause the failure of a process or unexpected results without any warning or notification. Type verification is also performed in the same manner when individual items or groups are automatically merged into a single list of entities, keeping the process of connecting data models as easy as possible.

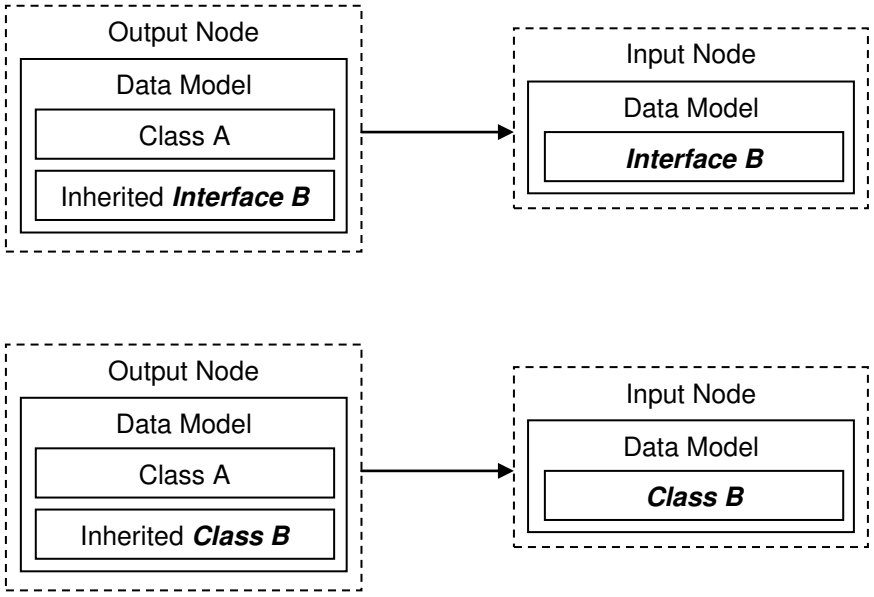


Figure 5.9 : Node Compatibility using Contravariance and Subtype Polymorphism

Creation of custom data models is very simple as they are merely containers that hold information, requiring only that an appropriate structure is defined and consequently involving little or no logic. An existing object, native value or bespoke item is viable to use, though inheritance hierarchies should be considered for effective node compatibility and elements would need public read and write access to

be fit for purpose. If models become particularly complex or the repository features are adopted, the incorporation of an enhanced cloning mechanism might also be essential in order for the framework to properly handle them.

5.4.5 Custom Nodes

Generic programming and reflection is heavily employed for the creation and manipulation of nodes, leading to processes that are reusable regardless of the embedded data model types. Every instance of a node also stores the name of the module field it is associated with and a unique identifying label, which is advantageous when designing user environments as it gives basic designation and multiple ways to distinguish between them.

All data sent to an output node is stored in the repository and is only removed when the node is disposed, however each copy made is held directly in main memory but is deleted automatically by the system when it goes out of scope. Data cloning is performed by the 'DataContractSerializer' class, which functions similar to standard XML serialisation but natively handles advanced structures like Interfaces or Dictionaries and is capable of producing deep copies of objects, though requires explicit metadata in certain situations.

Conversely input nodes do not use the repository and instead gather copies of the data from all linked output nodes and hold it directly in main memory, combining the difference sources into a single entity where necessary. Each input maintains a list of references to all the outputs it needs to query for data and is also responsible for checking data model compatibility whenever a new connection is attempted.

5.4.6 Custom Modules

A custom module refers to a single component defining validation logic, data processing, editable parameters, input nodes and output nodes, which is encapsulated by a core module for the purpose of framework integration. The wrapper is used to extract parameters, nodes and the associated field names at runtime via reflection, in order to make them available to projects and any implemented user environment. It is also provides initialisation and disposal of nodes, exception handling and calls to the methods that validate the information and execute the data processing.

From a research perspective the most important attribute of a framework is the integration of bespoke algorithms, so there is an emphasis on simplicity and rapid creation of custom components with minimal effort and programming experience. A secondary focus is also placed on keeping the algorithm details separate from elements of the framework for maximum reusability, though this is

ultimately the responsibility of the component designer and is not enforceable. Where possible the available programming tools and concepts have been utilised to integrate as much functionality within the framework itself, producing a high level of embedded complexity but also making the addition of new modules notably easy and flexible.

5.4.6.1 Imports and Inheritance

To assist in the development of custom modules just a single namespace needs to be imported, 'Core.Modules', which includes a small number of basic interfaces defining the required methods and components. In order for a class to be treated as a module within the framework, all it has to do is inherit the 'Module' interface and implement its two method signatures; Run() and Validate().

5.4.6.2 Attaching Nodes

Attaching a node merely involves declaring an instance of the specific node interface type, 'InputNode<DM>', 'OptionalInputNode<DM>' or 'OutputNode<DM>', where DM represents the type of data model embedded in the node. To be acknowledged the declaration must be a public and does not need to be initialised with a value, as this is done automatically by the framework as required.

5.4.6.3 Adding Parameters

If parameter values are needed for successful execution of the module they should just be added as a public field or read/write property within the class, using the appropriate system type and initialised with a suitable default value.

5.4.6.4 Validation

Just before a module is executed the validate method is called to ensure that input nodes reference appropriate data and that publically accessible parameters are assigned suitable values. Strong typing of data models guarantees that input data will always be compatible, but it is the responsibility of this method to ensure the legitimacy of any optional input node combinations, the magnitude of data and all associated parameter values. A value of true should be returned if everything is valid or false otherwise, with a possible future addition being a mechanism to report specific issues for the purpose of aiding users and error logging.

5.4.6.5 Execution

If a node successfully validates it is executed using the run method, which is in charge of processing the data supplied by the input nodes and then pushing the results to the output nodes. During development interactions with the nodes is done via the basic methods defined in the relevant interfaces, which provide the ability to check for and retrieve data from inputs and send data to outputs. Ideally the implementation of this method should be light weight and only convert data between the formats needed by the various data models and algorithms employed, retaining the majority of logic in a separate algorithm library.

5.4.7 Core Projects

In essence a project is just another wrapper that maintains a list of modules added to a solution, providing a number of actions for the manipulation of these modules and any attached nodes, but not the data they process. For flexibility multiple interfaces are defined that denote the possible interactions that can be performed on output nodes, input nodes and core modules, with every method using these generalised contracts in place of specific objects. Every project also satisfies the interface related to core modules, allowing them to be added to each other and directly substituted for modules wherever they are referenced.

When a module is inserted into a solution the project determines the actual object type, adding other projects directly or employing the module factory to attempt initialisation of an appropriate object and adding it if successful. Interaction with the solution components can be achieved by utilising either the unique identification labels, a mixture of module labels and node field names or actual object references, making it simple to work in either a static or dynamic environment.

Another major function handled by each project is the ability to embody itself in a suitable format for saving to, and conversely loading from, a mass storage device, though this functionality does not necessarily need to be embedded as part of the actual project. This is accomplished by the framework with selective component JSON serialisation and de-serialisation, giving a compact but understandable file representation that records details of the modules, nodes, and connections.

5.4.8 Core Status Events

Several parts of the framework trigger events whenever a significant change of state occurs and any element of the core functions or a user environment can subscribe to these, becoming instantly aware when an update takes place. This is the primary mechanism used to automatically manage and ensure valid ordering during execution of the various disjoint components, but also provides

information crucial for dynamically reporting solution state. Most aspects can further have their condition manually examined via defined interface methods, which reduces dependency on these events and permits several possible user interface architectures.

Internally the framework represents the state of key elements through the use of two basic constructs comprised of binary flags, with one present in each core module or project and the other embedded in the different types of node. These are maintained by a central process within each element that runs every time there is a potential change in state and are employed by a selection of methods that convey their information in a simplified format. Anything subscribed to an event is passed the particular item that triggered it and can access these methods to extract any necessary details, either by means of a single generic function or through delegation to any number of specialised operations.

5.4.8.1 Output Node Events

Output nodes can trigger two events, the first related to status and fires when they become enabled or disabled and have data added or removed, while the second fires to request disconnection from all the links they are part of, which is required as they do not store any link related information. The object passed to subscribers allows them to check if it is enabled, presently has available data or is ready for use by connected items and also to determine identifying characteristics.

5.4.8.2 Input Node Events

A status event is triggered by input nodes when they are enabled, disabled, receive their first connection, lose their last connection or if any items they are linked to report a change in state that affects it. The object passed to subscribers permits them to check if it is enabled, optional, connected, has all required data or is ready for use and also to determine identifying characteristics or examine the links it is currently part of. Input nodes are considered ready if they are enabled with at least one connection and if all of their connections are in a ready state, additionally optional inputs are also considered ready if they are disabled or have no connections.

5.4.8.3 Core Module and Project Events

As projects can be treated as a direct substitution for core modules they trigger identical events, under the same circumstances and with an object that implements the same set of accessible attributes. Status related events are fired and occur when they are enabled, disabled, begin execution, finish execution, clear output data, disconnect from other solution items or if an attached node reports a change in state that affects them. The object passed can be used check a variety of

different attributes, such as the validity of parameters, presence of input data or current state, and also to get identifying characteristics or perform modifications, for instance resetting nodes, disconnecting links or executing processes.

5.5 Expected Limitations

The current implementation of the proposed framework is a proof of concept rather than a finalised product and consequently there are several limitations that need to be addressed before it would be practical for widespread use. Though some of the issues are present in other existing frameworks a number of them are not, but all are possible to overcome with a variety of advanced programming techniques. Also despite these restrictions it is still capable of providing a sufficient platform for testing text categorisation solutions constructed from the independent stages and demonstrating how each affects the results gathered.

5.5.1 Cloning Output Node Data

Copying the data of an output node every time it is requested increases overall storage requirements of the framework, though it means non linear projects can be generated without unexpected side effects if modules directly modify their input data. This is less of an issue due to the ability of the repository to hold data outside of main memory on either local or remote storage devices, but the problem could be further reduced by clearing outputs once all linked inputs finish their requests. However this would also necessitate that the entire pipeline be rerun if modules are reset or have new links established and it limits the information that could be viewed, so the option for manual or automatic purging is preferable.

A bottleneck might be formed by the cloning mechanism as well, especially in linear pipelines, though in general it is expected to involve less computation that rerunning the data transformations and removes any constraints on how modules must process their inputs. The DataContractSerializer presently used may also prove complicated, as while it natively supports many data structures and permits deep copies metadata is sometimes necessary for certain elements to be recognised. Changing to basic XML serialisation negates this issue but restricts the composition of data models due to the limited structures it can handle, which actually includes all of those currently needing the additional information. Other alternatives are possible too, though most existing techniques still require code annotations and reliable bespoke methods are hard to achieve, while enforcing data models to include a specific cloning function adds undesirable development complexity.

5.5.2 Strongly Typed Data Models

Strongly typed data models are beneficial as they allow compatibility determination between input and output nodes, requiring no validation and less code than the weak column name references conventionally used in the generic table structures of most other frameworks. They also simplify the design and implementation of custom components, can be multidimensional and remove the ambiguity encountered when relying on ‘magic strings’ for accessing data. However they must be

entirely defined at compile time and all output and input node specifications need match existing interfaces or objects, with no capacity for individual elements to specify their own explicit prerequisites or constructs.

The advantages compensate for the drawback, though extra consideration of data model composition is necessary when producing modules so that legitimate connections can be formed. Due to principles of contravariance and covariance there are two main approaches that can be employed; the use of a complex all encompassing object that inherits every node type or multiple basic objects that each satisfy a particular format. Either way still has to be defined at compile time, but the first allows input objects to be directly amended and posted to the outputs, though at a cost of increased storage if modules do not delete unwanted data because of the cloning process. Conversely while the second is inherently more space efficient it may potentially require greater computation, involving the creation of new objects and the separate mapping of any information being retained.

Ideally the framework should automatically create data model objects based on input and output node types, thereby allowing custom code to only deal with modules and not have to worry about mapping data or explicitly stipulating appropriate objects. However this would potentially require the run time specification and initialisation of a previously undefined object based on an interface, which is a difficult and advanced programming task. Consequently both the options mentioned are presently available when producing modules, but the definition of data models must be done manually.

5.5.3 Data Model Storage

Use of the repository within a data model can boost its flexibility and scalability, but raises complexity and may lead to problems when an output node attempts to replicate the data. This is due to the cloning utility having to fetch the information from the repository and then re-add an exact copy in a different location, while keeping the references correctly aligned. An alternative to overcome this is enforcing data models to adhere to a contract that defines a self cloning function, though this elevates the development intricacy.

The ability to hold data on a long term storage device transfers much of the load from main memory and when suitable mechanisms for cache expiration or weak references are employed this load can be further reduced. However holding cloned objects may still prove a limiting factor when processing larger datasets and the only effective way to handle this is to provide items that optimise how they access the repository. This would involve just storing copies of information that will be altered and the efficient removal of superfluous data, but is likely to add too many complications to the framework and custom components.

5.5.4 Remote Updates

Presently the Component Loader relies on the items to be available on a local or network accessible storage device and requires them to be acquired manually before they can be loaded into the system. Larger systems solve this issue through the use of interfaces built into the framework that connect to centralised repositories and allow the searching, download and installation of remote plug-ins or add-ons in a semi automated manner. This is a considerable undertaking that entails publicly hosted servers, quality control and moderation of the components being hosted, which all necessitate that the framework be well established before they are feasible.

The Component Loader also currently stores all loaded assemblies in the same Application Domain, which is the execution environment, so it is not possible to load objects with the same assembly and class signatures and therefore custom components must be appropriately named. Once loaded this also makes it impossible to unload the assemblies without restarting the application, which might be irritating if a new or updated plug-in becomes available that uses the same signature as a previous one, especially if both are required in the same session. Advanced programming techniques could resolve this by employing different domains to encapsulate the assemblies that contain extensions.

5.5.5 Generic Modules

Compile time integration of algorithm libraries makes dynamic discovery and selection between similar types difficult when done within a generic module, which is one that allows a single algorithm to be chosen from a collection that all perform a related function. Regardless of the specific process taking place the entire group of associated elements must be determined and suitably handled after being loaded, despite the framework and encompassing components potentially having no prior knowledge of them.

The simplest way to prevent this issue is to bypass it completely by defining individual modules for every algorithm, which is how most existing frameworks deal with it but can also lead to an excessive number of components that crowd the user interface and become awkward to find. Another solution is the implementation of a mechanism to automatically identify similar algorithms, though this would require each group to share a common unique feature and introduce development restrictions solely for the purpose of the framework. The base class that forms the foundation of the factory method for initialising modules is actually already capable of this procedure, but the additional constraints have deliberately not been enforced.

5.5.6 Selective Repetition

Many stages of a text categorisation solution include aspects that entail multiple repetitions of a process, each with slightly altered inputs based on a feedback mechanism to achieve either parameter optimisation or a predefined set of criteria. These include elements such as cross validation for optimisation and both the primary evaluation metrics of break-even point and 11 point precision, all of which are difficult to accomplish in a transparent and modular way using the present implementation. A few existing frameworks are able to manage some of these features, for instance most have a dedicated cross validation component or can utilise meta classifiers to perform parameter tuning, though support is limited while the procedures seem crude and unintuitive.

Producing a complete copy of every module per iteration is the easiest and most flexible approach to cope with this, but while retaining a full set of results and permitting a global overview of the entire procedure this also requires greater memory consumption an may restrict dataset size. A more efficient way would be to keep the same modules and just reset them after each cycle, whist tracking the tested parameters and best result attained. This should generate less data and necessitate lower storage needs, though could prove challenging to develop when multiple elements are simultaneously optimised as the localised information might not allow full exploration of possible combinations.

5.6 Chapter Summary

The design and implementation of a new framework has been outlined in this chapter, which is based on previous findings and contains elements of both rapid prototyping and conventional frameworks. On its own rapid prototyping quickly generates disposable systems that are seldom recycled or made available for others to utilise, making precise duplication of results difficult. In contrast traditional frameworks tend be widely obtainable and replicate output exactly, but have limited functionality and regularly impose development constraints due to component integration mechanisms. By managing to amalgamate these concepts in a certain way the new framework exploits their strengths and reduces the weaknesses, whilst encouraging separation of algorithm definitions and features that use them.

Explicit implementation details are deliberately omitted from the basic design to avoid association to any specific programming language or environment. It is also not restricted to a particular research field or type of data processing, with this being controlled solely by the choice of components included on a per application basis. Components take the form of modules and data models, which process and store data respectively, they are fast to produce and have simplistic integration requirements that make them easy to reuse in other systems. Due to this flexibility many of the existing issues are negated, while all the potential benefits remain viable, though at a cost of added complexity to the core architecture and bespoke visual interfaces.

A trial implementation of the framework has been created as a proof of concept and for use in a number of practical exploratory investigations, the full details and components of which are described in the following chapter. Though this initial version contains several perceived limitations it still offers important advantages over the existing alternatives, principal among these being the speed of development and degree of modularity. Recommendations have also been suggested for possible methods to overcome or lessen the effects of the deficiencies and ideally these would be applied prior to it becoming generally available.

6 Proof of Concept

6.1 Chapter Contributions

This chapter details a set of text categorisation components built for the trial framework previously described and employs them in a number of controlled experiments that investigate the feasibility of the proposed solution architecture and potential impact of each stage it contains. An independent group of algorithm libraries related exclusively to text processing and classification were also created for the components to utilise, providing segregation of their constituent parts to a level that is seldom encountered in other utilities. In order to demonstrate this superior level of modularity, comparisons are made with some of the best text packages available for the frameworks reviewed in chapter four.

Using a selection of basic components from only the critical stages of the categorisation process a base solution has been formed that can generate a collection of reproducible benchmark performance and data statistics. These are analysed and directly compared against the output of several variants derived from the same solution, each with distinctive modifications that make them deviate from the base configuration in just one aspect of a particular stage. Experiments are conducted for every stage that is commonly disregarded or habitually merged with others already acknowledged as separate entities, with prominent information given for the variants examined that focuses on the differences between them rather than optimal performance.

As the experiments are designed to highlight the possible influence of individual stages, the results gathered are used to validate their existence and reinforce the benefits of treating them as separate elements encompassed by an overall solution. Due to the modular approaches employed an empirical evaluation of the implemented framework is also undertaken, with it being assessed according to multiple characteristics, including development speed and the capacity to handle complex processing pipelines.

6.2 Experiment Overview

Every experiment is constructed from the same basic solution design, which is used as a common benchmark for each of them to be compared against. It has been deliberately kept simple with no complicated or advanced techniques, but does contain components that are easily reproduced and regularly employed as part of more sophisticated solution compositions. All experiments run through the entire text categorisation process, taking an identical dataset as their initial input and performing at least the critical stages identified by the architecture to produce a final set of predictions that are always evaluated in a consistent manner.

For each experiment the foundation solution has a particular component inserted or substituted to produce a variant that differs for only a single stage of the proposed architecture. All others remain identical and statistics are gathered after they are processed or when an independent algorithm is applied to determine how the data has been affected and the magnitude of transformation that occurs. Using the benchmark as a reference the overall results and trends are then examined to ascertain the relative change in performance and how it might have been achieved.

There are no criteria for success and discovering optimal performance is not the primary objective of the trials, instead they are specifically intended to measure the impact of modification to the outlined stages. A number of comparative values and general details about the data are derived to accomplish this, both from statistics derived at various points throughout the process and once final categorisation has been completed. The timings of every stage are also collected to demonstrate the relative speed of algorithms they entail, as though overall performance may not alter significantly this could highlight an additional change that is beneficial in some circumstances.

Undertaking this procedure allows the importance of each stage to be assessed and helps to validate their treatment as individual aspects that must be considered when building or reproducing solutions. It also illustrates how the proposed architecture can be applied to the text categorisation problem and allows a review of suitability with regard to several frequently encountered approaches. Subsequently a direct appraisal of its worth in relation to current methodologies is possible, as they tend to neglect or combine certain areas together, which restricts flexibility and limits the information obtainable.

6.3 Text Processing Components

While the proposed framework is very general, merely allowing a set of operations to be executed in a controlled and reproducible manner, it can be easily adapted to deal with any particular problem through the construction of suitable data models and modules. This is evident in several text processing and classification components that have been created to investigate the composition of text categorisation solutions, the majority of which maps to just one of the distinct stages that may occur.

6.3.1 Data Models

Several options are possible when representing the data and constructing the models that hold it at each stage, but to keep the framework flexible and maximise the automatic node compatibility features they have been kept separate and broken down into their most basic forms. Consequently numerous smaller objects have been developed that are specific to the different types of processes encountered, meaning every stage in a categorisation solution has closely related data models that are reused where appropriate throughout all the experiments.

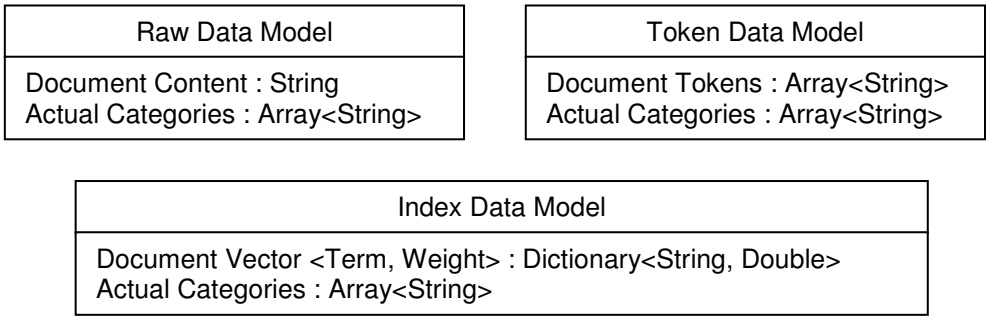


Figure 6.1 : Document Related Data Models

There are two kinds of data model; those that relate directly to documents and those that store additional information required by the various categorisation tasks. Individual document content is always recorded in a relevant format along with the list of manually assigned categories and any other associated scoring or weighting criteria. Also because these models are only capable of representing a single item of supervised text the framework utilises them in collections, allowing any number to be passed between modules and giving the ability to merge multiple sources together at each input node. By contrast the remaining data models do not relate to separate documents and instead embody more generalised information and objects that are derived from the dataset, as a whole or in part, permitting it to be shared with the modules that might need it.

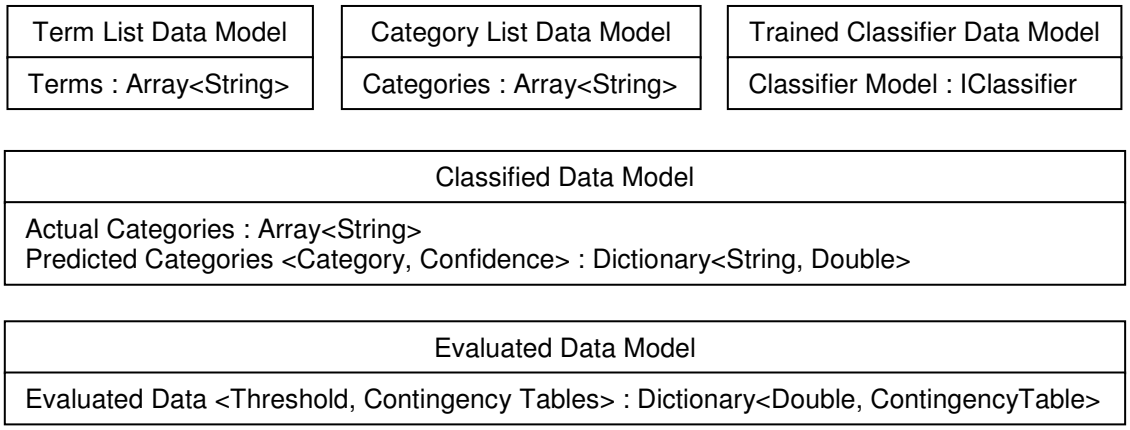


Figure 6.2 : Non Document Related Data Models

6.3.2 Modules

Every stage in the categorisation procedure has its own set of discrete modules that tend to follow a specific pattern dependent on what function they carry out, accepting input or supplying output data models according to their position within the overall solution. Aside from the most basic module, which merely determines the unique elements in a collection of strings, each of them also links to a separate library of text processing algorithms that handles any complex tasks, making development lightweight and relatively quick.

6.3.2.1 Data Loading

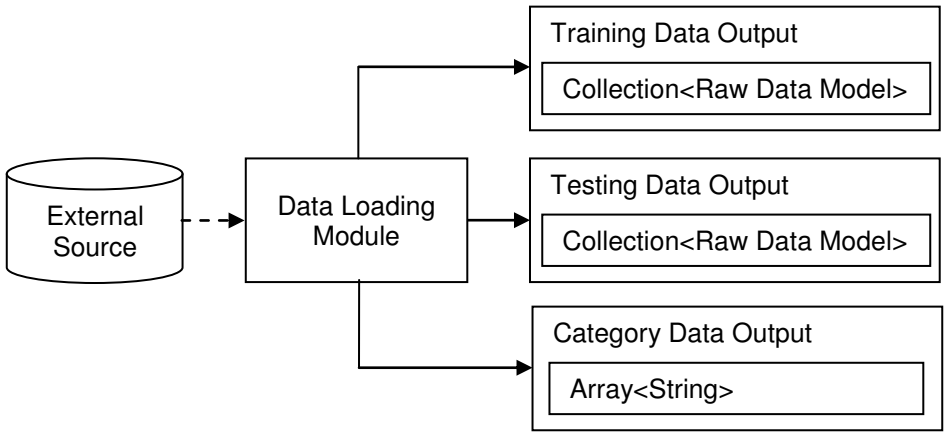


Figure 6.3 : Dataset Loading Module

Loading documents into the system and converting them to a viable format is controlled by a particular group of modules that have no inputs and the capacity to read from sources external to the framework. Defined parameters are used to locate the original dataset files which are then employed to produce the training and testing collections containing Raw Data Models that are pushed to the corresponding output nodes. As the collections are constructed the manual assignments of each document are also scanned and a complete list of unique categories over the entire dataset is compiled and sent to another output node ready for other solution components that might require it.

6.3.2.2 Pre-Processing

The nature of pre-processing transformations leads to a module design that seems trivial, which is true in the case of many simple operations where external input or the definition of parameters are not involved. However some sophisticated techniques entail accessing data sources located outside of the framework scope, like stopword and stemming dictionaries, or have variables that must be set appropriately, such as content truncation or customised character removal. These extra complications are not fully revealed in the outlines provided and may vary considerable based on the precise functionality of an individual module.

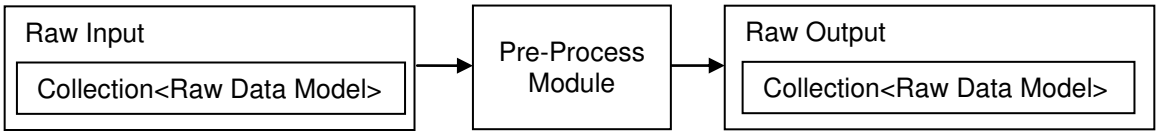


Figure 6.4 : Pre-Processing Module

Note that pre-processing techniques often require document content to be in a format equivalent to that stored by the Raw Data Models, though others may need the data to first undergo a particular representation, for instance stemming and stopword elimination. In these situations the pre-processing module can perform the relevant transformations itself or use Token Data Model collections for the input and output types and utilise an existing representation, which is probably the easiest approach to comprehend. Whichever method is employed it is important that the modularity and separation of the categorisation stages be retained, meaning the input and output types of a pre-processing module should always match.

6.3.2.3 Representation

Representation modules accept either a Raw or Token Data Model collection as input, apply their particular transformations to the individual document content and then output a new set of Token Data Models containing the alterations. Accepting both types of model as input allows the initial data to be

represented directly after loading and the combination of multiple different approaches, whilst also being useful when stages require terms in a specific format not desired by those that follow. In the majority of circumstances this functionality is easy to implement, as raw document content can simply be tokenised into single characters before conversion by a central algorithm.

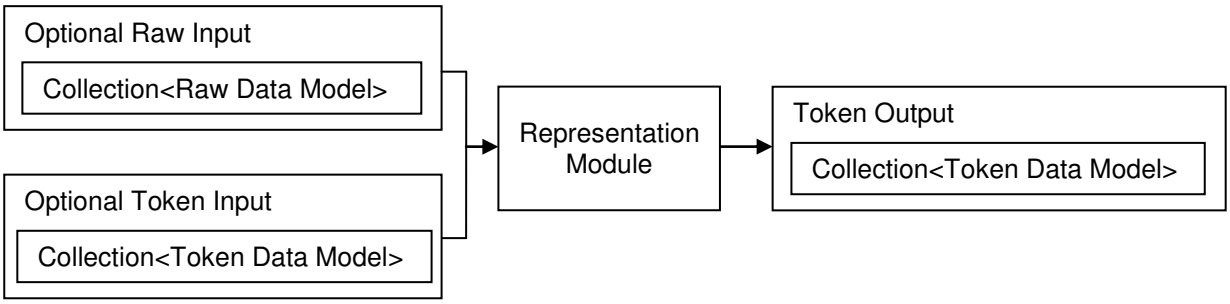


Figure 6.5 : Representation Module

6.3.2.4 Indexing

Accepting a collection of Token Data Models the indexing modules generate the document vectors necessary for classification and store them as part of the output Index Data Models. They range in complexity depending on the solution composition and have the possibility of adding two optional inputs that serve different purposes. One denotes a base reference collection for algorithms that exploit probability statistics as part of their calculations, but when this is compulsory and not explicitly supplied the input data is automatically used. While the other takes a list of terms and filters the final output, after determining weights, by removing any unspecified elements, effectively treating it in the opposite manner to a conventional stopword list though it is not meant as a standalone dimensional reduction technique.

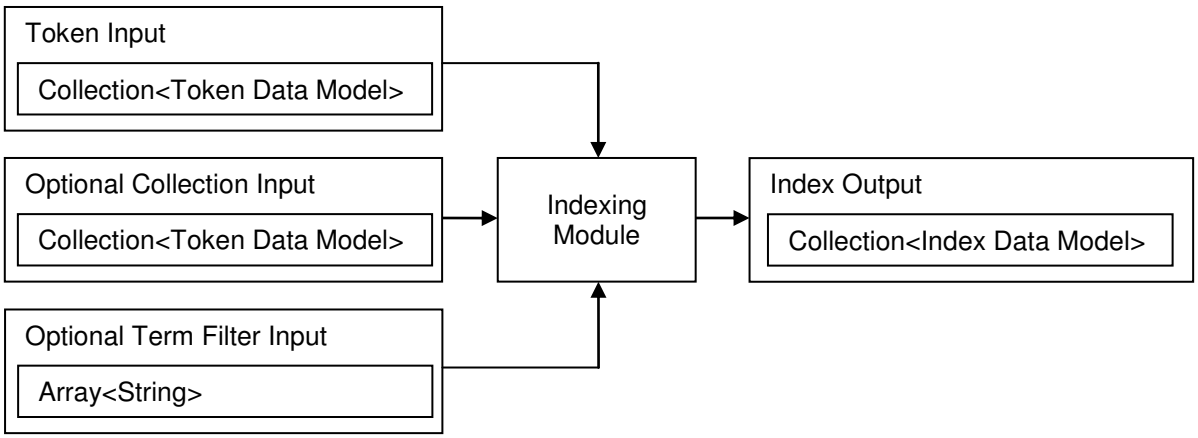


Figure 6.6 : Indexing Module

6.3.2.5 Dimension Reduction

Dimension reduction only needs to be inserted into the training processing pipeline, as a list of unique terms can be produced by a filter module and linked to the optional term filter input of the testing index module. However it is still possible to add them to both pipelines when necessary, for example if comparative statistical information of the two collections is required or the document content should be modified prior to indexing. Similar to the standard indexing modules an optional input is available to provide a base reference collection for algorithms that rely on one and it also behaves in the same manner, using the input data as a reference collection if no alternative is supplied.

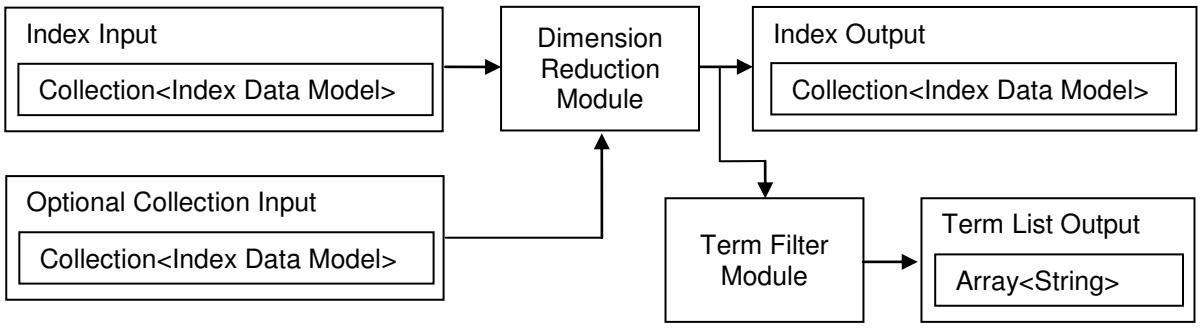


Figure 6.7 : Dimension Reduction Modules

Note that it is also possible to perform many dimension reduction techniques prior to indexing the document content, in which case all of the Index Data Models would become Token Data Models. The term filtering module has the ability to handle either of these input format types, but always supplies the same output regardless of which is used.

6.3.2.6 Classification

Classification involves construction and execution modules, the first of which generates a predictor from the collection of training Index Data Models, while the second applies this to each Index Data Model of the testing set. The predictor can be based on any algorithm but must follow a simple interface that accepts a document vector, comprised of unique tokens and their weightings, as input and outputting a corresponding object that contains the actual and predicted categories along with their confidence scores. By satisfying this constraint the execution module can be generalised to work with most types of classifier, providing a central mechanism to convert between data formats and apply the predictor whilst being kept separate to permit greater flexibility when building solutions.

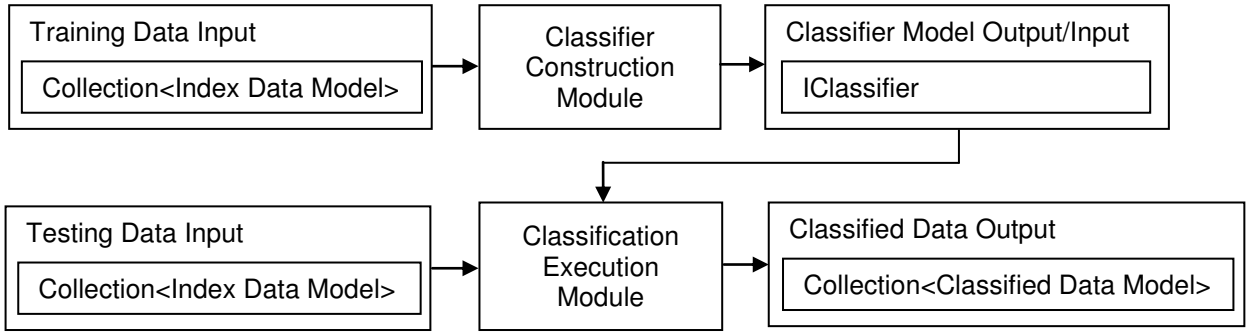


Figure 6.8 : Classifier Training and Execution Modules

6.3.2.7 Evaluation

An Evaluation module takes a collection of Classified Data Models as input and outputs a single Evaluated Data Model that contains all of the possible assignment thresholds and their associated contingency tables. The tables are category-pivoted and record the number of actual and predicted document assignments for each, while another input with a complete list of the unique categories present over the entire dataset is utilised to include any that do not occur in the classified data. Missing categories added in this way are automatically treated as though they have a confidence score of 0 for the purpose of assignment thresholds.

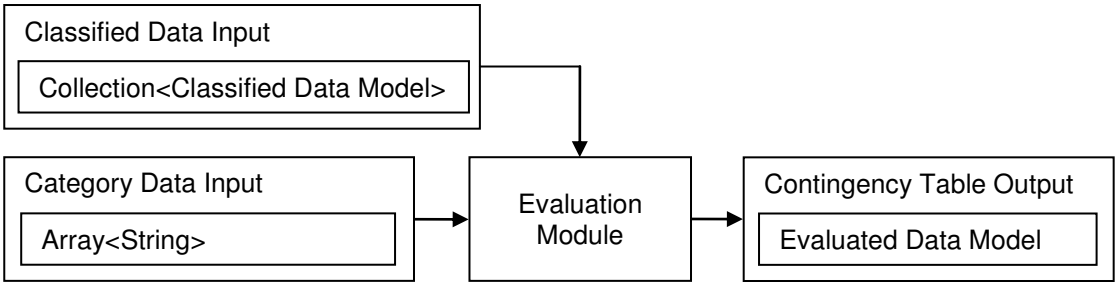


Figure 6.9 : Evaluation Module

6.3.2.8 Information Extractors

Statistics about the document content or category assignments can be gathered at certain points during the categorisation process to assist in determining the impact made by each stage. This is done by information extraction modules that are dedicated to particular data model formats, including those related to documents, category assignments and the evaluated data that contains the final contingency tables. Whenever one of these formats is produced it can be investigated by linking it to

the appropriate extractor, which is then executed to generate text or graphical information that is saved to an external source according to supplied parameters.

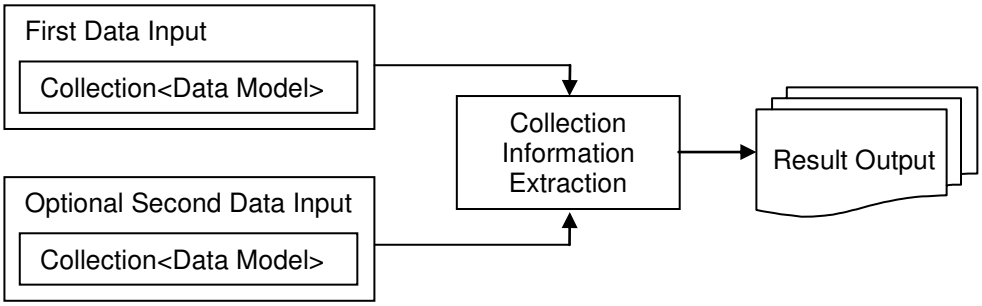


Figure 6.10 : Collection Information Extractors

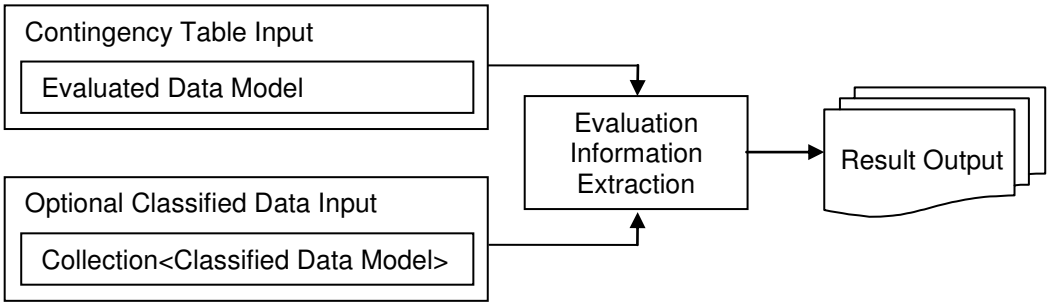


Figure 6.11 : Evaluation Information Extractor

Aside from the Classified and Evaluated Data Models statistics can be extracted from either a single or two separate collections, provided they consist of identical supported types, in which case they are considered independently, combined and in respect to their exclusive and common entities. This functionality offers the ability to make comparisons between different solutions, the data used for training or testing purposes and for the influence of modules that employ matching input and output data models, such as dimensional reduction and some pre-processing modules.

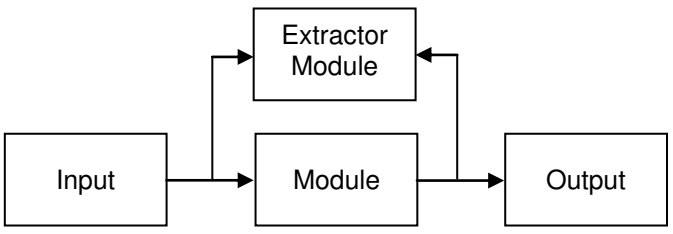


Figure 6.12 : Comparing Module Input and Output with Common Formats

Being standard modules that accept a collection of data models as input it is also possible to link multiple sources that share a format to the same node, whereby the framework will automatically join them into a single entity before processing. Again evaluated information extractors are the exception to this as the mandatory input only takes a solitary object, which is used to produce a set of globally averaged evaluation measures and a graph showing results for the different category acceptance thresholds. If the corresponding Classified Data Model collection is also supplied the confidence scores of predicted assignments are matched against specific thresholds and meaningful statistics will be determined. All the other extractors just derive basic occurrence and magnitude statistics depending on the particular format they handle, for instance minimum, maximum, mean and total values.

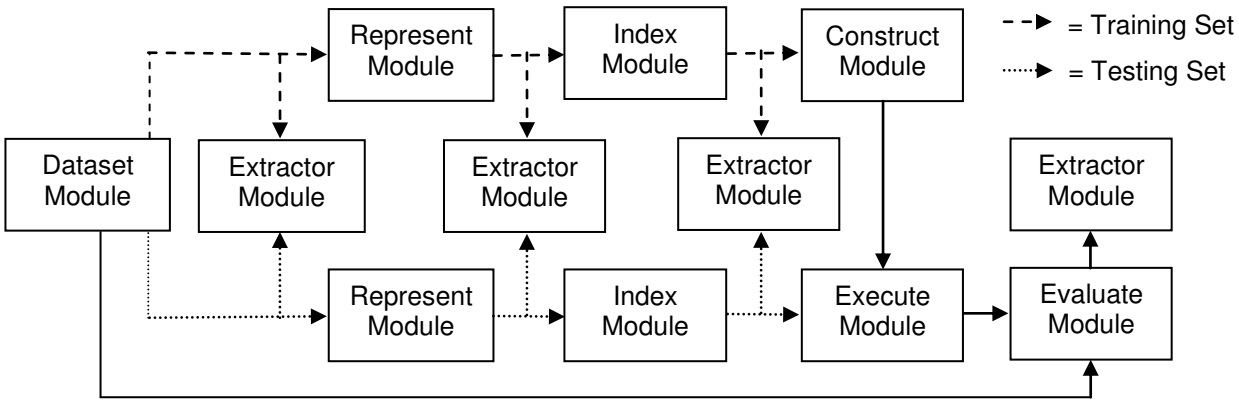


Figure 6.13 : Basic Solution with Training and Testing Data Comparisons

6.3.3 Configuration

Only module execution times are recorded as most framework overheads would not be a factor in applications dedicated to specific approaches, however these still include some inherent aspects such as obtaining data from input nodes and pushing it to the outputs. Elapsed time is recorded by a lightweight but accurate diagnostics stopwatch utility, which avoids the shortcomings of employing the system clock and does not suffer from the complexities involved when counting CPU cycles directly. Modules are also set to apply 'Thread Affinity' when executing, meaning they run entirely on a single processing thread and CPU core in order to compensate for architectures where multiple cores or processors might be available.

Further helping to keep times relative the ability of the repository to cache data locally in the main system memory has been disabled during experiments, with XML serialisation to and from a solitary long term storage device solely being relied on instead. This prevents issues where opportunistic garbage collection or sizable data structures unable to fit in main memory might bias results, giving the impression that some solutions are significantly more or less efficient in comparison to others.

6.4 Text Component Comparison

The principal purpose of these experiments is to assess the importance of granular definitions for text categorisation solutions, thereby illustrating the worth of the architecture proposed in chapter two. In order to achieve this, the stages outlined are being explicitly tested on an individual basis to gain a measure of the influence they each have on the results generated. As such, the experiments must be conducted on a framework that is capable of reproducing exactly the same base solution multiple times but with only a single component, which affects just one distinct stage, being changed during every test so that its impact can be observed.

Due to the nature of these trials, components of the initial base solution and those being substituted do not need to be particularly sophisticated, as they must merely demonstrate that the stage being investigated can have a significant effect on results. In addition the algorithms encompassed by them would always be the same regardless of which framework was actually chosen to run the tests. This means the output for each experiment would theoretically also remain identical, with perhaps only slight differences evident in the execution time due to variations in code efficiency and the languages used. Consequently, as the choice of framework will not modify the outcomes, it is not necessary to perform multiple instances of every test for the different frameworks and the only relevant criterion is the suitability of their text packages to construct the modular experiments.

6.4.1 Relative Modularity

As described in the previous section all of the components built for the new framework are as atomic as possible, with each representing just a solitary algorithm or isolated technique taken from a single stage. The existing frameworks also tend to use individual processes for the actual classification, data loading and output visualisations, so it is relatively safe to assume that these aspects do not require scrutiny and instead the focus should be on the other stages. Coincidentally, these are the stages that typically get neglected when text categorisation algorithms are defined in literature and therefore their influence is the most important to verify in order to validate the proposed architecture.

Using WEKA as an example it is immediately obvious the required level of modularity is not available, with the majority of text processing being done by the ‘StringToWordVector’ component that merges several of the stages together. It contains aspects of pre-processing, representation, dimensional reduction and indexing which it executes all in one go, permitting access to the transformed data only after everything has completed. The interface does give numerous options to configure the algorithms and settings to apply, so it might be possible to run some of the experiments, but not being able to extract statistics for the independent stages makes it more difficult to establish their impact.

Further issues that occur when so many algorithms related to different stages are grouped into a single component are that it can only contain a limited selection and forces them to execute in a fixed order. This already reduces flexibility, but also means any techniques not present in the selection must take place before or after all of the processes in this module, which further restricts the solution design options. The only ways to tackle this problem are to customise the component and insert the new method, which quickly becomes impractical and still forces it to run in a set order, or disassemble it and break it down into several distinct parts, effectively rewriting it to be more atomic.

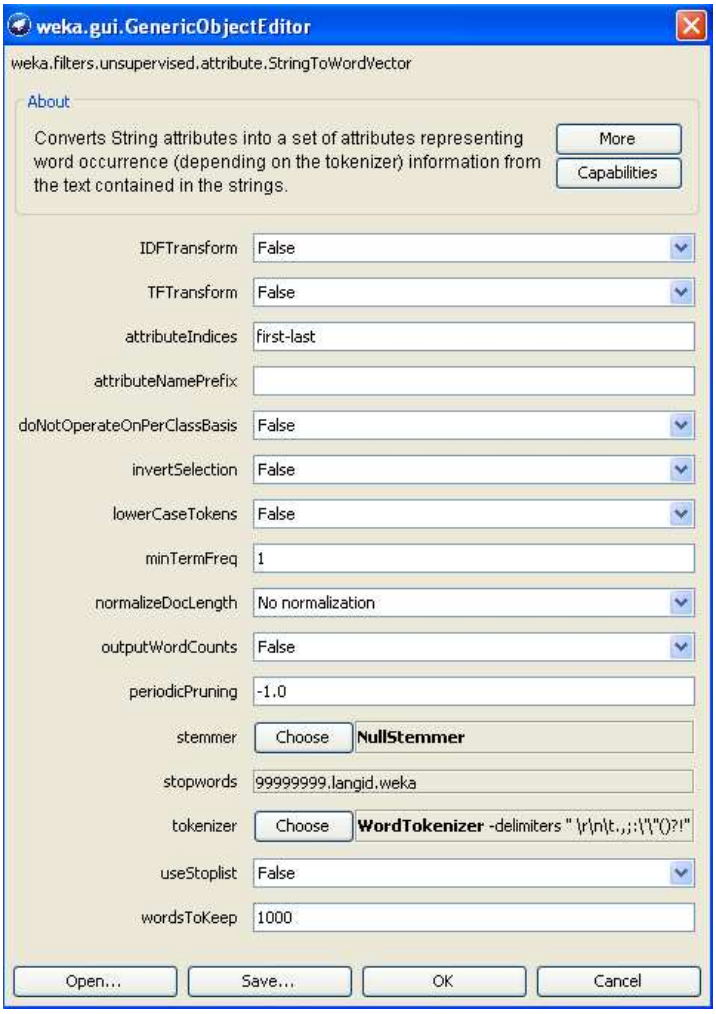


Figure 6.14 : WEKA 'StringToWordVector' Component

It is not only WEKA that suffers from these problems and they are apparent to differing degrees in all of the frameworks previously reviewed, though WEKA is undoubtedly one of the worst. Orange for instance also combines multiple algorithms into single components, like the 'Preprocess' widget that contains stopword elimination, formatting and language specific stemming. Although a positive aspect about this particular widget is that the input and output formats are identical and it can run just a lone operation, so several could be chained together and treated as atomic elements. Unfortunately this

benefit does not hold for some other widgets such as ‘BagOfWords’, ‘WordNgram’ and ‘LetterNgram’, which employ a mixture of pre-processing, representation, indexing and dimensional reduction.

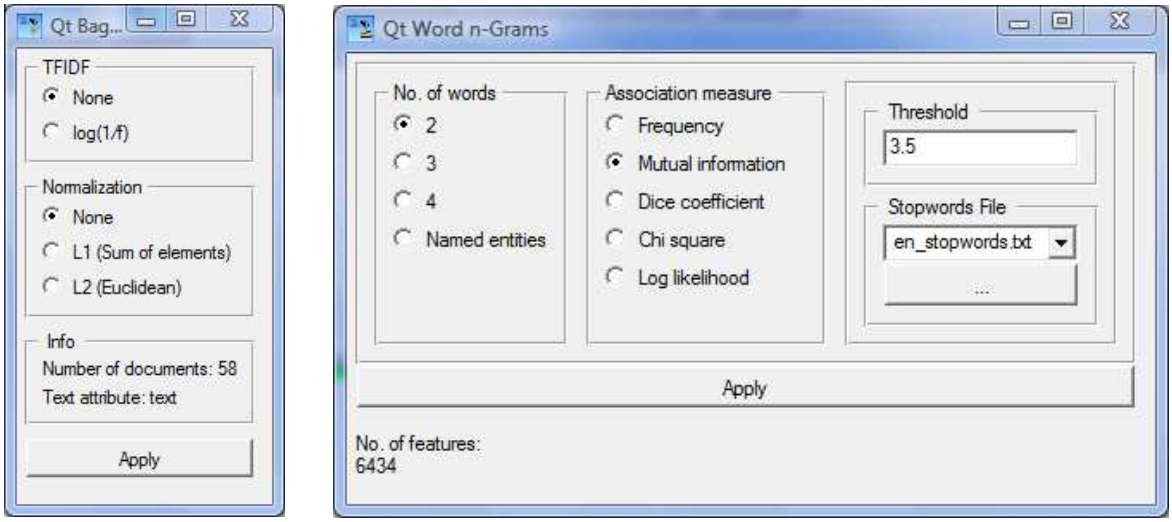


Figure 6.15 : Orange ‘BagOfWords’ and ‘WordNgram’ Components

KNIME and GATE show improvements in their level of modularity but still have a few issues, while R and RapidMiner both have a granularity similar to Orange, with some components distinguishing well between the proposed stages and others combining them. All but GATE also share the common trait of a restrictive set of available text components and astonishingly more than one currently lacks the ability to compute a simple Boolean term frequency weighting or generate representations other than a bag-of-words.

Due to the shortcomings of the other options the new framework and text components developed for it definitely seem to be better suited for the experiments being conducted. They are able to construct exact duplicates of the same base solution, are modular enough to modify just an isolated component during each trial and have the capacity to report statistical information after the execution of every module. Furthermore they are capable of doing these things regardless of the overall solution design, even if the work flow in nonlinear and executes components of the various stages in an atypical order.

6.4.2 Development Effort

It is evident that while some existing text processing packages are competitive, they still tend to lack the modularity required to provide statistics for the full range of stages and when more than just basic algorithms are considered customisation becomes unavoidable. Although it would be possible to rewrite or create new packages for any of the existing frameworks, and certainly that was done for the

new framework, what the previous comparisons do not make apparent is the amount of development effort it takes to accomplish this.

Due to the simplistic design and clean separation of the algorithms and user interface, fabrication of modules in the new framework require marginal code and rarely need extra validation beyond that already included as part of the core functionality. This means a basic component can be developed with minimal exertion in a matter of minutes by someone that has almost any level of programming experience. By contrast, as revealed by the reviews in chapter four, the existing frameworks involve considerably greater input, potentially entailing conversion of the data to bespoke formats, manual integrity validation and implementing multiple user interface events.

Direct code comparisons are very difficult due to the different structures and programming languages used by various components and is made harder because the algorithms and graphical aspects are not always kept separate from the framework. However, it is possible to compare basic requirements that need to be satisfied when making customisations, though this is still only partially feasible as they do not always include the mechanisms required to manipulate the internal data storage or perform other specialised functions.

```
using Core.Modules;
using MyDataModels;

namespace MyModules
{
    public class MyModule : Module
    {
        // Create any input and output nodes
        public InputNode<MyDataModel> dataIn;
        public OutputNode<MyDataModel> dataOut;

        // Define any module specific settings
        public int mySetting = 1;

        public void Run()
        {
            DataModel theData = dataIn.GetDataModel();
            // Process data here
            dataOut.SetDataModel(theData);
        }

        public bool Validate()
        {
            // Optional validation for module specific settings
            Return mySetting > 0 && mySetting < 11;
        }
    }
}
```

Figure 6.16 : New Framework Module Template

The only interface needed for any module in the new framework has just two methods that must be implemented, one of which is practically optional and can simply be set to return true, while the other processes the incoming data. Adding input and output nodes is also straightforward, with a single line

that states the type of node, its name and the compatible data model, then once inserted data can then be retrieved and set using the appropriate ‘GetDataModel’ and ‘SetDataModel’ methods. Module specific settings are again effortless, being public fields with optional default values that can be authenticated via the ‘Validate’ method of the interface if necessary. Everything else is automatically handled by the core framework, which is also responsible for firing any events related to the user environments, so there are no special requirements for manipulating the data or performing data integrity checks.

By contrast, except for R which is a statistical language with a steep learning curve, the procedure for creating components in other frameworks is significantly more complicated and each has dedicated guides or online tutorials that demonstrate how to achieve it. GATE for example has a full chapter in its user guide [208] that explains how to construct ‘Processing Resources’ and ‘Visual Resources’, offering sample code and descriptions of important events that should be included in every module, each with its own set of mandatory methods to implement.

KNIME also has a complicated procedure and each customisation must be comprised of multiple facets, namely the ‘NodeFactory’, ‘NodeDialog’, ‘NodeModel’, ‘NodeView’ and an XML snippet, all of which are vital for compatibility with the core framework. There is also a specialised mechanism for adding module specific settings that is necessary to retrieve, store and load the values. It has been recognised that this process can be quite convoluted and a number of tools are available to assist developers, though these introduce yet another aspect to learn.

As with other areas RapidMiner, WEKA and Orange all follow a similar pattern to that of GATE and KNIME, having a number of obligatory conditions that must be satisfied during development and a range of bespoke mechanisms to integrate with user environments and access internal data stores. Orange and R, assuming basic knowledge of the language, are perhaps the easiest of the existing frameworks to extend, but even these still require substantially more effort than the new framework. Both would also require the majority of their current text processing components to be modified and a selection of additional ones to be built in order to be viable options for these experiments and fit the architecture proposed in chapter two.

6.5 Benchmark Experiment

Full modularisation of the categorisation process allows isolation of individual stages so their impact can be independently analysed, whereas the framework permits reuse of core modules to produce accurate and directly comparable results that can be readily duplicated. Every test is based on and evaluated against the same benchmark solution, which consists of basic algorithms regularly found within the critical stages of text categorisation and provides a common point of comparison. All experiments were also conducted in an identical environment with consistent hardware and software, the exact specification of which is irrelevant for the purpose of this investigation, though the uniformity means execution times are approximately relative.

Throughout all investigations the aim is not perfection of the text categorisation process, but to emphasise the potential difference in outcomes as certain aspects of the various stages are modified or added. Consequently there is little focus on state-of-the-art or high performance algorithms and instead popular techniques that are frequently employed have been utilised, which consist of minimal data processing and only simple parameters that are not subject to extensive tuning or optimisation.

6.5.1 Solution Outline and Components

Only components that are necessary to complete the fundamental stages of text categorisation are part of the benchmark solution, intentionally keeping it simple by avoiding any form of pre-processing or dimensional reduction. It serves as the foundation for the solutions in every experiment and several of the base components remain unchanged for all of them, while others are reused where possible or altered according to the particular aspects being examined. When one of the absent stages is being investigated the new elements are just added to the existing structure as appropriate and everything else stays constant.

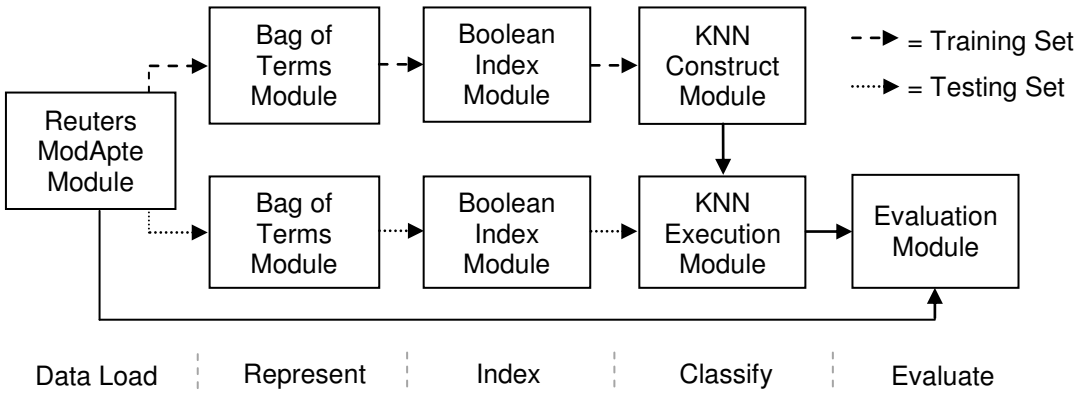


Figure 6.17 : Benchmark Solution Overview

6.5.1.1 Dataset Selection

It is widely acknowledged that different text document corpora, along with the subsets derived from them, have varying characteristics and relative hardness that affect the outcome of any applied solution. They obviously also need to be chosen to match the specific application or problem being studied, making it clear that the data selection stage is a crucial part of the categorisation process. Subsequently it is unnecessary to trial a selection of datasets and therefore the same collection of training and testing documents have been used throughout all experiments, though a similar degree of variation in results for the individual stages should not always be assumed.

Despite its age the popular ModApte subset of the Reuters-21578 corpus has been employed, as it is still commonly utilised for the appraisal of text classification algorithms in literature and is likely to provide appropriate comparative material. Each document is represented by its title added to the main body of text with a single space separating them or if only one of these is available it is used on its own. Content marked as 'unprocessed', where the data has not been stylised due to obscure formatting, is just taken in its raw unaltered form as no publications could be found that comment on how this is normally handled.

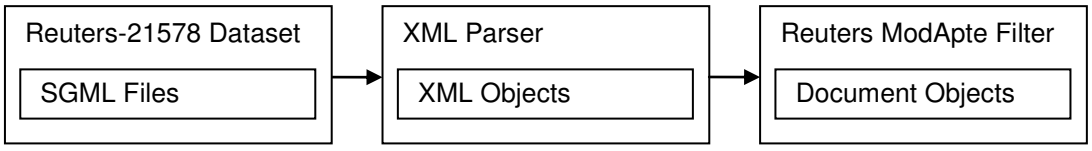


Figure 6.18 : Dataset Loading and Parsing Overview

The data was obtained from an official online resource in SGML format [82], which had non XML compliant characters removed before being parsed by a standard XML reader and converted into a basic document object. A specific filter was applied during the conversion to determine if items belong to the training or testing subsets and this information is stored as part of the object along with its textual content and list of manually assigned categories. Due to enforced line wrapping a space was added to the end of each line while being read, otherwise whitespace was preserved and no cleaning of the data attempted, so generated parser errors such as unrecognised special characters or unconventional formatting are retained.

6.5.1.2 Representation Format

Bag-of-Terms is used as the base representation, iterating over the content of each document and combining terms into a single entity until either the end is reached or a customisable sequence of

characters is encountered and triggers creation of another entity. Where content has not previously been converted into distinct tokens, as is the case in all experiments, individual characters are treated as terms and the same procedure is followed.

For every solution single space characters are used to distinguish how the document content should be split and are deleted as part of the process, meaning terms will never include single spaces but may consist purely of other types of whitespace. Any empty terms caused by adjacent spaces are automatically removed, otherwise no attempt is made to clean the data or handle formatting and all characters between the delimiters are retained in their entirety as a new term.

6.5.1.3 Index Weighting

Vector Space Model in combination with Boolean weighting form the principal indexing component, scanning though the terms of each document and recording a total weighting of 1 for those that occur. Any not present are simply ignored, as missing data is assumed to have a weighting of 0 by all modules throughout the entire framework, minimising both computational and storage requirements. If however an algorithm explicitly relies on information related to absent terms, for example those employing the concept of negative evidence, they must ensure that all relevant data is recorded in order to override this default behaviour.

6.5.1.4 Classification Algorithm

Similar to datasets it is accepted that the choice of classifier is an essential part of a categorisation solution, though they often mistakenly include a combination of multiple distinct stages as a single entity. Due to this the same basic classification algorithm has been used throughout all experiments, ensuring that the components and configuration are kept identical to provide unbiased results. In order to give an enhanced view of the impact made by the various stages a classifier with the ability to rank predictions according to a confidence score was also a priority.

K-Nearest Neighbour (KNN) is employed for the tests as it is commonly used in comparative analysis and previously shown to give reasonable performance for generalised text categorisation problems, including those based on Reuters-21578 dataset and its derived subsets. It is also relatively straightforward with minimum parameters that require direct input and a standard form that has no inadvertent merging of the other categorisation stages. Additionally it is susceptible to alterations in the data so any changes made should be adequately highlighted, demonstrating that even for a potentially superior classifier it is essential to consider the whole solution and not just certain elements.

The classifier is comprised of the most basic KNN algorithm, so there is a minimal construction stage that merely sets a parameter and records the training document vectors and their related categories, while the execution stage is responsible for the majority of processing. Unlike many other published versions of this technique the components involved are deliberately kept simple and there are no enhancements beyond the essential aspects, such as similarity modifiers or minimum acceptance criteria. An assignment threshold for generating absolute predictions is also not applied as part of this stage and instead a ranked list of candidate categories with their associated confidence scores is retained for every test document.

Vectors being tested and those stored during classifier construction are measured for likeness by Adjusted Euclidean Similarity and tie breaks between the neighbours with joint lowest similarity are chosen randomly. This decision process is done through the use of placeholders once all potential neighbours have been selected, giving each an equal chance of being added to the final group without the order they were encountered being a factor. After candidates have been compiled from the list of neighbours every category is ranked by confidence according to its number of occurrences, while those not present are automatically assumed to receive a score of zero.

Owing to the simplicity of the algorithm and the components involved only a single parameter needs to be defined during the classification stages, which is the value of k representing the number of nearest neighbours that candidate categories are determined from. As the main purpose of the experiments is to demonstrate the importance of change in categorisation solutions, rather than how well they can perform, no conventional optimisation or tuning is undertaken and instead a small set of trials were conducted to determine a practical value.

Based on the limited empirical tests a value of 30 was chosen for k and is kept the same throughout all experiments, as it does not require excessive resources and produces enough classification threshold points to give a reasonable indication of trends in performance. It also keeps the emphasis on the impact of individual stages unrelated to the classifier, by providing moderate results for the benchmark solution that are not inclined towards a particular extreme, though this was true for the majority of values trialled.

6.5.1.5 Evaluation Criteria

Several appraisal methods have been implemented to allow a variety of comparative information, including macro and micro averaged optimal F1, Break-Even Point and the relationship of Precision against Recall. Every stage investigated along with the entire categorisation process also has a number of associated statistics, showing relative execution times and a breakdown of prominent information into minimum, maximum and mean values. This includes classifier execution and the predicted category assignments, where the confidence score thresholds deduced in each of the final

performance evaluation measures are used to put the predictions into a suitable context before extracting the data.

A point of note is that it is never possible to achieve a Recall of 0 for every test document, as some do not have categories assigned and therefore default to a value of 1 even without predictions because all labels are present that should be. While this does not affect micro averaging it causes a significant but consistent rise in macro averaged performance, though is unavoidable given the dataset as using a default value of 0 would just have the opposite effect. Another option is to remove all documents displaying this characteristic from the test collection and some Reuters-21578 subsets are specifically designed to accommodate this, however they are rarely encountered which makes comparisons difficult.

6.5.2 Results and Statistics

For brevity and to focus attention on the degree of influence made by the particular stage being investigated only data relevant to those modifications or that differ from the benchmark figures are presented for each experiment. Also due to the volume of statistics generated by the information extractors and the potential for repetition between the stages not all of them have been portrayed, though a limited overlap is still evident in some experiments.

In order to aid readability values have been rounded to the nearest five decimal places, with the only exceptions being percentages and module execution times, which have been rounded to three decimal places to provide milliseconds. This reduction in precision in no way diminishes the perceived influence of each stage and any trends in the data are still plainly visible. For example the gathered statistics clearly suggest that, despite variation in content size and category assignment, the training and testing sets contain items of comparable length and subject diversity.

6.5.2.1 Execution Times

All experiments had a minimum of five runs and experienced similar execution times with the median values being recorded, which assists in preventing excessive bias from uncontrollable environmental differences that may occur during the tests. The original timings were in whole and fractional seconds to a precision of eight decimal places, though the final results have been rounded to the nearest millisecond as they will never be capable of showing more than an estimation of the module requirements.

Despite being approximations the relative needs of each stage are apparent, with classifier execution taking considerable longer than any other aspect, which is expected due to the lack of training in the

KNN algorithm and quantity of similarity calculations taking place. It is also possible to distinguish that indexing is undeniably the second longest process and is related closely to the size of dataset, a trait which is also true for the representation. A potentially unanticipated revelation is the relatively short time taken to load the initial data and parse it into a suitable format, though disabling the caching mechanism of the framework repository is likely the cause for this because it forces all modules to access an external storage device.

	Stage					Totals
	Data Load	Represent	Index	Classify	Evaluate	
Training	-	1.233	5.509	9.931	-	16.673
Testing	-	0.388	1.846	759.681	2.363	764.278
Combined	1.339	1.621	7.355	769.612	2.363	782.29

Table 6.1 : Benchmark Execution Times

Although not entered in the table it is worth noting that the Raw, Token and Index Data Model information extractors took several times longer to execute than the respective stages generating their inputs. However the Evaluated and Classified Data Model extractor was appreciably shorter, implying that the majority of time is spent processing data and deriving comparative statistics between two datasets rather than performing IO operations. If this is the case then it is probable that the same inference can be made regarding modules in general, though further examination of their internal workings and the separate operations involved is necessary to produce definite conclusions.

6.5.2.2 Dataset Statistics

Reuters-21578 is comprised of fully supervised data with 21,578 documents, while the ModApte split contains a subset of 12,902 documents in an attempt to mimic those from prominent experiments previously published [62]. It is divided into 9,603 training samples and 3,299 testing samples, which are assigned chronologically by creation date with earlier documents being assigned to the training set. A total of 8,676 are left unused which are considered to be exact duplicates or items that have not been appropriately classified during the original manual categorisation process.

Most documents consist of both a title and main section of text content, though a few only have a title available, while none are completely empty due to the filtering that occurs when the subset is created. Overall the mean content length is around 804 characters but there is wide variation in size, with a training sample containing just a title being the shortest at 37 characters, while the largest is a test sample of 13,432 characters. Both training and testing sets experience this level of variation, as the

mean length of test samples is approximately 764 characters with the shortest being 38, while the training sample mean length is roughly 817 characters with a maximum size of 8,604.

	Characters Per Document			Total Characters	Unique Characters	Total Documents
	Minimum	Maximum	Mean			
Train	37	8604	817.47725	7850234	87	9603
Test	38	13432	763.73568	2519564	81	3299
Combined	37	13432	803.73570	10369798	89	12902

Table 6.2 : Benchmark Dataset Statistics

Of the five different category groups available just the 'TOPICS' collection is utilised and only 118 of these are represented in the documents, with 115 assigned in the training set and 93 in the testing set. This means both sets share 90 common categories between them, while there are 25 exclusive to the training set and 3 that are exclusive to the testing set, preventing any categorisation solution from making completely correct predictions.

	Assignments Per Category			Assignments Per Document			Unique Categories	Total Assigned
	Min	Max	Mean[1]	Min	Max	Mean		
Train	1	2877	83.89565	0	16	1.00469	115	9648
Test	1	1087	40.29032	0	14	1.13580	93	3747
Combined	1	3964	113.51695	0	16	1.03821	118	13395

Table 6.3 : Benchmark Category Assignment Statistics

Increasing the hardness and potential for inaccurate predictions there are a total of 2,108 unlabelled documents, 280 of which belong to the test set, while the maximum categories assigned to a single document is 16 despite the mean per sample being just over 1. A significant disparity in the density of examples per category is also present as the mean is approximately 113.5, though 16 have just one assignment over the whole dataset and 5 have only one sample in both the training and testing collections, while another occurs 3,964 times.

[1] Training and testing values do not add up to the combined total as might be expected for this column because the means are calculated using only categories present in the particular subset and not over the entire dataset.

6.5.2.3 Representation Statistics

Even though the general dataset statistics indicate that there is a resemblance between the document length and category assignment densities of the training and testing sets, the basic bag-of-terms representation employed suggest that is the extent of their similarity. While sizes are comparable the variation in unique terms implies that the content is substantially different, with 52,155 being exclusive to the training set and 16,793 exclusive to the testing set, which leaves only 24,857 or roughly 26.5% common to them both. This means almost three quarters of the unique terms are unable to be of use when calculating document similarities or performing classification, although it is hard to ascertain what proportion of the data these actually depict.

	Characters Per Term			Terms Per Document			Unique Terms	Total Terms
	Min	Max	Mean	Min	Max	Mean		
Training	1	61	5.62807	5	1317	132.71519	77012	1274464
Testing	1	43	5.57161	5	1673	123.77811	41650	408344
Combined	1	61	5.61432	5	1673	130.43001	93805	1682808

Table 6.4 : Benchmark Representation Statistics

As the predominant language of the dataset is English a maximum term length of 61 characters may be surprising, but analysis reveals that these are not in fact real words and are due to obscure formatting. It is difficult to determine the ratio of genuine to invalid words in the data, though a quick inspection shows that there are many instances where punctuation causes undesirable alterations, which may be the reason for the large disparity in the unique terms.

SPOT.(DLRS/TONNE).....PROPANE.....BUTANE.....
MEDITERRANEAN.(CIF)....140/145.....175/180.....
ALGERIA.....(3/1).....120.....130.....

Figure 6.19 : Example of Terms with Maximum Length

6.5.2.4 Indexing Statistics

Statistics derived from the Indexed data have a degree of overlap with those of the representation stage, as the same term definitions are utilised to produce the document vectors. For instance the number of documents per term and unique terms, which signify the quantity of documents containing

a particular term and the amount of unique terms in the entire dataset, are guaranteed to match for both stages. However in contrast to representation terminology when an indexing statistic refers to a term it specifically denotes a unique term, so in many cases different information is portrayed, for example the terms per document figures do not count duplicates.

By comparing index statistics against those from the representation stage it appears that the dataset has a low level of term duplication, which is inferred because of the small ratio between the mean unique terms per document and the mean value when duplicates are included. The same ratio can also be established via the total terms, as the index statistic denotes the number of unique term and document combinations while the representation value gives the amount of overall term occurrences. In this case the ratio when considering the whole dataset is 1 to 1.52721, implying that when a term is present in a document it will occur approximately one and a half times on average.

	Documents Per Term			Terms Per Document			Unique Terms	Total Terms
	Min	Max	Mean	Min	Max	Mean		
Training	1	7115	10.82341	5	658	86.79923	77012	833533
Testing	1	2316	6.44300	5	613	81.34313	41650	268351
Combined	1	9328	11.74654	5	658	85.40412	93805	1101884

Table 6.5 : Benchmark Index Term Statistics

According to the term statistics there is at least one rare term that is in just a single document and therefore exclusive to either the training or testing data, but also common ones that occur in over 74% of the total documents. They also indicate that on average fewer documents contain any given term in the testing set than in the training set, though when considered in proportion to the size of collection this proves to be misleading. Using the ratio of testing to training documents the mean occurrence values can be scaled and made directly relative to one another, which subsequently shows that terms in general are actually around 73% more likely to be encountered throughout the testing set.

	Weighting Per Term			Global Term Weighting			Weighting Per Document			Total Weight
	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean	
Training	1	1	1	1	7115	10.823411	5	658	86.79923	833533
Testing	1	1	1	1	2316	6.44300	5	613	81.34313	268351
Combined	1	1	1	1	9328	11.74654	5	658	85.40412	1101884

Table 6.6 : Benchmark Index Weighting Statistics

In the benchmark solution and all experiments that employ the default Boolean weighting the term occurrence and weighting statistics are always equivalent, as a term can only be entered once into each vector and is given a weight of 1. This is also the reason why individual term weightings per document all have a value 1 and provide no meaningful information, since they are on a document basis and not aggregated over the whole collection. Consequently this data is not displayed for other experiments unless they use a different indexing technique that is able to offer additional insight.

6.5.2.5 Evaluation Measures

All results produced by the benchmark are near to 50% performance and would be considered below average in comparison to many existing solutions, which frequently achieve measures in excess of 80%. This is intentional though and provides a reliable foundation that is easily duplicated and suitable for other experiments to expand directly with minimal effort.

	Precision	Recall	Threshold	Value
Macro F1	0.57801	0.50506	1	0.53908
Micro F1	0.63682	0.47825	10	0.54626
Macro BEP	0.57801	0.50506	1	0.54154
Micro BEP	0.49925	0.53536	8	0.51731

Table 6.7 : Benchmark Evaluation Measures

Examination of the tabular and graphical data shows that macro averaging, which favours correct assignment of rarer categories, suffers from poor recall across all classification thresholds until the absolute lowest value that denotes the trivial acceptor. This suggests that the solution is unable to appropriately identify and assign several distinct categories, causing inferior macro averaged scores and the F1 and BEP measures to both employ the smallest possible non zero threshold. Reviewing the dataset and representation statistics, three categories are exclusive to the testing set and roughly 73.5% of the derived terms serve no purpose during classification, so significant improvement of these results is unlikely given the current term definitions.

Micro averaging follows a more conventional trend with recall steadily increasing and precision decreasing as the thresholds are lowered, though there is still a notable distance between the last threshold score and the trivial acceptor. A consistent high precision level at the beginning indicates a proportion of category assignments that are easy to achieve, but the location of first threshold after the trivial rejector insinuates many of these are probably documents without associated categories. This is confirmed by the abrupt start of the macro averaged results and the dataset statistics, with nearly 8.5% of the test documents being unlabelled.

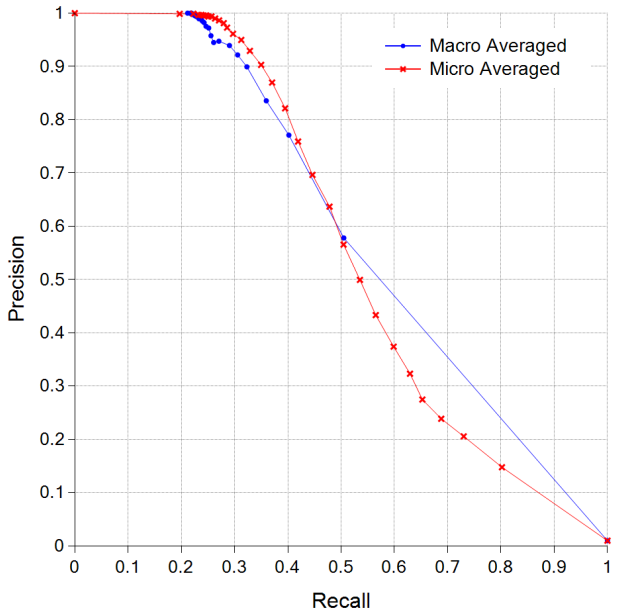


Figure 6.20 : Benchmark Micro and Macro Precision against Recall

For the greater half of the threshold values the micro averaged precision remains high but the corresponding recall is slow to increase, so although most of the categories predictions are correct there are not many being made. After the halfway point more predictions occur at each threshold however a larger percentage are inaccurate, leading to a steeper gradient as precision declines faster than the recall rises. By comparison macro averaging demonstrates that individual categories are assigned with minimal trouble until a threshold generates correct predictions for a selection of different categories that boosts both precision and recall, after which it struggles to assign those that remain.

	Predictions per Category			Predictions per Document			Unique Categories	Total Assigned
	Min	Max	Mean	Min	Max	Mean		
Actual	1	1087	40.29032	0	14	1.13580	93	3747
Macro F1	1	3150	295.13043	0	21	6.17278	69	20364
Micro F1	1	1146	201.00000	0	3	0.85299	14	2814
Macro BEP	1	3150	295.13043	0	21	6.17278	69	20364
Micro BEP	1	1313	200.90000	0	4	1.21794	20	4018

Table 6.8 : Benchmark Category Prediction Statistics

No document content related statistics remain after the classification stage so it is impossible to gather further information in regard to this aspect, however actual and predicted category assignments are available but need to be put in context before meaningful data can be extracted. This is achieved by using the thresholds determined for the macro and micro averaged F1 and BEP measures to produce static sets of predictions, which can then have relevant statistics derived.

Analysis of the prediction related statistics help understand trends present in the graphical data, with the low F1 and BEP macro averaged threshold causing considerable assignments to be made that account for the poor precision. In combination with the low recall and number of unique categories this reinforces the notion that some are difficult for the solution to predict, as it persistently makes incorrect choices. Most likely this is due to several high density categories that dominate the problem space and pollute the list of candidates obtained from the nearest neighbours during classification, which is possible to adjust with all stages of the categorisation process.

In contrast the macro averaging thresholds both lean towards more cautious tendencies, having fewer predictions per document and overall assignments that are closer to the actual figures. Despite this it is still evident that there are multiple dominating categories, with a low number of unique categories and a corresponding high quantity of predictions for each one, suggesting these are prevalent throughout the training data and responsible for the bias in candidate selection.

6.6 Pre-Processing Experiment

A wealth of pre-processing techniques are available that vary in methodology and complexity, but formatting and stemming have been chosen as they are commonly employed and based on different underlying concepts. Despite being in the majority of published text categorisation solutions they are normally applied as a matter of habit and it is rare for either to be accompanied by evidence, empirical or otherwise, that supports their use. Often both approaches are also combined together, though in these experiments they are utilised separately with the aim of investigating the immediate affect each has on the data and all subsequent stages.

6.6.1 Solution Outline and Components

While both pre-processing methods are designed to clean the data they accomplish this in different ways and require particular solution structures, with formatting occurring directly after the data is loaded and stemming relying on a bag-of-words representation for optimal effect. Despite these differences the base solutions retain all of the original components and the new modules are simple inserted into the training and testing pipelines in the necessary positions. Each also includes an extra information extractor of the appropriate data model format, which is linked to the outputs of the added modules so their statistics can be derived.

6.6.1.1 Format Pre-Processing

Four basic formatting procedures have been combined into a single module that is applied to the initial training and testing data immediately after being loaded and prior to any representation stages. Each process is used on the entire content of every document and aims to remove as many potential noise characters as possible and convert the rest into a standard format, attempting to reduce the negative impact of obscure structures and styling.

The first process to occur is capitalisation, which converts characters into their lowercase equivalent where available. Following this all forms of punctuation are deleted, including enclosing brackets, bracing symbols, dashes, hyphens and quote markings. Next digits are removed, which are numeric characters in the range of 0 to 9 but not glyphs, fractions, superscript, subscript, mathematical notation or non decimal symbols. Finally whitespace such as tabs, line feeds and carriage returns are converted into a single space and any that are adjacent are combined into a solitary character.

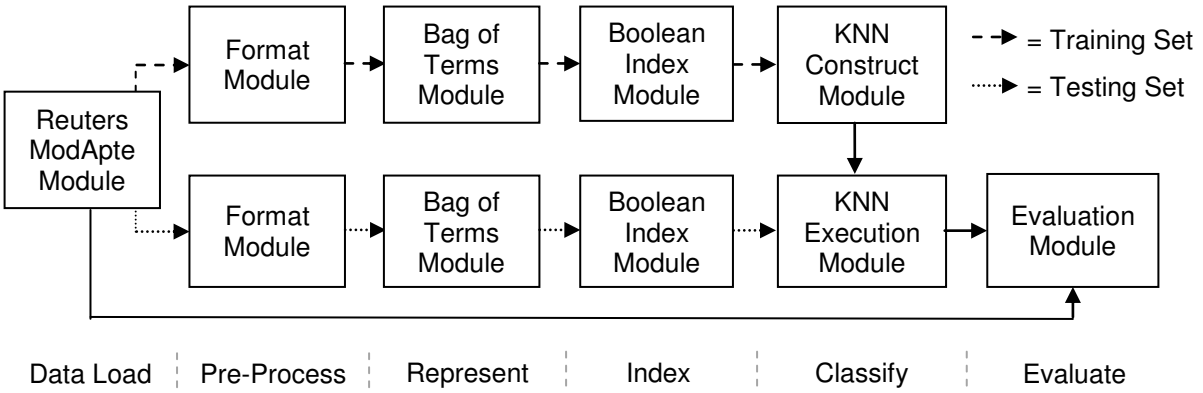


Figure 6.21 : Format Pre-processing Solution Overview

6.6.1.2 Stemmer Pre-Processing

In order to function properly stemmers require distinct words as input, which they transform into a root form that is common to all words with the same conceptual meaning. This entails conversion of the document content into a bag-of-words representation prior to application to achieve the maximum benefit and consequently the stemming is performed after this module in the solution. Porter stemmer is the algorithm used and no other data preparation or cleaning takes place, ensuring initial terms are identical to those in the benchmark and preventing any unwanted factors affecting results.

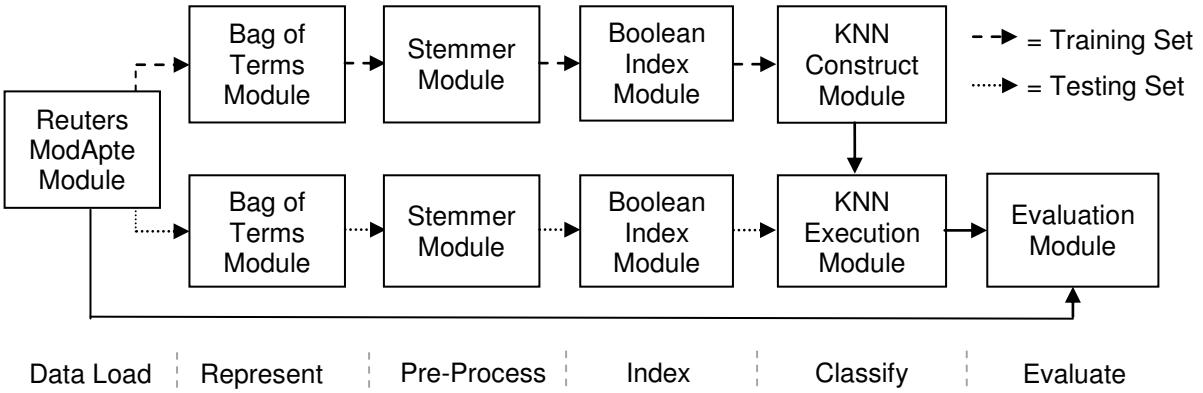


Figure 6.22 : Stemmer Pre-Process Solution Overview

6.6.2 Results and Statistics

Similar to the benchmark results all values are rounded to the nearest five decimal places except for percentages and module execution times, which are given to the nearest millisecond. Also only the

outcomes of stages relevant to these particular experiments or different to the base figures are supplied and to provide easier comparison pre-processing statistics are displayed as part of the core stages they affect instead of in a separate section. This means there are no explicit details about actual category assignments, index weightings or stemming statistics prior to the representation stage.

6.6.2.1 Execution Times

Although the execution times are just estimations it is possible to identify trends in the values that allow certain assumptions to be made with a degree confidence. For instance despite the added overheads of pre-processing the time taken for testing and the overall categorisation solution have been reduced, which is primarily due to an increase in classifier execution speed. This is an indication that less data is being processed during the KNN similarity calculations, most likely because of a general decline in the number of unique terms and therefore the length of document vectors.

		Stage						Totals
		Data Load	Pre-Process	Represent	Index	Classify	Evaluate	
Base	Training	-	-	1.233	5.509	9.931	-	16.673
	Testing	-	-	0.388	1.846	759.681	2.363	764.278
	Combined	1.339	-	1.621	7.355	769.612	2.363	782.29
Format	Training	-	1.128	1.116	5.215	8.071	-	15.530
	Testing	-	0.459	0.403	1.701	494.299	2.392	499.254
	Combined	1.441	1.587	1.519	6.916	502.37	2.392	516.225
Stemmer	Training	-	5.458	1.213	5.374	9.400	-	21.445
	Testing	-	1.732	0.417	1.719	635.154	2.486	641.508
	Combined	1.420	7.190	1.630	7.093	644.554	2.486	664.373

Table 6.9 : Pre-Process Execution Times

It is also clear that stemming causes a significant rise in the time taken to prepare the data, with it being the second longest process in the training and testing pipelines of its experiment, though this is easily offset by the savings made during classification. Further apparent is that the formatters have relatively lower overhead, but are still able to provide faster classification, reducing the total base solution time by approximately 34% compared to roughly 15% for the stemming. Even accounting for the inherent imprecision of the data these margins are substantial, whereas the values for the other modules are too similar and prevent the inference of reliable conclusions.

6.6.2.2 Dataset Statistics

Stemming is excluded from this section as it does not affect the benchmark values due to occurring after term representation takes place, however formatting is present and the statistics demonstrate a notable change in the data. Though basic in nature the employed procedures have reduced the volume of data by removing 1,007,146 characters, representing approximately 9.7% of the overall content. Reflected in the mean characters per document this loss also influences both the training and testing sets in almost equal proportions, further attesting the similarity between them.

		Characters Per Document			Total Characters	Unique Characters
		Minimum	Maximum	Mean		
Base	Train	37	8604	817.47725	7850234	87
	Test	38	13432	763.73568	2519564	81
	Combined	37	13432	803.73570	10369798	89
Format	Train	29	8276	741.95137	7124959	34
	Test	30	5119	678.29433	2237693	32
	Combined	29	8276	725.67447	9362652	35

Table 6.10 : Pre-Process Dataset Statistics

Elimination of the data has caused the maximum length of test document to decline by nearly 62%, implying it is mainly comprised of punctuation, numerical characters or atypical whitespace and formatting. The minimum length has also experienced a drop in size, though as would be expected with the smaller content this is not as severe and is a decrease of around 21% for both training and testing sets. Another observation is a more than 60% decline in unique characters, again affecting both datasets evenly, which limits combination possibilities and consequently should reduce the quantity of unique terms derived during the representation stage.

6.6.2.3 Representation Statistics

Applying the formatting procedures has cut the maximum term length considerably, with the training value being more than halved, which is consistent with findings from the representation stage of the benchmark experiment where the longest terms were comprised of excessive punctuation. There has not been such a significant effect on the mean term length however, with only trivial differences that imply it is not typical for terms to contain a large amount of the removed noise characters.

The total quantity of terms has also reduced by almost 6%, with drops in the region of 6.9% for training and 5.6% for testing data, which is unexpected as it means removing noise that would

potentially join multiple terms together has actually caused a decrease. This suggests that several of the benchmark term definitions consist of just noise in the form of punctuation, numbers and whitespace, while the substantial decline in maximum terms per test document indicates these are prevalent in some content.

As deduced from the dataset statistics the number of unique terms has also dropped by around 58.7% for the whole dataset, meaning there are now 12,733 common terms between the training and testing datasets. This is approximately 32.9% of the total available and an increase compared to the base figure of 26.5%, though it still leaves 5,930 unique terms that are exclusive to the test set and 20,055 exclusive to the training set, giving more than 67% that have no use in the classification process.

		Characters Per Term			Terms Per Document			Unique Terms	Total Terms
		Min	Max	Mean	Min	Max	Mean		
Base	Training	1	61	5.62807	5	1317	132.71519	77012	1274464
	Testing	1	43	5.57161	5	1673	123.77811	41650	408344
	Combined	1	61	5.61432	5	1673	130.43001	93805	1682808
Format	Training	1	26	5.62433	5	1305	125.22670	32788	1202552
	Testing	1	31	5.56937	5	925	115.32464	18663	380456
	Combined	1	31	5.61109	5	1305	122.69478	38718	1583008
Stemmer	Training	1	61	5.08226	5	1317	132.71519	70026	1274464
	Testing	1	43	5.05070	5	1673	123.77811	37479	408344
	Combined	1	61	5.07457	5	1673	130.43001	85871	1682808

Table 6.11 : Pre-Process Representation Statistics

Stemming is intended to modify terms into a generalised form rather than remove them, therefore it has not altered the number of terms per document or the overall total and these remain identical to the base figures. It also had no effect on the maximum length terms, as they do not follow a valid affix pattern recognised by the Porter stemming algorithm. The mean length has changed and exhibits a reduction of about 9%, which shows there are enough valid affixes in the base terms for the slight change caused by their transformation into a root form to make an appreciable difference.

The quantity of unique terms has fallen by nearly 8.5%, which is markedly smaller than the change experienced by formatting and reveals there are only 7,934 potential legitimate words converted by the stemmer[1]. Also this decrease is not evenly dispersed between the training and testing data,

[1] Based on limited trials with both techniques combined the noise characters eliminated by formatting seem to be part of the cause for this low value, which helps to validate them being used together.

leading to 21,634 common terms or less than 25.2%, which is lower than the base percentage. Of the remaining terms 48,392 are restricted to the training set and 15,845 are exclusive to test documents, giving over 74.8% that serve no purpose during classification.

6.6.2.4 Indexing Statistics

Formatting has produced roughly 18.7% fewer term and document combinations throughout the data, along with a corresponding drop in the average number of terms per document and just over a 15.5% decrease in the range. Considering this in conjunction with the 97% increase of documents containing each term it signifies that the problem space has become denser, which may lead to a different selection of neighbours and candidate assignments during the classification stage.

In line with compaction of the problem space the ratio for duplicate terms has also increased to 1 in 1.76713, making it 24% more likely for a term to be replicated in a document compared to the benchmark experiment. Additionally the chance of encountering a term in the test data compared to the training set has decreased from being 73% more likely to slightly over 62%, though this is still a sizeable margin and hints at a lack of similarity between the compositions of their content.

		Documents Per Term			Terms Per Document			Unique Terms	Total Terms
		Min	Max	Mean	Min	Max	Mean		
Base	Training	1	7115	10.82341	5	658	86.79923	77012	833533
	Testing	1	2316	6.44300	5	613	81.34313	41650	268351
	Combined	1	9328	11.74654	5	658	85.40412	93805	1101884
Format	Training	1	8726	20.74006	5	556	70.81381	32788	680025
	Testing	1	3002	11.56202	5	411	65.40831	18663	215782
	Combined	1	11728	23.13671	5	556	69.43164	38718	895807
Stemmer	Training	1	7115	11.58867	5	604	84.50568	70026	811508
	Testing	1	2348	6.98090	5	603	79.30797	37479	261637
	Combined	1	9328	12.49718	5	604	83.17664	85871	1073145

Table 6.12 : Pre-Process Index Term Statistics

By contrast to the formatters stemming has not had much effect, with less than a 6.4% increase in the average number of documents containing a term and a duplication ratio of 1 in 1.56811, which is only a 4.1% rise in probability above the benchmark value. The difference in likelihood of a testing term being contained by a document in relation to one from the training set is also minor, though it grows

by almost 2.1% to approximately 75.3% and insinuates that the density of unique terms in the testing and training documents has actually diverged slightly.

6.6.2.5 Evaluation Measures

It is immediately obvious that the data cleaning applied by the formatters at the start of the solution has a notable effect on the final results, with all precision, recall and composite values differing from the base figures. By contrast stemming produces similar output to the benchmark, which may be surprising given how prevalent it is in published solutions, though it is rarely used without the data first being prepared by a separate method.

Formatting and the base solution share two common thresholds but the corresponding values are not the same, with macro averaged BEP being 0.04274 lower than its counterpart and micro averaged F1 being 0.11939 higher. The other measures both demonstrate increases as well and the degree of variation between the types of averaging is also similar, as macro averaged F1 only rises by 0.02247 while micro averaged BEP grows by 0.11904.

		Precision	Recall	Threshold	Value
Base	Macro F1	0.57801	0.50506	1	0.53908
	Micro F1	0.63682	0.47825	10	0.54626
	Macro BEP	0.57801	0.50506	1	0.54154
	Micro BEP	0.49925	0.53536	8	0.51731
Format	Macro F1	0.68097	0.47777	2	0.56155
	Micro F1	0.77584	0.58287	10	0.66565
	Macro BEP	0.40682	0.59078	1	0.49880
	Micro BEP	0.61645	0.65626	7	0.63635
Stemmer	Macro F1	0.56963	0.51211	1	0.53934
	Micro F1	0.63604	0.48412	10	0.54978
	Macro BEP	0.56963	0.51211	1	0.54087
	Micro BEP	0.50421	0.54283	8	0.52352

Table 6.13 : Pre-Process Evaluation Measures

Precision appears to be the main reason for disparity in performance as it has been affected the most, with every change being greater than 0.1 in a positive direction aside from a drop in macro averaged BEP, which conincidentally also has the largest variation of 0.17119. Recall is not as consistent with differences ranging from a decrease of 0.02729 for macro F1 to an increase of 0.12090 for micro

BEP, it also has a positive figure of 0.08572 for macro BEP which partly offsets the substantial decline in associated precision.

Stemming utilises the same thresholds as the base results and all values are markedly similar, with micro BEP having the greatest differences in recall and performance, which increase by 0.00747 and 0.00621 respectively, while both macro averages show the biggest drop in precision of 0.00838. Despite most losing precision the performance values are marginally better than the benchmark due to each having an increased level of recall, with the sole exception of macro BEP where the 0.00705 rise in recall does not compensate for the 0.00838 drop in precision and causes a lower overall value. This indicates that stemming makes more predictions for each of the chosen thresholds, managing to correctly assign a larger number of categories though with a higher proportion of error.

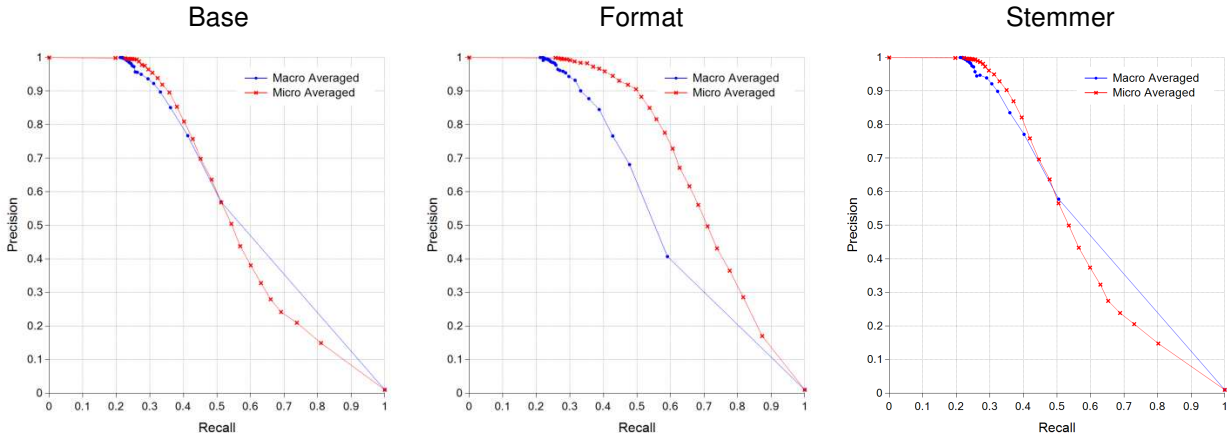


Figure 6.23 : Pre-Process Micro and Macro Precision against Recall

The graphs clearly show that formatting has produced significant improvements for micro averaged thresholds and a small selection of macro averaged points that exhibit higher recall than any of the base scores, though the relatively low precision may affect their choice as optimal thresholds. Also although the micro and macro averaged scores are generally higher than the benchmark after a certain point they display a similar gradient, except for the last few thresholds where micro averaging stays consistent and macro averaging skips to the trivial acceptor. This means that once the initial assignments have been made the denser problem space caused by formatting still tends to struggle in a comparable manner with the remaining categories.

Formatting has allowed the macro averaged measures to be more selective in their assignments, with fewer maximum and mean predictions per category and a lower average per document. Macro F1 also has the largest decline in total and per category predictions with almost 47.4% less than the benchmark, suggesting it avoids excessive assignments to satisfy rare categories, which is supported by the chosen threshold not being the lowest possible and a drop in unique categories. This is not

necessarily the case for macro BEP though, as the decrease in total and per category predictions is less than 5.4% and it uses the lowest threshold that is not the trivial acceptor, while the number of unique categories has also increased by nearly 20.3%.

Micro averaged results have also benefitted from the application of formatting by becoming less selective in their assignments, with the number of unique categories increased by 45% for micro BEP and more than 57.1% for micro F1. This coincides with a decline in the quantity of predictions per category and less than a 1% change in the overall totals, which is the reason for the precision and recall trends that give the superior performance.

		Predictions Per Category			Predictions Per Document			Unique Categories	Total Assigned
		Min	Max	Mean	Min	Max	Mean		
Actual		1	1087	40.29032	0	14	1.13580	93	3747
Base	Macro F1	1	3150	295.13043	0	21	6.17278	69	20364
	Micro F1	1	1146	201.00000	0	3	0.85299	14	2814
	Macro BEP	1	3150	295.13043	0	21	6.17278	69	20364
	Micro BEP	1	1313	200.90000	0	4	1.21794	20	4018
Format	Macro F1	1	2498	181.71186	0	15	3.24977	59	10721
	Micro F1	1	1205	127.95455	0	4	0.85329	22	2815
	Macro BEP	1	2813	232.15663	0	26	5.84086	83	19269
	Micro BEP	1	1420	137.55172	0	5	1.20915	29	3989
Stemmer	Macro F1	1	3091	286.94366	0	21	6.17551	71	20373
	Micro F1	1	1134	219.38462	0	3	0.86450	13	2852
	Macro BEP	1	3091	286.94366	0	21	6.17551	71	20373
	Micro BEP	1	1312	192.09524	0	4	1.22279	21	4034

Table 6.14 : Pre-Process Category Prediction Statistics

Stemming yields values that are much closer to the base figures and has little appreciable change for any of the prediction related statistics. However with the exception of micro F1 it does increase the number of unique categories, while reducing the predictions made for each and having almost no variation in the overall total assignments. In combination these traits show that stemming has enhanced the spread of categories present in the candidate lists generated by KNN, which has improved recall enough to have a positive effect on performance.

6.7 Representation Experiment

Despite a variety of possible options being available the dominance of bag-of-words means it is unusual to encounter any other types of representation technique. Therefore the n-gram and term grouping approaches have been chosen because they are the frequently mentioned in connection with text categorisation, which suggests they are likely alternatives when bag-of-words is not employed. Both can also be applied in a number of ways and attempt to retain limited information about the ordering of terms, though each achieves this in a different manner.

6.7.1 Solution Outline and Components

Each approach has a definable parameter relating to the size of derived terms and is able to take individual or a combination of other representations as input, though in the experiments they have been used in their main recognisable formats. This means that although they could employ the same solution structure they do not in this particular instance, with the first being substituted for the bag-of-terms module of the original base solution and the second being inserted directly after it. In each case any changes made are identical in the training and testing pipelines and aside from components of the representation stage, including an additional information extractor for the term grouping statistics, no aspects of the benchmark have been modified.

6.7.1.1 N-Gram Representation

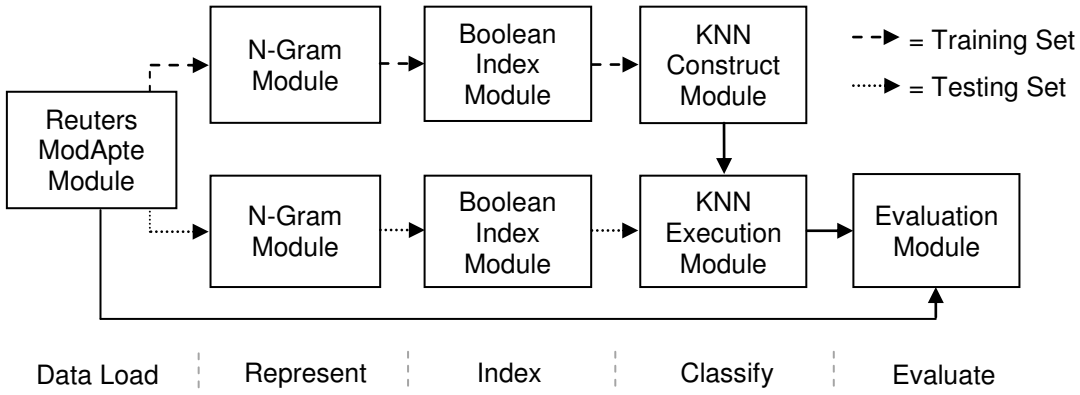


Figure 6.24 : N-Gram Representation Solution Overview

An n-gram overlaps content with those adjacent to it and may consist of any type or quantity of elements, with a maximum size that is controlled by a single definable parameter. In this case the individual characters from the unprocessed document content are supplied as the elements and a

fixed length of 3 has been selected. The algorithm is also set to work from the beginning of the input and ignore any partial items that are created at the end, meaning a portion of the data might potentially be excluded from the final representation and take no further part in the categorisation process.

6.7.1.2 Term Group Representation

A term group has no shared content with those adjacent to it and contains sequential elements that are separated by a customisable set of characters, while the maximum length is determined by a definable parameter. Word pairs have been used as they are the most familiar kind of term grouping and are created by applying a standard bag-of-words representation to produce the input elements, with a single space as the separating character and a maximum size of 2. The algorithm is also configured to work from the start of the input and retain any surplus elements at the end, permitting the inclusion of all data in the final representation though possibly generating terms that are not true word pairs.

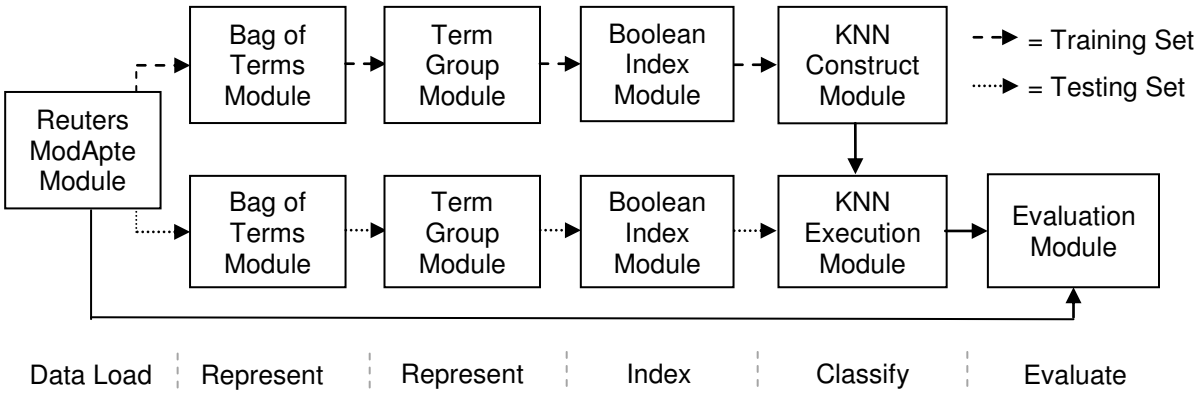


Figure 6.25 : Term Group Representation Solution Overview

6.7.2 Results and Statistics

As previously all values are rounded to the nearest five decimal places except for percentages and module execution times, which are to the nearest millisecond. In addition only the outcomes of stages relevant to these particular experiments or different to the base figures are supplied, meaning no statistics prior to the representation stage or index weightings are shown.

6.7.2.1 Execution Times

By comparing the data loading stages of each experiment, which perform identical amounts of data processing, it is clear that there is discrepancy between the values but shows they are reasonably similar. This is also true for the evaluation stage, though this does not always have exactly the same computational requirement and varies depending on the classification output. Even with these inconsistencies it is immediately obvious that n-gram representation is considerably slower than the other methods, taking almost 484.85% longer in total than the base configuration and 648.99% longer that term grouping. Another point of note is that term grouping is over 13.25% faster than n-gram at generating the final representation, despite it utilising two separate techniques

		Stage						Totals
		Data Load	Represent	Represent	Index	Classify	Evaluate	
Base	Training	-	1.233	-	5.509	9.931	-	16.673
	Testing	-	0.388	-	1.846	759.681	2.363	764.278
	Combined	1.339	1.621	-	7.355	769.612	2.363	782.29
N-Gram	Training	-	6.637	-	32.435	48.327	-	87.399
	Testing	-	2.123	-	10.341	3689.401	2.323	3704.188
	Combined	1.318	8.760	-	42.776	3737.728	2.323	3792.905
Group	Training	-	1.236	4.591	3.354	7.452	-	16.633
	Testing	-	0.350	1.422	1.068	561.243	2.400	566.483
	Combined	1.315	1.586	6.013	4.422	568.695	2.400	584.431

Table 6.15 : Representation Execution Times

Classification is the primary reason for the substantial difference in n-gram execution time, though the representation and indexing stages also contribute. In fact for the results given both have higher variation from the benchmark figures, with representation being approximately 540.41% slower and indexing 581.59%, while classification is only around 485.66%. Without doubt n-grams are clearly a time consuming option, which is due to a rise in the number of derived terms per document and subsequent increase in vector sizes and amount of data involved in the KNN similarity calculations.

Notably different trends are present in term grouping, which has reduced times for the indexing and classification stages, though the additional representation is the second longest process in both the training and testing pipelines. In combination the two representations actually take more than 468.78% longer than the one used by the base solution, but this is offset to give a total decrease of over 25.29%. The largest relative saving in execution time occurs during indexing, which shows nearly a 40% decline in contrast to roughly 26.11% for classification.

6.7.2.2 Representation Statistics

Use of the n-gram representation has led to a considerable increase of almost 614.69% in the number terms per document and total terms, while the maximum and minimum values have also grown, which is expected given the nature of the algorithm. This is part of the cause behind the longer execution times and one of the main factors that prevent n-grams being employed more frequently, as memory consumption becomes prohibitive. Values related to characters per term remain constant for the entire dataset as they have a fixed length and all partial terms are removed.

Due to the small term size fewer character combinations are available and the quantity of unique terms has consequently decreased by over 62.44%, with the biggest drop of nearly 57.61% occurring in the training data against less than 40.25% for the test data. This has generated 22,307 terms that are common to both, accounting for more than 68.51% of the total compared to only 26.5% for the benchmark, but still leaves 10,342 exclusive to the training set and 2,580 to the testing set, meaning about 31.46% have no practical use.

		Characters Per Term			Terms Per Document			Unique Terms	Total Terms
		Min	Max	Mean	Min	Max	Mean		
Base	Training	1	61	5.62807	5	1317	132.71519	77012	1274464
	Testing	1	43	5.57161	5	1673	123.77811	41650	408344
	Combined	1	61	5.61432	5	1673	130.43001	93805	1682808
N-Gram	Training	3	3	3	35	8602	815.47725	32649	7831028
	Testing	3	3	3	36	13430	761.73568	24887	2512966
	Combined	3	3	3	35	13430	801.73570	35229	10343994
Group	Training	1	110	11.04960	3	659	66.60658	277219	639623
	Testing	1	87	10.97377	3	837	62.13822	113239	204994
	Combined	1	110	11.03122	3	837	65.46404	349701	844617

Table 6.16 : Representation Statistics

Average characters per term for groups has almost doubled to over 196.48% of the base figure, while the total number of terms has halved to just under 49.81%, though this is expected because each of them effectively refers to a pair from the original benchmark representation. These values might not be as close to the mid points as predicted since the odd terms left at the end of each document are kept, which is also the reason why the minimum characters per term are still 1.

Unique terms have risen in quantity for the term groups, being more than 372.79% of the base amount, in conjunction with a general decline in terms per document by nearly half. Collectively these changes mean a larger variety of terms are spread less densely throughout the data, so if those shared between the training and testing sets hold little information the classification accuracy is likely to suffer. Further compounding the issue only 11.65% or 40,757 unique terms are common between the datasets, giving just under 88.35% that cannot be utilised, including 72,482 from the test set and 236,462 from the training set.

6.7.2.3 Indexing Statistics

The indexing statistics for n-grams differ from the base solution, with an increase in excess of 1353.10% in the average number of documents per term and corresponding maximum value that shows at least one term is present in over 90.68% of the data. A rise of almost 508.17% in the mean terms per document and total unique document term combinations is also evident, which causes the generation of large vectors and explains the long execution times previously observed.

Duplication of the terms has increased by approximately 32.01% to give a ratio of 1 in 1.84734, which is similar for both the training and testing data, while the chance of encountering a term in the test set has experienced a 50% drop and is now just above 23.09%. These figures show the problem space has become denser than the benchmark and, in combination with the values gathered for the unique terms, demonstrate that n-grams have potential to enhance the similarities between the training and testing sets.

		Documents Per Term			Terms Per Document			Unique Terms	Total Terms
		Min	Max	Mean	Min	Max	Mean		
Base	Training	1	7115	10.82341	5	658	86.79923	77012	833533
	Testing	1	2316	6.44300	5	613	81.34313	41650	268351
	Combined	1	9328	11.74654	5	658	85.40412	93805	1101884
N-Gram	Training	1	8704	129.69742	32	2128	440.95501	32649	4234491
	Testing	1	2996	54.84414	33	1827	413.73325	24887	1364906
	Combined	1	11700	158.94283	32	2128	433.99450	35229	5599397
Group	Training	1	3189	2.20772	3	624	63.73248	277219	612023
	Testing	1	1151	1.72905	3	632	59.35011	113239	195796
	Combined	1	4340	2.31003	3	632	62.61192	349701	807819

Table 6.17 : Representation Index Term Statistics

The quantity of unique terms per document for the group representation is just under 26.69% lower than the benchmark, producing smaller vector sizes and giving the faster execution times. However there is a decline of nearly 48.17% in the level of term duplication to only 1 in 1.04555 and more than an 80.33% decrease in the number of documents per term, which reveals that the problem space has become much sparser. A proportionally low drop in maximum documents per term also suggests that the average values may be optimistic for much of the data, hiding the extent of change. In addition the chance of encountering a test term has increased by 54.7% to roughly 127.98%, meaning the training and testing term densities have diverged.

6.7.2.4 Evaluation Measures

Results are different for the two experiments, with only one common threshold between them and disparate precision and recall figures, though both have higher macro precision and lower recall than the benchmark. Overall performance values also differ and are generally worse than the base solution, with n-gram values being relatively similar, while term groups vary considerably.

		Precision	Recall	Threshold	Value
Base	Macro F1	0.57801	0.50506	1	0.53908
	Micro F1	0.63682	0.47825	10	0.54626
	Macro BEP	0.57801	0.50506	1	0.54154
	Micro BEP	0.49925	0.53536	8	0.51731
N-Gram	Macro F1	0.66347	0.46951	2	0.54989
	Micro F1	0.65677	0.40753	13	0.50296
	Macro BEP	0.66347	0.46951	2	0.56649
	Micro BEP	0.48954	0.49319	9	0.49137
Group	Macro F1	0.85151	0.35919	1	0.50525
	Micro F1	0.34156	0.35095	9	0.34619
	Macro BEP	0.85151	0.35919	1	0.60535
	Micro BEP	0.34156	0.35095	9	0.34625

Table 6.18 : Representation Evaluation Measures

N-grams values are all within 0.07512 of each other and provide minor increases to macro averages but lower micro averages, with the largest drop being 0.04330 in micro F1 and the biggest rise being 0.02495 in macro BEP. Except for a slight decrease of 0.00971 in micro BEP, precision is greater than the base figures and both macro values grow by 0.08546, while micro F1 gains just 0.01995. In all cases recall is markedly worse, counteracting the rise in precision with declines ranging from

0.03555 for the macro figures up to 0.07072 for micro F1. All of the thresholds are different from those of the base solution, though the macro averaged measures still share a low value that is near to the lowest possible.

The term group results significantly range in value, with the biggest difference being 0.25734 between micro F1 and macro BEP. Aside from a 0.06381 gain in macro BEP, they are also inferior to the benchmark, dropping by 0.03385 for macro F1 and up to 0.20007 for the micro averages. Precision is the cause behind such large variation, as it is 0.27350 higher than the base for both macro averages and correspondingly low for the micro averages, changing by 0.29526 for F1 and 0.15769 for BEP. In contrast recall remains consistently poor, having a disparity of only 0.00824 between the figures, but decreasing by 0.14587 for the macro values, 0.12730 for micro F1 and 0.18441 for micro BEP when compared against the originals.

From the precision recall graph it seems n-grams are superior at predicting rarer categories than the base solution, having a selection of macro averaged points available that display greater recall. When assigning common categories it appears to struggle though, failing to match the base recall until the threshold is reduced to relatively low values. This is caused by the generation of more distinguishing features per category, which has succeeded to an extent for rarer cases but blurred the boundaries between the others as their content is already quite similar.

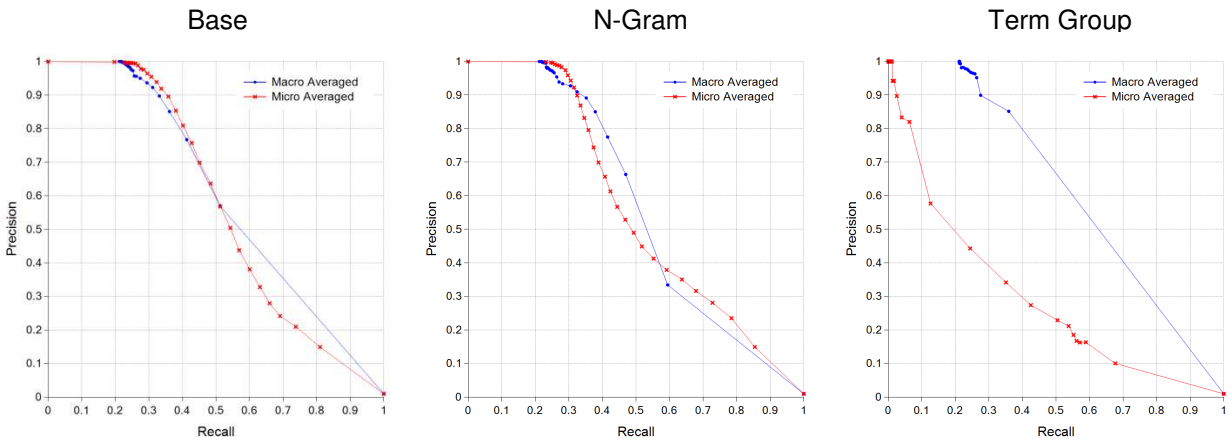


Figure 6.26 : Representation Micro and Macro Precision against Recall

Minor changes to macro averaging are shown for the term groups, with just the smallest thresholds displaying any difference as they sharply reduce in precision to gain recall. This suggests that the rarer categories tend to share word pairs in a similar manner to individual words, though the representation still fails to produce a selection of macro averaged points that contain reasonable recall. Subsequently the lowest threshold above the trivial acceptor is chosen by both corresponding

evaluation measures as it is the only viable choice. In contrast the micro averaged results appreciably differ, revealing that the increased sparseness of the problem space has left too few distinguishing features and the solution immediately struggles to predict the common categories.

		Predictions Per Category			Predictions Per Document			Unique Categories	Total Assigned
		Min	Max	Mean	Min	Max	Mean		
Actual		1	1087	40.29032	0	14	1.13580	93	3747
Base	Macro F1	1	3150	295.13043	0	21	6.17278	69	20364
	Micro F1	1	1146	201.0000	0	3	0.85299	14	2814
	Macro BEP	1	3150	295.13043	0	21	6.17278	69	20364
	Micro BEP	1	1313	200.90000	0	4	1.21794	20	4018
N-Gram	Macro F1	1	2315	201.90323	0	14	3.79448	62	12518
	Micro F1	1	1360	122.36842	0	3	0.70476	19	2325
	Macro BEP	1	2315	201.90323	0	14	3.79448	62	12518
	Micro BEP	1	1606	151.00000	0	5	1.14429	25	3775
Group	Macro F1	1	3299	495.82353	1	14	7.66505	51	25287
	Micro F1	1	2431	550.00000	0	3	1.16702	7	3850
	Macro BEP	1	3299	495.82353	1	14	7.66505	51	25287
	Micro BEP	1	2431	550.00000	0	3	1.16702	7	3850

Table 6.19 : Representation Category Prediction Statistics

A decrease of nearly 31.59% in mean predictions per category and almost 38.53% in the number of predictions per document implies that n-grams lessen the scope of macro averaged measures. This can be confirmed by more than a 10.14% decline in unique categories and a lower maximum quantity of predictions per document that is equal to the true value. However, while this leads to slightly better performance, the sizeable difference in actual and predicted unique categories clearly shows a lack of diversity, while the smaller amount of total assignments highlights that excessive predictions are still made in an attempt to raise recall.

With regard to micro averaging n-grams have broadened the range of categories and decreased the predictions for each, with declines of around 39.12% for F1 and 24.84% for BEP, but even the rise in unique categories of over 35.14% by micro F1 is not enough to permit a reasonable level of recall. It is also apparent that micro F1 has become more selective with assignments, reducing the base total and number of predictions per document by about 17.34%, despite them already being lower than the real values. There is a similar trend for micro BEP as well, though the benchmark figures are higher than the actual values and get decreased less than 6.05%.

Although the lowest viable threshold is employed by both term group macro averaged measures, the number of unique categories has unexpectedly fallen by over 23.88%, which is likely due to a general lack of discriminating attributes amongst those unassigned. Otherwise the total assignments and predictions per document have increased by almost 24.18% and the predictions per category by over 68.00%, as is the standard trend when no thresholds are available with reasonable recall. However, despite the abundant assignments the precision still remains above 85%, meaning several categories must contain distinctive word pairs that assist in their accurate prediction.

The term group micro averaged measures have also become more selective, with unique categories dropping by 50% for F1 and 65% for BEP, which is again probably due to the absence of sufficiently descriptive terms. Unlike the macro averaged results the precision is affected considerably, suffering continual decline as the limited categories are erroneously assigned to progressively more documents in an attempt to improve recall. It is this behaviour that causes the trend depicted in the graph and the quantity of predictions per category to grow substantially larger than the base figures, both rising by over 173.6%, even though the total assignments and predictions per document show relatively little change.

6.8 Indexing Experiment

Similar to other stages in the text categorisation process there are a range of techniques available for indexing, which each have a selection of possible weighting algorithms for the generation of machine readable document vectors. Due to its popularity the Vector Space Model (VSM) has been employed in these experiments, combined with a Normalised Term Frequency weighting and a simple version of Term Frequency- Inverse Document Frequency (TF-IDF). Both are fundamentally based on counting terms, but the normalisation restricts scope to individual documents, while TF-IDF derives occurrence statistics from a reference collection.

6.8.1 Solution Outline and Components

Similar solutions are used with the same basic design and components as the benchmark, except the standard Boolean indexing modules are replaced by the particular techniques being investigated. One additional link is also required by the TF-IDF weighting, which connects the representation module of the training pipeline to the indexing module of the testing pipeline, making the training data become its reference collection.

6.8.1.1 Normalised Term Frequency Indexing

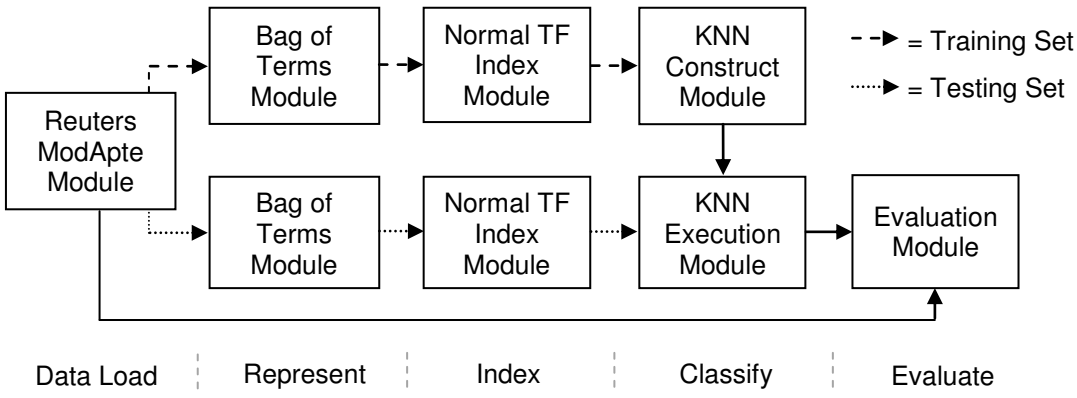


Figure 6.27 : Normalised Term Frequency Indexing Solution Overview

Term frequency weightings are based on the principal that the more commonly a term appears in a document the more discriminative it is for determining the topics described by the content. Whereas the intention of normalisation is to make every weighting adhere to a scale that is universal throughout the dataset, whilst still accurately reflecting term importance on a per document basis. A number of techniques are possible to achieve this, but the one used in the experiment simply divides each term

frequency by the maximum frequency encountered in the document it was derived from. This means all weightings are given a value between 0 and 1, where only the most common term has a weighting of 1 and the others receive values that become lower as the term appears less often.

6.8.1.2 TF-IDF Indexing

Complimenting the notion of Term Frequency (TF) that common terms within a single document are highly informative, the concept of Inverse Document Frequency (IDF) is that terms occurring in fewer documents must also have good discriminative capability. Therefore the TF-IDF weighting favours terms that are present many times in only a small range of documents, while giving lower weightings to those that rarely occur in a specific document or that appear in a wide selection.

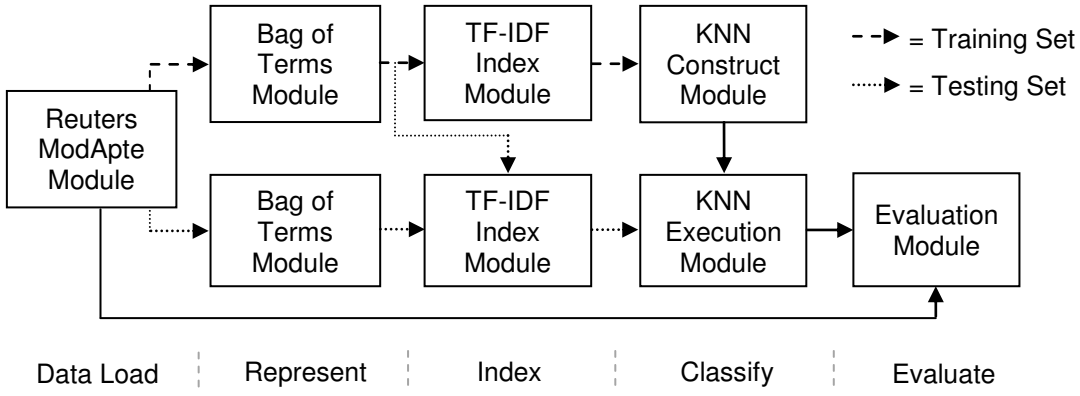


Figure 6.28 : TF-IDF Indexing Solution Overview

Several algorithms are referred to as TF-IDF throughout literature but the experiment employs the most common form, which is the raw term frequency count multiplied by the natural logarithm of dataset size divided by the number of documents containing the term. This means no normalisation is applied, while required dataset and document frequency information for IDF calculations is always obtained from the training set, as this ensure all values use a relative scale[1].

6.8.2 Results and Statistics

As before values are rounded to the nearest five decimal places except for percentages and module execution times, which are given to the nearest millisecond. Due to the absence of pre-processing

[1] Weighting dynamics of the test data change if it uses itself as the reference collection, with limited trials showing a decrease in all macro and micro averaged performance measures, though the precision against recall trends remained almost identical.

and retaining the same bag-of-words representation as the benchmark, all figures prior to indexing are unaltered and the statistics for these stages are not duplicated here. Reuse of the Vector Space Model to generate document vectors also means their structure is unchanged, consequently only data relating to the different weightings is supplied.

6.8.2.1 Execution Times

Overall the results indicate that Normalised Term Frequency is approximately 20.08% slower that the benchmark, whereas TF-IDF is roughly 29.24% slower. As expected the differences between the data loading and representation stages of each experiment are trivial because they are all performing the same operations. Indexing times for the normalised TF and base solution are also similar with less than a 9.15% discrepancy, which could potentially be due to abnormally inaccurate timings. However, TF-IDF indexing is definitely slower, being more than 138.08% above the benchmark, showing that considerable overhead is required to gather the term occurrence statistics.

		Stage					Totals
		Data Load	Represent	Index	Classify	Evaluate	
Base	Training	-	1.233	5.509	9.931	-	16.673
	Testing	-	0.388	1.846	759.681	2.363	764.278
	Combined	1.339	1.621	7.355	769.612	2.363	782.290
Norm TF	Training	-	1.218	6.013	10.891	-	18.122
	Testing	-	0.425	1.863	915.079	2.534	919.901
	Combined	1.329	1.643	7.876	925.970	2.534	939.352
TF-IDF	Training	-	1.236	10.963	11.566	-	23.765
	Testing	-	0.416	6.548	976.033	2.952	985.949
	Combined	1.313	1.652	17.511	987.599	2.952	1011.027

Table 6.20 : Indexing Execution Times

Classification construction and execution times have increased for both weighting methods, though as construction only involves storing training vectors the rise for this must be caused by extra characters being retrieved from and inserted into the framework repository. This may be partially responsible for the variation in execution as well, but with Normalised Term Frequency almost 20.46% and TF-IDF nearly 28.48% larger than the base figure the absolute differences are too substantial for it to be the sole reason and evidently more data is being processed. As nearest neighbour discovery remains constant throughout experiments, since the quantity and length of vectors are identical, it must mean more candidate categories are being found, which would also explain the gain in evaluation times.

6.8.2.2 Indexing Statistics

Due to using the Vector Space Model and same bag-of-words representation as the benchmark all index term statistics are identical to the base solution, so vector structures, chance of term duplication and relative density of training and testing sets have not changed. Weightings are different though and no longer mimic term statistics as they did previously with Boolean indexing, which merely sets all term weights to 1. Instead they are based on the actual term frequencies within each document, but despite this aspect being shared by both techniques under investigation they are difficult to compare directly because of the way they utilise it.

		Weighting Per Term			Weighting Per Document		
		Min	Max	Mean	Min	Max	Mean
Base	Training	1	1	1	5	658	86.79923
	Testing	1	1	1	5	613	81.34313
	Combined	1	1	1	5	658	85.40412
Norm TF	Training	0.01075	1	0.19195	3	48	16.66121
	Testing	0.00408	1	0.19421	3.5	55.33333	15.79784
	Combined	0.00408	1	0.19250	3	55.33333	16.44045
TF-IDF	Training	0.29987	331.45467	5.27189	23.12359	4752.48599	457.59556
	Testing	0	1088.39854	4.71865	23.01111	6966.12449	383.82946
	Combined	0	1088.39854	5.13715	23.01111	6966.12449	438.73380

Table 6.21 : Indexing Term and Document Weighting Statistics

An interesting distinction between Normalised Term Frequency and the base solution is that figures for the minimum weighting per document do not relate to those with the shortest lengths and are in fact due to obscure content structuring. They are caused by a term used purely for formatting that is present an excessive number of times, which subsequently reduces the weightings of all other terms. Low values for the minimum and mean weightings per term reflect this behaviour, showing that noise in the data can have a considerable affect when this form of normalisation is applied[1]. In contrast the maximum weighting per term is always set to 1, as this is the weight associated with the most common term of every document that becomes the point of reference for normalisation.

[1] Limited trials with raw term counts gave improved performance over normalised values for every evaluation measure, whilst exhibiting similar precision against recall trends.

With TF-IDF the minimum weighting per document does relate to those having the shortest length and potential noise in the data actually has an opposite effect on the calculation. This is apparent in the maximum weighting per term for the testing set, where the abnormally large value is due to a term with moderately low occurrence in the reference collection and unusually high frequency in a single document. Several other terms also display similar traits and are responsible for elevation in relative difference between the mean and maximum weighting per document, which increases by over 106.08% compared to the base solution, while Normalised Term Frequency drops by almost 54.48%.

Another area of note is the reason behind the 0 minimum weighting per term in the TF-IDF testing set, which occurs when the reference collection does not contain an encountered term. This invalidates the IDF part of the calculation as it attempts to divide by 0 and consequently the algorithm resorts to the default weighting. It would also be possible to set the value to an arbitrarily high number, but unless explicitly instructed otherwise the classification process effectively ignores any terms exclusive to only one dataset.

		Global Term Weighting			Total Weight
		Min	Max	Mean	
Base	Training	1	7115	10.823411	833533
	Testing	1	2316	6.44300	268351
	Combined	1	9328	11.74654	1101884
Norm TF	Training	0.01075	5139.61699	2.07757	159997.61520
	Testing	0.00408	1598.53166	1.25131	52117.07012
	Combined	0.00408	6738.14866	2.26123	212114.68532
TF-IDF	Training	9.16983	21044.37804	57.05981	4394290.16040
	Testing	0	7995.03554	30.40224	1266253.37382
	Combined	0	27584.33638	60.34373	5660543.53423

Table 6.22 : Indexing Global Weighting Statistics

Considering the variation in statistics for each method, with regard to the relative change between the training and testing data and markedly different causes for values that initially seem similar, it is clear they all alter the problem space. However, unlike other categorisation stages, where information is in a format that is typically easier to understand, it is hard to speculate how indexing will affect overall performance. Though it is obvious that removing noise from the dataset could significantly change the outcome, but it is difficult to determine exactly what should be filtered.

6.8.2.3 Evaluation Measures

As before there are notable differences between the precision, recall and final evaluation measures of all the experiments, but just half of the values show improved performance over the benchmark solution. While no obvious trends are apparent in the figures, it is plain that standard TF-IDF by itself yields low micro averaged values and is not a particularly superior weighting algorithm[1], which is surprising considering its high level of popularity.

		Precision	Recall	Threshold	Value
Base	Macro F1	0.57801	0.50506	1	0.53908
	Micro F1	0.63682	0.47825	10	0.54626
	Macro BEP	0.57801	0.50506	1	0.54154
	Micro BEP	0.49925	0.53536	8	0.51731
Norm TF	Macro F1	0.68137	0.41365	3	0.51478
	Micro F1	0.61180	0.51188	8	0.55740
	Macro BEP	0.53787	0.47301	2	0.50544
	Micro BEP	0.55677	0.54577	7	0.55127
TF-IDF	Macro F1	0.62329	0.49204	1	0.54994
	Micro F1	0.14142	0.63598	2	0.23139
	Macro BEP	0.62329	0.49204	1	0.55767
	Micro BEP	0.18204	0.17748	9	0.17976

Table 6.23 : Indexing Evaluation Measures

Despite Normalised Term Frequency measures not sharing thresholds with the base experiment there is only moderate variation in their final values, as macro averages decline by 0.02430 and 0.03610 for F1 and BEP respectively, while micro averages increase by 0.01114 and 0.03396. The differences between F1 and BEP are also minimal for both types of averaging, being separated by less than 0.01 in either case, though the corresponding precision and recall values alter significantly. Unlike the benchmark precision is higher than recall for all measures, but their range in disparity is larger and goes from 0.26772 for macro F1 to 0.01100 for micro BEP, compared to 0.15857 for micro F1 and 0.03611 for micro BEP, making it more sensitive to bias of either value.

Only the macro averaged measures of TF-IDF share the same thresholds as the base solution and both show minor improvement, with 0.01086 for F1 and 0.01613 for BEP, which is due to a rise of 0.04528 in precision as recall actually drops by 0.01302. Conversely the micro averages are much

[1] A version of TF-IDF using Normalise Term Frequency instead of raw term counts was also tested and gave minor improvement to micro averages, but they still remained lower than the base figures.

lower, decreasing by 0.31487 for F1 and 0.033755 for BEP, being caused mainly by corresponding declines in precision of 0.47038 and 0.31721. This suggests TF-IDF negates the distinctive features used to discriminate between common categories, while the low micro F1 threshold and micro BEP recall imply this is not necessarily counteracted by an excessive number of assignments.

Micro BEP is the smallest value because of an inferior combination of precision and recall, which is a consequence of the algorithm forcing selection of a threshold where they are similar. This means it has been unable to compensate for the poor quality of one by choosing a point that maximises the other, leading to a difference of only 0.00456, unlike micro F1 where recall is 0.49456 higher than precision. However, as both micro averaged measures exhibit low precision over a large variation in recall it indicates that TF-IDF generally experiences difficulty in making the right predictions.

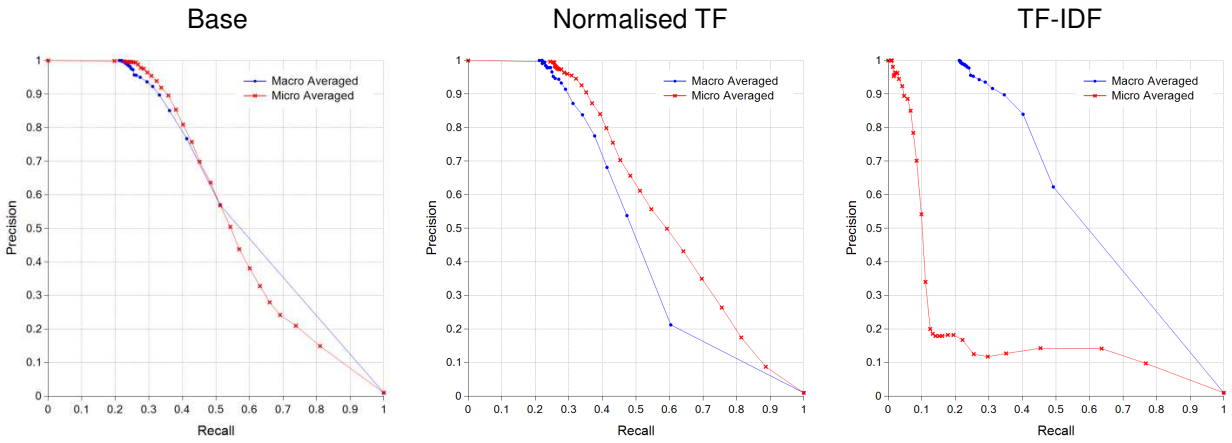


Figure 6.29 : Indexing Micro and Macro Precision against Recall

Precision against recall trends explain the overall evaluation measures gathered, with Normalised Term Frequency having a greater variation between the lower thresholds for macro averaging, but quickly dropping in precision to make them worse than the benchmark. Micro averaging is initially inferior as well, then steadily gains recall at an increased rate that leads to improvement until the last two points. This indicates that the base solution typically has more correct predictions at higher thresholds, while Normalised Term Frequency gradually becomes superior before sacrificing accuracy to cover a larger range of categories at the lowest.

TF-IDF has similar trends for macro averaging as the base results and also lacks available thresholds with reasonable levels of recall, meaning it shares the same optimal value of the lowest one above the trivial acceptor. However, the last few points have increased precision and only slight reduction in corresponding recall, which proves enough to boost the final evaluation measures. In contrast micro averaging is considerably worse and suffers an immediate decline in precision as the majority of initial

predictions are erroneous. After approximately half the thresholds follow this pattern it changes and recall begins to rise, though there is little change in precision.

This behaviour shows that TF-IDF generally makes more mistakes than the benchmark, but is better at predicting some of the categories because it manages to maintain macro averaged precision while micro averaged precision falls. Once it reaches lower thresholds it also displays marked improvement and even produces proportionally more correct assignments for certain values, although the damage is already done and it is never able to achieve enough positive gain to compensate for the previously encountered issues.

		Predictions Per Category			Predictions Per Document			Unique Categories	Total Assigned
		Min	Max	Mean	Min	Max	Mean		
Actual		1	1087	40.29032	0	14	1.13580	93	3747
Base	Macro F1	1	3150	295.13043	0	21	6.17278	69	20364
	Micro F1	1	1146	201	0	3	0.85299	14	2814
	Macro BEP	1	3150	295.13043	0	21	6.17278	69	20364
	Micro BEP	1	1313	200.9	0	4	1.21794	20	4018
Norm TF	Macro F1	1	1928	202.67925	0	14	3.25614	53	10742
	Micro F1	1	1331	116.11111	0	5	0.95029	27	3135
	Macro BEP	1	2185	253.23188	0	21	5.29645	69	17473
	Micro BEP	1	1413	122.43333	0	5	1.11337	30	3673
TF-IDF	Macro F1	1	3255	399.39189	0	23	8.95878	74	29555
	Micro F1	1	3250	312.03704	0	12	5.10761	54	16850
	Macro BEP	1	3255	399.39189	0	23	8.95878	74	29555
	Micro BEP	1	2956	228.31250	0	3	1.10731	16	3653

Table 6.24 : Indexing Category Prediction Statistics

Predictions statistics for Normalised Term Frequency confirm a drop in diversity for macro F1, with reductions in unique categories of almost 23.19%, total predictions by over 47.25% and the maximum per document by a third. Whereas the decline in mean predictions per category of fewer than 31.33% and a higher corresponding drop in the maximum per category of over 38.79% suggests that those remaining may be dispersed more evenly. Macro BEP has no change in the range of diversity, though there is a decrease of approximately 14.19% in predictions per document, mean per category and in total. These are combined with more than a 30.63% decline in maximum predictions per category, which implies those removed were probably responsible for biasing the distribution.

Conversely for micro averaging the variety of predictions increases compared to the benchmark with rises in unique categories of almost 92.86% for F1 and 50% for BEP, while the corresponding totals have less change gaining roughly 11.41% and decreasing by fewer than 8.59% respectively. Due to this rise in diversity and overall predictions the mean per category has declined by more than 42.23% for F1 and nearly 39.06% for BEP, though the maximum values have grown by 16.143% for F1 and 7.616% for BEP. Only the precision of the F1 measure is negatively affected by these alterations and everything else is improved by them.

TF-IDF macro averaging share same thresholds as the base solution and shows that significantly more assignments were made, with a growth of over 45.13% in the total predictions and mean per document, which helps to explain the greater execution times. This increase also leads to an excess, with nearly 8 times the number of assignments necessary, raising maximum predictions per document by over 9.52%, the maximum per category more than 3.33% and the average per category by almost 35.33% when compared to base figures. However, it also improves the unique range by around 10.45% and this is the most likely reason for the high level of precision experienced, as though it has difficulty accurately predicting common categories it seems to get them right for some rarer ones.

From the precision against recall trend it can be seen that micro averaged TF-IDF initially has trouble making correct predictions but the sharp fall in precision at the start has a slight increase for the larger recall values, which is the reason for the F1 measure selecting a low optimal threshold. It is this choice that causes differences in the predictions, as the total number and mean per document are over 498.79% more than the benchmark solution. All other figures also experience such change, with the maximum per document rising 300%, the maximum per category nearly 183.60% and the mean per category by over 55.24%. Unique categories increase by roughly 280.57% too and this, combined with the volume of predictions, is why the recall is relatively high, though it is still almost 41.94% lower the actual quantity required.

In contrast micro BEP does not share a low threshold similar to F1 and at first glance may appear superior, with the total amount of predictions and mean per document being just under 2.51% short of the actual values and slightly over 9.08% below the base. However, the amount of unique categories drops by 20% and an increase of more than 125.13% in the maximum predictions per category shows nearly 80.92% of the total assignments are for only one specific category, which according to the corresponding precision value are mostly incorrect. This highlights dominance of a single category in the training data and a lack of capability by TF-IDF in this particular situation to generate term weights that adequately discriminate between documents.

6.9 Dimension Reduction Experiment

Dimensional reduction is possibly the most researched area of text categorisation, though is regularly treated as part of a classification algorithm and the two stages are often merged together. It is also commonly acknowledged as having a potentially significant affect on overall performance and a number of publications focus on it directly [16][146][18][149]. Consequently only Document Frequency has been presented for completeness, confirming that dimensional reduction can be deployed as an independent element and verifying the impact it can have.

6.9.1 Solution Outline and Components

The solution configuration keeps all of the original benchmark components unaltered but inserts two additional modules. Dimensional reduction is performed on the training data by the first and this is placed between the indexing and classification modules of the training pipeline. While the second uses the output of this to create a list of terms that it supplies to the indexing module on the testing pipeline, which subsequently employs them to filter the test data.

6.9.1.1 Document Frequency Reduction

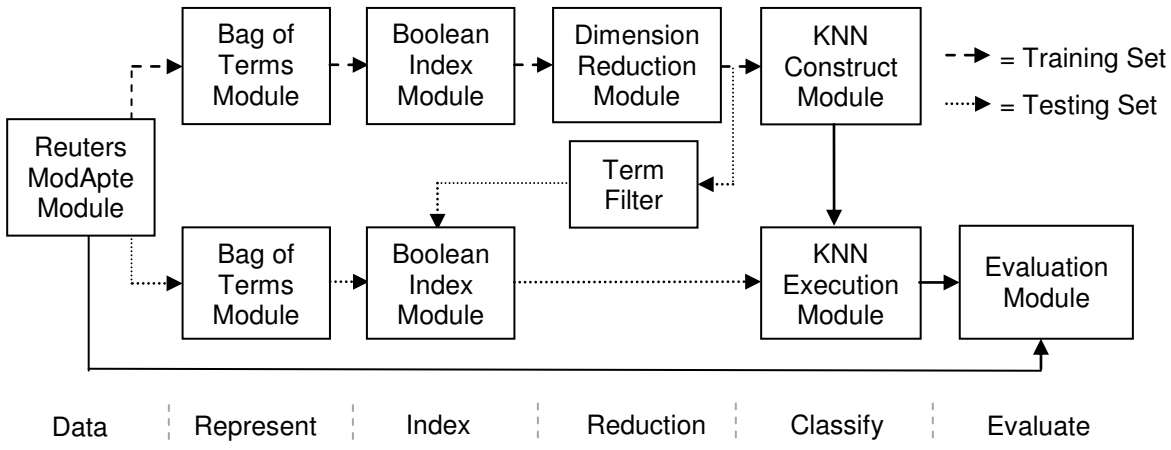


Figure 6.30 : Dimension Reduction Solution Overview

Being one of the simplest dimensional reduction techniques Document Frequency is a form of feature selection and merely records the number of documents each term occurs in and only retains those that meet certain thresholds. In the majority of published trials only a minimum constraint is applied to remove rarer terms, though both minimum and maximum boundaries are present in this experiment

and have been set to the values of 2 and 400 respectively. This means terms occurring in less than 2 or more than 400 documents are ignored and the remainder are used to train the classifier.

6.9.2 Results and Statistics

As with all the other experiments values are rounded to the nearest five decimal places except for percentages and module execution times, which are given to the nearest millisecond. Pre-processing and representation statistics are not given as they are identical to the base, while the indexing figures refer those after reduction has taken place. Representation data matches that of the benchmark in this case because dimensional reduction is applied post indexing, though for this particular technique it is also possible to perform it immediately following the representation stage in either the training or testing pipelines.

6.9.2.1 Execution Times

For the testing data reduction time refers to producing the terms that become the filtering reference during indexing, as no explicit reduction module is present in the testing pipeline. Technically filtering is not required with Boolean weighting as the classifier simply ignores any terms not present in the training data, though it is necessary for some complex algorithms that alter weights based on term statistics. It also serves to speed up execution of the KNN classifier since removing terms from an index is faster than repeatedly checking for them during similarity calculations. This is evident in trials where the filter is removed, as classification time increases by roughly 33% while the quicker indexing and lack of filtering make the overall testing process around 12.5% slower.

		Stage						Totals
		Data Load	Represent	Index	Reduction	Classify	Evaluate	
Base	Training	-	1.233	5.509	-	9.931	-	16.673
	Testing	-	0.388	1.846	-	759.681	2.363	764.278
	Combined	1.339	1.621	7.355	-	769.612	2.363	782.290
DF	Training	-	1.191	5.603	10.535	6.264	-	23.593
	Testing	-	0.357	51.728	5.324	438.033	2.375	497.817
	Combined	1.310	1.548	57.331	15.859	444.297	2.375	522.72

Table 6.25 : Reduction Execution Times

Data loading, representation and the training index modules undertake exactly the same processes in both cases and their times reflect this showing relatively small disparity. Dimensional reduction seems

a comparatively slow process, taking the longest time in the training pipeline and being accountable for almost half the total. It also has a considerable affect on the testing indexer due to filtering with an increase of over 2702.17% above the base value, which rises to more than 2990.57% if production of the filtered term list is included. By comparison classification times have dropped by over 36.92% for construction and almost 42.34% for execution, giving a combined reduction of around 42.27%.

Training time has growing by 41.50422%, but testing declines by 34.8644% and due to the relatively large time of the classifier execution stage costs incurred by dimensional reduction are negated and the complete system is around 33.18% quicker. A change of this magnitude suggests an appreciable amount of data has been eliminated as noise and highlights the possible savings in regard to solution efficiency, which might be appealing to time critical applications if effectiveness remains acceptable.

If dimensional reduction is applied prior to the indexing stage these timings would alter, speeding up indexing but increasing testing representation and also the reduction for both pipelines, as the entire content needs to be processed rather than just compact indexes. This means pre indexing reduction will always be slower or at best equal to post indexing and depending on composition of the data it could entail a sizeable rise in overall execution time [1]. However, the faster option is not possible for all methods and provides fewer statistics relating to modified document content, unless term frequency is used for the weighting.

6.9.2.2 Indexing Statistics

As Boolean indexing is being used the weight and term statistics for the indexing stage match and the individual term weightings for each document are always 1, so only the term statistics are given here. These clearly show the affect of dimensional reduction, with a decrease of almost 43.04% in mean terms per document and total term and document combinations, which indicates that rare and highly frequent terms account for a substantial volume of the data.

Decline in unique terms is also evident with a drop of over 63.78%, leaving 19,540 common to both datasets that account for nearly 57.52% of those present, compared to 26.5% for the benchmark. All remaining 14,432 are exclusive to the training set, as this is always the reference data and any terms it does not include are removed from the testing set when filtering is applied, meaning about 42.48% have no practical use during classification.

Influence of the Document Frequency boundaries is also visible, with minimum training documents per term equal to the lower constraint and the maximum being just under the upper limit. Though as

[1] Limited trials to explore this effect on the experiment data showed the testing representation stage became over 230 times slower than the benchmark, though execution of the overall solution was still approximately 30% faster due to savings made during classification.

revealed by the corresponding minimum documents per term for the test figures, these restrictions are not enforced on the test data because it is not part of the reference set. Removing the potential noise has raised the overall mean value by more than 57.29% as well, which denotes a greater proportion of rare terms have been eliminated and will be amplified by the lack of initial data cleaning. This is reflected in the decreased minimum and maximum terms per document too, with the larger change to the testing statistics again due to referencing the training set for the filter.

		Documents Per Term			Terms Per Document			Unique Terms	Total Terms
		Min	Max	Mean	Min	Max	Mean		
Base	Training	1	7115	10.82341	5	658	86.79923	77012	833533
	Testing	1	2316	6.44300	5	613	81.34313	41650	268351
	Combined	1	9328	11.74654	5	658	85.40412	93805	1101884
DF	Training	2	399	14.22330	1	492	50.31698	33972	483194
	Testing	1	353	7.39452	1	336	43.79782	19540	144489
	Combined	1	657	18.47648	1	492	48.65005	33972	627683

Table 6.26 : Reduction Index Term Statistics

Two equivalent calculations are typically used to measure the degree of reduction, ‘aggressivity’ and ‘reduction factor’, but both require the number of terms throughout an entire dataset before and after dimensional reduction occurs, which is only possible if it happens prior to indexing. Corresponding directly to the difference in size of data between the stages, these values are just a single statistic that can be deduced when the proposed architecture is employed. A provisional test was conducted to verify this and the necessary values were extracted, resulting in a total reduction factor of 0.54873, comprising of 0.54525 for the training set and 0.55958 for the testing set.

6.9.2.3 Evaluation Measures

Final results have mixed performance, with macro averages showing marginal improvement and micro averaging declining by a significant amount in comparison to the benchmark. This variation is due to precision as it is high for macro averaging and low for micro averaging, while all display a similar and relatively poor level of recall. Selected thresholds are also markedly low for every measure and the macro values match the base, which is the lowest option possible other than the trivial acceptor.

Macro averaged performances increase by about 1.05% for F1 and under 5.07% for BEP, caused by a rise of almost 17.32% in precision that is partially offset by a decline in recall of nearly 7.18%, which prevents a more substantial overall change. These values imply that fewer total predictions are being

made, but with a greater degree of accuracy, likely because removal of common terms means those occurring less frequently throughout the data have a larger influence when nearest neighbours are determined during classification.

In contrast micro averaged measures demonstrate worse performance than the base solution, with F1 dropping more than 16.17% and BEP by almost 14.60%. A fall in precision of over 27.45% is the main reason F1 is inferior, as recall decreases less than 6.86%, whereas both are lower for BEP, having declines of over 13.12% for precision and just below 16.07% for recall. Since all values are smaller it indicates predictions are not as accurate or able to cover as many assignments as the benchmark, which may be due to the elimination of discriminative terms that are prevalent in common categories. For instance one category is associated with 2,877 test documents, so an upper limit that deletes terms occurring in 400 or more could damage the predictive power of this category, highlighting a potential flaw in Document Frequency.

		Precision	Recall	Threshold	Value
Base	Macro F1	0.57801	0.50506	1	0.53908
	Micro F1	0.63682	0.47825	10	0.54626
	Macro BEP	0.57801	0.50506	1	0.54154
	Micro BEP	0.49925	0.53536	8	0.51731
DF	Macro F1	0.75118	0.43329	1	0.54958
	Micro F1	0.36228	0.40966	3	0.38452
	Macro BEP	0.75118	0.43329	1	0.59223
	Micro BEP	0.36802	0.37470	4	0.37136

Table 6.27 : Reduction Evaluation Measures

Macro precision and recall trend is more linear than the base results, though there is still an issue achieving sufficient recall and it is worse over the lower thresholds. This may be caused by removal of rare terms representing the majority of discriminative information for the content that contain them, as there are 42,766 terms in the training set that appear in only one document and are subsequently erased by the Document Frequency lower boundary. In trials exploring alteration of these boundaries enlarging the upper constraint had little effect on macro averaging, but did gradually increase recall while precision dropped at a higher rate, whereas raising the lower limit amplified this behaviour.

Poor recall is also apparent in micro averaging, with it initially rising for the largest thresholds before a high loss in precision occurs, followed by a period of minimal change until the last few values where recall grows and precision falls in similar proportions. This pattern is undoubtedly due to excessive deletion of frequent terms that specifically relate to common categories, implying that the Document Frequency upper boundary is too restrictive in this case. Investigation confirmed this and revealed

that increasing the maximum limit improved results, with the trend moving towards that of the base solution and then surpassing it to produce slightly higher micro precision and recall throughout all thresholds. Raising the lower limit also enhanced recall, but the variation is smaller and mostly affects the mid range thresholds.

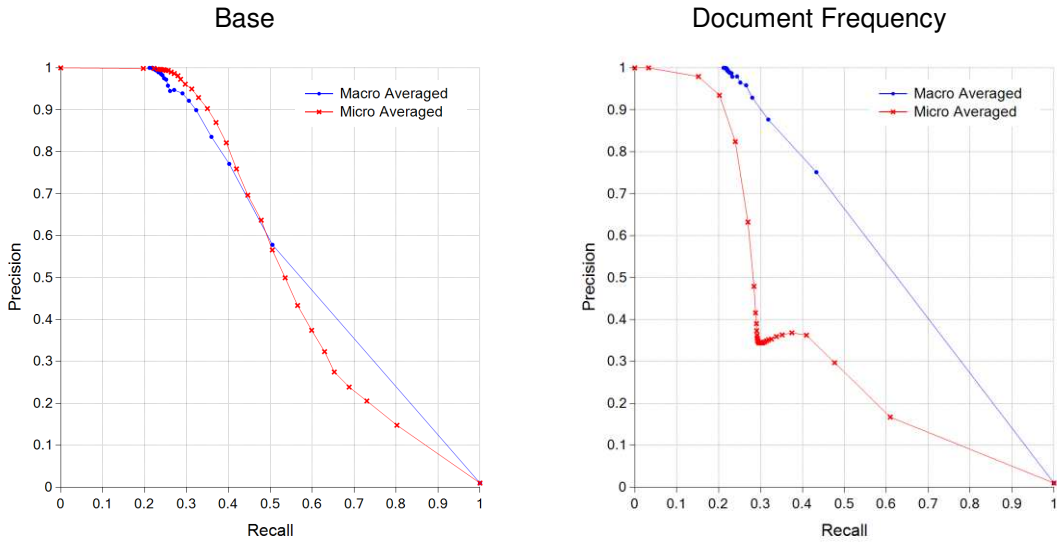


Figure 6.31 : Reduction Micro and Macro Precision against Recall

From the maximum predictions per category it can be seen that nearly all 3,299 test documents have the same category applied to them by every measure, though despite this being the one that should be assigned most often its excessive use means precision remains low. It accounts for almost 24.08% of the total macro averaged predictions, compared to less than 15.47% for the benchmark, whilst also representing just under 77.70% for micro F1 and over 86.26% for micro BEP, which correspond to base values of around 40.72% and 32.68% respectively.

Unique categories has declined by over 14.49% for the macro measures, while the total and mean predictions per document drop approximately 32.80%, so the category set is more restrictive and there are roughly 21.41% fewer predictions made for each. Conversely the set of unique categories rises for both micro criteria and F1 demonstrates an increase of almost 121.43%, combined with a rise in total predictions of about 50.57% and double the number of maximum per document. Micro BEP predictions figures do not change as much and the overall quantity falls little more than 5.05%, but category diversity enlarges 35% and there are nearly 29.67% less predictions for each, while the maximum per document has grown by half.

Given the low threshold values the small amount of total predictions may be surprising, especially considering the comparatively substantial drop for macro averaged measures and increase in unique categories for micro averaging. However, this rise in unique categories is only due to the thresholds

allowing disparate nearest neighbours to be used during evaluation, while the general decrease in predictions is because documents containing rare or highly frequent categories are being disregarded by the KNN classifier. These are clear indicators that the Document Frequency boundaries are overly restrictive and lead to the removal of too many informative terms in this situation.

		Predictions per Category			Predictions per Document			Unique Categories	Total Assigned
		Min	Max	Mean	Min	Max	Mean		
Actual		1	2877	83.89565	0	16	1.00469	115	9648
Base	Macro F1	1	3150	295.13043	0	21	6.17278	69	20364
	Micro F1	1	1146	201	0	3	0.85299	14	2814
	Macro BEP	1	3150	295.13043	0	21	6.17278	69	20364
	Micro BEP	1	1313	200.9	0	4	1.21794	20	4018
DF	Macro F1	1	3295	231.93220	0	17	4.14792	59	13684
	Micro F1	1	3292	136.67741	0	6	1.28433	31	4237
	Macro BEP	1	3295	231.93220	0	17	4.14792	59	13684
	Micro BEP	1	3291	141.29630	0	6	1.15641	27	3815

Table 6.28 : Reduction Category Prediction Statistics

6.10 Chapter Summary

The experiments undertaken in this chapter were designed to demonstrate how the stages previously proposed could be employed in a practical setting and also to reveal the potential impact changes to any of them individually might have on categorisation performance. Only a limited selection of the possible options is represented by the utilised components and results gathered, but the importance of separating the stages and feasibility of the recommended solution structure are still validated.

Every aspect of the outlined architecture has been shown to have the potential to affect not just the particular stage it is associated with but all of those that follow, including classifier predictions and the outcome of applied evaluation measures. Segmentation of the algorithms has also made construction of solutions a simple and rapid process, with a single base configuration forming the foundation of every trial conducted while relevant modules are easily added or exchanged as necessary. These traits, combined with the ability to exactly duplicate any results, verify the primary benefits that can be gained and confirm that none of the stages should be merged together, disregarded or intentionally ignored as trivial.

Aside from the expected advantages a number of others are apparent as well, as statistics gathered from stages prior classification indicate what is happening to the data and allow certain hypotheses to be formed about overall performance in terms of both effectiveness and efficiency. For example the quantity of unique terms shared between or exclusive to the training and testing sets can be extracted after representation and used to determine the exploitable problem scope, which may highlight issues with the suitability of those being generated. In addition the average terms per document and overall total provide guidance on the length of time needed to perform indexing and classification, in respect to each document and the entire dataset.

More information about composition of the data is also obtainable, meaning similarities or differences between the training and testing sets can be examined, such as actual category assignments and details of their relative densities. These are beneficial because they make it possible to ascertain the maximum and minimum performance boundaries, as exclusive test categories cannot be predicted and unlabelled documents receive perfect precision and recall by default, even with a trivial rejector. Certain statistics could guide the selection of appropriate components as well, for instance excessive noise might require extensive pre-processing, while a sparse problem space would favour specific dimension reduction and classification techniques.

Separation of algorithms into distinct stages also permits various observations to be made that may not otherwise be evident in conventional experiments, the most prominent being that no particular component or solution is definitively superior. In fact the trials exposed several methods routinely utilised that had little effect or actually lead to poorer performance than basic alternatives with similar functionality, including the popular options of stemming and TF-IDF index weighting. This could be

handled by building a solution that is either robust to changes in the data or receptive to modification, allowing elements with desirable characteristics to be inserted or substituted.

Another point that became apparent during the investigations was the benefit of providing precision and recall both independently and combined in the final evaluation measures, as they were rarely similar even for methods that attempted to balance them. This is also true for the overall measures employed, with the F-Measure and Break-Even Point varying by more than 30% in one experiment, while macro and micro averaging had a disparity of over 25%. Published results typically represent categorisation effectiveness by just a single value, but this is too restrictive and does not explain why it was chosen or how sensitive it is change. In contrast supplying precision and recall trends, chosen thresholds and prediction statistics helps to recognise the capabilities of a solution, which is essential in many real world situations.

Specifically regarding the findings of the tests conducted, it seems accurately predicting all categories of the dataset is a difficult task, as every macro averaged measure has poor recall. Consequently this leads to low thresholds being chosen for macro criteria in an attempt to deal with these challenging assignments, which produces a disproportionate number of predictions and causes a general drop precision. However, micro precision always drops faster than macro precision, meaning more incorrect predictions are made overall but the majority are confined to a particular group of categories. This is the main reason why optimal thresholds are not consistent for evaluation measures, making it difficult to choose one over another.

7 Final Conclusions

7.1 Thesis Overview

A systematic architecture for the accurate description and practical application of text categorisation solutions has been defined and evaluated in this thesis. By examining the shared characteristics and areas of similarity between existing interpretations of the process, a universal set of distinct stages has been identified and subsequently mapped to a collection of existing algorithms. Current text processing utilities and applications were then reviewed to determine their capacity to deal with this proposed solution structure and, finding them unsuitable, a new framework was introduced that has the required level of flexibility and capacity for rapid development. Finally controlled experiments were conducted using a basic implementation the new framework, which provided a proof of concept for the proposed text categorisation architecture and allowed realistic assessment of its feasibility and potential benefits.

7.1.1 Objectives

Establishing and appraising a universal architecture for defining standardised text categorisation solutions is the principal intention of this research. However, before this can be properly achieved the completion of a preliminary objective is necessary, which is the discovery of a text processing utility that is able to execute a series of trials suitable for empirically validating the recommended approach. Several other closely related issues are also considered during the course of achieving the main task, involving the correct definition of multiple concepts and specific terminology, though these are seen as lower priority secondary aims.

7.1.1.1 Primary Motivation

Text categorisation is an established research area that relates to many real world applications, though it is relatively recent that full automation of the process has become a viable option. A variety of techniques have been adopted in order to achieve this, including numerous algorithms from the domains of machine learning and artificial intelligence. However, no formal standards have been outlined to control the composition and evaluation of approaches, which has led to multiple instances of misinterpretation and inconsistent findings. These consequently make it difficult to successfully duplicate and appraise new techniques or compare their fundamental components to those used in other solutions.

The primary motivation of this research is to resolve or reduce the problem of insufficient comparative results, which is achieved by recommending a novel architecture for the text categorisation process that is based on the parallels found within existing solutions. By adhering to a standardised format it should be possible to generate fully reproducible benchmarks that can be used as the basis for comparative analysis, while allowing the individual algorithms and components to be easily recycled.

Aside from detailed descriptions of the structure and aspects of the proposed architecture, a viable proof of concept is also necessary. Therefore discovery of an application capable of building solutions that map directly to the various stages was an important factor, but reviewing existing options found them to be inadequate and lead to the construction of a new framework. This was then utilised to undertake a selection of experiments that were critically analysed by empirical means, allowing the validity and potential advantages of the suggested architecture to be determined.

7.1.1.2 Secondary Motivation

Despite the majority of techniques having roots from recognised research circles and utilising known algorithms, there is general level of confusion and contradiction evident in some text categorisation literature. This appears to be due to a combination of things, including imprecise explanations of modifications made to transition methods from other research areas and a general inattentiveness by those that employ them. Regrettably, both these issues compound the problem that forms the primary motivation for this work, but even broad acknowledgement and successful integration of the proposed solution architecture would not eliminate them. Subsequently they have become the focus of a secondary set of objectives, to outline common concepts and provide a comprehensive survey of popular techniques and algorithms.

7.1.2 Contributions

The foremost contribution of this work is the definition of a modular architecture for structuring text categorisation solutions, along with descriptions of the stages it contains and details of their potential interactions, as outlined in chapter two. With a level of detail not previously achieved the full extent of complexities involved in the process are illustrated and chapter three surveys common algorithms and demonstrates how they map onto the individual stages. A proof of concept is also provided and a number of experiments are undertaken in chapter six that utilise empirical evidence to evaluate the feasibility of the architecture, the results of which reveal the importance of such a granular approach and the benefits to be gained.

In order to carry out the trials used as a proof of concept several prominent open source data mining frameworks were reviewed during chapter four, being compared against each other and assessed for

their suitability to handle the proposed architecture. An emphasis was placed on the text related components currently available and how well they could be mapped to the distinct stages of the text categorisation process. Finding them to be unsuitable and determining that the effort required to modify them was too great, designs for a new framework were introduced and implemented in chapter five, before being employed in the experiments of chapter six. Like the existing frameworks this has the capacity for rapid prototyping, reusable components and exact duplication of results, though is simple enough to customise that even users with little programming knowledge will be able to readily extend it.

If the concepts contained by this work are adopted by the general research community they could potentially yield improvements in all text categorisation related research, due to standardising the manner and level of detail in which solutions are defined. This is especially important since the basic experiments from chapter six clearly show the impact that each individual stage can have. However, should it not be adopted it might still serve an equally important role, exposing areas of weakness within the domain and highlighting parts that need further consideration to attain a reasonable rate of progression.

7.2 Solution Architecture Evaluation

Evidence has been gathered that does imply the separation of text categorisation solutions into a set of discrete stages has several benefits over current informal approaches. Use of independent stages to group atomic components employed by techniques gives a clearer perspective on the processes that occur, simplifying the description and duplication of systems by allowing the reuse and easy modification of predefined base configurations. Extraction of statistics for each of them also offered a variety of useful information, though the results collected are by no means exhaustive and future research may yield further interesting discoveries.

No alterations were required to the proposed architecture during trials and it seems to fit the problem well, the statistics derived from each stage were also informative, though some had a high degree of overlap with the components tested. However, they did highlight it is not only important to discover how effective a solution can be, but also to determine how robust it is in respect to changes to the input at various stages, as a wide range of results were produced by the same basic configuration.

It was found that analysis of document content statistics throughout the process gave a reasonable indication of relative hardness, with noise or unsuitable data being highlighted when it was initially generated. Comparison between training and testing documents also contributed toward this and it was possible to ascertain details of term occurrence densities in regard to either set individually or as a ratio of the difference between the two. This subsequently allowed performance boundaries and estimations to be determined by discerning information related to their mutual and exclusive traits, for instance test items containing few shared terms are unlikely to be classified by positive samples.

While the composition of stages seems suitable, expansion of the basic information gathered from them is definitely feasible, for example details of specific terms could be extracted instead of just generalised information. Characteristics of the terms and how they impact performance might also be possible to distinguish earlier in the process, such as which term densities or sizes are expected to be valuable based on the proportion of categories and documents that contain them. This could provide guidance when selecting components to build solutions and narrow the problem space sooner, saving resources and allowing a more elegant mechanism than the current repetitive empirical trial and error.

Statistics related to documents rather than terms may also be useful and could be applied in the same manner, for instance whether longer or shorter documents are preferable in the training data or if term duplication is a factor. This would permit investigation into general trends of document characteristics and could lead to specialised components that tailor the training set or multipart solutions that use different procedures based on test document features. If obscure term representations are employed this might become important, as certain traditional techniques like stemming and stopword elimination are ineffective when applied to a concatenation of nonsensical phrases or symbolic sequences.

7.3 Framework Evaluation

Visual interfaces and an extensive variety of components are the main appeal of most established applications, though they increase the complexity involved when creating bespoke elements and limit practicality when rapidly prototyping novel techniques. This becomes apparent when considering the construction of text categorisation solutions that employ the proposed architecture and highlights a number of weaknesses in current data mining frameworks. Consequently the new framework was designed and implemented with an emphasis on the separation of functional and graphical layers, which has lead to greater flexibility and faster development of custom components.

The present implementation is only a trial version and does not address all of the issues discovered in other frameworks, but this is to be expected as the majority have been under development for more a decade. Modules and data models produced are also not necessarily the most efficient in terms of data structures or programming techniques, though they were capable of conducting the experiments required without difficulty and all generated results can be duplicated exactly.

Utilising the framework for successfully building several complex processing pipelines has led to a high degree of confidence that, through various types of extensions and additions, it will be able to resolve every major drawback encountered in existing frameworks. Nevertheless, even if these features do not get realised it has been clearly demonstrated that the current options would benefit from an increase in flexibility for prototyping applications, primarily in the form of simpler construction and integration of bespoke elements.

Dealing with the intricacies of text categorisation and the diverse algorithms it entails provided enough evidence for a fair assessment of the framework design, along with the practical aspects of recycling components and speed of solution prototyping. In general the design seems sound and everything functioned as expected with only a few minor programming anomalies that were quickly resolved, while custom development has also reduced and now takes a matter of minutes rather than hours. However, the actual implementation of the design is not perfect and needs adjustments to improve performance, data extraction and the manual coding of core projects.

Although data processing occurring within a module will probably have the largest affect on execution times, the experiment timings were inconsistent and closer inspection revealed that a notable portion was spent performing cloning and I/O operations for the repository. While these were actually quite consistent it did emphasise that a more comprehensive break down of execution times could prove advantageous, as some of the text processing functions were being artificially amplified. Enhancing the cloning mechanism to use less verbose representations than XML, such as JSON or binary, would help to lessen this problem and potentially lower the working size of the data, but finding a suitable method that does not necessitate code annotations is challenging.

More noticeable is the effort required to obtain data from the independent module nodes, as data models can only be accessed by analysing the code during execution because of the repository. Information extractors avoid this outputting their statistical information directly to an external text file, though this is impractical for every module as it involves filename parameters and formats that are individually defined. Other frameworks deal with it by forcing all data to adhere to tabular structures, but this is restrictive, so the best option is a graphical interface with default views or dedicated generic output modules capable of displaying different data types.

Another area that negatively impacts usability is manually defining projects, in particular forming links between nodes, as all instance specific modules are created by the factory and returned as standard core modules that do not allow direct access to nodes. This means the objects must be cast to a suitable type in order to gain references or the links must be made with error prone 'magic' strings that represent the variable names. Unfortunately there is no simple answer for this, though it is only an issue when manually coding projects and would be completely negated by the addition of an appropriate user interface.

Specifically in regard to the text categorisation components created, the dimensional reduction and term filtering should be reviewed as there are multiple ways for it to be applied to the test data. The current mechanism is embedded in the index module and calculates all weightings first before custom term filtering is applied, which means the algorithm controls how filtering occurs but flexibility is reduced as indexing cannot be executed independently of dimension reduction. If extracting statistics for both stages the module must consequently be executed twice, once with the filtering and once without it.

An alternative method is to have a completely separate module that accepts an input data model and filter reference then outputs the filtered results in an appropriate format. This keeps the indexing and dimensional reduction processes independent and makes simultaneous statistic extraction feasible, though if filtering is done prior to indexing the document content will be affected and might adjust the calculated weights in an undesirable manner. It is impossible to choose which option is better, as both have potential to change the outcome in different ways, but due to speed of component development it is trivial to create either of them.

7.4 Future Directions

Two major directions have been established from this research; the first is ensuring adoption of the proposed solution architecture, while the second focuses on enhancing the created framework. Of these only the solution architecture is specific to text categorisation and offers potential benefits that may help to advance the subject domain by reforming the basic approach taken to solve the problem. However, getting it accepted will be difficult to accomplish, as bad habits need to be undone and work already completed would have to be revisited and fitted to the model, though once embraced its sustained use will be self perpetuating.

Conversely the framework improvements have a larger scope and are relatively easy to develop, but involve a continual process that must be able to compete with other frameworks as they are inevitably upgraded, otherwise it would quickly become obsolete. As it grows in popularity this task will also get harder and require additional resources, though opening the source and allowing development by the community could reduce this effort, but even then it ideally requires some form of support, quality control and public access.

7.4.1 Solution Architecture Adoption

It is important that measures are taken to ensure that text categorisation becomes more standardised in its nature, either by attempting to gain acceptance of the solution stages and their structuring or by getting the issues they address to be recognised as a genuine problem. This can only be achieved by performing further experiments for common algorithms and publishing the results in a manner that draws attention to the benefits gained and the negative aspects of current methodologies.

Conversion of existing solutions and production of a viable comparative analysis between them is a worthwhile task because it will directly illustrate the flexibility of the architecture and emphasise how it can be adapted to suit different techniques. It also shows the capacity for accurate duplication and generation of critical information that is not available with present approaches, doing so in a modular manner that allows investigation of different algorithm combinations and appraisal of their resistance to change in certain elements.

Studying the impact of different independent components is another area that might prove interesting, as was highlighted by the limited experiments conducted during this research. Although it is clear that different techniques employed for each stage do affect overall results, finding ways to minimise or bypass variation in particular traits would be advantageous. For example discerning if cosine similarity in a KNN classifier is able to negate the effect of index weight normalisation and whether this makes execution more efficient. Uncertainties such as this require in depth exploration of the relationships

linking the statistics of each stage, but also have the potential to expose reliable indicators that could determine optimal parameters or solution composition without empirical trial and error.

7.4.2 Framework Enhancement

There are many improvements that can be made to the trial version of the framework, but perhaps the most notable is addition of a graphical interface, as this would instantly boost overall ease and speed of use. It would also serve as a complete proof of concept for the outlined designs and solve many of the current issues related to data extraction and project construction that were revealed during the conducted experiments.

Publicising the core framework and available modules is imperative as well, not just to enlarge the user base but to grow a community that can provide support, create new components and generate benchmark solutions. Due to the strong emphasis on prototyping novel techniques, the most appropriate way to achieve this is probably through joint collaborations with academic institutions or research focused organisations and referring to it in published literature. Independent release of the text algorithm library developed for the experiments may also contribute toward this, whilst integration of existing code libraries from other disciplines might attract a wider range of followers and promote rapid expansion.

Numerous updates to central functionality are necessary too, like logging descriptive information and proper error handling, which assist in customisations and general troubleshooting. Module execution times also need refinement as the experimental results gathered show the present diagnostic utility is not consistent enough to be of use, though this will mean employing a method based on CPU cycles and entail a resource overhead. Investigation of efficient data structures and cloning mechanisms are other key aspects that require further consideration, as they both increase the practical limitations of data samples being processed.

7.5 Concluding Remarks

The domain of text categorisation is in need of reform due several recurrent issues that consistently prove a hindrance to its reasonable progression. In an attempt to initiate a change this research convincingly shows that it is important to formalise the text categorisation solution process, whether it be by the proposed architecture or another mechanism that offers similar advantages. With even limited experiments exposing a wealth of interesting and potentially valuable information that can only be determined by consideration of individual aspects instead of just a global overview, though much exploration is still required to discover the extent of the benefits to be gained.

Previous attempts to describe the general steps involved in text categorisation are incomplete or vague, either relating to a specific form of algorithm [54], merging stages together [57] or missing the initial elements of data selection and preparation [59][53][40]. None of them properly account for the full process complexity, identify the importance of all required aspects or supply definitive structures that can represent most existing techniques, typically focusing on algorithms employed for classification rather than how to build a entire viable solution. Aside from the limited ‘aggressivity’ [136] and ‘reduction factor’ [59] values, there is also no mention of statistical information obtainable from each stage, despite it giving constructive indications of method suitability and data composition.

Compounding the current problem a shortage of adequate text processing utilities and frameworks has been revealed, with those presently available providing superior graphical interfaces, but lacking suitable means for rapid creation of new algorithms and solution components. Initial steps have been taken to resolve this issue through, with the introduction of designs for a new framework specifically focused on research based prototyping. A viable proof of concept has also been implemented, though a degree of finalisation is yet to be completed before it is ready to be made publically accessible.

8 References [1] [2]

[1] F. Sebastiani, “A Tutorial on Automated Text Categorisation,” in *1st Argentinian Symposium on Artificial Intelligence, ASAI 1999*, Buenos Aires, 1999.

[2] K. Aas and L. Eikvil, “Text Categorisation: A Survey.,” Technical Report, Raport NR941, Norwegian Computing Centre, 1999.

[3] M.-L. Antonie and O. R. Zaiane, “Text Document Categorization by Term Association,” in *IEEE International Conference on Data Mining, ICDM 02*, 2002.

[4] F. Sebastiani, “Machine Learning in Automated Text Categorization,” *ACM COmputing Surveys (CSUR),* vol. 34, no. 1, pp. 1-47, March 2002.

[5] T. Bladwin and M. Lui, “Language Identification: The Long and the Short of the Matter,” in *HLT '10 Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

[6] B. Pang and L. Lee, “Opinion Mining and Sentiment Analysis,” *Foundations and Trends in Information Retrieval,* vol. 2, no. 1-2, pp. 1-135, January 2008.

[7] S. Danso, E. Atwell and O. Johnson, “A Comparative Study of Machine Learning Methods for Verbal Autopsy Text Classification,” *International Journal of Computer Science Issues,* vol. 10, no. 6, November 2013.

[8] B. Kessler, G. Nunberg and H. Schutze, “Automatic Detection of Text Genre,” in *35th Annual Meeting of the Association for Computational Linguistics, ACL 98, and 8th conference of the European Chapter of the Association of Computational Linguistics, EACL 97*, 1997.

[9] Y.-B. Lee and S. H. Myaeng, “Text Genre Classification with Genre-Revealing and Subject-Revealing Features,” in *25th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR 02*, 2002.

[10] O. d. Vel, M. C. A. Anderson and G. Mohay, “Mining E-mail Content for Author Identification Forensics,” *ACM SIGMOD,* vol. 30, no. 4, pp. 55-64, December 2001.

[11] M. Gamon, “Linguistic correlates of style: authorship classification with deep linguistic analysis features,” in *International Conference on Computational Linguistics, COLING 2004*, 2004.

[12] F. Sebastiani, “Classification of Text, Automatic,” in *The Encyclopedia of Language and Linguistics*, vol. 2, Elsevier Science Publishers, 2006, pp. 457-463.

[13] F. Sebastiani, “Text Categorization,” in *Text Mining and its Applications to Intelligence, CRM and Knowledge Management*, WIT Press, 2005, pp. 109-129.

[14] E. K. Jacob, “Classification and Categorization: A Difference that Makes a Difference,” *Library Trends,* vol. 52, no. 3, pp. 515-540, 2004.

[15] S. Arumugam, “Classification Techniques for Categorization of Hypertext Documents,” Pennsylvania State University, 2005.

[16] E. Gabrilovich and S. Markovitch, “Text Categorization with Many Redundant Features: Using Agressive Feature Selection to Make SVMs Competitive with C4.5,” in *21st International Conference on Machine Learning, ICML 04*, Banff, 2004.

[17] T. Joachims, “Text Categorization with Support Vector Machines: Learning with Many Relevant Features,” in *10th European Conference on Machine Learning ECML '98*, 1998.

[18] Y. Yang and J. O. Pederson, “A Comparative Study on Feature Selection in Text Categorization,” in *14th International Conference on Machine Learning, ICML 97*, 1997.

[19] T. Theeramunkong and V. Lertnattee, “Multi-Dimensional Text Classification,” in *19th*

[1] Note that references with limited publication information are likely to be internal company reports as the original source did not list additional details and alternative sources were unavailable.

[2] Online references relate to specific sections of company, product or text mining related websites and hard copy equivalents were not available at the time of writing. There is also no guarantee these links will remain active and, in a small number of cases, they have been included specifically because they are not presently active.

international conference on Conputational linguistics, COLING 02, 2002.

[20] D. Koller and M. Sahami, “Hierarchically classifying documents using very few words,” in *14th International Conference on Machine Learning*, 1997.

[21] A. McCallum, R. RosenFeld, T. Mitchell and A. Y. Ng, “Improving Text Classification by Shrinkage in a Heirarchy of Classes,” in *15th International Conference on Machine Learning, ICML 98*, 1998.

[22] S. Kiritchenko, S. Matwin and A. F. Famili, “Functional Annotation of Genes Using Hierarchical Text Categorization,” in *SIG: Linking Literature, Information and KNowledge for Biology, BioLINK*, 2005.

[23] T. Hedlund, A. Pirkola, H. Keskustalo, E. Airio and K. Jarvelin, “Cross-Language information retrieval: using multiple language pairs,” in *2nd Bi-annual DISSAnet Conference, ProLISSA*, 2002.

[24] J. A. Goldsmith, D. Higgins and S. Soglasnova, “Automatic Language-Specific Stemming in Information Retrieval,” in *Workshop of Cross-Language Evaluation Forum on Cross-Language Information Retrieval and Evaluation, CLEF 00*, 2000.

[25] Snowball, “Multi-lingual text processing resources,” [Online]. Available: http://snowball.tartarus.org/algorithms/.

[26] Universite de Neuchatel, “IR Multilingual Resources at UniNE,” [Online]. Available: http://members.unine.ch/jacques.savoy/clef/index.html.

[27] S. Godbole and S. Sarawagi, “Discriminative Methods for Multi-Labeled Classification,” in *8th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2004.

[28] G. Tsoumakas and I. Katakis, “Multi-Label Classification: An Overview,” *Int J Data Warehousing and Mining,* vol. 2007, pp. 1-13, 2007.

[29] A. Kyriakopoulou and T. Kalamboukis, “Text Classification Using Clustering,” in *ECML-PKDD Discovery Challenge Workshop*, 2006.

[30] M. Sanderson, “Word Sense Disambiguation and Information Retrieval,” in *17th ACM International Conference on Research and Development in Information Retrieval, SIGIR 94*, 1994.

[31] G. Ramakrishnanan and P. Bhattacharyya, “Text Representation with WordNet Sysnsets using Soft Sense Disambiguation,” in *8th International Conference on Applications of Natural Language to Information Systems, NLDB 2003*, 2003.

[32] D. Nadeau and S. Sekine, “A survey of named entity recognition and classification,” *Lingvisticae Investigationes,* vol. 30, no. 1, pp. 3-26, 2007.

[33] E. Brill, “A Simple Rule-Based Part Of Speech Tagger,” in *Third conference on Applied natural language, ANLC 92*, 1992.

[34] E. Gabrilovich and F. Sebastiani, “Bibliography on Automated Text Categorization,” [Online]. Available: http://liinwww.ira.uka.de/bibliography/Ai/automated.text.categorization.html.

[35] Y. Yang, “An Evaluation of Statistical Approaches to Text Categorization,” *Information Retrieval,* vol. 1, no. 1-2, pp. 69-90, 1999.

[36] S. Buddeewong and W. Kreesuradej, “A New Association Rule-Based Text Classifier Algorithm,” in *17th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 05*, 2005.

[37] M. J. Pazzani, “Representation of Electronic Mail Filtering Profiles: A User Study,” in *5th International conference on Intelligent user interfaces, IUI 00*, 2000.

[38] L. S. Larkey, “Automatic Essay Grading Using text Categorization Techniques,” in *21st annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR 98*, 1998.

[39] F. Sebastiani, “Text Categorization,” in *The Encyclopedia of Database Technologies and Applications*, Idea Group Publishing, 2005, pp. 683-687.

[40] F. Sebastiani, “Machine Learning in Automated Text Categorisation,” Technical Report, 1999.

[41] Y. Yang, “An Evaluation of Statistical Approaches to Text Categorization,” Technical Report, Carnegie Mellon University, 1997.

[42] Y. Yang and X. Liu, “A re-examination of text categorization methods,” in *22nd annual international ACM SIGIR conference on Research and development in information retrieval,*

SIGIR 99, 1999.

[43] F. Debole and F. Sebastiani, “An Analysis of the Relative Hardness of Reuters-21578 Subsets,” *Journal of the American Society of Information Science and Technology,* vol. 56, no. 6, pp. 584-596, April 2005.

[44] A. Esuli and F. Sebastiani, “Evaluating Information Extraction,” in *International Conference of the Cross-Language Evaluation Forum, CLEF 2010*, Padua, 2010.

[45] F. Thabtah, “A review of associative classification mining,” *The Knowledge Engineering Review,* vol. 22, no. 1, pp. 37-65, March 2007.

[46] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer and R. Wirth, “CRISP-DM 1.0, Step-by-step data mining guide,” SPSS Inc., 2000.

[47] Ó. Marbán, G. Mariscal and J. Segovia, “A Data Mining & Knowledge Discovery Process,” in *Data Mining and Knowledge Discovery in Real Life Applications*, J. Ponce and A. Karahoca, Eds., I-Tech Education and Publishing, 2009, p. 436.

[48] A. Azevedo and M. F. Santos, “KDD, SEMMA and CRISP-DM: A Parallel Overview,” in *IADIS European Conference on Data Mining*, 2008.

[49] S. S. Rohanizadeha and M. B. Moghadam, “A Proposed Data Mining Methodology and its Application to Industrial,” *Journal of Industrial Engineering,* vol. 4, pp. 37-50, 2009.

[50] R. D. Jarrard, “Scientific Methods,” University of Utah, Utah, 2001.

[51] Y. H. Li and A. K. Jain, “Classification of Text Documents,” *The Computer Journal,* vol. 41, no. 8, pp. 537-546, 1998.

[52] C. C. Aggarwal and C. X. Zhai, “A Survey of Text Classification Algorithms,” in *Mining Text Data*, Springer US, 2012, pp. 163-222.

[53] A. Mahinovs and A. Tiwari, “Text Classification Method Review,” Decision Engineering Report (DEG) Series, Cranfield University, 2007.

[54] W. B. Cavnar and J. M. Trenkle, “N-Gram-Based Text Categorization,” in *3rd Annual Symposium on Document Analysis and Information Retrieval, SDAIR-94*, 1994.

[55] A. T. Sadiq and S. M. Abdullah, “Hybrid Intelligent Technique for Text Categorization,” in *Advanced Computer Science Applications and Technologies (ACSAT)*, Kuala Lumpur, 2012.

[56] I. Moulinier, “A Framework for Comparing Text Categorization Approaches,” AAAI, Paris, 1996.

[57] M. Ikonomakis, S. Kotsiantis and V. Tampakas, “Text Classification Using Machine Learning Techniques,” *Wseas Transactions on Computers,* vol. 4, no. 8, pp. 996-974, August 2005.

[58] S. C. Dharmadhikari, M. Ingle and P. Kulkarni, “A Novel Multi label Text Classification Model using Semi supervised learning,” *International Journal of Data Mining & Knowledge Management Process (IJDKP),* vol. 2, no. 4, pp. 11-21, July 2012.

[59] F. Debole and F. Sebastiani, “Supervised Term Weighting for Automated Text Categorization,” in *ACM Symposium on Applied computing, SAC 03*, 2003.

[60] D. Davidov, E. Gabrilovich and S. Markovitvh, “Parameterized Generation of Labelled Datasets for Text Categorization Based on a Hierarchical Directory,” in *27th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR 04*, 2004.

[61] B. Klimt and Y. Yang, “The Enron Corpus: A New Dataset for Email Classification Research,” in *15th European Conference on Machine Learning, ECML 2004*, Pisa, 2004.

[62] D. D. Lewis, “Reuters-21578 ReadMe - version 1.3,” 14 May 2004. [Online]. Available: http://www.daviddlewis.com/resources/testcollections/reuters21578/readme.txt.

[63] A. Esuli and F. Sebastiani, “Training Data Cleaning for Text Classification,” in *ICTIR '09 Proceedings of the 2nd International Conference on Theory of Information Retrieval*, Cambridge, 2009.

[64] W. Hersh, “OHSUMED ReadMe,” 07 December 2007. [Online]. Available: http://ir.ohsu.edu/ohsumed/readme.

[65] D. D. Lewis, Y. Yang, T. G. Rose and F. Li, “RCV1: A New Benchmark Collection for Text Categorization Research,” *Journal of Machine Learning Research,* vol. 5, pp. 361-397, 2004.

[66] W. W.Cohen, V. R. Carvalho and T. M. Mitchell, “Learning to Classify Email into "Speech Acts",” in *EMNLP 2004*, Barcelona, 2004.

[67] G. Rios and H. Zha, “Exploring Support Vector Machines and Random Forests for Spam

Detection," in *First Conference on Email and Anti-Spam (CEAS)*, 2004.

[68] J. Itskevitch, "Automatic Hierarchical E-Mail Classification Using Association Rules," Thesis, Belorussian State Polytechnic Academy, 1997.

[69] W. W. Cohen, "Learning Rules that Classify E-Mail," in *AAAI Spring Symposium on Machine Learning in Information Access*, 1996.

[70] T. Urvoy, T. Lavergne and P. Filoche, "Tracking Web Spam with Hidden Style Similarity," in *AIRWeb2006, Second International Workshop on Adversarial Information Retrieval on the Web, co-located iwht SIGIR 2006*, Seattle, 2006.

[71] Y. Yang, S. Slattery and R. Ghani, "A Study of Approaches to Hypertext Categorization," *Journal of Intelligent Information Systems,* vol. 18, no. 2-3, pp. 219-241, 01 March 2002.

[72] Y. Yang and J. Wilbur, "Using Corpus Statistics to Remove Redundant Words in text Categorization," *Journal of the American Society for Information Science,* vol. 47, no. 5, pp. 357-369, May 1996.

[73] W. Hersh, C. Buckley, T. J. Leone and D. Hickam, "OHSUMED: An Interactive Retrieval Evaluation and New Large Test Collection for Research," in *17th annual international ACM SIGIR conference on Research and development in information retrieval*, 1994.

[74] Y. Yang, J. Zhang and B. Kisiel, "A Scalability Analysis of Classifiers in text Categorization," in *26th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR 03*, 2003.

[75] Y. Yang, "Noise Reduction in a Statistical Approach to Text Categorization," in *18th annual international ACM SIGIR conference on Reasearch and development in information retrieval, SIGIR 95*, 1995.

[76] D. Mladenic, "Feature subset selection in text-learning," in *10th European Conference on Machine Learning, ECML 98*, 1998.

[77] R. Kohavi, "A Study of Cross-Validatio and Bootstrap for Accuracy Estimation and Model Selection," in *14th international joint conference on Artificial intelligence, IJCAI 95*, 1995.

[78] P. J. Hayes and S. P. Weinstein, "Construe-TIS: A System for Content-Based Indexing of a Database of News Stories," in *2nd Conference on Innovative Applications of Artificial Intelligence*, 1990.

[79] M. Sanderson, "Reuters test collection," in *Proceedings of the Sixteenth Research Colloquium of the British Computer Society Information Retrieval Specialist Group, Drymen*, Glasgow, 1994.

[80] Z. Xu, M. Chen and K. Q. Weinberger, "An alternative text representation to TF-IDF and Bag-of-Words," in *Proceedings of 21st ACM Conference of Information and Knowledge Management (CIKM)*, 2012.

[81] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," in *Information Processing and Management*, 2009.

[82] "reuters21578.tar.gz," 16 February 1999. [Online]. Available: http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.tar.gz.

[83] NIST, "Reuters Corpora (RCV1, RCV2, TRC2)," National Institute of Standards and Technology, 2004. [Online]. Available: http://trec.nist.gov/data/reuters/reuters.html.

[84] D. D. Lewis, "RCV1-v2/LYRL2004: The LYRL2004 Distribution of the RCV1-v2 Text Categorization Test Collection," 2004. [Online]. Available: http://www.jmlr.org/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm.

[85] NIST, "TREC-9 Filtering Track Collections," National Institute of Standards and Technology, 2007. [Online]. Available: http://trec.nist.gov/data/t9_filtering.html.

[86] E. D. Weiner, J. O. Pederson and A. S. Weigend, "A Neural Network Approach to Topic Spotting," in *4th Annual Syposium on Document Analysis and Information Retrieval, SDAIR-95*, 1995.

[87] E. Baralis and P. Garza, "Associative Text Categorization exploiting Negated Words," in *ACM Symposium on Applied computing, SAC 06*, 2006.

[88] S. Ahmed and F. Mithun, "Word Stemming to Enhance Spam Filtering," in *1st Conference on Email and Anti-Spam, CEAS*, 2004.

[89] J. Carlberger, H. Dalianis, M. Hassel and O. Knutsson, "Improving Precision in Information Retrieval for Swedish using Stemming," in *13th Nordic Conference on Computational*

Linguistics, NODALIDA 01, Uppsala, 2001.

[90] C. Liao, S. Alpha and P. Dixon, “Feature Preparation in text Categorisation,” Internal Report, Oracle Corporation.

[91] C. D. Paice, “Method for Evaluation of Stemming Algorithms based on Error Counting,” *Journal of the American Society for Information Science,* vol. 47, no. 8, pp. 632-649, August 1996.

[92] R. T.-W. Lo and L. O. Ben He, “Automatically Building a Stopword List for an Information Retrieval System,” in *5th Dutch-Belgium Information Retrieval Workshop, DIR 2005*, 2005.

[93] C. T. Yuang, R. E. Banchs and C. E. Siong, “An Empirical Evaluation of Stop Word Removal in Statistical Machine Translation,” in *Joint Workshop on Exploiting Synergies between Information Retrieval and Machine Translation, ESIRMT, and Hybrid Approaches to Machine Translation, HyTra (EACL)*, 2012.

[94] LEXTEK International, “Onix Text Retrieval Toolkity API - Stopword list 2,” [Online]. Available: http://www.lextek.com/manuals/onix/stopwords2.html.

[95] M. F. Porter, “English stop word list,” Snowball, [Online]. Available: http://snowball.tartarus.org/algorithms/english/stop.txt.

[96] SAO/NASA, “SAO/NASA Astrophysics Data System - Stopword lists,” [Online]. Available: http://adsabs.harvard.edu/abs_doc/stopwords.html.

[97] MySQL, “MySQL Fulltext-Stopwordwords,” [Online]. Available: http://dev.mysql.com/doc/refman/5.1/en/fulltext-stopwords.html.

[98] LEXTEK International, “Onix Text Retrieval Toolkity API - Stopword list 1,” [Online]. Available: http://www.lextek.com/manuals/onix/stopwords1.html.

[99] Rank.NL, “English Stopwords,” [Online]. Available: http://www.ranks.nl/tools/stopwords.html.

[100] J. B. Lovins, “Development of s stemming algorithm,” *Mechanical Translation and Computational Linguistics,* vol. 11, pp. 22-31, 1968.

[101] M. F. Porter, “Lovins revisited,” in *Charting a New Course: Natural Language Processing and Information Retrieval. Essays in Honour of Karen Sparck Jones*, vol. 16, Springer Netherlands, 2005, pp. 39-68.

[102] M. F. Porter, “An algorithm for suffix stripping,” *Program,* vol. 14, no. 3, pp. 130-137, 1980.

[103] D. D. P. Barrenechea, “A Spanish Stemming Algorithm Implementation in PROLOG and C#,” Artificial Intelligence Center, The University of Georgia, 2006.

[104] K. Taghva, R. Beckley and M. Sadeh, “A Stemming Algorithm for the Farsi Language,” in *International Conference on Information Technology: Coding and Computing, ITCC 05*, 2005.

[105] M. F. Porter, “The Porter Stemming Algorithm,” 2006. [Online]. Available: http://tartarus.org/martin/PorterStemmer/.

[106] M. F. Porter, “The English (Porter2) stemming algorithm,” [Online]. Available: http://snowball.tartarus.org/algorithms/english/stemmer.html.

[107] G. A. Miller, “WordNet: A Lexical Database for English,” *Communications of the ACM, CACM,* vol. 38, no. 11, pp. 39-41, 1995.

[108] N. S. Giridhar, K. V. Prema and N. V. S. Reddy, “A Prospective Study of Stemming Algorithms for Web Text Mining,” *Ganpat University Journal of Engineering & Technology,* vol. 1, no. 1, 2011.

[109] R. Krovetz, “Viewing Morphology as an Inference Process,” in *16th annual international ACM SIGIR conference on Research and development in information retrieval*, 1993.

[110] A. G. Jivani, “A Comparative Study of Stemming Algorithms,” *International Journal of Computer Technology and Applications,* vol. 02, no. 06, pp. 1930-1938, 2011.

[111] T. Berners-Lee, J. Hendler and O. Lassila, “The Semantic Web,” *Scientific American Magazine,* 26 March 2008.

[112] World Wide Web Consortium (W3C), “Extensible Markup Language (XML) 1.0,” 26 November 2008. [Online]. Available: http://www.w3.org/TR/xml/.

[113] World Wide Web Consortium (W3C), “Resource Description Framework (RDF),” 10 February 2004. [Online]. Available: http://www.w3.org/RDF/.

[114] World Wide Web Consortium (W3C), “Web Ontology Language (OWL),” 11 December 2012. [Online]. Available: http://www.w3.org/2001/sw/wiki/OWL.

[115] A. Gruzd, “Folksonomies vs. Bag-of-Words: The Evaluation & Comparison of Different Types of Document Representations,” in *17th Annual ASIS&T SIG/CR Classification Research Workshop*, Austin, 2006.

[116] J. Trant, “Studying Social Tagging and Folksonomy: A Review and Framework,” *Journal of Digital Information,* vol. 10, no. 1, 2009.

[117] H. Halpin, V. Robu and H. Shepherd, “The Complex Dynamics of Collaborative Tagging,” in *16th International conference on World Wide Web, WWW 07*, 2007.

[118] A. Nouruzi, “Folksonomies: (Un)Controlled vocabulary?,” *Knowledge Organization,* vol. 33, no. 4, 2006.

[119] A. Passant, “Using Ontologies to Strengthen Folksonomies and Enrich Information Retrieval in Weblogs,” in *International Conference on Weblogs and Social Media*, 2007.

[120] A. Moschitti and R. Basili, “Complex Linguistic Features for Text Classification: a comprehensive study,” in *26th European COnference on Information Retrieval, ECIR*, 2004.

[121] D. Molla, M. v. Zaanen and D. Smith, “Named Entity Recognition for Question Answering,” in *Australasian language technology workshop*, Sydney, 2006.

[122] T. Poibeau and L. Kosseim, “Proper Name Extraction from Non-Journalistic Texts,” *Language and Computers,* pp. 144-157, December 2001.

[123] S. J. DeRose, “Grammatical Category Disambiguation by Statistical Optimization,” *Computational Linguistics,* vol. 14, no. 1, pp. 31-39, 1988.

[124] K. W. Church, “A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text,” in *Second conference on Applied natural language, ANLC 88*, 1988.

[125] D. Widdows, “Unsupervised methods for developing taxonomies by combining syntactic and statistical information,” in *Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, NAACL 03*, 2003.

[126] J. Lin and D. Demner-Fushman, “"Bag of Words" is not enough for Strength of Evidence Classification,” in *Annual Symposium of the American Medical Informatics Association (AIMIA 2005)*, Washington, 2005.

[127] T. Scheffer and S. Wrobel, “Text Classification beyond the Bag-of-Words Representation,” in *Workshop on Text Learning, 19th International Conference on Machine Learning, ICML*, Sydney, 2002.

[128] K. Denecke, T. Risse and T. Baehr, “Topic Classification Using Limited Bibliographic Metadata,” in *4th International conference on Digital Information Management, ICDIM 2009*, Ann Arbor, 2009.

[129] R. Ghani, S. Slattery and Y. Yang, “Hypertext Categorization using Hyperlink Patterns and Meta Data,” in *19th International Cnference on Machine Learning, ICML 01*, 2001.

[130] K. Thiel and M. Berthold, “The KNIME Text Processing Feature: An Introduction,” User Guide, KNIME AG, 2012.

[131] P. Soucy and G. W. Mineau, “Beyond TFIDF Weighting for Text Categorization in the Vector Space Model,” in *19th international joint conference on Artificial intelligence, IJCAI 05*, 2005.

[132] F. W. Lancaster and E. G. Fayen, Information Retrieval On-Line, Melville Publishing Co., 1973.

[133] G. Salton, E. A. Fox and H. Wu, “Extended Boolean information retrieval,” *Communications of the ACM,* vol. 26, no. 11, 1983.

[134] G. Salton, A. Wong and C. Yang, “A Vector Space Model for Automatic Indexing,” *Communications of the ACM,* vol. 18, no. 11, pp. 613-620, November 1975.

[135] P. D. Turney and P. Pantel, “From Frequency to Meaning: Vector Space Models of Semantics,” *Journal of Artificial Intelligence Research,* vol. 37, no. 1, pp. 141-188, 2010.

[136] L. Galavotti, F. Sebastiani and M. Simi, “Experiments on the Use of Feature Selection and Negative Evidence in Automated Text Categorization,” in *4th European Conference on Research and Advanced Technology for Digital Libraries, ECDL 00*, 2000.

[137] J. Zobel, A. Moffat and K. Ramamohanarao, “Inverted files versus signature files for text indexing,” *ACM Transactions on Database Systems, TODS,* vol. 23, no. 4, pp. 453-490, 1998.

[138] T. Jo, “Inverted Index based Modified Version of KNN for Text Categorization,” *Journal of Information Processing Systems,* vol. 4, no. 1, pp. 17-26, 2008.

[139] R. Bayer and K. Unterauer, “Prefix B-Trees,” *ACM Transactions on Database Systems, TODS,*

vol. 2, no. 1, pp. 11-26, March 1977.

[140] V. R. Shanks, H. E. Williams and A. Cannane, “Indexing for Fast Categorisation,” in *26th Australasian computer science conference, ACSC 03*, 2003.

[141] G. Salton and C. Buckley, “Term-Weighting Approaches In Automatic Text Retrieval,” *Information Processing and Management: an International Journal,* vol. 24, no. 5, pp. 513-523, 1988.

[142] S. Robertson, “Understanding Inverse Document Frequency: On theoretical arguments for IDF,” *Journal of Documentation,* vol. 60, no. 5, pp. 503-520, 2004.

[143] C. Buckley, G. Salton, J. Allan and A. Singhal, “Automatic Query Expansion Using SMART : TREC 3,” in *The third Text Retrieval Conference, TREC 3*, 1994.

[144] S. T. Dumais, “Improving the retrieval of information from external sources,” *Behaviour Research Methods, Instruments and Computers,* vol. 23, no. 2, pp. 229-236, June 1991.

[145] I. Guyon and A. Elisseeff, “An Introduction to Variable and Feature Selection,” *The Journal of Machine Learning Research, JMLR,* vol. 3, pp. 1157-1182, 2003.

[146] M. Rogati and Y. Yang, “High-Performance Feature Selection for Text Classification,” in *11th international conference on Information and knowledge management, COKM 02*, 2002.

[148] S. Li, R. Xia, C. Zong and C.-R. Huang, “A Framework of Feature Selection Methods for Text Categorization,” in *47th Annual Meeting of the ACL and 4th International Conference on Natural Language Processing, ACL 09 and AFNLP*, 2009.

[149] Z. Zheng, X. Wu and R. Srihari, “Feature Selection for Text Categorization on Imbalanced Data,” *ACM SIGKDD Explorations Newsletter - Special issue on learning from imbalanced datasets,* vol. 6, no. 1, pp. 80-89, June 2004.

[150] S. D. Simon, “Understanding the Odds Ratio and the Relative Risk,” *Journal of Andrology,* vol. 22, no. 4, pp. 533-536, July/August 2001.

[151] H. Toutenburg, R. Fisher and F. Yates, in *Statistical Tables for Biological, Agricultural and Medical Research*, 6 ed., Longman, 1974.

[152] J. M. Utts and R. F. Heckard, “More About Categorical Variables,” in *Statistical Ideas And Methods*, 1 ed., Duxbury Press, 2005, p. 538.

[153] D. C. Howell, “Chi-Squared Test - Analysis of Contingency Tables,” in *International Handbook of Statistical Sciences*, 2011.

[154] H. T. Ng, W. B. Goh and K. L. Low, “Feature Selection, Perceptron Learning, and a Usability Case Study for Text Categorization,” in *20th annual international ACM SIGIR conference on Research and development in information retrieval*, 1997.

[155] C. Cortes and V. Vapnik, “Support-Vector Networks,” *Machine Learning,* vol. 20, no. 3, pp. 273-297, 01 September 1995.

[156] S.-H. Cha, “Comprehensive Survey on Distance/Simiarity Measures between Probability Density Functions,” *International Journal of Mathematical Models and Methods in Applied Sciences,* vol. 1, no. 4, pp. 300-307, 2007.

[157] S. Niwattanakul, J. Singthongchai, E. Naenudorn and S. Wanapu, “Using of Jaccard Coefficient for Keywords Similarity,” in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 2013.

[158] Y. Yang, “A Study on Thresholding Strategies for Text Categorization,” in *24th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR 01*, 2001.

[159] D. D. Lewis, “Evaluating and Optimzing Autonomous Text Classification Systems,” in *18th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR 95*, 1995.

[160] H. Cunningham, “Text Processing with GATE (Version 6),” User Guide, University of Sheffield Department of Computer Science, Sheffield, 2011.

[161] C. J. v. Rijsbergen, Information Retrieval, London: Butterworths, 1979.

[162] C. D. Manning, P. Raghavan and H. Schutze, “Evaluation in Information Retrieval,” in *An Introduction to Information Retrieval*, Cambridge University Press, 2008, pp. 151-175.

[163] M. Everingham, L. V. Gool, C. K. Williams, J. Winn and A. Zisserman, “The PASCAL Visual Object Classes (VOC) Challenge,” *International Journal of Computer Vision,* vol. 88, no. 2, pp.

303-338, June 2010.

[164] S. I. Hill, H. Zaragoza, R. Herbrich and P. J. W. Rayner, “Average Precision and The Problem of Generalisation,” in *ACM SIGIR Workshop on Mathematical and Formal Methods in Information Retrieval*, 2002.

[165] KDnuggets, “Poll Result: Top Analytics, Data Mining, Big Data sotware used,” KDnuggets, May 2012. [Online]. Available: http://www.kdnuggets.com/2012/05/top-analytics-data-mining-big-data-software.html.

[166] W. N. Venables and D. M. Smith, “An Introduction to R,” 2013.

[167] Rapid-I, “RapidMiner 5.0 User Manual,” Rapid-I, 2010.

[168] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kotter, T. Meinl, P. Ohl, K. Thiel and B. Wiswedel, “KNIME - The Konstanz Information Miner: Version 2.0 and Beyond,” *ACM SIGKDD Explorations Newsletter,* vol. 11, no. 1, pp. 26-31, June 2009.

[169] R. R. Bouckaert, E. Frank, M. Hall, R. Kirby, P. Reutermann, A. Seewald and D. Scuse, “WEKA Manual for Version 3-7-9,” The Univeristy of Waikato, 2013.

[170] J. Demsar and B. Zupan, “ORANGE: DATA MINING FRUITFUL AND FUN,” University of Ljubljana, 2012.

[171] S. Chapman and C. Parkinson, “SimMetrics library v 1.5 for .Net 2.0 - System and Reference Manual,” 2006.

[172] A. K. Callum, “Mallet: A Machine Learning for Language Toolkit,” 2002. [Online]. Available: http://mallet.cs.umass.edu/.

[173] M. F. Porter, “Snowball: A language for stemming algorithms,” October 2001. [Online]. Available: http://snowball.tartarus.org/texts/introduction.html.

[174] Apache Software Foundation, “Apache Lucence,” Apache Software Foundation, [Online]. Available: http://lucene.apache.org/.

[175] Revolution Analytics, “Revolution R-Enterprise,” Revolution Analytics, 2013. [Online]. Available: http://www.revolutionanalytics.com/products/revolution-enterprise.php.

[176] F. Wild, “CRAN Task View: Natural Language Processing,” R-Project, 22 May 2013. [Online]. Available: http://cran.r-project.org/web/views/NaturalLanguageProcessing.html.

[177] The R Foundation, “CRAN Repository Policy,” R Project, [Online]. Available: http://cran.r-project.org/web/packages/policies.html.

[178] I. Feinerer, K. Hornik and D. Meyer, “Text Mining Infrastructure in R,” *Journal of Statistical Software,* vol. 25, no. 5, pp. 1-54, 31 March 2008.

[179] T. P. Jurka, L. Collingwood, A. E. Boydstun, E. Grossman and W. v. Atteveldt, “RTextTools: A Supervised Learning Package for Text Classification. R package version 1.3.9.,” 2012.

[180] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutermann and I. H. Witten, “The WEKA Data Mining Software: An Update,” *ACM SIGKKD Explorations Newsletter,* vol. 11, no. 1, pp. 10-18, June 2009.

[181] University of Waikato, “Projects related to Weka,” 2013. [Online]. Available: http://weka.wikispaces.com/Related+Projects.

[182] University of Waikato, “Text categorization with Weka,” 2013. [Online]. Available: http://weka.wikispaces.com/Text+categorization+with+Weka.

[183] C. Rose, “TagHelper Tools: Facilitating Reliable Content Analysis of Corpus Data,” Carnegie Mellon University, 2013. [Online]. Available: http://www.cs.cmu.edu/~cprose/TagHelper.html.

[184] C. P. Rose, “TagHelperTools: Tools for Supporting the Analysis of Verbal Data,” in *Artificial Intelligence in Education: Building Technology Rich Learning Contexts That Work*, 2007.

[185] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz and T. Euler, “YALE: Rapid Prototyping for Complex Data Mining Tasks,” in *12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006.

[186] Rapid-I, “Data core,” Rapid-I, 2013. [Online]. Available: http://rapid-i.com/wiki/index.php?title=Data_core.

[187] Rapid-I, “How To Extend RapidMiner 5,” Rapid-I, 2012.

[188] M. Wurst and I. Mierswa, “The Word Vector Tool and the RapidMiner Text Plugin,” Rapid-I, 2009.

[189] Rapid-I, “RapidDoc,” Rapid-I, 2013. [Online]. Available: http://rapid-i.com/content/view/185/195/lang,en/.

[190] D. M. North, Data Mining for the Masses, Global text Project, 2012.

[191] M. R. Berthold, N. Cebron, F. Dill, T. R. G. F. G. T. M. Giuseppe Di Fatta, P. Ohl, C. Sieb and B. Wiswedel, “KNIME - The Konstanz Information Miner,” in *4th Industrial Simulation Conference, ISC, Workshop on Multi-Agent Systems And Simulation, MAS&S*, 2006.

[192] KNIMETech, “KNIME - License Terms and Conditions,” KNIMEtech, [Online]. Available: http://www.knime.org/downloads/full-license.

[193] KNIMEtech, “KNIME - Services,” KNIMEtech, [Online]. Available: http://www.knime.org/services.

[194] K. Thiel, “The KNIME Text Processing Plugin,” KNIMETech.

[195] KNIMEtech, “KNIME - New Node Wizard,” KNIMEtech, [Online]. Available: http://tech.knime.org/new-node-wizard.

[196] KNIMETech, “KNIME Noding Guidelines,” KNIMETech, 2011.

[197] KNIMETech, “KNIME - Default Dialog Components,” KNIMETech, [Online]. Available: http://tech.knime.org/default-dialog-components.

[198] KNIMETech, “KNIMETech,” KNIMETech, [Online]. Available: http://tech.knime.org/home.

[199] M. R. Berthold, C. Borgelt, F. Hoppner and F. Klawoon, Guide to Intelligent Data Analysis, Springer, 2010.

[200] KNIMETech, “KNIME - Known Issues,” KNIMETech, [Online]. Available: http://tech.knime.org/known-issues.

[201] biolab, “Orange Text Mining,” [Online]. Available: https://bitbucket.org/biolab/orange-text.

[202] The University of Sheffield, “Sponsor GATE,” The University of Sheffield, [Online]. Available: http://gate.ac.uk/sponsor.html.

[203] The University of Sheffield, “GATE Partners,” The University of Sheffield, [Online]. Available: http://gate.ac.uk/partners.html.

[204] The University of Sheffield, “Plugins included in the GATE distribution,” The University of Sheffield, [Online]. Available: http://gate.ac.uk/gate/doc/plugins.html.

[205] B. Carpenter and B. Baldwin, Text Analysis with LingPipe 4, 0.5 ed., LingPipe Publishing, 2011.

[206] Apache Software Foundation, “Apache OpenNLP,” Apache Software Foundation, [Online]. Available: http://opennlp.apache.org/.

[207] T. Joachims, “SVMlight - Support Vector Machine,” Cornell University, [Online]. Available: http://svmlight.joachims.org.

[208] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, N. Aswani, I. Roberts, G. Gorrell, A. Funk, A. Roberts, D. Damljanovic, T. Heitz, M. A. Greenwood, H. Saggion, J. Petrak, Y. Li and W. Peters, “Developing Language Processing Components with GATE Version 7 (a User Guide),” The University of Sheffield, 2012.

[209] The University of Sheffield, “Simmetrics,” The Univeristy of Sheffield, [Online]. Available: http://sourceforge.net/projects/simmetrics/.

[210] Rapid-I, “Services - Professional Support,” Rapid-I, [Online]. Available: http://rapid-i.com/content/view/60/89/lang,en/.

[211] KNIMEtech, “KNIME - Partner Extension Compatibility,” KNIMEtech, [Online]. Available: http://www.knime.org/partner-extension-compatibility.

[212] Apache Software Foundation, “Apache Hadoop,” Apache Software Foundation, [Online]. Available: http://hadoop.apache.org/.

[213] Microsoft, “Microsoft Open Specifications Promise,” Microsoft, [Online]. Available: http://www.microsoft.com/openspecifications/en/us/programs/osp/default.aspx.

[214] Wikipedia, “List of CLI Languages,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/List_of_CLI_languages.

[215] IKVM.NET, “IKVM.NET Home Page,” IKVM.NET, [Online]. Available: http://www.ikvm.net/index.html.

[216] Wikipedia, “List of JVM languages,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/List_of_JVM_languages.

[217] O. Shilo, “CS-Script - The C# Script Engine,” [Online]. Available: http://www.csscript.net/.

[218] Xamarin, “Mono - Multi-platform .Net development framework,” xamarin, [Online]. Available: http://mono-project.com/Main_Page.

[219] DotGNU Project, “DotGNU Portable.Net,” DotGNU, [Online]. Available: http://www.dotgnu.org/pnet.html.

9 Glossary

* Denotes that the definition is as used throughout this thesis. Note that it might relate to a variety of different definitions when used by other sources.

Affix - A morphological addition to a word to change function or meaning. See also Prefix and Suffix.

Annotation - Dataset metadata, created prior to any phases in the classification process, which might be included in addition to or generated directly from the data content.

Basic Combined Programming Language - A simple type less language designed in 1966 by Martin Richards. Now considered obsolete it indirectly influenced the 'C' programming Language.

BCPL - See Basic Combined Programming Language.

Big Data - As yet a vaguely defined term, taken to mean a quantity of data impossible to process on a single hardware machine, instead requiring the use of cluster, cloud or grid computing.*

Category - A specific subject from the pool of subjects associated with all of the data of a particular dataset.

Class - See Category.

Cloud Computing - A collection of similar data devices that are remote to the central data requests being made. See also Cluster Computing and Grid Computing.

Cluster Computing - A collection of data devices that are localised to the central data requests being made. See also Cloud Computing and Grid Computing.

Conjunction - A word that links other words, phrases or clauses, e.g. 'and', 'or', 'when', 'but', 'because'.

Corpus - See Dataset.

Dataset - Collection of documents to be used in the categorisation process. See also Training Set and Testing Set.

Deep Copy (Programming) - Duplicating an entire object graph including all associated data and other objects it references, see also Shallow Copy.

Derivation - The modification of a word root to alter the syntactic category, often in an unpredictable manner, e.g. the noun “glory” can become the verb “glorify”, the verb “derive” can become the adjective “derivational”.

Dynamic Dispatch - The process of deciding at runtime which implementation of a polymorphic operation should be called.

Encapsulation - The process of restricting access to certain components and embedded data that form part of an entity.

Feature - See Term.

Framework - A programming API, software package or suite of software applications designed to solve a specific problem or provide a foundation for the creation of prototypes that solve a specific problem. In the context of this thesis the problem is defined as the categorisation of text documents or one of the stages involved in the process.*

Gazetteer - A geographical index or dictionary that contains reference information about places and place names.

Grid Computing - A collection of diverse data devices that are remotely scattered in relation to the central data requests being made. See also Cloud Computing and Cluster Computing.

Homonymy - Words with the same spelling and/or pronunciation that have different meanings, e.g. the ‘stalk’ of a plant and to ‘stalk’ a person.

Inflection - The modification of a word root to change the grammatical category, generally in a predictable manner by means of a prefix and/or suffix. See also Prefix and Suffix.

Intelli-sense - A programming environment feature that simplifies development by the provision of helpful attributes, such as auto completion, syntax correction and functional details.

Interface (Programming) - A software contract outlining a set of method signatures that must be implemented in its entirety by any object that inherits it.

Label - A reference to a particular category that is assigned to a document.

Lemmatisation - The process of grouping related words together under a single term in order to analyse them as one item.

Multi-Class - Denotes that a text categorisation problem has more than a single class that documents are grouped under.*

Multi-Label - Denotes that a text document can be correctly categorised under more than a single label. A Multi-Label text categorisation problem defines a Multi-Class problem where at least one of the documents being categorised has multiple labels associated with it.*

N-Label - Denotes each text document is correctly categorised under exactly 'n' labels. An N-Label text categorisation problem defines a Multi-Label problem where all of the documents being categorised have exactly 'n' labels associated with them.*

Phase - See Stage.

Platform - Combination of hardware and software technologies that comprises a computing system.

Polysemy - One word with several distinct but related meanings, often considered a distinct form of homonymy, e.g. the 'mouth' of a river and the 'mouth' of an animal. See also Homonymy.

Prefix - A morphological addition to the beginning of a word to alter its context, e.g. 'un', 'dis', 'im', 'sub', 'pre'.

Preposition - A word that links nouns, pronouns and phrases to other words in a sentence, e.g. 'on', 'beneath', 'above', 'during', 'with'.

Problem Scope - The range of possible variable, parameter or textual symbol permutations associated with a particular combination of corpus and solution strategy being employed.*

Pronoun - A word or form that substitutes for a noun, noun phrase or another pronoun, e.g. 'I', 'we', 'you', 'they', 'which'.

Shallow Copy (Programming) - Duplicating an object by only copying its basic variables and reference pointers to the other objects it contains, see also Deep Copy.

Single-Class - A binary text categorisation problem containing only a single class, where each document either does or does not belong to that class.

Single-Label - A text categorisation problem that explicitly allows only a single label to be assigned to each text document.*

Stage - One of the main parts required in the process of categorising text documents.*

Stemming - The process of converting a word to its stem, base or root form, commonly by the removal of a prefix and/or suffix. See also Prefix and Suffix.

Subtype Polymorphism - The ability to define multiple forms for a single element, allowing different processes to occur through means of a common interface.

Suffix - A morphological addition to the end of a word to alter its context, e.g. 'ing', 'er', 'ed', 's', 'ly'.

Symbol - See Term.

Synonymy - Different words with varied spelling that have the same inherent meaning, e.g. 'vehicle' and 'automobile'.

Term - A single unit of information that represents a portion of and is derived from a textual document, it may also contain a level of semantic relationship to other terms in the document.

Test/Testing Document - A previously unlabelled document supplied to a classification system in order to be categorised.

Testing Set - A collection of Testing Documents, used to test the performance of a classifier.

Text Categorisation - Refers to the entire process of assigning labels to text documents, including all of the stages involved.* It is often shortened to 'Categorisation'.*

Text Classification - Refers only to the stage of text categorisation that involves the use of an actual classification algorithm.* It is often shortened to 'Classification'.* See also 'Text Categorisation'.

Token - See Term.

Topic - See Category.

Training Document - A previously categorised document used as comparative data for the construction or definition of a classification system.

Training Set - A collection of Training Documents, used to construct a classifier.

WordNet - An extensive semantically-oriented dictionary, similar in structure to a thesaurus but containing semantic relations, definitions and example use-cases for any given term.

10 Mathematical Notation

10.1 Symbolic Notation

C	Set of categories $\{C0, \ldots, Cn, \ldots, C\|C\|\}$.
$\|C\|$	Total number of categories for an entire dataset, the magnitude of C.
C_i	Specific category from the set of categories C.
c	Category, from the set of categories C.
c_d	Category associated with document d.
$c_{\bar{d}}$	Category not associated with document d.
c_t	Category associated with term t.
$c_{\bar{t}}$	Category not associated with term t.
c_{dt}	Category associated with document d and term t.
$c_{d\bar{t}}$	Category associated with document d but not term t.
$c_{\bar{d}t}$	Category not associated with document d but with term t.
$c_{\bar{d}\bar{t}}$	Category not associated with document d or term t.
D	Set of documents $\{D0, \ldots, Dn, \ldots, D\|D\|\}$.
$\|D\|$	Total number of documents, the magnitude of D.
D_j	Specific document from the set of documents D.
d	Document from the set of documents D.
d_c	Document associated with category c.
$d_{\bar{c}}$	Document not associated with category c.
d_t	Document containing term t.
$d_{\bar{t}}$	Document not containing term t.
d_{ct}	Document associated with category c and containing term t.
$d_{c\bar{t}}$	Document associated with category c but not containing term t.
$d_{\bar{c}t}$	Document not associated with category c but containing term t.
$d_{\bar{c}\bar{t}}$	Document not associated with category c or containing term t.
T	Set of derived terms $\{T0, \ldots, Tn, \ldots, T\|T\|\}$.
$\|T\|$	Total number of derived terms, the magnitude of T.
T'	Modified set of derived terms T.
$\|T'\|$	Total number of terms for modified dataset, the magnitude of T'.
T_k	Specific term from the set of derived terms T.
$\|T_{kd}\|$	Total occurrence of specific term T_k in document d.
t	Term from the set of derived terms T.
t_d	Term contained by document d.
$t_{\bar{d}}$	Term not contained by document d.

t_c	Term associated with category c.
$t_{\bar{c}}$	Term not associated with category c.
t_{dc}	Term contained by document d and associated with category c.
$t_{d\bar{c}}$	Term contained by document d but not associated with category c.
$t_{\bar{d}c}$	Term not contained by document d but associated with category c.
$t_{\bar{d}\bar{c}}$	Term not contained by document d or associated with category c.
$[a]$	Absolute value of a.
$P(a)$	Probability of a.
$P(a\|b)$	Probability of a given b already happened (conditional probability).
$P(a\&b)$	Probability of a and b, when a follows b (conjunctional probability).
$F(a\|b)$	Frequency of a in the restricted scope of b.
$\{a:b\}$	Set of a conforming to condition b.
$\|\{a:b\}\|$	Total number of a conforming to condition b, the magnitude of $\{a:b\}$.
w_{td}	Weight of term t with respect to document d for weighting function w.
w_{kj}	Weight of specific term T_k with respect to specific document D_j.
w_{Tj}	Total weighting for all terms in set T with respect to specific document D_j.
w_{kD}	Total weighting of specific term T_k with respect to all documents in set D.

10.2 Explanation of Definitions

The dataset of all documents in a corpus is denoted as D, the total number of documents as $|D|$, an individual document as d and a specific document as D_j, where $D = \{D_1, \ldots, D_j, \ldots, D_{|D|}\}$.

The set of categories relating to an entire dataset is denoted as C, the total number of categories as $|C|$, an individual category as c and a specific category as C_i, where $C = \{C_1, \ldots, C_i, \ldots, C_{|C|}\}$.

The set of unique terms derived from the dataset, which may have already been subject to one or more processing stages, is denoted as T , an individual term as t and a specific term as T_k, where $T = \{T_1, \ldots, T_k, \ldots, T_{|T|}\}$.

In addition a document d that is associated with a category c is designated as d_c, which contains a term t as d_t and which has a amalgamation of both as d_{ct}, with a corresponding lack of relation being denoted by either $d_{\bar{c}}$, $d_{\bar{t}}$ or $d_{\bar{c}\bar{t}}$. This format is also employed in a similar fashion for both categories (c_d, c_t, c_{dt}, $c_{\bar{d}}$, $c_{\bar{t}}$, $c_{\bar{d}\bar{t}}$) and terms (t_d, t_c, t_{dc}, $t_{\bar{d}}$, $t_{\bar{c}}$, $t_{\bar{d}\bar{c}}$) and allows a mixture of associations and non associations

to be represented by a combination of formats, for example $d_{c\bar{t}}$ designates a document d that has an association with category c but does not contain term t.

10.3 Probability Calculations

Probability of a document being associated with a specific category, marginal probability of c in relation to d:

$$P(d_c) = |\{d_c : d_c \in D\}| / |D|$$

Probability of a document containing a particular term, marginal probability of t in relation to d:

$$P(d_t) = |\{d_t : d_t \in D\}| / |D|$$

Probability of a document being associated with a specific category and containing a particular term:

$$P(d_{ct}) = |\{d_{ct} : d_{ct} \in D\}| / |D|$$

Conditional probability of a given b, $P(a|b)$ is the probability of a given that b has already happened.

Probability of a document being associated with a specific category on the condition that it already contains a particular term:

$$P(d_{ct}|d_t) = (|\{d_{ct} : d_{ct} \in D\}| / |D|) / (|\{d_t : d_t \in D\}| / |D|)$$
$$P(d_{ct}|d_t) = |\{d_{ct} : d_{ct} \in D\}| / |\{d_t : d_t \in D\}|$$
$$P(d_{ct}|d_t) = P(d_{ct}) / P(d_t)$$

Probability of a document containing a particular term on the condition that it is already associated with a specific category:

$$P(d_{ct}|d_c) = (|\{d_{ct} : d_{ct} \in D\}| / |D|) / (|\{d_c : d_c \in D\}| / |D|)$$
$$P(d_{ct}|d_c) = |\{d_{ct} : d_{ct} \in D\}| / |\{d_c : d_c \in D\}|$$
$$P(d_{ct}|d_c) = P(d_{ct}) / P(d_c)$$

Joint probability of a and b happening in conjunction, $P(a\&b)$ is probability that a happens and then b happens in that order.

Probability of a document being associated to a specific category in conjunction with it containing a particular term first:

$$P(d_{ct}\&d_t) = (|\{d_{ct}:d_{ct} \in D\}| / |\{d_t:d_t \in D\}|) \cdot (|\{d_t:d_t \in D\}| / |D|)$$
$$P(d_{ct}\&d_t) = P(d_{ct}|d_t) \cdot P(d_t)$$

Probability of a document containing a particular term in conjunction with it being associated to a specific category first:

$$P(d_{ct}\&d_c) = (|\{d_{ct}: d_{ct} \in D\}| / |\{d_c:d_c \in D\}|) \cdot (|\{d_c:d_c \in D\}| / |D|)$$
$$P(d_{ct}\&d_c) = P(d_{ct}|d_c) \cdot P(d_c)$$

Note that $P(d_{ct}\&d_t)$ and $P(d_{ct}\&d_c)$ are equivalent, despite the ordering, along with $P(d_{ct})$:

$$P(d_{ct}\&d_t) \equiv P(d_{ct}\&d_c) \equiv P(d_{ct})$$

10.4 General Notes

Example of equality between different definition formats.

$$D = d_t + d_{\bar{t}}$$

$$|D| = |d_t| + |d_{\bar{t}}|$$

$$|\{dt:dt \in D\}| = F(d_t|D)$$

$$P(d_t) = |d_t| / |D| = |d_t| / (|d_t| + |d_{\bar{t}}|)$$

11 Appendices

11.1 Appendix A: Reuters-21578 ModApte Split

The dataset used throughout the experiments was the ModApte split of the Reuters-21578 corpus.

Documents from April 7th 1987 and before are assigned to the training set and documents from April 8th 1987 and after to the test set.

Content length is denoted in characters and all values have been rounded to five decimal places.

11.1.1 Raw Data Structure

Individual document entries in the original SGML files use the following formatting:

```
<REUTERS TOPICS=?? LEWISSPLIT=?? CGISPLIT=?? OLDID=?? NEWID=??>
<DATE> </DATE>
<MKNOTE> </MKNOTE>
<TOPICS>...<D></D>... </TOPICS>
<PLACES> </PLACES>
<PEOPLE> </PEOPLE>
<ORGS> </ORGS>
<EXCHANGES> </EXCHANGES>
<COMPANIES> </COMPANIES>
<UNKNOWN> </UNKNOWN>
<TEXT> <TEXT TYPE="BRIEF"> <TEXT TYPE="UNPROC">
<AUTHOR> </AUTHOR>
<DATELINE> </DATELINE>
<TITLE> </TITLE>
<BODY> </BODY>
</TEXT>
</REUTERS>
```

11.1.1.1 Training and Testing Set Designation

Training Set: LEWISSPLIT="TRAIN"; TOPICS="YES"

Test Set: LEWISSPLIT="TEST"; TOPICS="YES"

Unused: LEWISSPLIT="NOT-USED"; TOPICS="YES"
or TOPICS="NO"
or TOPICS="BYPASS"

11.1.2 Dataset Statistics

After applying the split using the designations outlined above the data has the following statistics.

11.1.2.1 Entire Dataset

Number of Documents: 12902
Unique Categories: 118
Unique Characters: 89
Category Assignments: 13395

Minimum Document Length: 37
Maximum Document Length: 13432
Mean Document Length: 803.73570
Total Content Length: 10369798

Minimum Assigned Categories per Document: 0
Document with Minimum Assignments: 2108
Maximum Assigned Categories per Document: 16
Document with Maximum Assignments: 1
Mean Assigned Categories per Document: 1.03821

Minimum Assignments per Category: 1
Categories with Minimum Assignments: 16
Maximum Assignments per Category: 3964
Categories with Maximum Assignments: 1 (“Earn”)
Mean Assignments per Category: 113.51695

11.1.2.2 Training Set

Number of Documents: 9603
Unique Categories: 115
Unique Characters: 87
Category Assignments: 9648

Minimum Document Length: 37
Maximum Document Length: 8604

Mean Document Length: 817.47725
Total Content Length: 7850234

Minimum Assigned Categories per Document: 0
Document with Minimum Assignments: 1828
Maximum Assigned Categories per Document: 16
Document with Maximum Assignments: 1
Mean Assigned Categories per Document: 1.00469

Minimum Assignments per Category: 1
Categories with Minimum Assignments: 20
Maximum Assignments per Category: 2877
Categories with Maximum Assignments: 1 (“Earn”)
Mean Assignments per Category: 81.76271 (categories in entire corpus)
Mean Assignments per Category: 83.89565 (categories in training set)

Categories Assignments Common to Train and Test Sets: 9585
Mean Assignments per Common Category: 106.5

11.1.2.3 Testing Set

Number of Documents: 3299
Unique Categories: 93
Unique Characters: 81
Category Assignments: 3747

Minimum Document Length: 38
Maximum Document Length: 13432
Mean Document Length: 763.73568
Total Content Length: 2519564

Minimum Assigned Categories per Document: 0
Document with Minimum Assignments: 280
Maximum Assigned Categories per Document: 14
Document with Maximum Assignments: 2
Mean Assigned Categories per Document: 1.13580

Minimum Assignments per Category: 1
Categories with Minimum Assignments: 17

Maximum Assignments per Category: 1087
Categories with Maximum Assignments: 1 (“Earn”)
Mean Assignments per Category: 31.75424 (categories in entire corpus)
Mean Assignments per Category: 40.29032 (categories in testing set)

Categories Assignments Common to Train and Test Sets: 3744
Mean Assignments per Common Category: 41.6

11.1.2.4 Common Data

Unique Characters: 79
Unique Categories: 90
Category Assignments: 13329

Categories with One Training Document: 7
Categories with One Testing Document: 3
Categories with One Document in Both: 5

Mean Assignments per Category: 148.1
Mean Categories per Document: 1.03310

Train to Test Assignments per Category Ratio: 2.56010 to 1

11.1.2.5 Exclusive Training Data

Unique Characters: 8
Unique Categories: 25
Category Assignments: 63

11.1.2.6 Exclusive Testing Data

Unique Characters: 2
Unique Categories: 3
Category Assignments: 3

11.2 Appendix B: Programming Language Notes

The majority of existing data mining frameworks and text processing utilities are composed mainly in the Java or Python programming languages, which are critically examined here.

11.2.1 Java Issues

Started at Sun Microsystems in June 1991, Java was first made publically available in 1995 and much of it was released as free and open source software (FOSS) on 13th November 2006 under the terms of the GNU General Public License (GPL). Aside from a small portion of code that Sun did not hold copyright for, all remaining core libraries were eventually released under open source distribution by 8th May 2007. Since then development was conducted only by the public community, until ownership transferred to Oracle Corporation when they acquired Sun Microsystems on 7th January 2010. A new version was subsequently released, though it has never been officially formalised and is considered a de facto language.

For a number of years Java seems to have become less active and popular among developers and in commercial environments, with an unstable future that is dependent on whether Oracle decides to give it an appropriate level of support. One of its three initial creators, James Gosling, also left Oracle shortly after the takeover of Sun Microsystems on 2nd April 2010, due to several reasons related to unfavourable changes in the work environment. Currently it remains active mostly due to deployment in mobile devices as part of the 'Android' SDK produced by Google, though Oracle immediately began a legal case against them directly because of this. Another area where it is heavily used is enterprise level applications, though this is sometimes due to tradition and an unwillingness to move to different languages rather than being based on technical motives.

Recently major security risks have been revealed in the language, including the latest release and dating back to 2004, causing prominent manufacturers to boycott it. A flaw was discovered on 31st August 2012 that affected Microsoft Windows, Mac OS X and Linux that allowed remote exploits to occur simply by loading a malicious webpage. While three computer specialists were prompted to speak out against Java again on 10th January 2013, informing Reuters that it was not secure and advising the general public to disable it. Security alerts from Oracle announced that critical patches were scheduled for production, but as of 14th January 2013 experts stated that prior attempts to protect from known attacks had failed. Consequently the issue provoked response from the United States Department of Homeland Security, which encouraged every user to completely disable or uninstall Java.

Apple has since been blacklisting all insecure versions of Java on products running their proprietary Mac OS X operating system. While previous to this in 1997 Sun Microsystems sued Microsoft over a

breach of terms relating to an incomplete implementation of Java version 1.1 and won a settlement of US$20 million, along with a court order that enforced the terms of the license. However, as a direct result of this Microsoft also stopped including implementations of the Java run time environment with releases of the Widows operating System and other products.

11.2.1.1 Java Timeline

January 23, 1996	JDK 1.0 released
February 19, 1997	JDK 1.1 released
October, 1997	Legal case begins against Microsoft
December 8, 1998	J2SE 1.2 released (Playground)
May 8, 2000	J2SE 1.3 released (Kestrel)
January, 2001	Microsoft starts to phase out Java
February 6, 2002	J2SE 1.4 released (Merlin)
September 30, 2004	J2SE 5.0 released (Tiger)
November 13, 2006	Start of release as free and open source software (FOSS)
December 11, 2006	Java SE 6 released (Mustang)
May 8, 2007	Completion of release to FOSS
January 7, 2010	Oracle Corporation purchase Sun Microsystems
April 2, 2010	James Gosling ‘the father of Java’ leaves Oracle
July 28, 2011	Java SE 7 released (Dolphin)
August 31, 2012	Major security flaw discovered in Java version 5, 6 and 7
January 10, 2013	Experts advise against Java
January 14, 2013	Security patches prove inadequate - Apple starts blacklisting Java

11.2.2 Python Issues

Python is deliberately designed with the notion of clarity over performance, so code is easier to read without being complicated by optimisations, meaning it is not always the best choice for memory or computationally intensive tasks like data mining. It also requires third party utilities to be compiled into an executable format, otherwise everything is done at runtime, which makes it slower and prevents many of the automatic enhancements available to other languages. Some versions suffer from multi-threading issues as well and are unable to utilise multiple processors where they are present in a system, causing graphical interfaces to slow down or freeze during moments of heavy computation.

While the lack of compilation can lead to rapid deployment it does mean that errors occur at runtime and are generally more difficult to debug. Python also cannot distinguish between the declaration and assignment of a variable and has no read only construct, making it prone to errors if existing variables

are overwritten or left undefined. Another caveat is its use of restrictive formatting, with code grouping being determined by exact indents rather than the braces similar to many other languages, which results in compact code but can cause issues when whitespace errors need to be resolved. However, there are a number of third party development environments available that are suitable for reducing these types of problem.

From an architectural perspective there are no access modifiers present for variables and everything is stored in plaintext without compilation, so it is impossible to limit alteration of particular components or guide development through the use of specific publicly accessible interfaces. Adding complexity to this issue is the ability to deal with multiple class inheritance, as while this can be a positive feature it may also be confusing due to the typed order of inheritance defining which method is called if two inherited classes both have instances with the same method signature.

Python generates the list of inherited classes and methods using the C3 linearization algorithm, which enforces two constraints; children precede their parents and if a class inherits from multiple classes they are kept in the order specified by the tuple that defines them. For example given a class D that inherits B and C in that order, where B and C both inherit A, the resultant method resolution would be D, B, C, A. This means that in some cases it is possible for generalised classes to precede those typically considered more specialised.

Continuously evolving there are regular updates to the language, though the last major release was version 3.0 on 3rd December 2008 and included several aspects that broke backward compatibility with prior versions. However, these changes are well documented and reasonable online resources are available for all versions dating back to 1996. A sizeable number of manifestations of the python interpreter along with various utilities and extensions also exist, each exhibiting varied semantics and functionality that overcomes a limitation of the standard language, such as debugging, static typing, performance or the ability to compile it. Most of these are not part of the core package though and are independently maintained by individuals or companies, so they must be separately acquired and installed, while the level of support, documentation and quality varies significantly.

11.3 Appendix C: Weka Data File Formats

Weka is the only framework reviewed that explicitly requires data to be entered in a bespoke format.

The information compiled here has been gathered from various sources, including the main Weka online documentation and the official user manual.

11.3.1 Attribute-Relation File Format (ARFF)

Attribute-Relation File Format files are composed of ASCII characters and employ a custom layout to describe data instances that all share a common set of attributes.

11.3.1.1 General Details

```
@relation <relation-name>

@attribute <attribute-name> <datatype>

@data
<data-instance>, {<data-instance-weight>}
...
<data-instance>, {<data-instance-weight>}
```

- <relation-name>/<attribute-name> must be surrounded by quotes if it contains spaces.
- <attribute-name> must start with an alphabetic character.
- <datatype> can be any of the types supported by Weka: numeric (integer, real), string, <nominal-specification>, date [<date-format>], relational (reserved for future use by multi-instance data).
- <nominal-specification> takes form {<nominal-name1>, <nominal-name2>, ...}
- <date-format> supports date parsing formats of the 'java.text.SimpleDateFormat' Java class.
- relational attributes take the following format:

```
@attribute <name> relational
@attribute <attribute-name> <datatype>
...
@attribute <attribute-name> <datatype>
@end <name>
```

- <data-instance> is a comma separated list of attribute values given in the order defined.
- <data-instance-weight> is optional (defaulting to a weight of 1) and relates to a specific weighting for each individual data instance.

- The @RELATION, @ATTRIBUTE and @DATA declarations are case insensitive.
- "%" denotes a comment and is ignored during processing.
- Attributes are in a strict order related to the data/column ordering.
- Data type names are case insensitive.
- Commas can have zero or more spaces before them.
- Missing values are depicted by a single question mark in the appropriate attribute ordering.
- String data containing spaces or the comment symbol '%' must be quoted.
- Date formats containing spaces must be quoted.
- Relational data must be enclosed by double quotes.

11.3.1.2 ARFF Example

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
%
@RELATION iris

@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}

@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
```

11.3.2 Sparse Attribute-Relation File Format

Sparse ARFF files are similar to standard ARFF files but do not need to explicitly define 0 values, instead they can declare values along with their zero-based index in the data string.

In prior versions there was a bug when using string values in Sparse ARFF files, due to them being stored in a vector with a zero-based numerical index pointer which causes the first string to be considered invalid. This bug was reported as fixed on 29/01/2013.

```
@data
0, X, 0, Y, "class A", {0.3}
0, 0, W, 0, "class B", {0.55}
```

is equivilent to

```
@data
{1 X, 3 Y, 4 "class A"}, {0.3}
{2 W, 4 "class B"}, {0.55}
```

- Missing values must be explicitly marked with a question mark.

11.3.3 Extensible Attribute-Relation File Format (XRFF)

Data files can also be created in XML to make them easier to understand by using the eXtensible Attribute-Relation File Format, which provides all the functionality of the standard ARFF but is more verbose.

11.3.3.1 General Details

- XRFF files can be gzipped to compensate for their verbosity and WEKA automatically recognises this by the addition of '.gz' to the filename.
- XRFF contains all ARFF features but allows inclusion of other specialised values: instance weights, class attribute specification and attribute weights.
- Instance weights are defined by the addition of a "weight" attribute to the individual instance elements (the default value is 1):

```
<instances>
        <instance weight="0.75">
                <value>4.9</value>
        </instance>
        ...
</instances>
```

- If the 'class="yes"' attribute is added to an attribute element in the header it denotes is should be used as the class for the purpose of classification.

```
<attribute class="yes" name="class" type="nominal">
```

- Attribute weights can be added as meta data in the header section as follows:

```
                <attribute name=" sepallength" type="numeric">
                        <metadata>
                                <property name="weight">0.9</property>
                        </metadata>
                </attribute>
```

11.3.3.2 XRFF Example

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE dataset
      [
      <!ELEMENT dataset (header,body)>
      <!ATTLIST dataset name CDATA #REQUIRED>
      <!ATTLIST dataset version CDATA "3.5.4">

      <!ELEMENT header (notes?,attributes)>
      <!ELEMENT body (instances)>
      <!ELEMENT notes ANY>   <!--  comments, information, copyright, etc. -->

      <!ELEMENT attributes (attribute+)>
      <!ELEMENT attribute (labels?,metadata?,attributes?)>
      <!ATTLIST attribute name CDATA #REQUIRED>
      <!ATTLIST attribute type (numeric|date|nominal|string|relational) #REQUIRED>
      <!ATTLIST attribute format CDATA #IMPLIED>
      <!ATTLIST attribute class (yes|no) "no">
      <!ELEMENT labels (label*)>   <!-- only for type "nominal" -->
      <!ELEMENT label ANY>
      <!ELEMENT metadata (property*)>
      <!ELEMENT property ANY>
      <!ATTLIST property name CDATA #REQUIRED>

      <!ELEMENT instances (instance*)>
      <!ELEMENT instance (value*)>
      <!ATTLIST instance type (normal|sparse) "normal">
      <!ATTLIST instance weight CDATA #IMPLIED>
      <!ELEMENT value (#PCDATA|instances)*>
      <!ATTLIST value index CDATA #IMPLIED>   <!-- 1-based index (only for format "sparse") -->
      <!ATTLIST value missing (yes|no) "no">
      ]
>

<dataset name="iris" version="3.5.3">
      <header>
            <attributes>
                  <attribute name="sepallength" type="numeric"/>
                  <attribute name="sepalwidth" type="numeric"/>
                  <attribute name="petallength" type="numeric"/>
                  <attribute name="petalwidth" type="numeric"/>
                  <attribute class="yes" name="class" type="nominal">
                        <labels>
                              <label>Iris-setosa</label>
                              <label>Iris-versicolor</label>
                              <label>Iris-virginica</label>
                         </labels>
                  </attribute>
```

```
        </attributes>
    </header>

    <body>
        <instances>
            <instance>
                <value>5.1</value>
                <value>3.5</value>
                <value>1.4</value>
                <value>0.2</value>
                <value>Iris-setosa</value>
            </instance>
            ...
        </instances>
    </body>
</dataset>
```

11.3.4 Sparse Extensible Attribute-Relation File Format

An equivalent to Sparse ARFF can also be represented using the following XML format.

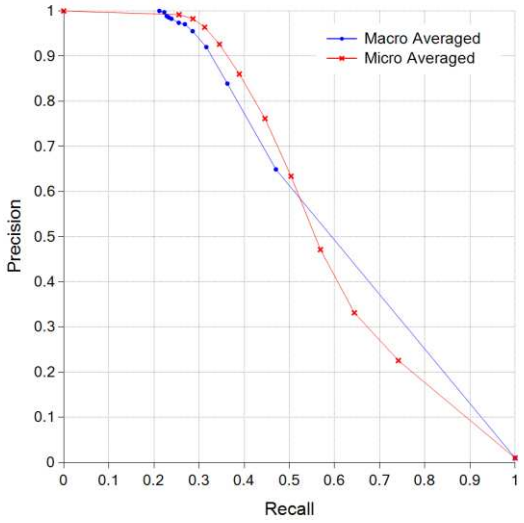
```
<instances>
        <instance type="sparse">
                <value index="1">5.1</value>
                <value index="2">3.5</value>
                <value index="3">1.4</value>
                <value index="4">0.2</value>
                <value index="5">Iris-setosa</value>
        </instance>
         ...
</instances>
```

- The “index” attribute is the 1-based position of the related attribute in the header definition.
- A “type” attribute of "sparse" must be applied to particular instance element using this format.

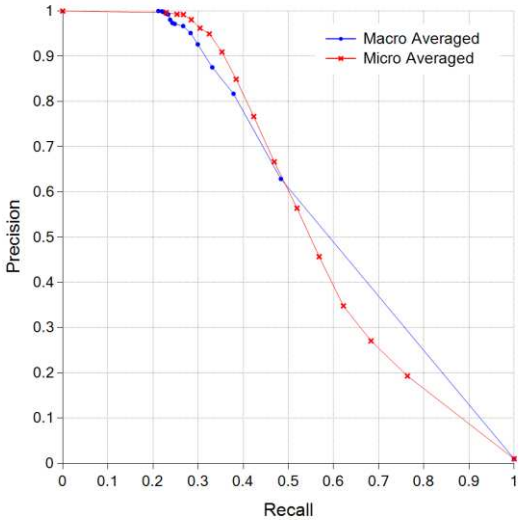
11.4 Appendix D: Experimental Data

Higher resolution and supplemental results gathered during experiments are provided here, with the value in brackets being the selected optimal threshold that corresponds to the evaluation measure.

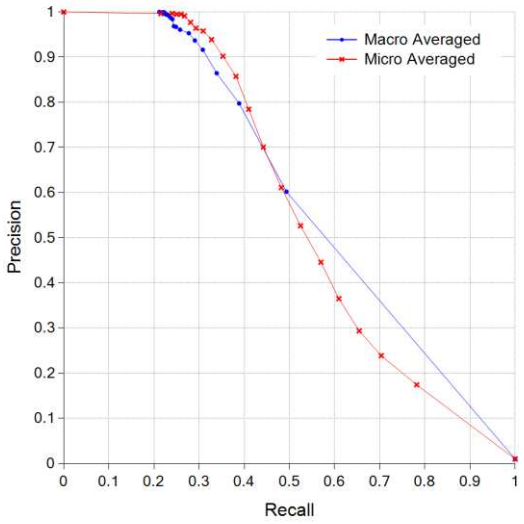
11.4.1 Benchmark K-Value Variations



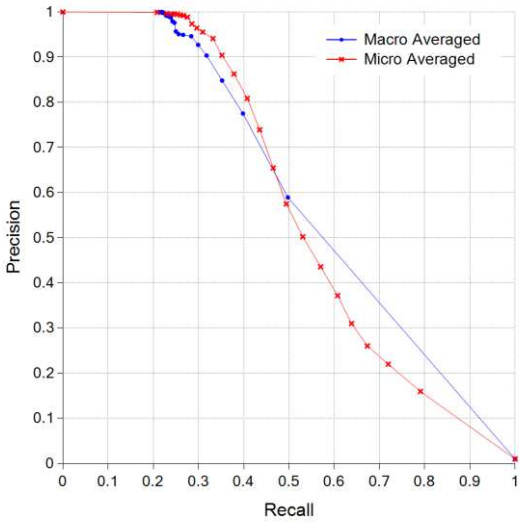
K-Value: 10
Macro F1: 0.54526 (1)
Micro F1: 0.56253 (5)
Macro BEP: 0.55954 (1)
Micro BEP: 0.52004 (3)



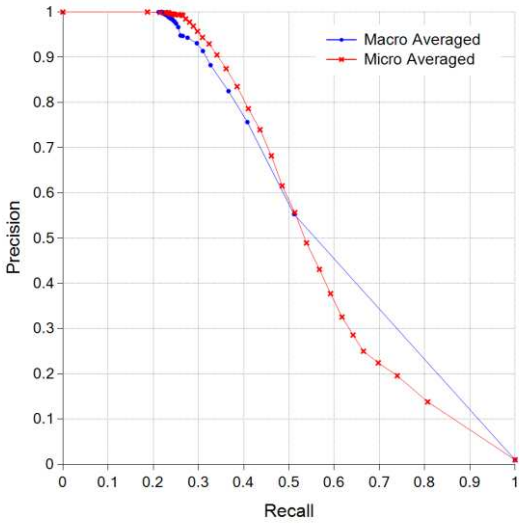
K-Value: 15
Macro F1: 0.54649 (1)
Micro F1: 0.55030 (6)
Macro BEP: 0.55600 (1)
Micro BEP: 0.54151 (5)



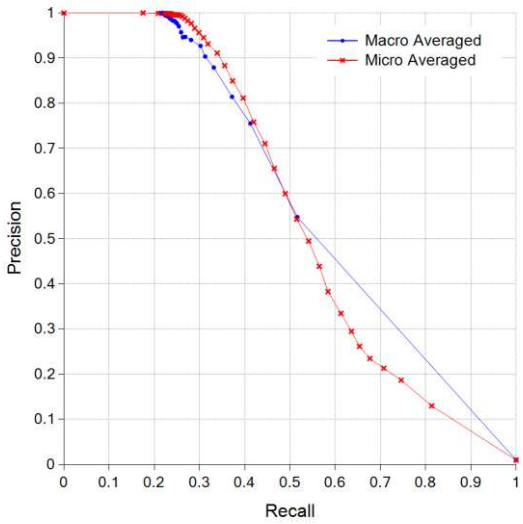
K-Value: 20
Macro F1: 0.54225 (1)
Micro F1: 0.54221 (8)
Macro BEP: 0.54761 (1)
Micro BEP: 0.52546 (6)



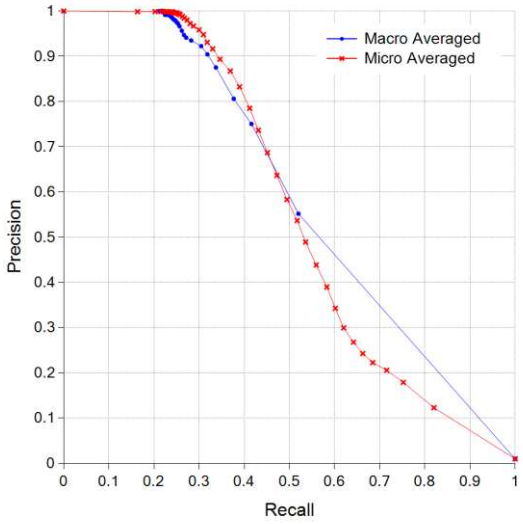
K-Value: 25
Macro F1: 0.53949 (1)
Micro F1: 0.54787 (10)
Macro BEP: 0.54334 (1)
Micro BEP: 0.51623 (7)



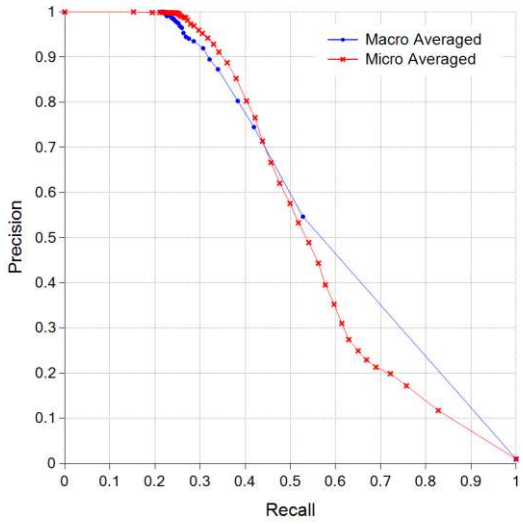
K-Value: 35
Macro F1: 0.53119 (1)
Micro F1: 0.55023 (12)
Macro BEP: 0.53201 (1)
Micro BEP: 0.53470 (10)



K-Value: 40
Macro F1: 0.53350 (2)
Micro F1: 0.54719 (14)
Macro BEP: 0.53209 (1)
Micro BEP: 0.52907 (11)

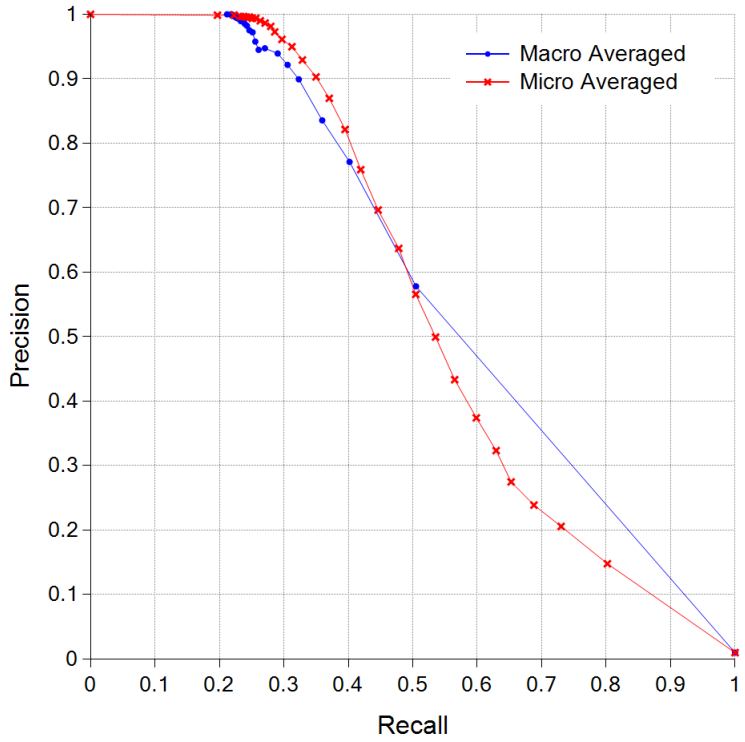


K-Value: 45
Macro F1: 0.53536 (1)
Micro F1: 0.54436 (15)
Macro BEP: 0.53583 (1)
Micro BEP: 0.52695 (12)



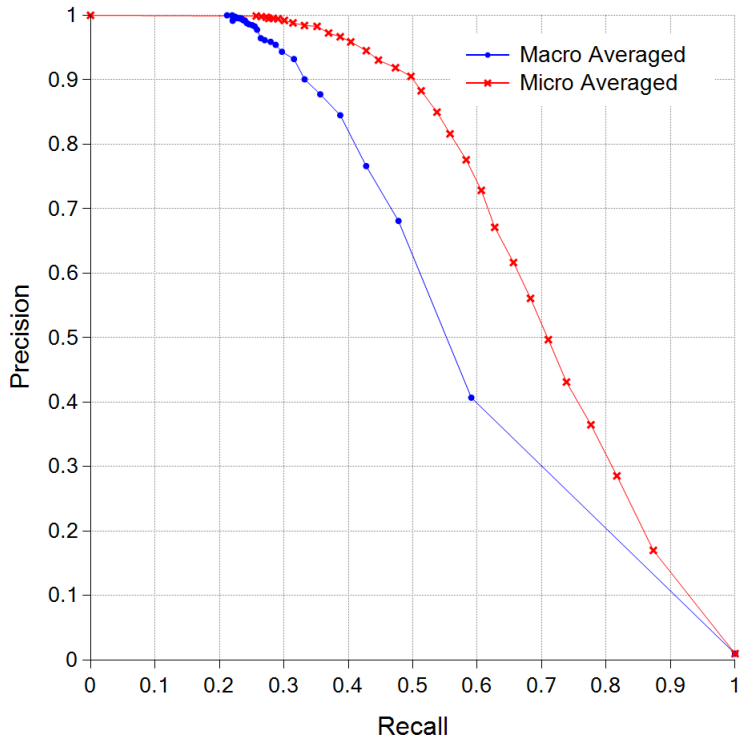
K-Value: 50
Macro F1: 0.53692 (1)
Micro F1: 0.54380 (18)
Macro BEP: 0.53708 (1)
Micro BEP: 0.52528 (13)

11.4.2 Benchmark



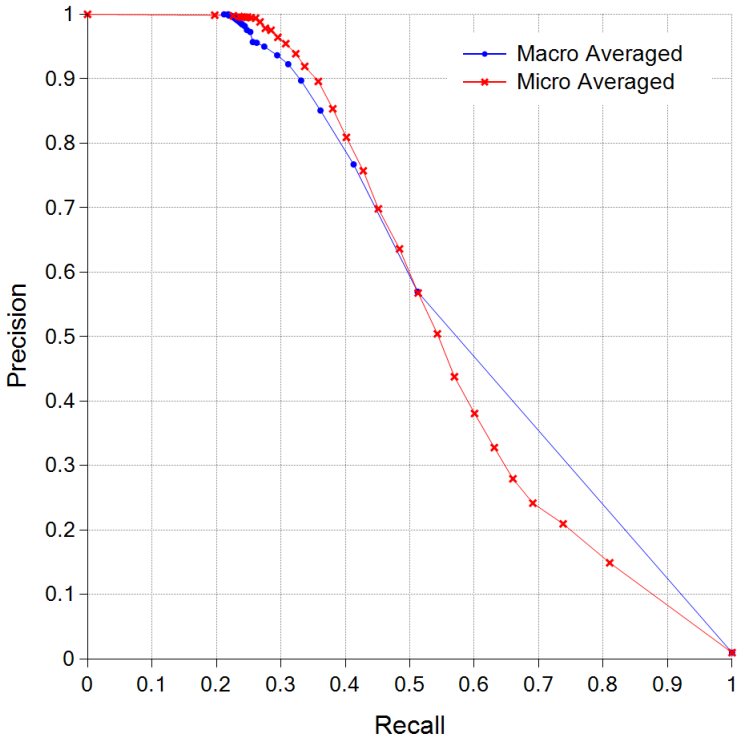
K-Value:	30	
Macro F1:	0.53908	(1)
Micro F1:	0.54626	(10)
Macro BEP:	0.54154	(1)
Micro BEP:	0.51731	(8)

11.4.3 Pre-Processing - Formatting



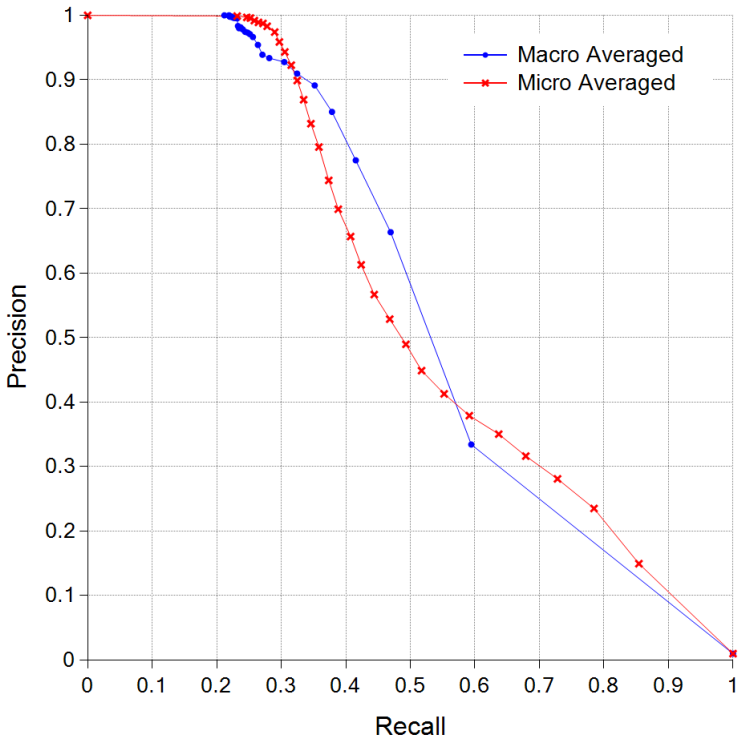
K-Value:	30	
Macro F1:	0.56155	(2)
Micro F1:	0.66565	(10)
Macro BEP:	0.49880	(1)
Micro BEP:	0.63635	(7)

11.4.4 Pre-Processing - Stemming



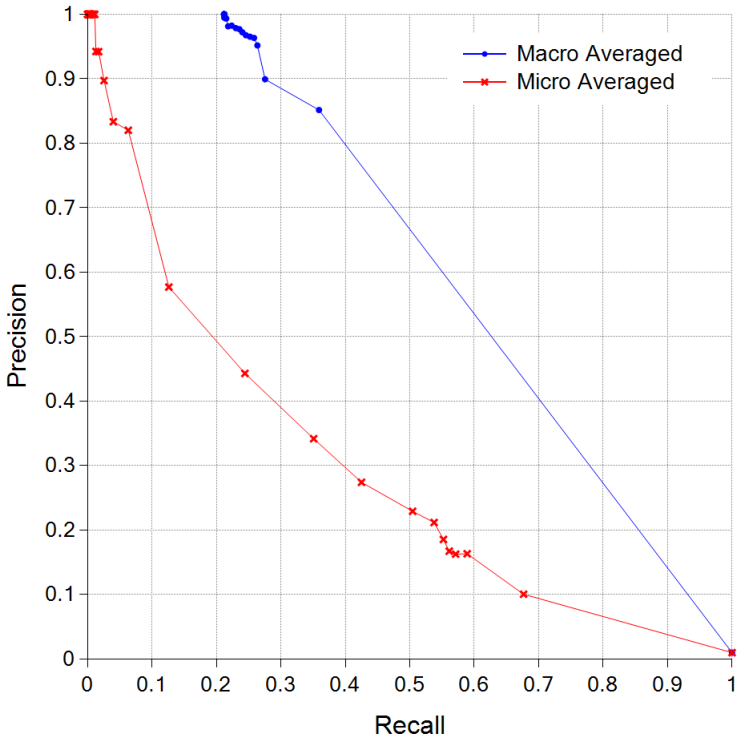
K-Value:	30	
Macro F1:	0.53934	(1)
Micro F1:	0.54978	(10)
Macro BEP:	0.54087	(1)
Micro BEP:	0.52352	(8)

11.4.5 Representation - N-Grams



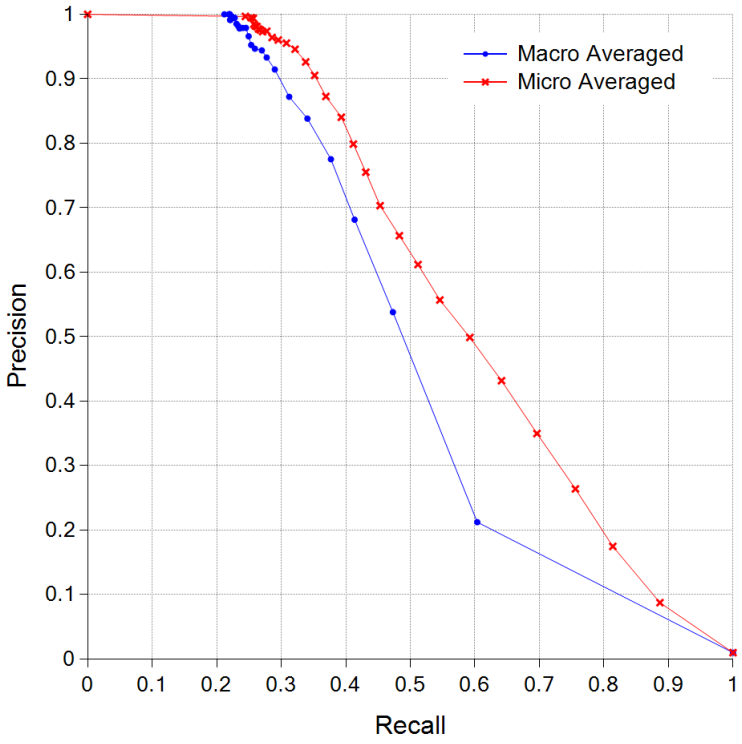
Term Type:	Characters	
N-Gram Size:	3	
K-Value:	30	
Macro F1:	0.54989	(2)
Micro F1:	0.50296	(13)
Macro BEP:	0.56649	(2)
Micro BEP:	0.49137	(9)

11.4.6 Representation - Term Grouping



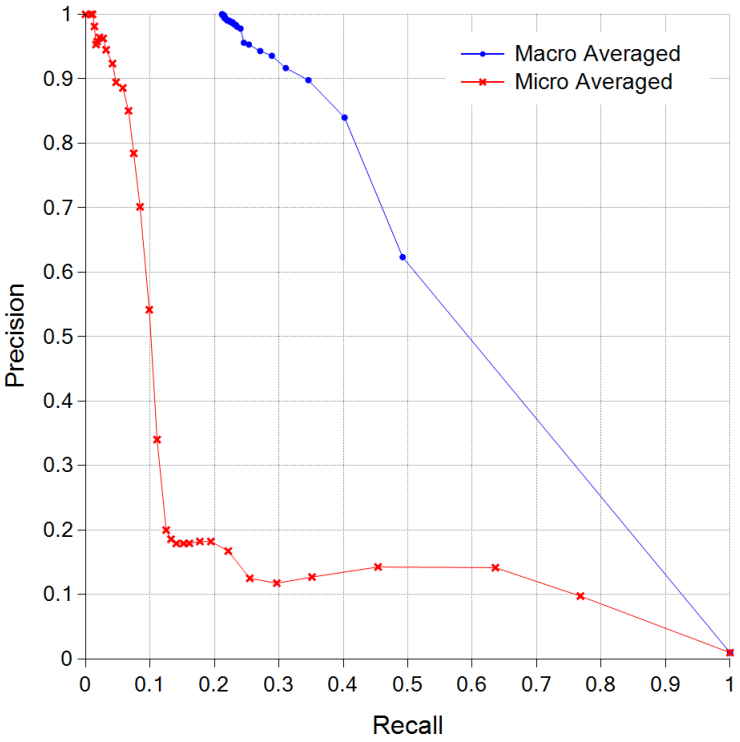
Term Type:	Words	
Group Size:	2	
K-Value:	30	
Macro F1:	0.50525	(1)
Micro F1:	0.34619	(9)
Macro BEP:	0.60535	(1)
Micro BEP:	0.34625	(9)

11.4.7 Indexing - Normalised Term Frequency



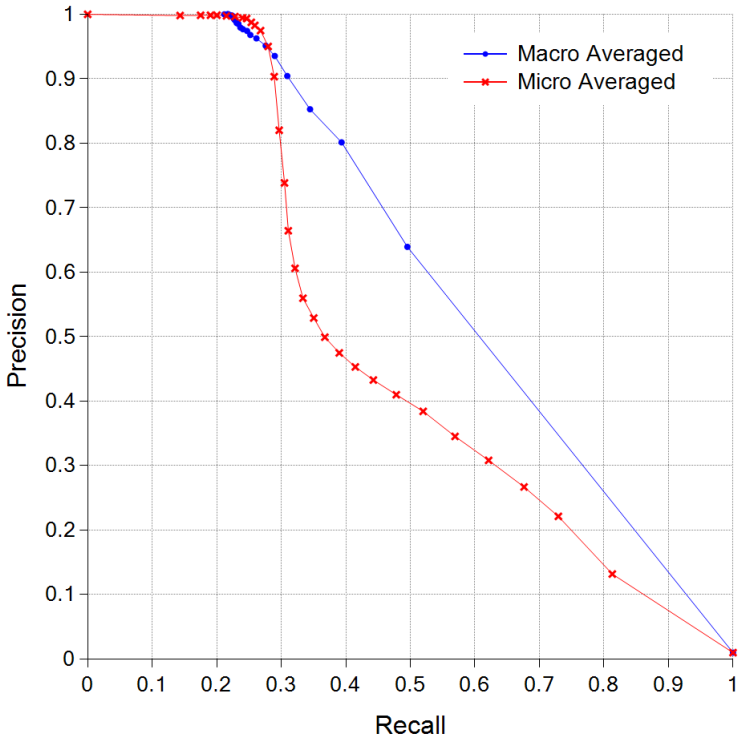
K-Value:	30	
Macro F1:	0.51478	(3)
Micro F1:	0.55740	(8)
Macro BEP:	0.50544	(2)
Micro BEP:	0.55127	(7)

11.4.8 Indexing - TF-IDF

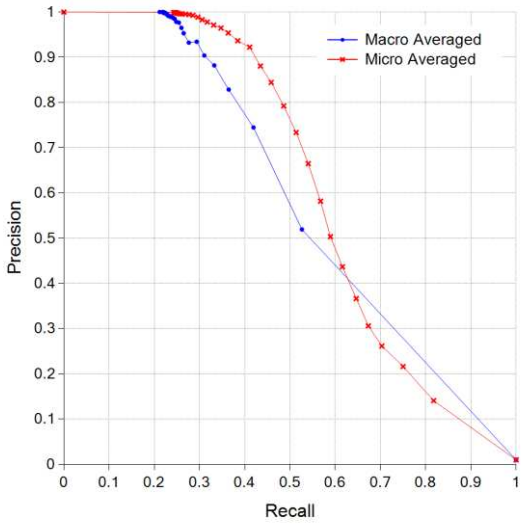


K-Value:	30	
Macro F1:	0.54994	(1)
Micro F1:	0.23139	(2)
Macro BEP:	0.55767	(1)
Micro BEP:	0.17976	(9)

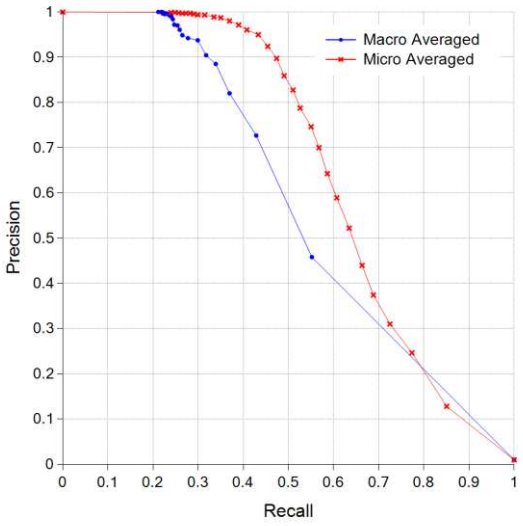
11.4.9 Dimensional Reduction - Document Frequency



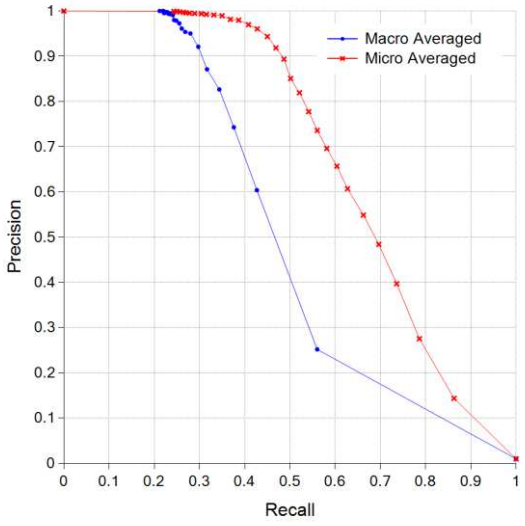
Lower Limit:	2	
Upper Limit:	1000	
K-Value:	30	
Macro F1:	0.55821	(1)
Micro F1:	0.44172	(6)
Macro BEP:	0.56731	(1)
Micro BEP:	0.43767	(8)



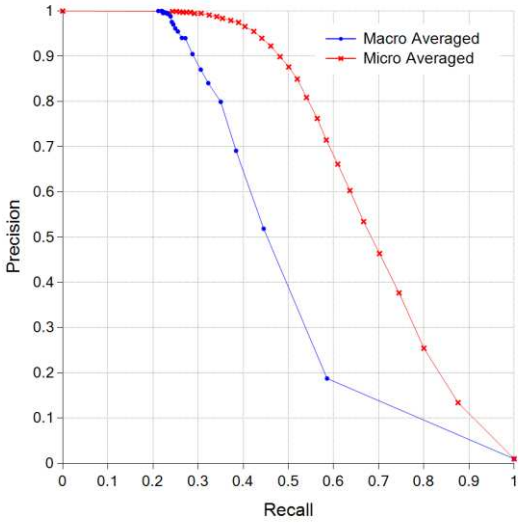
Lower Limit: 2
Upper Limit: N/A
K-Value: 30
Macro F1: 0.55821 (1)
Micro F1: 0.44172 (6)
Macro BEP: 0.56731 (1)
Micro BEP: 0.43767 (8)



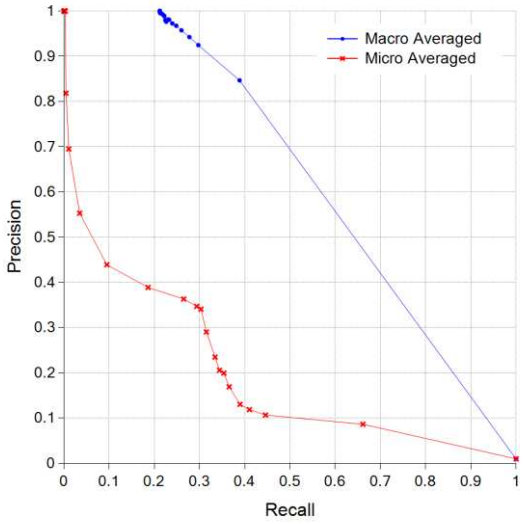
Lower Limit: 10
Upper Limit: N/A
K-Value: 30
Macro F1: 0.53931 (2)
Micro F1: 0.63349 (10)
Macro BEP: 0.50499 (1)
Micro BEP: 0.59811 (7)



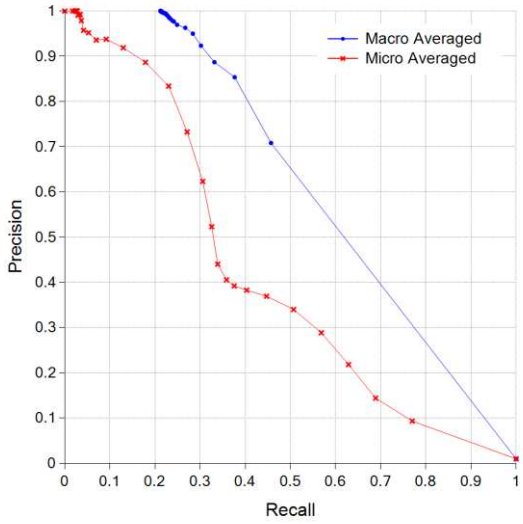
Lower Limit: 50
Upper Limit: N/A
K-Value: 30
Macro F1: 0.50014 (2)
Micro F1: 0.63835 (10)
Macro BEP: 0.51533 (2)
Micro BEP: 0.61715 (6)



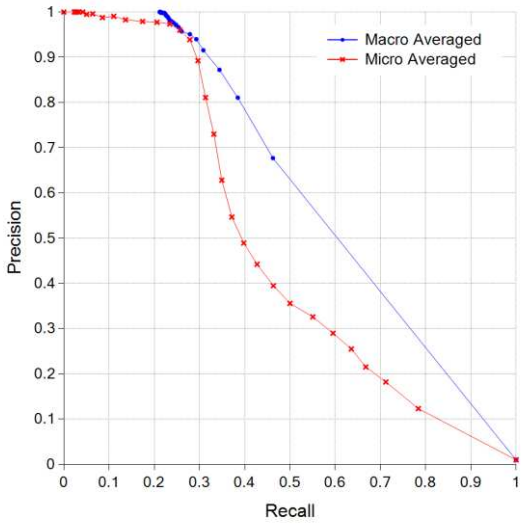
Lower Limit: 100
Upper Limit: N/A
K-Value: 30
Macro F1: 0.49363 (3)
Micro F1: 0.64826 (9)
Macro BEP: 0.48184 (2)
Micro BEP: 0.61959 (6)



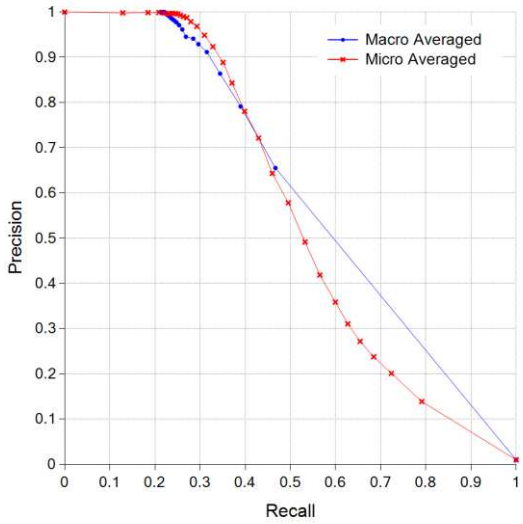
Lower Limit: N/A
Upper Limit: 50
K-Value: 30
Macro F1: 0.53261 (1)
Micro F1: 0.32081 (10)
Macro BEP: 0.61756 (1)
Micro BEP: 0.30271 (9)



Lower Limit: N/A
Upper Limit: 500
K-Value: 30
Macro F1: 0.55555 (1)
Micro F1: 0.41031 (12)
Macro BEP: 0.58259 (1)
Micro BEP: 0.38371 (8)

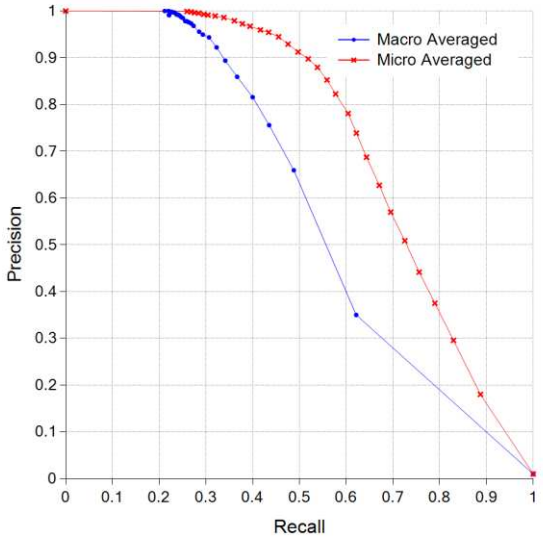


Lower Limit: N/A
Upper Limit: 1000
K-Value: 30
Macro F1: 0.54940 (1)
Micro F1: 0.45615 (13)
Macro BEP: 0.56960 (1)
Micro BEP: 0.43471 (9)



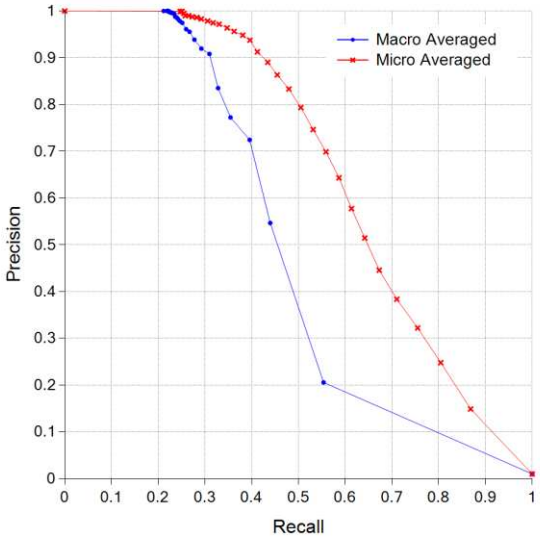
Lower Limit: N/A
Upper Limit: 2500
K-Value: 30
Macro F1: 0.54514 (1)
Micro F1: 0.53855 (11)
Macro BEP: 0.56092 (1)
Micro BEP: 0.51196 (8)

11.4.10 Supplemental Results



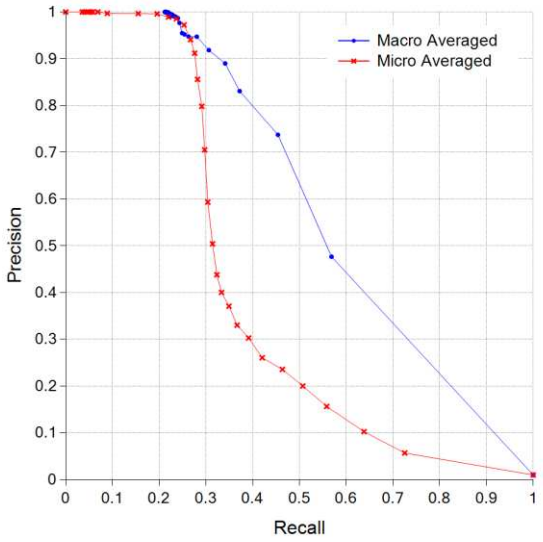
Pre-Processing - Formatting and Stemming Combination

K-Value:	30	
Macro F1:	0.56081	(2)
Micro F1:	0.68121	(10)
Macro BEP:	0.57361	(2)
Micro BEP:	0.66527	(8)



Indexing - Raw Term Frequency

K-Value:	30	
Macro F1:	0.51172	(3)
Micro F1:	0.62100	(9)
Macro BEP:	0.49306	(2)
Micro BEP:	0.59545	(7)



Indexing - Normalised TF-IDF

K-Value:	30	
Macro F1:	0.56196	(2)
Micro F1:	0.42667	(15)
Macro BEP:	0.52256	(1)
Micro BEP:	0.36003	(9)