Improved Rule-based Document Representation and Classification using Genetic Programming

Yasaman Soltan-Zadeh

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

October 2011

Royal Holloway, University of London
Egham, Surrey TW20 0EX, England

To my parents
Who always put their children first

Declaration

The work presented in this thesis is the result of original research carried out by myself, whilst enrolled in the School of Management as a candidate for the degree of Doctor of Philosophy. Where other published and unpublished source materials have been used, those have been acknowledged and referenced. This work has not been submitted for any other degree or award in any other university or educational establishment.

Yasaman Soltan-Zadeh

October 2011

Abstract

In the field of information retrieval and in particular classification, the mathematical and statistical rules and classifiers are not human readable. Non-human readable rules and classifiers act as a barrier in utilizing "expert knowledge" to improve results.

Such barriers can be overcome using genetic programming. The aim of this thesis is to produce classifiers and in particular document representatives which are human readable using genetic programming. Human readability makes these representatives more interactive and adaptable by providing the possibility of integrating expert knowledge.

Genetic programming as a non-deterministic method with high flexibility is among the best options to produce human readable document representatives. To test the results of the chosen method, standard test collections are used. These standard test collections guarantee that the experiments are replicable and the results are reproducible by other researchers.

This thesis demonstrates the process of producing human readable document representatives with transparency for further modification and analysis by expert knowledge, while retaining the performance.

To obtain these findings, this thesis has contributed to the field by developing a system that introduces a novel tree structure to improve the feature selection process, and a novel fitness function to improve the quality of representative generator.

To produce a human readable representative the tree structure is changed into a new shape with more control on the number of children. This reduces the depth of each tree for certain number of features and results in a flatter structure. A fitness function is constructed by combination of classification accuracy on training and validation sets and a parsimony component. This study found that the order of matched document with representatives can improve overall performance. Different feature selections are investigated and integrated into our genetic programming based feature selection method which is based on a probability distribution derived from the feature weights.

Acknowledgements

I would like to thank my advisory team, Dr. Masoud Saeedi and Dr. Ashok Jashapara, for their advice and their confidence in my work. I would also like to thank Professor Duska Rosenberg for her encouragement and moral and practical support when I needed it.

Many thanks to the examination team, Professor Fionn Murtagh and Dr. Richard Hill, for their valuable and constructive comments.

My special thanks to my friend and brain storming team Sirvan Yahyaei.

Finally the love and support from my parents have been continuous, unflinching and unlimited. I cannot thank them enough.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

1.1 Background

During recent decades, the fast growing amount of digitally available information has resulted in an information explosion and brought new challenges to organisations and businesses. This has happened due to advancement of technology in hardware, while new efficient and effective computational models are being developed at a much slower pace. While the amount of available digital information is growing, they lack sufficient means to search and retrieve it. The creation of the World Wide Web (WWW) and the appearance of search engines have provided a comprehensive understanding of digital search concept for entities and the essentialness of comparable tools for their digitised information. For example, currently *CiteSeerX* searches over 1.5 million documents and 29 million citations on the web[1].

This excessive availability of digital information passes the user's cognitive capacity while only a very small fraction of the available information is relevant and matches the user's information need. One of the most effective and widely used methods to deal with the information overload is the use of Information Retrieval (IR) techniques. Information retrieval is a discipline concerned with organisation, analysis, extraction and storage of the information for the future use [van Rijsbergen, 1979].

Information retrieval includes tasks such as search, classification, clustering, filtering,

[1]`http://citeseerx.ist.psu.edu`

novelty detection and routing to tackle the information overload problem and identifies relevant results for users' information needs. Each of these tasks are implemented differently and require different algorithms to perform their jobs, however, they share principles and techniques.

A technique employed in most of these tasks is generating a string (of terms and logical functions) to represent a set of documents that are related to each other context-wise and convey the same content. This technique is known as *generating a document representative*. For example, a news story and its follow-ups can be considered as a set of related documents. To determine if a new document belongs to the set, a representative is built for the document set and the new document is compared and evaluated against the representative instead of being compared with each individual document. The same technique can be applied to a classification problem which requires an algorithm that labels a new document to one of the pre-defined classes. A representative is generated based on the labelled documents (examples) for each class and new documents are tested against the representative.

Using a string as the representative for a set of documents to solve the classification problem, provides the opportunity to re-formulate most of the other problems of text management to be similar to classification. Document retrieval can be seen as categorising documents under two classes of relevant and non-relevant [Robertson and Jones, 1976]. Filtering can be defined as classification under allowed and not-allowed (filtered) categories [Robertson, 2002a] and so on. Once we have a flexible classification system, it can be tailored to perform many other tasks in text management.

Machine Learning (ML) is one of the most popular and successful approaches to implement IR tasks. A machine learning system refers to any system that learns from experience, analytical observation and other means. Machine learning has two main approaches for learning: supervised and unsupervised learning [Sebastiani, 2005]. Supervised learning which is the base of our approach includes: classification, filtering, novelty detection and routing. In the case of classification, a set of manually labelled examples are provided for the system to learn a model or a collection of rules and use it to perform the task on future documents.

There are different approaches of machine learning to solve the classification problems, such as statistical, rule-based and evolutionary based [Sebastiani, 2002]. In statistical methods, different features are selected and tuned on the tuning set to classify the documents. Features may vary relating to the purpose of the system. Each feature has a weight based on its importance and the job of the statistical system is to find the best combination of weights for the features, based on the examples provided as training and tuning data.

In rule-based methods, a hierarchical tree structure classifies documents using rules that apply each class attributes to a document's features. The tree starts with user information need or its representative query, and applies each of the class rules in order to narrow down the number of documents in each class.

Evolutionary based computation is a broad term to address all the computation methods that employ an iterative process often simulates biological mechanisms of evolution. In these methods there is a population[2] and solutions grow or develop in this population. Evolutionary based algorithms simulate the process of biological evolution by relying on random mutations and selecting the fittest to breed more and transfer its characteristics to the next generation. When all the progress is done, the desired result, which is the fittest individual in the final generation, is chosen.

In this thesis, we propose a method based on evolutionary computation to perform representative generation and classification.

1.2 Motivation

Evolutionary based computational techniques are inspired by natural biological systems. They are stochastic methods which makes them very flexible to solve complex problems and create candidate solutions. This flexibility gives them the power to solve problems in the manner which is not available to other commonly used classification techniques like, statistical classification, decision trees and neural networks. In general, there is a random element in evolutionary algorithms that makes them different

[2]There are methods with more than one population, but the basic concepts are the same.

from other deterministic approaches.This guided random nature, gives the Evolutionary Algorithm (EA) the chance to overcome problems that a deterministic algorithm can not.

Genetic Programming (GP) [Koza, 1992] is an evolutionary learning technique, specifically useful for classification. GP's inherited flexibility as a heuristic technique makes it possible to use complex pattern presentations such as trees. This presentation includes any kind of operations or functions and accepts any domain knowledge in the learning process.

GP makes few assumptions about how good solutions should be characterised, which makes it specifically suitable to search large complex solution spaces. GP produces solutions as a symbolic representation which can be analysed further. This feature is very useful when a general solution is needed at the beginning, before moving to more focused and specific issues. GP has a random nature which makes it a better approach to deal with complex solutions. However, as a non-deterministic method it is necessary to run the GP multiple times to result in useful solutions.

GP automatically generates computer programs by simulating biological evolution although it is different from other evolutionary algorithms in its representation of candidate solutions. For example, in genetic algorithms, a solution is usually a binary vector or a vector of numbers. In genetic programming the solutions are computer programs that can be run, which enables genetic programming to deal with problems that require complex solutions.

Providing solutions in the form of computer programs or tree-shaped structures is an ideal way to generate the representatives for a set of documents. A representative is a program that selects the documents which belong to the set and deselects those which do not.

There are many applications of GP in IR [Fan et al., 2004b; Trotman, 2005; Cummins and O'Riordan, 2007; Kro andmer et al., 2010] and Classification [Lin et al., 2007; Faraoun, 2006; Segond et al., 2009; Espejo et al., 2010], however most of the work has been done on learning ranking functions, query expansion or similarity methods of Classifiers.

The representatives that we generate are logical sentences that can be used as classifiers. They are combined sets of features extracted from the training samples and logically connected to each other to select or deselect future documents. This thesis work is close to the work of [Hirsch et al., 2007] more than any other. However, this thesis approach is different from this work in many aspects. This thesis focus is not only on the quality of the classification system, but also improves the readability of the learnt representatives to be as important. Therefore, we have chosen a different structure for our representatives. Apart from the tree structure, our feature selection and fitness evaluation are also significantly different from the mentioned work above.

1.3 Experiments

In this research, we use experiments to test the theory and explore the reality against the predictions. Experiments can show if there are bugs in a theory, however it can not guarantee their absence. While an experiment's inductive approach can extract the area by observation, there are some areas in reality where theory and deductive analysis cannot reach. The best example for this process is artificial neural networks. Based on theoretical ground they were discarded, however researchers later developed better theories based on the properties that experiments had demonstrated [Tichy, 1998].

It is very important for an experiment to be repeatable. Repeatability makes it possible to check the results independently so thereby raising the confidence in the results, identifying and eliminating errors and frauds. Experiments provide a reliable foundation of knowledge to decrease the ambiguity of theories, methods and tools. Experiments may derive unexpected, new understanding of the area which indicates new investigations in the field. As unsuccessful approaches and incorrect assumptions are identified and eliminated quickly by experiments, the progress accelerates [Sjøberg et al., 2005].

Experiments can be extremely expensive in terms of both money and time, however there are meaningful approaches to fit different budgets. *Benchmarking* provides an effective and affordable way to experiment. Benchmarking takes a sample of a task domain, human and computer execute this sample while well-defined measures are used to evaluate the performance. Benchmarking is a successful approach that provides a level field to compare ideas, while adequately chosen benchmarks are sufficiently representatives, repeatable and objective comparisons are allowed there for unpromising approaches and exaggerated claims are identified and eliminated [Sim et al., 2003].

1.4 Research Questions and Hypotheses

This thesis addresses the following research questions:

1. How can we automatically improve the representative generating for a document set?
2. Can GP improve the readability of a document set representative?
3. Which feature selection method results in the best performance of classification based on GP learnt rules?
4. How to increase the readability of the generated representatives by incorporating a readability component in the fitness function?
5. Does increasing the readability damage/degrade the quality of classification or representative generation?

Based on these research questions, the thesis hypotheses are:

1. ***Hypothesis 1:*** *The GP-based representative generation achieves classification results comparable to, or even better than state-of-the-art classification methods*
2. ***Hypothesis 2:*** *The readability of generated representatives can be improved by defining more complex functions for the simpler GP tree structure*

1.5 Aims and Objectives

In this thesis, we aim to address the following points :

1. We design and develop a machine learning system based on GP to learn a representative for a set of documents. This system is used for classification and performs other information retrieval tasks.

2. One of the most important advantages of GP is human readability of the generated representatives, however this readability can be reduced and degraded by complex combinations and numbers of duplicated terms. This thesis improves the readability by defining a new tree structure and adding a component to the fitness function.

3. In this work we use *Ephemeral Random Constants*[3] to improve the feature selection and simplify the tree structure by this sophisticated method.

4. This work investigates different feature selection methods and reports the results.

5. We design a new fitness method to consider the quality and readability at the same time. By introducing a new component we hope that the performance of the algorithm on the unseen data will improve.

6. We outline how to use a representative generation algorithm to perform classification, filtering and query expansion.

1.6 Thesis Outline

This thesis is structured into 7 chapters (including the present chapter). An outline of the contents of the remaining chapters is as follows:

Chapter 2, introduces the fundamental concepts of information retrieval and text processing with a review of the previous work in the field.

Chapter 3 , discusses biologically inspired models and their applications in information retrieval and text management.

Chapter 4, explores the related work and state of art in the field.

[3]see Chapter 5

Chapter 5, presents this thesis classification system based on genetic programming and reports an initial empirical results.

We improve all the aspects of our system in **Chapter** 6 and present results of various experiments.

Finally, **Chapter** 7 concludes the thesis and points out some areas for future work.

CHAPTER 2

Background: Basic Concepts

2.1 Introduction

In this chapter we are going to introduce the importance and need of information retrieval. We will look at classification, clustering and document retrieval as three main parts of information retrieval. The different methods (such as supervised and unsupervised learning) and applications (such as web-based search or enterprise search) will be surveyed and we will look in detail at the different types, approaches and applications of information retrieval.

2.2 Information Retrieval

The meaning of information retrieval can be defined as finding unstructured material (usually text documents) from large collections (usually stored on computers) to satisfy a user information need [Manning et al., 2008].

As the above definition shows there are three major parts in information retrieval. The first part is information need, the user's request for information which is expressed by a query, although the queries are not the same as information need, they are used to represent the user's information need in a format which can be understood by information retrieval systems and retrieval mechanisms. The second part is a collection of documents which is expected to match the query against. The third part is the matching process, which shows the result of retrieval process in a list of relevant documents to the query.

We refer to the unit that the information retrieval system has been built over as *document*. Users are looking for documents in the retrieved list which contains a list of documents. A document can be a web-page, a PDF file, an email message or a simple plain text file. Also, a group of documents that we are dealing with is referred to *collection*[1]. Thus, we will have a collection of email messages or web-pages that our information retrieval system is working on.

In this chapter, we claim that there are three main parts in information retrieval:

- Information organisation : classification and clustering
- Information search and retrieval : matching process and document retrieval
- Information management : information processing

A system capable of performing techniques of these three topics is able to effectively organise and manage text documents.

Clustering

Clustering is the task of grouping a collection of documents into sub-collections which are called *clusters*. A clustering algorithm aims to make clusters that contain similar documents, but different from other documents in other clusters. Clustering is an unsupervised procedure (see Section 3.2.1 for types of learning methods), which means the documents are categorised based on a set of unlabelled samples. Although, it may seem unpromising to address such a problem, there are good reasons that make clustering very useful and worth investigating [Basu et al., 2008].

[1]Sometimes it is called corpus.

Classification

In *Classification*, in contrast to clustering, there are predefined classes[2] and the classification job is to label future documents to one or more of these predefined classes. Since the classes are manually defined, classification is a supervised procedure (see Section 3.2.1 for types of learning methods). Classification is very useful for many search-related tasks such as *Spam detection*, *Sentiment classification*, *Online advertising* [Croft et al., 2009].

Matching process and document retrieval

Matching process and document retrieval is finding documents that satisfy a user's information need by comparing the document representatives against the query which represents the user's information need. The result of matching process is a ranked list of documents of which web search engines are good examples. They work by crawling and storing many web-pages and retrieve a list of relevant documents (web-pages in this case) upon the user's request based on the query provided by the user. Another application is *enterprise search* which deals with documents and contents across an enterprise. An enterprise search system indexes and retrieves the contents within the organisation to the authorised users [Göker and Davies, 2009].

Web-based search is finding relevant web-based documents which match the user information need. Web-based search has different requirements and characteristics compared to traditional search because of distinctive web document characteristics as well as a user's behaviours and requirements. Web-based documents are known as *pages*. Pages are addressed by a *uniform resource locator (URL)* and are grouped as *sites*. Pages are exceptionally versatile, a page can provide simple information such as introduction to a site or can play an interactive role by accepting information from a user as an online form or simply can contain a news article.

Web-pages refer to each other by using hyper-links. This is a reference from source page to the destination page URL. Web-pages are built by *HyperText Markup Language*

[2]In this thesis, terms 'class' and 'category' are used interchangeably

(HTML) which make it possible to provide a uniform and standard view for all different platforms and web browsing technologies. Though users may see a specific sort of information on browsers, HTML makes it possible to add extra information, namely *meta-data* for web search engines.

User's behaviours vary from finding the latest financial news, using interactive pages, shopping online, searching a library archive, joining a social network, to simply checking their emails [Craswell and Hawking, 2004; Davison et al., 2010].

Enterprise Search is a relatively new concept, its applications are used where the search is taking place on an organisation's data to complete a task. The data is a collection of diverse types, such as emails, employees forums, Intranet web sites and published reports. The ultimate aim is to have a search system that is able to include and deal with all different data types. Enterprise search is interesting as it works on two main issues :

- new data: organisations have different type of documents, such as, web pages, emails, archives, each type has its specific characteristics.
- new tasks: enterprise users may search for a single data type such as a specific item in their emails or may request an adhoc search on a mixture of data types such as search for a specific topic across all available documents including emails, archives.

[Voorhees, 2005; Halvorsen et al., 2008].

Enterprise search can improve the result of search for organisations by shorter search sessions which means more time to spend on other searches or activities. Also it reduces the number of failed search sessions which results in less unnecessary activities inside the organisations. However there are two main potential problems. The first one is based on the fact that each organisation (Enterprise) is different from the other one, therefore the result of one single enterprise may not reveal anything about enterprise search in general. The second one is related to confidentiality. The real world enterprise search needs to have access to confidential information. Though this information may be available inside the company, enterprise information may have been accessible to a subset of employees. Even if some documents are publicly available, there are always users who resist to use documents which were supposed to be private [Bailey et al., 2007].

The rest of this chapter is structured as follows: first we discuss the information retrieval techniques common to all of the above tasks. Then we discuss three main problems in more detail and introduce some of their models . Finally, an introduction to evaluation of information retrieval is given.

2.3 Information Retrieval Processes/Techniques

Documents in their original form are not suitable for retrieval processes. Retrieval systems need to receive an accurate pattern to represent the original information content. This pattern, which is the internal representation of the document is the result of the *indexing* process. Indexing process presents each document as a set of *indexing features*.The indexing process extracts words, known as *terms*, with a specific value related to the purpose of indexing from the original documents.

To identify these indexing features some pre-processing is required. These processes are specific forms of lexical processing which focuses on the first two main parts of an information retrieval system.

1. Reduce the complexity of document's structure and remove unnecessary parts. These processes are applied to all the documents in the collection to simplify representing the documents.

2. Optimise the configuration of user information need for a better query. All the processes that have been applied to the documents should be applied to the query as well, so the comparison and similarity measures can be computed. In addition, there are techniques to improve the query to better represent the user's information need.

[Herrero et al., 2010; Zhang, 2005].

2.3.1 Document Presentation

Document pre-processing changes the documents from their original form into an internal representation. These pre-processes are mainly lexical-processes. Some of routine lexical processing can be named as follows.

Tokenisation

It is the process of identifying the terms (tokens) in the document as the meaningful units of the text. Tokenisation is a simple algorithm that looks for white-spaces and punctuations, works efficiently in many languages. However, for some languages such as Chinese, identifying the boundary of the words is not trivial.

Case Folding/Normalisation

Case folding changes the document's terms into a unified format, either all lower-case or upper-case. Case folding increases the number of terms that match the query terms and improve the search results in most instances, however it can reduce the precision[3] as some terms are distinguished in their capital letters. For example, Turkey the country and turkey the bird [Jurafsky and Martin, 2008].

Stop-words Removal

In traditional language structure, there are certain high frequency function words known as stop-words. Typical examples are articles such as *the* and prepositions such as *in*, *at* and *on*. This operation reduces the text volume up to 50 percent without losing any significant information [van Rijsbergen, 1979]. However there are disadvantages to this process. Phrases that contain words from stop lists may be eliminated from search results and decrease the recall[4] [Jurafsky and Martin, 2008], as an example the phrase *To be or not to be* may be lost during the stop-word removal process. With more powerful computers and compression techniques modern information retrieval

[3]For definition of precision see Section 2.7.1
[4]For definition of recall see Section 2.7.1

systems use a very short stop-word list. In other words, they prefer to deal with more terms in the collection rather that hurting the performance of the system.

Non-alphabetical Character Removal

Apart from stop-words, there are some other concerns such as special symbols (!, ? and %) or sometimes numbers. It can be helpful to remove all these non-alphabetical characters. This increases the recall, however some lost punctuations may change the meaning of a word or phrase so the precision will be decreased [Iliopoulos et al., 2001].

Morphological Normalisation

Traditional document structure uses different forms of a root word, *manufacturer* and *manufacturing* are both generated from the same entity *manufacture*. There are many lexical processing methods to normalise the morphology of words in the documents. A very accurate, though difficult, algorithm is *lemmatisation* that determines the *lemma* of a word. An English lemmatiser can determine that word *worst* has *bad* as its lemma, or *say* is the lemma for *said*. However, it requires context understanding and extra algorithms such as part-of-speech tagging to perform correctly.

A widely used automatic approach in information retrieval is called stemming which maps two different children to the same root. Stemming is less accurate than lemmatisation, but it is faster and does not require extra tools. Two of the main methods for stemming are:

1. Rule-based: there are hand-crafted rules for each natural language to reduce the words to their roots [Nahm and Mooney, 2001].

2. Automatic suffix algorithms: the process that removes the frequent and ordinary morphological and inflexional word endings that exist in the language grammar [Porter, 1997].

The stemming algorithm developed by [Porter, 1980, 1997] is widely used by information retrieval researchers [Robertson et al., 1980]. An extensive general survey of stemming algorithms is provided in [Frakes and Baeza-Yates, 1992].

Weighting

Extracting the terms from the document includes estimating the importance of each term in order to find particular documents. The indexing feature must have a discriminant power to differ between the document it has been extracted from and the rest of documents in the collection. These discrimination powers are measured by a numeric values called weights. There are different types of weighting methods. Here, we briefly introduce the classic ones:

- [Luhn, 1958] used *term frequency* to measure the effectiveness of each term. Term frequency is estimated by counting the occurrences of a term in a document[5]. He concludes that terms with medium frequency are the most discriminant. Terms with high frequency have very little information and terms with low frequency are unlikely to appear in a query, so they are discarded.

- [Sparck-Jones, 1971] used *inverse document frequency* to emphasise the term importance within the entire document collection. The intuition behind *idf* is that the less documents a term occurs in, the more significant is in discriminating the documents. For a document collection with size of N when term i occurs in n_i documents then the *idf* weight is calculated for that term as $log\frac{N}{n_i}$.

- Another weighting scheme is known as *tf-idf* weight, which is a combination of the above methods. It has been used by [Salton, 1971]. The *tf* is used in this weighting scheme is normalised by the length of the document [Salton and Buckley, 1988] to prevent any misleading effects of long documents. *tf-idf* calculates the term weight as follows:

$$w_{ij} = \frac{log(freq_{ij} + 1)}{log(length_j)} log(\frac{N}{n_i}) \qquad (2.3.1)$$

where, w_{ij} is the $tf - idf$ weight for term i in document j,

[5]Usually there are normalisation factors in estimating the term frequency.

$freq_{ij}$ is the frequency of term i in document j,

$length_j$ is the length (in words) of document j,

N is the number of documents in the collection and

n_i is the number of documents that term i is assigned to.

For more information and a discussion on weighting schemes refer to [Salton and Buckley, 1988]. An extensive survey which also compares the effects.

When these processes are done, in order to store all the information about extracted terms, a specific data structure is used. One which is typically used is *inverted file* structure [Frakes and Baeza-Yates, 1992]. In retrieval systems, this means to identify immediately all the documents in the collection that contains a specific given keyword.

2.3.2 Query Representation

Query pre-processing focus is mainly on improving the user information need representation. A user information need is represented by relevant keywords which are understood by retrieval mechanisms and are chosen by user. These keywords or any other types of information need expression are called *query* [Spoerri, 1995; Zhang, 2005; Meadow et al., 2007].

One of the fundamental types to formulate the user information need into a query is *Boolean Query*. In this formulation, Boolean operators such as *AND*, *OR*, and *NOT* join the terms to generate the query. An example query may look like: *(Information AND Retrieval) NOT Evaluation*. This method is the easiest way to formulate a query for an experienced user. However [Sparck-Jones and Willett, 1997] has claimed, it is difficult and somehow impractical for non-experienced users to formulate effective queries.

When a query is generated, lexical processing and term-weighting such those mentioned for the documents, are required (see Section 2.3.1). A generated query may be expanded before or after the retrieval to better represent the user's information need. To expand the query before retrieval, a thesaurus can offer semantically related terms to those already in the query [Voorhees, 1994]. After retrieval, relevant retrieved documents may add extra terms that the original query does not include [Magennis and van

Rijsbergen, 1997]. The process of *query expansion* can be either automatic, which takes place by the system, or controlled, which is done by user through user interaction.

Relevance feedback is one of the techniques for query expansion. The relevance feedback process is a cycle. The system provides a set of retrieved documents, user identifies the relevant documents, the system uses the relevant retrieved documents to extract new terms from relevant documents and modify the query. The point is to increase the possibility of retrieving more documents similar to documents identified as relevant and less similar the rest of the collection. The modified query is then used to retrieve a new set of documents [Ruthven and Lalmas, 2003]. Studies have examined the information retrieval systems that use relevance feedback methods. Most of these studies have shown positive results [Harman, 1992b]. Meanwhile, the lack of interest from the user to engage into these sorts of processes, query expansion and relevance feedback, is an ordinary reported problem [Ruthven et al., 2001].

2.3.3 Matching Documents and Queries

So far, we have described two common processes of information retrieval document and query representation. The next step is estimating the similarity of a document and a query or more generally estimating the similarity of two documents. Consider a document retrieval system, which aims to retrieve a list of relevant documents against a given query. The system should identify relevant documents which are the most similar to the given query.

Boolean Query

One way to deal with this is to formulate the query as a Boolean query and return the documents that fulfill the logical requirements of the query in an unranked list. However, this way of matching does not show the degree of relevancy.

The Best Match Searching

Another way to find relevant documents is *the best match searching*. In these methods, query is compared to documents in the collection and the similarity between them is measured. There are many methods to estimate the similarity, which will be discussed later. Based on the similarity scores, a list of relevant documents is created. This list is sorted in decreasing order and presented to the user, the final answers are chosen from the top of the retrieved list.

Vector Space Model

There are different retrieval models (see Section 2.4) that use variety of similarity methods to calculate the relevancy of a document against a query. For example, in *Vector Space Model* [Salton and McGill, 1986] vectors of terms are used to represent documents and queries. In a simplified example each term shows one dimension and each document coordinates with each dimension based on that term's weight in the mentioned document. The queries are represented in the same way. The angles between the vector which represents each document and the vector that represents the given query, measures the similarity between that document and the given query.

Coordination Level

Another way of measuring the similarity between a document and a query is calculating how many words they have in common. This measure known as *coordination level* matching function [Losee, 1987]. Consider vectors of length n representing the document D, and the query Q, the similarity is calculated as:

$$Sim(d, q) = \sum_{i=1}^{n} d_i q_i \tag{2.3.2}$$

where, d_i is the ith term in the document vector and q_i is the ith term in the query vector. To avoid the negative effect of long documents with high occurrences of terms, we need to normalise the estimate. To normalise the similarity by the length of documents and queries, Dice coefficient can be used:

$$Sim(d,q) = \frac{2 \times \sum_{i=1}^{n} d_i \times q_i}{\sum_{i=1}^{n} d_i \times \sum_{i=1}^{n} q_i} \tag{2.3.3}$$

To this point, we have discussed techniques that are common in all information retrieval problems. In the next three sections, different models used in document retrieval, clustering and classification will briefly be discussed.

2.4 Document Retrieval

Document retrieval systems try to recognise relevant and non-relevant documents with respect to a given query which represents the user's information need. Apart from the binary decision of relevant and non-relevant, sometimes it is required to measure the level of relevancy. *Retrieval Models* are the main part of information retrieval process. These retrieval models are used to predict the relevancy of a document and most of the times calculate a numeric value as a score of each document to compare the relevancy of different documents. There are different approaches among these models and a few of the main approaches are introduced here.

An information retrieval model is a quadruple (d, q, F, R) [Baeza-Yates and Ribeiro-Neto, 1999], where:

1. $\mathcal{D}$ is a set composed of logical views (or presentations) for the documents in the collection (see Section 2.3.1).

2. $\mathcal{Q}$ is a set composed of logical views (or presentations) for the user information needs. Such representations are called queries (see Section 2.3.2).

3. F is a framework for modelling document presentations, queries, and their relationships.

4. $R(q, d)$is a ranking function which associates a real number with a query $Q \in \mathcal{Q}$

and a document representation $D \in \mathcal{D}$. Such ranking defines an ordering among the documents regarding to the query Q.

There are two main steps concerning each model [Meadow et al., 2007; Zhang, 2005]:

- How to represent the documents and user's information needs.

  We discussed the problem of representing the user's information need and the possibility of representing it with queries in Section 2.3.2 and document presentation in Section 2.3.1. Also, each retrieval model represents the documents and the queries differently.

- The framework that builds a ranking process.

  As mentioned in Section 2.3.3 about the similarity methods, there are many ways to model and compute the relevancy of a document and a query.

Building the models and representing the user's information need and the documents also have some drawbacks. For example, using a set of keywords and indices as described before, will affect the semantic of a document and the information need. The situation will become worse when users do not have any knowledge on how to transfer their information needs into the query string [Bai, 2007].

In this section, we will introduce a few classical models of document retrieval that have extensively been used and improved over the years. The basic retrieval models, that only consider the existence of a term in a document are the so-called *set models*. In such models documents and queries are considered as a *SET* of their terms. Terms in a query set will be compared with terms in a document set. As joint terms in a query set and a document set increase, the document will become more relevant to the query. Therefore, where Q is the set of query terms and D is the set of document terms. Ranking in these models are based on the cardinality of $D \cap Q$, which is the number of common terms between the document and the query [Baeza-Yates and Ribeiro-Neto, 1999; Grossman and Frieder, 2004]:

$$RSV(d,q) = |d \cap q| \tag{2.4.1}$$

Where, RSV is the *retrieval status value* which is a numeric value associated by the retrieval model to the query and document pair, in order to compare different documents. There are also methods to normalise the Equation 2.4.1 to compensate for the effect of the document and the query length:

$$\text{Cosine:} \quad RSV(d,q) = \frac{|d \cup q|}{\sqrt{|d| \cdot |q|}} \tag{2.4.2}$$

$$\text{Jaccard:} \quad RSV(d,q) = \frac{|d \cup q|}{|d \cap q|} \tag{2.4.3}$$

$$\text{Dice:} \quad RSV(d,q) = \frac{2 \cdot |d \cup q|}{|d| + |q|} \tag{2.4.4}$$

The set-based models have advantages, because of their simplicity.

These models are simple so their implementation is easy and does not add any complication to the process. In addition, they are very clear for individual subsets of a collection. Furthermore, the whole characteristics of Boolean Algebra are applicable.

On the other hand, there are also disadvantages in using them: In these models partial matching is not possible. There are some difficulties in understanding complex queries. Not only they do not support *null* output but also the information overload causes difficulties that are not handled properly. There is no meaningful style(such as ranking) to order the output. Users search and their individual search techniques are not taken into consideration [Willie and Bruza, 1995] and different systems may retrieve different documents for the same query [Borgman et al., 1996].

An improvement over the set-based models is a model that represents the documents and the query as vectors. *Vector Space Model* considers documents and query vectors and ranks the documents based on the closeness of their vectors to the query's vector [Salton, 1971; Salton et al., 1975]. Each term in the query or document is an axle and has a weight. There are many methods to calculate these weights, some approaches use 1 if a term occurs and 0 if it does not. Most methods employ a form of $tf \times idf$ weighting, that is discussed in Section 2.3.1. There have been many studies in finding the best weighting schemes on different collections. For different weighting methods and a comparison of their effectiveness and performance see [Salton and Buckley, 1988] and [Zobel and Moffat, 1998].

Consider an ordered set of all the terms in the collection as $C = \{t_1, t_2, \ldots, t_n\}$, each document D is represented by a vector with n dimensions: $\vec{d} =< d_1, d_2, \ldots, d_n >$ and each query Q as $\vec{q} =< q_1, q_2, \ldots, q_n >$. In these vectors, d_i is the weight of term i in document $\vec{d}$ based on a weighting scheme such as $tf \times idf$ and q_i is the weight of the term i in the query such that if $t_i \in Q$ then q_i is 1, otherwise it is 0. Now, we have the document and query representations as vectors, we can compute their similarity based on the closeness of the vectors. Figure 2.1 shows a special case of a document and a query with only two terms. The angle α between the vectors can measure the similarity of the document and the query.

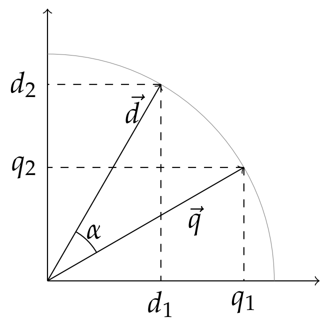


Figure 2.1: Vector space model - Graphical interpretation

Thus, the retrieval status value for vector space model is defined as:

$$
\begin{aligned}
RSV(d,q) &= \frac{\sum_{i=1,n} d_i q_i}{(\sum_{i=1,n} d_i^2)^{1/2} (\sum_{i=1,n} q_i^2)^{1/2}} = RSV(\vec{d},\vec{q}) && (2.4.5)\\
&= \frac{\vec{d} \cdot \vec{q}}{\sqrt{\vec{d}^2} \cdot \sqrt{\vec{q}^2}} && (2.4.6)\\
&= \cos\alpha && (2.4.7)
\end{aligned}
$$

Vector space model is simple and easy to implement, unlike set-based models it uses term weighting to improve the quality of matching, which also makes partial matching possible. In addition, the cosine formula in Equation 2.4.5, shows the similarity of the document and the query which makes ranking possible. On the other hand, it does not support logical expressions such as "mouse & NOT cat" and does not understand term dependencies by default[6].

[6]Term dependencies are modelled in *generalised vector space model* [Wong et al., 1985]

As discussed already, a retrieval system tries to determine the relevancy of different documents against a query by considering the representations of the documents and the query. The query represents the user information need. The document representations contain, in fundamental, less information than the actual documents. *Probabilistic models* give a theoretical foundation to the notion of relevance and the retrieval process. Probabilistic models that were developed by [Robertson and Jones, 1976] and [van Rijsbergen, 1979] rely on *probabilistic ranking principle* [van Rijsbergen, 1979]:

> If a reference retrieval system's response to each request, is a ranked list of documents in the collection in order to decrease the probability of relevance to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data.

There are many probabilistic models for information retrieval. *Binary Independence Retrieval Model* (BIRM) [Robertson and Jones, 1976] is a classical model, which estimates the probabilities based on the presence and absence of the terms in relevant and non-relevant information, acquired through relevance feedback by the user. BIRM performs poorly in practice because it does not take advantage of tf and document length, which causes problems for long documents. The $BM25$ model[7] [Jones et al., 2000] considers document normalisation and also query normalisation, and similar to vector space model, adds a term frequency component.

[7]Often called Okapi BM25 weighting

$$RSV(d,q) = \tag{2.4.8}$$

$$= \textstyle\sum_{t \in q} \left(w_t \cdot \frac{(k_1+1)tf(t,d)}{K+tf(t,d)} \cdot \frac{(k_3+1)tf(t,q)}{k_3+tf(t,q)} \right) \tag{2.4.9}$$

$$+ \quad k_2 \cdot |q| \cdot \frac{avgdl-|d|}{avgdl+|d|} \tag{2.4.10}$$

$$\text{where} \quad w_t = \log \frac{N-df(t)+0.5}{t+0.5} \tag{2.4.11}$$

$$\text{and} \quad K = k_1((1-b)+(b \cdot |d|)/avdl) \tag{2.4.12}$$

where w_t is the term weight as above or *idf*. $tf(t,d)$ is term frequency within document d and $tf(t,q)$ is term frequency within the query. k_1, k_2, k_3 and b are tuning parameters and *avgdl* is the average document length in the collection.

In addition to classical models described above, there are many other models built on the classical ones or ideas from other domains. For example, in Boolean retrieval there is no provision for term weighting and no ranking is generated. As a result the size of output might be too large or too small. *Extended Boolean Retrieval* [Salton, 1989; Baeza-Yates and Ribeiro-Neto, 1999] extends the Boolean model with the functionality of partial matching and term weighting. This strategy allows one to combine Boolean query formulations with characteristics of the vector model. Despite more than twenty years of using language models for speech recognition [Kuhn, 1988; Jelinek, 1997] and language translation [Brown et al., 1990], their use for information retrieval started only in 1998 [Ponte and Croft, 1998]. In their language modelling approach to IR, a language model is built for each document, and the likelihood that the document will generate the query is computed.

2.5 Clustering

Clustering is one of the popular problems in IR. Clustering focuses on putting documents into groups known as *clusters* when they are similar to documents in that cluster and dissimilar to those in other clusters [Gordon, 1987].

Figure 2.2 shows a simple example of clustering. It can be seen that there are four

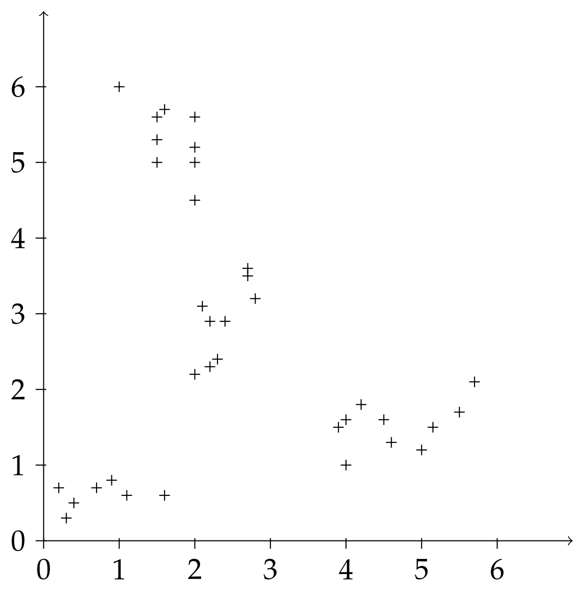


Figure 2.2: A clustering example

clusters of elements in the picture. Clustering algorithms aim to find such clusters without labelled data.

2.5.1 Clustering Types

Clustering is trying to determine the natural similarities in a set of unlabelled documents and group them into clusters. This can be done by employing different algorithms. Some of these algorithms are mentioned as follows [Gulli, 2008].

Flat Clustering

Flat clustering, also known as partitioning or exclusive clustering is when documents can belong to only one cluster. *K-means* [MacQueen, 1967] is one of the perhaps most common flat clustering algorithms. K-means' goal is to minimise the average squared distance of each document in a cluster from the cluster centre. The cluster centre known as *mean* or *centroid* is calculated:

$$\overline{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\overline{\chi} \in \omega} \overline{\chi} \qquad (2.5.1)$$

Where the ω represents the cluster, X represents the documents, with assumption of document-length normalised. The measure *residual sum of squares* or *RSS* is used to evaluate how well the centroids represent each cluster's members:

$$RRS_k = \sum_{\overline{X} \in \omega_k} |\overline{\chi} - \overline{\mu}(\omega_k)|^2 \tag{2.5.2}$$

The squared distance of each vector from its centroid summed over all vectors:

$$RRS_k = \sum_{k=1}^{K} RSS_k \tag{2.5.3}$$

Hierarchical Clustering

Hierarchical clustering uses the combination of the most similar clusters. The algorithm starts by clustering any single fact as a cluster and after few iterations unites nearest clusters and reaches the desired clusters.

Hierarchical clustering [Johnson, 1967], for a given set of N documents, consists of following steps:

1. Assign each document as a cluster. It means there are N clusters to start with.
2. Find the most similar pair of clusters then merge them into one new cluster. The number of clusters will be $N - 1$.
3. Compute the similarities between the new cluster and all the old ones.
4. Repeat steps 2 and 3 to reach the most desired results or reach to the requested number of clusters.

To compute the similarity between two clusters, the similarity between all the document pairs in the two clusters are computed. So, there are similarity scores between each pair of document that each one belongs to a cluster. Having computed the document similarities, similarity computation of clusters can mainly be done in three different ways:

- *Single-linkage clustering* where the similarity of two clusters is considered the similarity of their *most similar* documents. In other words, the similarity of the clusters is estimated by the similarity of the two closest documents in the two clusters.
- *Complete-linkage clustering* where the similarity of two clusters is considered the similarity of their *most dissimilar* documents.
- *Average-linkage clustering* where the similarity of two clusters is considered the *average of the similarities* of the documents. In other words, for all the document pairs the similarity is computed and the similarity of the clusters is the average of all those similarity scores.

Overlapping Clustering

Overlapping clustering is based on fuzzy sets therefore each clustered document can belong to more than one cluster. The membership of each cluster is associated with a degree (weight).

One of the popular algorithms in this group is *Fuzzy C-Means (FCM)* [Dunn, 1973], in this algorithm one document can belong to more than one cluster and each document is associated with a set of membership degrees (weights). This algorithm is widely used in pattern recognition and image processing[Bezdek, 1981] .

FCM focus is on using the membership degrees that minimise the total mean-square error of membership degree:

$$J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} {U_{ij}}^m |X_i - C_j|^2, 1 <= m < \infty \tag{2.5.4}$$

Where X_i is document i from the given finite document set, U_{ij} is the degree of member ship of document X_i in the cluster j, C_j is the centre of cluster j and $|X_i - C_j|$ expresses the similarity between the document and the centre.

The membership degree can be calculated using the following formula:

$$U_{ij} = \frac{1}{\sum_{k=1}^{C} (\frac{|X_i - C_j|}{|X_i - C_k|})\overline{\frac{2}{m-1}}} \tag{2.5.5}$$

Where k is the iteration steps. This iteration will terminate when:

$$max\{|U_{ij}{}^{(k+1)} - U_{ij}{}^{(k)}|\} < \xi \tag{2.5.6}$$

where ξ is the termination criterion.

Probabilistic Clustering

Probabilistic clustering uses probabilistic algorithms to group documents into different clusters.

Model-based clustering [Wolfe, 1963] techniques are another way to solve clustering problems. In this approach, clusters are mathematically represented using a parametric distribution such as a Gaussian (continuous) or a Poisson (discrete).

There are assumptions for this approach:

- Documents are collected from a finite document collection.
- Document within each collection can be modelled using standard statistical model.

Model-based clustering has some advantages:

- Available statistical inference techniques are well-studied.
- It is very flexible to choose the component distribution.
- For each cluster a density estimation is obtained.
- Soft-clustering is available.

2.6 Classification

Classification refers to the problem where there are pre-labelled classes and documents are labelled to those specific classes, when a new document arrives, it should be labelled to one of the class it belongs to [Dominich, 2008].

In text classification, there is a document space $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$, containing all the available documents to classify. A set of classes $\mathcal{C} = \{c_1, c_2, \ldots, c_j\}$, these classes are human-predefined labels to address different and desired information needs for applications. There is a set of labelled documents which are manually labelled for each class. This is called *training set* and is used as a set of examples for classification, the classifier can be built based on these examples or can be validated if they have not been used for training. There are more discussions on training set, its usage and machine learning in general in Section 3.2.1.

A classification system can be developed by employing experts knowledge to conduct rules to discriminate between documents and select the relevant ones to the class. These classification systems can be very accurate, because they benefit from the experts knowledge in the field that no systems can acquire from the labelled examples. However, maintaining the system and modifying it for future needs, are very expensive processes. In addition, for some problems, the complexity of features is a substantial obstacle for the human expert.

Another way to build a classification system is to use automatic algorithms to create a model based on the labelled examples to predict future classifications. A learning method or algorithm is used to go through all training documents and generate a classifier or a classification function. The responsibility of this classifier is to map the documents in document space $\mathcal{D}$ to the related class or classes in the class set $\mathcal{C}$.

These labels are symbolic notations and they do not add extra information. Also all the information about these labels are from inside the collection set and the document, no extra knowledge is available from outside [Sebastiani, 2002].

Nowadays modern applications increasingly require a classification system. Based on the application purpose, a specific approach is applied to the classification problem.

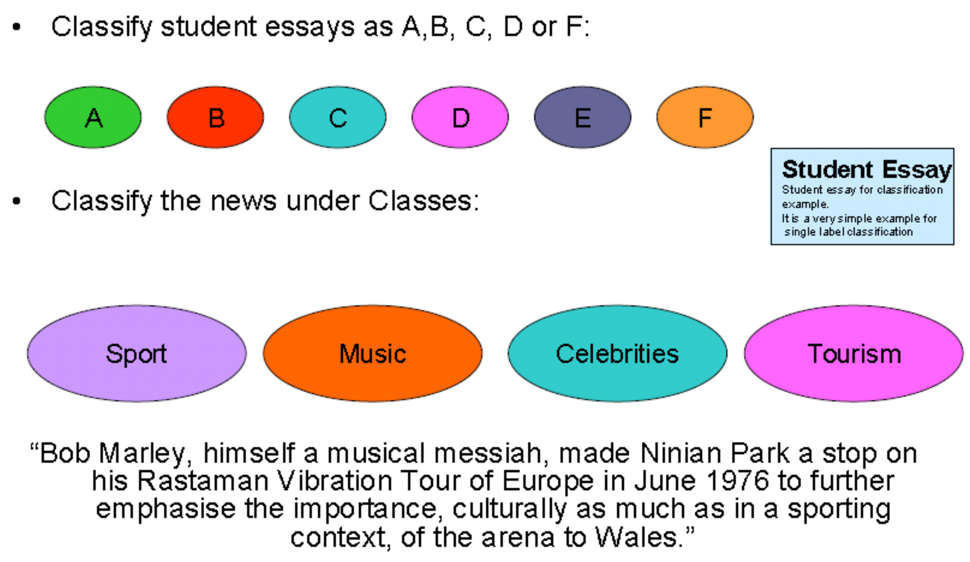

• Classify student essays as A,B, C, D or F:
A
B
C
D
E
F
Student Essay
Student essay for classification example.
It is a very simple example for single label classification
• Classify the news under Classes:
Sport
Music
Celebrities
Tourism
"Bob Marley, himself a musical messiah, made Ninian Park a stop on his Rastaman Vibration Tour of Europe in June 1976 to further emphasise the importance, culturally as much as in a sporting context, of the arena to Wales."

Figure 2.3: Classification

2.6.1 Classification Types

As it was mentioned the task of assigning documents to predefined labels (that represent predefined classes) has become a very promising field. This field targets the hidden information in text documents, where the length of each document and number of words vary and the documents structure are in natural language. In addition, the vagueness, uncertainty and fuzziness should be handled [Hotho et al., 2005].

Single-label Classification vs. Multi-label Classification

Single-label classification, is assigning each document (natural language text) to only one of the predefined labels (classes) based on its content. In this method, classes are assumed to be disjointed, each document belongs to only one of these classes [Gao et al., 2004].

Some examples for single-label classification can be: Students' marks (A, B, C, . . .), age range (infant, youngster, teenage, adult, middle age, old), education degree (Bachelor degree, Master degree, MPhil, PhD, higher), level of proficiency (beginner, intermediate, advanced), type of media (TV, radio, newspaper, web).

In the real world, most cases are related to each other and very few cases can be totally independent, in fact one document can be labelled by more than one class. The task of assigning one document to more than one class is called Multi-label classification [Ghamrawi and McCallum, 2005].

Examples for multi-label classification can be: The courses that one student takes during one term (management, business, marketing, ...), different media that broadcasts news (TV, radio, newspaper and web), different aspects of one news (economy, social life, tourism, industry, ...)

In Single-label classification the assumption is based on class independence whereas in Multi-label classification the assumption is based on class dependence and classes' semantics.

Binary Classification vs. Multi-class Classification

In single-label classification task each document is labelled by only one label, however there is no restriction on number of classes available. There is a very specific form of single-label classification where there are only two classes available. This special single-label classification is known as Binary classification. A binary classification can be notated as, where the document (d_j member of D) belongs to class (c_j member of C) or its complement ($C - c_j$) [Hatano, 2001].

Binary classification examples such as the number of students who have passed a test, either a student has passed or not, the news has been televised or not, the emails are spam or not. In these cases there are always two classes: pass or fail, televised or non-televised, spam or non-spam. One of the most popular uses of binary classification is filtering [Pon and Cárdenas, 2007; Robertson, 2002b].

Though the binary classification, especially filtering are widely used, in most cases there are more than two classes where an email can be from work, friends, home, online shopping etc. Where the number of available classes is more than two, it is a multi-class classification problem [Tax and Duin, 2002].

It is crucial to understand that a multi-class classification problem is different from a multi-label classification problem.

Class-oriented vs. Document-oriented

Traditionally classification is referred to classify a document under a predefined class. However there are two approaches towards the classification problem.

Sometimes a collection of documents is available and the classifier provides a list of relevant documents per each individual class from the set of predefined classes. This approach is known as class-oriented or class-pivoted classification, where the emphasis is on classes. This is what happens during training and testing the classifiers, where a collection of train documents is ready and classifier has to find the relevant documents for each class. However, in the real world the collection of documents is not ready and normally documents will be identified one by one.

In the second approach, the classifier provides a list of relevant predefined classes for each individual document and the emphasis is on the documents. This approach is known as document-oriented or document-pivoted classification.

Though these two approaches are more practical than conceptual, in most cases either the document collection or the set of predefined classes are not complete from the beginning [Sebastiani and Ricerche, 2002].

The choice of orientation is more practical than conceptual, but very important as in most of the cases the document collections or class sets are not complete right from the beginning [Sebastiani, 2002].

Hard Classification vs. Ranking Classification

Measuring the relevance of each document to a class can be considered as a "YES/NO" job, where the classifier either labels each combination of a class and a document as relevant or not. This approach is called hard-classification [Qi and Davison, 2009; Wu and Oard, 2008].

Though hard-classification is widely used, it is not a sufficient way of classifying real world documents. Where document and class are not totally irrelevant, there is always a level of relevancy which is ignored in hard-classification. There is, however another approach to consider this level of relevancy known as ranking-classification or simply ranking. In this approach the classifier measures the similarity or relevance between each document and each class, sorts the similarity measures and provides the result as a ranked-list of document-class pair based on their ranks [Elsas et al., 2008; Sun et al., 2009; Elsas and Dumais, 2010].

Hierarchical Classification vs. Flat Classification

In the classification task, there are different predefined classes which are treated as isolated and totally independent classes. These classes do not have any relationship with each other; this approach to classification is known as flat-classification [Zimek et al., 2008; Li et al., 2007; Nguyen et al., 2005].

In contrast, there is an approach which considers a tree-shaped relationship between classes. In hierarchical-classification, the classifier uses a tree-shaped structure which represents, the general to specific levels and assigns suitable classes to one document from one branch. There are two methods for this approach: where the document is labelled by one class in one step process, the task is known as big-bang method. Top-down level-based documents pass through classifiers in each level till they get labelled to the suitable class [Ee-Peng et al., 2003; Cesa-Bianchi et al., 2006; Gauch et al., 2009].

2.6.2 Applications of Text Classification

Text Classification is based on Maron's [Maron, 1961] original work, however it has been applied to many different applications. Some of these applications are very similar to each other or sometimes one is a special case for another one. Such applications include speech classification which is a combination of speech recognition and text classification [Myers et al., 2000; Schapire and Singer, 2000], multimedia document classification by the means of textual captions [Sable and Hatzivassiloglou, 2000] author identification for literary texts of unknown or disputed authorship [Pavelec et al., 2009], language identification for texts of unknown language [Cavnar and Trenkle, 1994; Martins and Silva, 2005; Rehurek and Kolkus, 2009], automated identification of text genre [Kessler et al., 1997; Goldstein-Stewart and Winder, 2009] and automated essay grading [Larkey, 1998] . Some of the most common and widely used applications have been discussed as follows.

Automatic Indexing for Boolean Information Retrieval Systems

Most of the early research in the field of IR [Field, 1975] has been regenerated by automatic document indexing for IR systems. These applications depend on infinite sets, called *controlled dictionaries* and are made up of hierarchical thesauri, such as the NASA thesaurus for aerospace and MESH thesaurus for medicine. The most distinguished example of these applications is *Boolean Systems*. They use the controlled dictionary to assign one or more key words or key phrases to each document, in order to describe the document's content. Where key words and key phrases in the controlled dictionary indicate classes, the task of indexing documents can be considered as a text classification in the form of document oriented (pivoted) classification (see Section 2.6.1). More examples and descriptions for text classifiers that clearly illustrate document indexing are available in the literature [Robertson, 1984; Tzeras and Hartmann, 1993].

As automatic indexing for Boolean IR systems is a very broad task (concept), one of its widely used sub-tasks is *automated metadata generation*. In a digital library, documents may be described under a variety of aspects, such as document type, modification data, owner, size, etc, or sometimes these aspects are thematic and they revile the document's semantics. These aspects or *metadata* can be seen as a task of document indexing with a controlled dictionary [Jackson and Moulinier, June 2007].

Document Classification

Although using a controlled vocabulary in order to index document is a widely used application, it is a sub-task of a more general task called *Document Organisation*. Document organisation can address personal documents filing as well as building a corporate document base [Sebastiani, 2002].

For instance, in a university registration office, each lecturer should be assigned to a related department, students are categorised under the discipline they have registered for and type of founding and financial issues.

Text Filtering

A text filtering system focuses on the incoming information flow. These systems work with a long term user information need, represented by profiles. Dissimilar to traditional search query, these profiles are continual and can be optimised by user feedback, which improves the overall system performance gradually.

A text filtering system main job is to sieve the incoming information flow based on existing user information need and identify the relevant or non-relevant documents. Filtering systems should be able to present relevant documents immediately to the user, so there is no time to collect and rank a set of documents and rational evaluation is purely based on the quality of the retrieved documents.

There are three main filtering sub-tasks: adaptive filtering, batch filtering and routing. Adaptive filtering starts with a user information need and a small set of relevant documents as its training set, the system retrieves each document, assigns relevant or non-relevant labels. Then the system is able to use it to update the user profile. Batch filtering and routing start with a large training dataset to build their profiles. While batch filtering should assign the relevant or non-relevant labels to the retrieved documents, routing system returns a ranked list of documents [Robertson, 2002a].

Novelty Detection

Novelty detection is to find the relevant and novel sentences within a given topic and a list of documents ordered based on their relevancy [Soboroff, 2005].

One of the applications for IR is *Novelty detection*. The basic idea was introduced by *Prof. Jamie Carbonnell* during a talk at the *Automatic Summarisation Workshop* at *NAACL*

(North American Chapter of the Association for Computational Linguistics) in May 2001. He argues that, in order to optimise search results relevance ranking is not the only option. Documents can be ranked based on punctuality of the article, legitimacy of its source, understandability and *Novelty* of its information. While some of the mentioned features are very easy to identify such as punctuality (timestamps) some others are very difficult and complicated to detect such as source validity and user comprehensibility, in between novelty can be handled by the hypothesis that there is no prior knowledge or information about the document and all the learning process happens during the document retrieval [Zhao et al., 2006].

Word Sense Disambiguation

In many languages there are words or phrases that are spelled the same and pronounced the same but they have totally different meaning (polysemous or homonymous). For instance, bank can identify a financial organisation, or means the river side. In medicine bread diet can present a person who usually eats bread; or prescribed bread to treat a condition. *Word Sense Disambiguation* is the task to clarify the sense of each word in an ambiguous sentence. WSD can be considered as a classification task, where each sentence is assumed as the document and the word sense as classes [Tufis et al., 2005; Escudero et al., 2000]. Needless to say it is a single-label (see Section 2.6.1), document-oriented classification (see Section 2.6.1) task.

Hierarchical Categorisation of Web Page

Hierarchical structure is one of the most efficient structures for large sets, this approach improves the scalability of building the classifier for such sets. For the same reason lots of popular internet portals host the web pages and sites based on a hierarchical structure. When online documents are itemised or catalogued hierarchically, a searcher finds the appropriate sub-hierarchy first and limits the domain to that specific class rather than performing a general search over the whole collection. Although web pages do have two strange characteristics:

- The hypertextual nature of the web means that links provide an instant access to vast amounts of information. Links may lead to other relevant pages as well as pages with totally different information. There are techniques to deal with this characteristic [Oh et al., 2000; Yang et al., 2002, 2003].
- The hierarchical structure of the web makes it possible to break down each classification problem into smaller problems [Dumais and Chen, 2000].

2.7 Evaluation of IR system

So far the need and design of IR systems in general have been discussed and the final step in the series is to evaluate the system. The evaluation reveals which technique is the most effective one, whether the stop word list is complete or not, which method of stemming improves the result, and if the used weighting scheme is an appropriate one. Each of the mentioned issues may or may not improve or optimise the result of an information retrieval system. They are tested to see if they have any effects on user happiness which is assumed as the key utility measure. User happiness also depends on response time/speed; no one is satisfied to wait too long for an accurate result. In reality no user waits more than a minute to get the result. The next point is index size which can affect both the response time/speed and a key point in user satisfaction, *relevant results*. However the result relevance and response time/speed are very important, user interface must make the user feel comfortable to work with its layout which should be very clear and responsive towards user's needs.

To evaluate an information retrieval system there are three standard elements:

1. A document set
2. A set of representatives (queries) for information needs
3. A query-document pair of relevance judgements

There has been lots of studies to measure the effectiveness of an information retrieval system. Techniques, models and design choices in such systems can affect the outcome.

A careful and complete evaluation is an essential part in the designing of these systems. Also, any system should be evaluated and have feedback in order to optimise and improve its performance. A typical evaluation approach relies on the concept of *relevant* and *non-relevant* documents. The user information need and a document from the set are assigned to each other with a binary classification as either relevant or non-relevant. The variety of information need representatives and the number of documents should be considered to provide an accurate result.

To find the relevant documents to one user information need, queries are used, but the relevancy is estimated based on the user information need and not the query. To make it clearer, consider an information need as:

```
The advantages or disadvantages of computer employment in
students education
```

while the query that represents it might be :

```
computer AND employment AND students AND education
```

which may represent another information need such as:

```
Employment opportunity for students with education in computer fields
```

The fact that relevancy should be measured between documents and information needs and ***NOT*** queries, is often misunderstood or ignored in practice. The reason is, information needs are not normally clear and plain. To somehow overcome this issue and also tune the system performance, many systems use various weights. To have a fair and realistic evaluation it is very important not to use the same collection that has been used for these weights and the system adjustment. As these documents are used to optimise the system performance over those specific weights, it will lead to a biased evaluation [Fransson, 2009].

2.7.1 Unranked Retrieval Evaluation

The first step in retrieval evaluation is to identify if the document and information need are relevant based on a binary decision. *Precision* and *recall* are the two most popular and widely used measures to present the systems effectiveness.

Precision shows how many retrieved documents are actually from the relevant document set, and is calculated as follows:

$$Precision = \frac{\#(\text{relevant documents retrieved})}{\#(\text{retrieved documents})} = P(relevant|retrieved) \tag{2.7.1}$$

Recall shows how many relevant documents have been actually retrieved, and is calculated as follows:

$$Recall = \frac{\#(\text{relevant documents retrieved})}{\#(\text{relevant documents})} = P(retrieved|relevant) \tag{2.7.2}$$

A system retrieves non-relevant documents as well as relevant documents, and identifies them as relevant documents by mistake. Also some of relevant documents will be overlooked, treated like non-relevant documents and not retrieved. To clarify, this can be shown in Table 2.1:

	Relevant	Non-relevant
Retrieved	true positives(tp)	false positives(fp)
Not retrieved	false negatives(fn)	true negatives(tn)

Table 2.1: True and false, positive and negative retrieval.

based on this table and the above definitions, precision (P) and recall (R) can be interpreted as follows:

$$P = tp/(tp + fp) \tag{2.7.3}$$

$$R = tp/(tp + fn) \tag{2.7.4}$$

Another option to evaluate an information retrieval system is by its *accuracy*:

$$accuracy = (tp + tn)/(tp + fp + fn + tn) \tag{2.7.5}$$

The majority of the documents in almost all cases, belong to the non-relevant class. Since the data is extremely biased, the accuracy is not an appropriate measure. A system with a very high accuracy can simply identify all the documents as non-relevant which is unsatisfactory for users.

Precision and recall focus on the proportion of the number of returned true positive items relative to the number of returned or relevant items respectively. Based on the user's intention either precision or recall can identify the more accurate systems. General web surfers prefer more relevant results on the first page, which means high precision will serve them best, while, professional searchers prefer systems with high recall, they care how many of the relevant documents are retrieved. Precision and recall are two measures that trade off against each other. When the number of documents retrieved increases and the recall value, referring to its formula, is a non-decreasing function; the fraction does not change if a new non-relevant document is retrieved. While in a desired system by increasing the number of retrieved documents the precision decreases. In general, we can say, a system with $Precision = 1$ retrieves only relevant documents but misses a lot of them. On the other hand, a system with $Recall = 1$ retrieves mostly relevant documents but the result contains a lot of non-relevant ones as well [Meadow et al., 2007; Manning et al., 2008].

Despite their popularity and usefulness, precision and recall measures have some drawbacks, namely:

- Do not show the efficiency of the system
- Do not provide any information about number of non-relevant documents in the collection
- In the case of no relevant documents, recall will be undefined
- In the case of no retrieved document, precision will be undefined

2.7.2 Ranked Retrieval Evaluation

Unordered sets of documents are used to measure precision, recall and F-measure so they are known as *set-based* measures. In ranked retrieval methods, a set of top k retrieved documents is presented, precision and recall are calculated and a precision-recall curve is plotted. $(K+1)^{th}$ document is retrieved, if it is non-relevant, precision will decrease while the recall stays the same, but if it is relevant, both precision and recall will increase.

The first solution to take this increasing and decreasing into consideration is to calculate the average for different precisions at the same recall level, also known as *average precision*. This new measure, average precision, has improved the evaluation of process, there are other more common measures. One method, that has become very popular especially as a standard in the TREC[8] community is *Mean Average Precision*, in which a single-figure measure is used for all recall levels. Consider the set of $\{d_1, ..., d_{m_j}\}$ as the relevant documents for an information need representative $q_j \in Q$ and R_{jk} as a set of top k ranked retrieved documents, then Mean Average Precision, *MAP*, can be calculated as follows:

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk})$$

However, MAP is a single measure for whole system, it makes more sense to calculate MAP for an individual information need in different systems than MAP for different information needs in one system.

MAP has the advantage that it does not require any size estimation for relevant documents set. In contrast with other commonly used evaluation measures, there is a disadvantage too. The total number of relevant documents has a strong influence on precision so MAP is the least stable and does not average well [Croft et al., 2009; Göker and Davies, 2009].

[8]Text REtrieval Conference. For more information refer to `http://trec.nist.gov`

2.7.3 Assessing Relevance

To evaluate a system properly user information need should be closely related to the test document collection. The provided user information need also must be able to present and predict the system's goal. It is therefore better that a domain expert designs the queries that represent user information need. These experts randomly combine discriminant query terms, then they compare information need and representatives and documents against each other and finally; *relevant assessments* are measured. However, not only are these assessments time consuming and expensive but also there is no standard judgement for human users or experts. The relevance judgement from different experts relies on their knowledge background and their personal or professional preferences. How much agreement exists between the experts about the relevance is ambivalent. Based on TREC tradition, alongside the train and test sets of documents and sets of classes, predefined relevance judgements are provided, to make it possible to evaluate different systems based on standard pairs of information need and document [Meadow et al., 2007; Dominich, 2008].

2.8 Summary

The main objective of this chapter is to provide a solid background in information retrieval in general and specifically classification. The chapter has clarified the meaning of document representative, user information need and its difference with query. Also the process of generating representatives for documents or generating queries based on user information need and their optimisations as well as matching the document representatives and queries are explained.

This chapter has reviewed the need for IR. It has provided an insight into *information retrieval (IR)* processes; from document representation techniques and queries. More details for each part of the process, such as tokenisation, case-folding, stop-word removal and weighting for document representation, have been provided. Boolean query, vector space model and coordination level for matching documents and queries are explained step by step.

Different methods of Machine Learning (ML), namely supervised and unsupervised learning are introduced. Some applications that use IR to improve the quality of services they provide such as web-based search and enterprise search are named. Three main problems in IR: document retrieval, clustering and classification are introduced, while clustering and classification are explained with more details on clustering and classification types and classification applications.

Evaluation of IR systems is divided into three sections to address unranked retrieval, ranked retrieval and assessing relevance to finish this chapter.

CHAPTER 3

Biologically Inspired Models

3.1 Introduction

In the field of *Artificial Intelligence (AI)*, machine learning is a major approach with highly appreciated results in many problems. In this chapter, we start by briefly introducing machine learning. We will discuss biologically inspired models also known as *evolutionary computation (EC)* models. The basic concepts are based on the principles of biological evolution. However, there are differences in their representations of solutions and operators. We introduce genetic programming in more detail, since it is the main algorithm used in this thesis.

3.2 Machine Learning

Machine learning is a sub field of *Artificial Intelligence* that transforms a system to an intelligent and automated one. Machine Learning is referred to a system that learns from experience, analytical observation and other means. After training, the system can improve its performance, efficiency and effectiveness [Mitchell, 1997]. Machine learning algorithms have various applications in many fields such as natural language processing [Manning and Schuetze, 1999], computer vision [Duda et al., 2000] and data mining [Witten and Frank, 1999].

Machine learning has been used for a long time to create self-improving systems, systems that learn to optimise and improve themselves. As Oliver G. Selfridge puts it: if one finds and fixes a bug in a program, the program works today, however if one shows the program how to find and fix the bug, the program will work forever [Hearst and Hirsh, 2000].

It is used to create systems that learn how to do tasks in a quality level which is comparable with human being performance, meanwhile it reduces the time and expenses that the work done by human needs.

3.2.1 Machine Learning Algorithm Types

Supervised Learning

One of the algorithms of machine learning is supervised learning. A training dataset is used to train the system and create a function to deal with new requests. The training data is normally a pair represented as a vector. This vector assigns input objects to preferred outputs [Duda et al., 2000].

In this technique, the system receives a pre-categorised set of data. The system uses this pre-categorised set to find the logical relationships between terms by creating a prototype or a concept model for itself. When the system receives a new request, it repeats the same processes in order to conceptualise the new request. To achieve this conceptualisation, the system should see enough samples to be able to generalise the characteristics of the training data in order to apply it to unseen data. This process in human being and animals is known as *concept learning*. If the output of the system is a continuous value, the process is called *Regression*, and if the output is a set of classes of labelled input documents, it is called *classification*.

Unsupervised Learning

Unsupervised learning is another family of algorithms used in machine learning. In such algorithms there are some inputs that need to be put together, but unlike supervised learning, there is no specified output set [Mitchell, 1997]. In unsupervised learning set of inputs are assumed as a set of random variables. This kind of learning is used to:

- Discover the structure of the data
- Encode the data
- Compress the data
- Transform the data

Semi-supervised Learning

This family of algorithms is a combination of the two previous ones. In such algorithms a small training set and a large amount of raw data is used to train the system. It is reported that they show better practical results than others [Mitchell, 1997].

Reinforcement Learning

In reinforcement learning, for each input an action will be taken. The system's action makes changes in the environment. Based on these changes, it receives rewards or punishments (feedbacks). The system's aim is to learn how to maximise the rewards [Mitchell, 1997].

3.3 Introduction to Biologically Inspired Computing

Biological processes are used as a source for models, inspiration and ideas in developing new computational architectures based on the principles of natural biological systems. The primary objective of models based on biological evolution, is to study and extract the essential working principles from biological systems and translate them into useful software algorithms or architectures [Bäck et al., 2000].

One way in which bio-inspired computing differs from other artificial intelligence (AI) algorithms is in how it takes a more evolutionary approach to learning, as opposed to what could be described as "creationist" methods. In traditional AI, intelligence is often programmed from above and the programmer is the creator. The programmer creates something and makes it intelligent. Bio-inspired computing, on the other hand, takes a more bottom-up, decentralised approach; bio-inspired techniques often involve the method of specifying a set of simple rules, a set of simple organisms which adhere to those rules, and a method of iteratively applying those rules. After several generations of rule application it is usually the case that some forms of complex behaviour arise. Complexity gets built upon complexity until the end result is something markedly complex, and quite often completely counter-intuitive from what the original rules would be expected to produce [Bäck et al., 2000].

Natural evolution is a good analogy to this method, the rules of evolution (*selection*, *recombination/reproduction* and *mutation*) are in principle quite simple rules, yet over thousands of years have produced remarkably complex organisms. The technique used in evolutionary computation is very similar.

To understand biologically inspired algorithms we need to be familiar with some biological terms [Luke, 2009; Bäck et al., 2000; Luke, 2000]:

- **Individual**: A single member of a population, representing a candidate solution. In Evolutionary computation, each individual represents a possible solution to the problem being tackled, i.e. a single point in the search space. Other information is usually also stored in each individual, for example, its fitness.
- **Breeding**: Producing one or more off-springs from a set of parents by means of genetic operators.
- **Genotype** or **genome**: the data structure of the individual used for breeding.
- **Chromosome**: In biology, it is strings of DNA that serve as a "blueprint" for the organism. In EC, it is vector-based genome.
- **Gene**: Each chromosome can be conceptually divided into genes. In other words, each element of the chromosome vector.

- **Reproduction**: The creation of one or more new individuals from two or more parents (sexual reproduction). Also, asexual reproduction can be the creation of a new individual from a single parent.
- **Crossover** or **recombination**: A reproduction operator which forms two children by combining parts of two parents.
- **Mutation**: A reproduction operator which forms a new individual by making minor changes to the parent's gene.
- **Fitness**: In general, it means quality of an individual. It is a numeric value assigned to an individual which shows how well the individual solves the problem.
- **Fitness function**: A function that maps individuals to fitness values.
- **Population**: A collection of individuals.
- **Generation**: An iteration of the fitness assessment and the creation of a new population by means of reproduction operators.

3.3.1 Evolutionary Algorithms

Evolutionary algorithms is an umbrella term used to describe the algorithms inspired by biological models. The major ones are: genetic algorithms, evolutionary programming and genetic programming. All evolutionary algorithms share a number of common properties:

- They all utilise the collective learning process of a population of individuals [Bäck et al., 2000]. They are generally re-sampling techniques and new populations are created by revising previous ones [Luke, 2009].
- New individuals based on the selection of older ones are generated by randomised processes intended to model mutation and crossover.
- The quality of the individuals are assessed separate from the others or in comparison to the others. The assigned fitness is used in selection processes for breeding.

Therefore, fitter individuals are more likely to breed new children than the less fit ones.

An evolutionary algorithm usually starts with constructing an initial population, assesses the fitness of each individual in the population and uses that information for breeding. Breeding process then produces a new generation and a new cycle starts. After a certain number of generations, or some other termination criterion, the algorithm returns the fittest individual as the solution for the problem.

3.3.2 Genetic Algorithm

Genetic algorithms are a class of evolutionary algorithms first proposed and analysed by John Holland [Holland, 1975]. A typical genetic algorithm requires two things to be defined:

1. a genetic representation of the solution candidates.
2. a fitness function to evaluate the solution candidates.

A common representation of the solution candidates is a vector of bits or a string of bits. Vectors of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, that facilitates simple crossover operation. Once we have the genetic representation and the fitness function defined, genetic algorithm proceeds to initialise a population of solutions randomly, then improve it through repetitive application of mutation, crossover, and maybe other operators [Bäck et al., 2000].

Initialisation

Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Traditionally, the population is generated randomly, covering the entire range of possible solutions (the search space).

Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found. Then each individual is evaluated for fitness [Luke, 2009].

Selection

Some of these individuals are selected to breed a new generation. Individual solutions are selected through a fitness-based process where fitter solutions (as measured by a fitness function) are typically more likely to be selected. However, there are many approaches for selection that select a combination of high fitness and low fitness individuals in order to make it possible to explore a larger part of the search space.

Reproduction

The next step is to generate the next generation of candidate solutions from those selected through genetic operators, such as: crossover (also called recombination), and/or mutation. The rate at which mutation and crossover are applied is an implementation decision. By producing a *child* using the crossover and mutation, a new solution is created which typically shares many of the characteristics of its *parents*. New parents are selected for each child, and the process continues until a new population of solutions of appropriate size is generated. These processes ultimately result in the next generation population of individuals that are different from the initial generation. Generally, the average fitness will have increased, since only the best individuals from the first generation are selected for breeding, along with a small proportion of less fit solutions [Bäck et al., 2000].

For a crossover example consider two selected parents as in Figure 3.1:

1	**Parent1**	10101001
2	**Parent2**	00111111

Figure 3.1: A cross-over example in GA with binary vector representation.

Suppose the vectors is a binary vector with the length of 8. A one-point crossover at point 5 gives us the fragments in Figure 3.2:

	Fragment 1	**Fragment 2**
Parent1	10101	001
Parent2	00111	111

Figure 3.2: One-point cross-over example. The cross-over point is 5.

By combining **Fragment 1** from **Parent1** with **Fragment 2** from **Parent2** and **Fragment 1** from **Parent2** with **Fragment 2** from **Parent1**, two new children will be produced as in Figure 3.3:

1	**Child1**	10101111
2	**Child2**	00111001

Figure 3.3: Produced children in result of a cross-over.

The mutation operator works on one parent by first selecting it from the population and then selecting a random point on the vector and complementing it. Cross-over and mutation operators for other representations (non-binary vectors) are very similar.

Termination

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria.
- Fixed number of generations reached.
- The highest ranking solution's fitness has not improved in a series of generations.
- Manual inspection.
- Combinations of the above.

Figure 3.4 shows the general process in genetic algorithms. The process starts with an initial population in which each individual in this population is randomly chosen. Each individual should fulfill a series of evaluations were its fitness is calculated: either

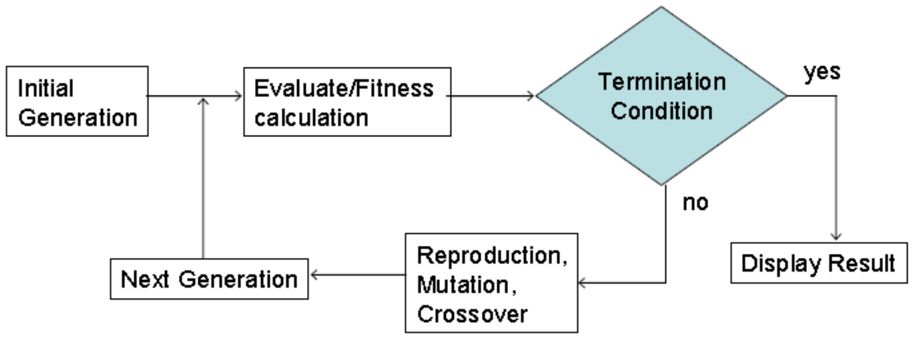


Figure 3.4: Genetic Algorithm

the evaluated individuals meet the termination condition and the results are displayed. Or, the process will repeat as a recursive process, in order to generate a new population and re-evaluate the new population till it meets the termination condition.

3.3.3 Genetic Programming

Genetic programming is one of the evolutionary algorithms that differ from the genetic algorithm approach in representing individuals, genetic operator designs and fitness function evaluation.

In genetic programming individuals are represented by computer programs and a fitness evaluation for them means executing these programs. Genetic programming like genetic algorithm involves an evolve-based search on possible computer programs to find the one that produces the best fitness value [Koza, 1992].

Genetic programming starts with an initial population of computer programs which are generated randomly. Computer programs are composed of functions and terminals which are specific to the problem domain. The fitness value for each individual is measured in a particular problem domain. Typically, each computer program is run over a number of fitness cases so that fitness is measured as a function of these execution values. The genetic operators are used to create new offspring computer programs. Algorithm 1 shows the structure of a genetic programming process [Luke, 2009]. In general, the process can be summarised as [Koza, 1992]:

1. Generate an initial population of computer programs randomly which are composed of the functions and terminals of the problem.

2. Repeat until the termination criterion has been occurred.

   (a) Evaluate the fitness value for each program according to how well it solves the fitness cases.

   (b) Create a new population of computer programs by applying genetic operators to probability-based on fitness selection of parents.

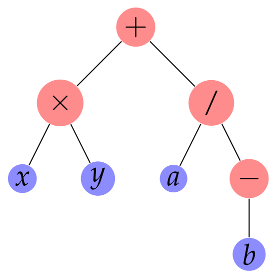
3. The best computer program that appeared in any generation is provided as the result of problem.

Algorithm 1 A genetic programming process.

```
Require: class $c$ and the training data set
Require: parameters of GP such as: $popSize$, operators rate, ...
1: $P \leftarrow$ randomly generate an initial population
2: repeat
3:   for all $p_i \in P$ do
4:     assessFitness($p_i$)
5:     $best \leftarrow$ findNewBest($best$, $P$)
6:   end for
7:   $P' \leftarrow \{\}$
8:   while $|P'| < popSize$ do
9:     $c \leftarrow$ generateNewChild($P$)
10:    $P' \leftarrow P' \cup \{c\}$
11:  end while
12:  $P \leftarrow P'$
13: until termination criteria met
14: return $best$
```

The representation used by genetic programming is in the form of executable computer programs. There are many different forms of computer programs which are used in genetic programming implementations, but most of them use a tree-structured representation. Figure 3.5 shows an example of tree-structured representation for a genetic programming implementation.

Most of the genetic programming implementations use two types of nodes in representing individuals: Terminals and Functions. Terminals are inputs to the program, they can be constants or variables. In the above example a, b, x and y are terminals.



$(x \times y) + \ (a/(-b))$

Figure 3.5: An example of tree structured genetic programming representation.

Functions take inputs which produce outputs and have some possible side-effects. The inputs can be terminals or the output of other functions. In the above example $+$, $/$, $-$ and $\times$ are functions. The functions may be standard arithmetic operations, standard mathematical functions, logical functions, standard programming functions or domain specific functions.

The representation for the program is the definition of the functions and terminals. There are some important points that should be considered when defining the terminal and functions sets. First, the set of terminals and the set of primitive functions must be capable of expressing a solution to the problem. Each one of the functions in the function set should accept any value and data type, that are produced by any other function or terminal, as argument. Each function in the function set should be well defined and closed for any combination of arguments that it may encounter. One way to define a function set is to use only a single data type, so the output of any function or the value of any terminal is acceptable as input for any function. Another way is to use *strongly-typed genetic programming* which enforces data types constraints [Montana, 1995]. The definition of functions is not restricted to functions that only produce an output, many genetic programming functions are used because of their side-effects. These functions should return a value as well, but the main purpose of them is affecting the environment.

Fitness Measure

In genetic programming fitness value for each individual is evaluated by executing the program on a set of fitness cases and summarising the results. Fitness cases are a set of inputs in which the correct answers are known. In other words, the fitness of a program is a function of the number of fitness cases which produces the correct result.

Genetic Operators: Reproduction, Crossover and Mutation

Reproduction, crossover and mutation are three basic operators in genetic programming. They have different performances in different domains, however [Luke and Spector, 1997] have done some research about the comparison of mutation and crossover and have concluded that crossover has some advantages over mutation.

- **Reproduction**, First a single individual is selected from the population according to some selection method based on fitness. Second the selected individual is copied without alteration from the current population into the new population.

- **Crossover**, The crossover operation generates a new offspring that consists of parts of its parents. According to Figure 3.5 there are two parents; A and B. Each parent independently selects a point for crossover operation randomly. Then two offspring will be generated by swapping two selected fragments of parents. For example, we have following selected fragments (Figure 3.6), and by swapping the fragments we have the following children (Figure 3.7).

- **Mutation** In genetic programming a point is selected at random within a selected individual. The mutation operation then removes whatever is currently at the selected point and whatever is below the selected point and inserts a randomly generated sub-tree. Another method is to randomly find a function node anywhere on the tree and then replace the function with another selected at random, from the set of functions, where the function takes the same number of arguments as the function to be replaced.

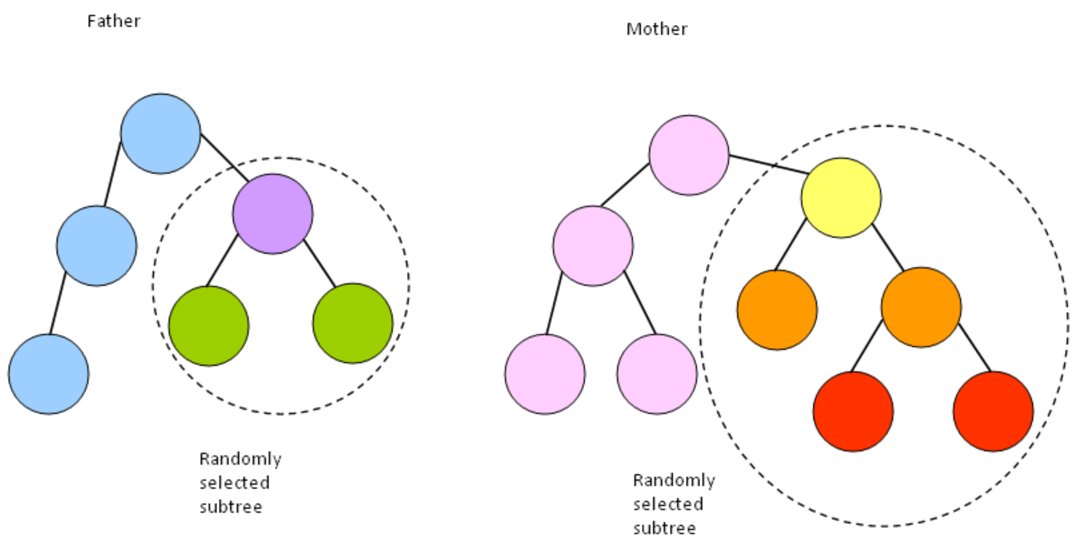


Figure 3.6: Two parents selected for crossover.

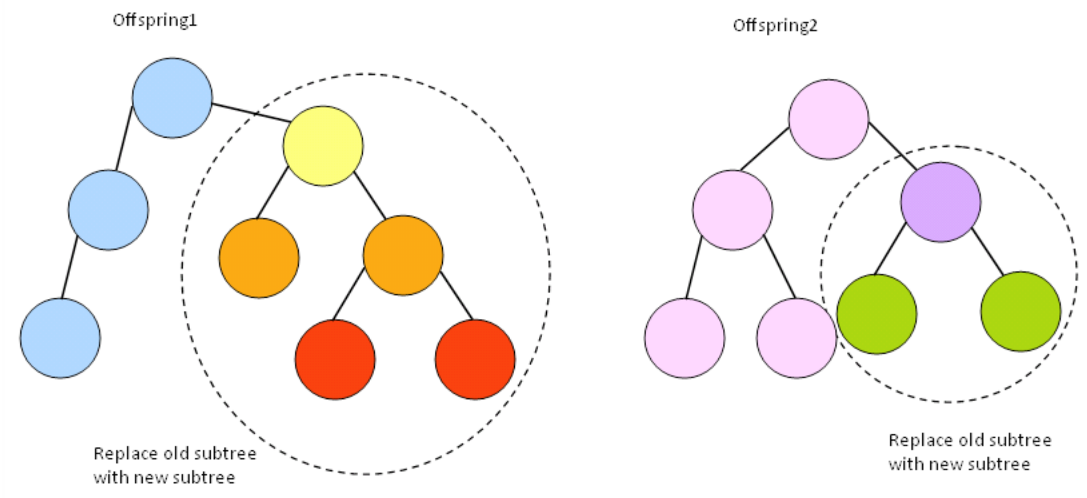

Offspring1
Offspring2
Replace old subtree with new subtree
Replace old subtree with new subtree

Figure 3.7: Two children generated by the crossover operator.

3.4 Summary

In this chapter different machine learning algorithms are explained. We discussed how biologically inspired computing uses the natural biological systems principle to improve the computing results. In order to understand these principles, biological terms such as Individual, Crossover, Reproduction, Mutation, etc were explained. Genetic Algorithm was explained and how it works. Genetic Programming is shown to employ evolutionary algorithm and implement them in the form of computer programs. In order to use evolved-based search, computer programs are composed of functions and terminals which are specific to a problem domain and genetic operators such as reproduction, crossover and mutation are used to produce new individuals which are more capable of meeting the fitness measure.

These definitions and principles are used in this thesis to generate new population of fitter individuals through the evolution-based search. This is in order to create better representatives for documents and to address the user information need.

CHAPTER 4

Related Work

4.1 Introduction

In this chapter we look into the use of Evolutionary Algorithms (EA) mainly in the field of information retrieval and also in artificial intelligence and natural language processing. In particular, genetic programming has gained more attention in recent years and it has grown to many applications in different areas of artificial intelligence, data mining, text mining and natural language processing.

4.2 Data Mining and Knowledge Discovery

[Freitas, 2008] has discussed the use of evolutionary algorithms, particularly genetic algorithms and genetic programming, in data mining and knowledge discovery. This study discusses GA and GP in order to deal with the discovery of classification rules, clustering, attribute selection and attribute construction. The focus of study is on the individual representation and fitness function.

In addition, [Bojarczuk et al., 1999]'s study proposes a genetic programming framework for classification and generalised rule induction. It also discovers some comprehensible rules and claims that the results look promising concerning predictive accuracy. However there are certain issues that the research does not address. The length of the generated rules can be very long and the fitness function does not consider this problem. The preprocessing, which is used in this study, needs improving in order to perform a careful selection of attributes, also the used data set is a small data set focusing on patients chest pain, whereas the result on different and larger data sets may be different.

Textual relations in documents and semantic networks are extracted by genetic programming in the work of [Bergström et al., 2000] to enhance information retrieval and the development of user interface, particularly for small screen devices.

4.3 Natural Language Processing

[Araujo, 2006] has used genetic programming in natural language parsing and tagging. This work has attempted to develop a multi-objective genetic program which performs statistical parsing and tagging simultaneously. Grammar rules and tag sequences are used to train for the best sentence available. Different models (aggregative function, MOGA, NSGA and NSGA-II)[1] have been adapted to these specific problems. The results show improvements for each of these evolutionary multi-objective models, when parsing and tagging are handled separately. The results also show that MOGA outperforms the other methods.

[Araujo, 2007] has studied evolutionary algorithms in statistical natural language processing. This survey has reviewed several works that have applied evolutionary algorithms to different *Natural Language Processing (NLP)* tasks . In most tasks GA has performed with comparable and satisfactory results. The main drawback is the high cost of the computation power and time. Usually the NLP problems have a very large search space, so the GA's execution time is practical only when the cost of fitness function computation is low.

As mentioned earlier, [Bojarczuk et al., 2000] have used genetic programming for knowledge discovery in a specific domain (i.e. chest pain discovery) that involves natural language techniques to extract information.

[1]MOGA is Multi-Objective Genetic Algorithm, NSGA is Non-dominated Sorting Genetic Algorithm, NSA-II is improves NSGA with elitism and a crowded comparison operator that keeps diversity without additional parameters

4.4 Query Expansion

An area of information retrieval, which is being approached by evolutionary algorithms is query expansion and reformulating queries in order to improve the retrieval quality. [Vrajitoru, 1998] introduces a new crossover operation in genetic algorithm specifically for creating new queries.

In [Araujo et al., 2010], an evolutionary algorithm is employed to combine clauses to reformulate a user query in order to improve the results of a similar search. The study has two parts: Firstly, the study starts with a review of the query expansion algorithms and discusses the negative effects of term correlation used in query expansion. A method, which is called query clauses are used to address the problems pointed out caused by term correlation methods. The second part of the study combines the query clauses with genetic algorithm in order to create a method to improve the result of stemming by reformulating the user query. Though this study has ignored the user and pseudo relevance feedback as well as any re-weighting formulas, it has focused on the term dependencies, their occurrences in simple experiments and their effects on performance. An unsupervised method is suggested to note and include simple term dependencies. To reformulate the query the candidate terms are specifically selected from a morphological thesaurus. The fitness function used in this study measures the proximity of selected terms from the query and the top ranked retrieved documents. Different proximity measures such as cosine, square cosine and square-root cosine are used as fitness functions in the unsupervised method.

[Araujo and Pérez-Iglesias, 2010] have proposed a way to train a classifier for query expansion of too short or unspecific queries. In this study, the user's relevance judgements on a document set are used as fitness function for the genetic algorithm to train the classifier to identify distinguished terms for query expansion. The authors conclude that the genetic algorithm training can improve the query expansion quality. The main focus is on a pseudo relevance feedback method and genetic algorithm is used to build a set of suitable terms for query expansion from the top documents of the initial ranked list.

4.5 Weighting Schemes

[Cummins and O'Riordan, 2006] employed genetic programming to find a term weighting scheme for the vector space model. The fitness function to evaluate the individual weighting schemes is average precision and statistics of the terms are derived from two domains: Firstly, term weightings based on each document alone, which is called *local domain*. Secondly, term weightings based on the whole collection, which is called *global domain*. They argue that in their approach terms with low or high frequencies do not affect the weight of the term, also for large collections local weighting does not have a substantial effect. In their experiments, three small document collections (*Medline*, *CISI* and *Cranfield*), one medium sized collection and one large document collection *TREC-9* are used. All the collections are preprocessed using Brown corpus stop-words [Kuĉera and Francis, 1967] and Porter's stemming algorithm [Porter, 1997].

The GP settings in the experiments are: 50 generations, initial population of 1000, tournament selection with the size of 10. Each collection is divided into two sets: train set and test set. The train set is used to train the solutions and the test set to evaluate the weighting schemes. The ramped half and half[2] [Koza, 1992] is used as the creation method of the individuals. As mentioned before, the average precision is used as the fitness function and the traditional $tf\text{-}idf$ and Okapi-BM25 weighting schemes are used for comparison. The Okapi-BM25 model is used with the default parameters of $b = 0.75$ and $k_1 = 1.2$.

The function set for the experiment is consisted of the four main mathematical operators $(+, -, \times, /)$, the natural log, trigonometric functions, square root function and square. The terminals include constants[3], raw term frequency within a document, number of documents a term occurs in, number of documents in the collection, collection frequency, document length, collection size and so on.

Four main experiments were conducted: the first experiment aims to show that on small collections, the general weighting schemes can be evolved into schemes with

[2]Means half of the random individuals must have the maximum tree depth (known as full) while half of them may not have all the branches (known as grow)

[3]For example, 1

better performance than the $tf\text{-}idf$ and even the modern BM25 weighting scheme. The second experiment has used the average precision to estimate the effect of each terminal which has been used in the weighting scheme. Based on the result of this phase, terminals with high advantage will be included in the weighting scheme where as those with low or no effects are ignored. The third and fourth experiments, terminals and their domains, either in document (local) or in collection (global), are combined. The aim is to show that evolving the global weighting with local weighting in a binary method can promote those important terms which support retrieval. Also the local weighting improves while combined with the best global weighting. This helps to evaluate and analyse the worth of the weighting scheme. The formula that has been used for this purpose is:

$$W_t(d_i, q) = \sum_{t \in q \cup d} (lw_t \times gw_t \times qrtf) \tag{4.5.1}$$

where lw_t is the local weight and gw_t is the global weight and the $qrtf$ is the raw term frequency in the query. In this formula the weighting schemes are divided into their relevant parts which shows the significant trait of each one in the final evolved scheme. The novelty of the research is related to the set of text features that they collect, so the advantage of each feature is measured based on their relevancy. The idea of local and global weighting scheme reduces the search space. Also those text features with certain relevance info are identified and those with no specific effect can be eliminated to reduce the search space more.

Similarly, [Fan et al., 2000] use genetic programming to automatically generate term weighting schemes for different contexts. The terminals are very similar to the ones described above, but the functions are limited to $(+, -, \times, /)$ and log. In addition, [Oren, 2002b] applies the same techniques to evolve better weighting methods to outperform $tf.idf$. The principles of both works are very similar to those of [Cummins and O'Riordan, 2006], however the [Cummins and O'Riordan, 2006] provide more analysis into the contribution of each terminal in the overall performance of the weighting schemes and hence more insight into the effectiveness of the approach.

4.6 Retrieval Functions

Since ranking functions are central to the document retrieval problem and genetic programming is well suited to discover complex formulas for different problems, several research projects have been carried out to discover best performing ranking functions and similarity measures for document retrieval.

[Fan et al., 2004b] propose a generic framework to be based on genetic programming to automatically discover and evaluate new ranking functions for different collections and different scenarios. In [Fan et al., 2004a], they investigate the effect of the fitness function in the quality of the discovered ranking function for web retrieval. The authors conclude that the design of the fitness function is very important in the success of the ranking function discovery. They suggest to use a fitness function that considers all the relevant information and also takes into account the order of retrieval of the relevant documents.

In [Yeh et al., 2007] an evolutionary algorithm approach is employed to automatically generate an effective ranking or retrieval function for information retrieval. The learning method, *RankGP*, which has been presented in this research, employs the genetic programming as its engine for finding the best ranking algorithm. It has merged different IR features, such as content features, structure features and query-independent features to learn the ranking function. The proposed learning method, receives two sets of input and one set of output. The first input set contains pairs of queries and documents along with a vector for their features and their relevance judgement. The second input identifies the GP-related parameters, such as number of generations, size of population, crossover rate and mutation rate. The output is a ranking function which is believed to measure the real similarity between a query and a document. The learning procedure is summarised as follows:

1. A set of individuals are randomly initialised for the initial population.
2. The performance of each individual in the initial population is scored on the training set used by a fitness function.

3. The fittest individual from initial population is chosen for the output set.
4. Crossover and mutation are used to generate the individuals for the new generation with the specific population size.
5. These steps are repeated till the initial number of generations is fulfilled.
6. A score function evaluates the performance of each individual in the output set and the best one is chosen as the result.

Each individual is a functional expression of variables, constants and operators, where variables are figurative details for training set's feature, constants are a set of predefined numbers and operators are a set of mathematics operators. The overall representation is a binary tree structure with internal nodes and leaves. The variables and constants are leaves and operations are nodes. Only simple mathematical operations $\{+, -, \times, /\}$ are used as the researchers believe they are sufficient enough to accomplish good results in for this problem, also they have less computational cost. Mean average precision (MAP) has been chosen as the fitness function based on the idea that is a widely used as a retrieval measure. During the evolution process, the initial population is created by the ramp half-and-half method [Koza, 1992]. All three methods of generating individuals are used; reproduction to follow the natural selection principle for elitism, mutation, and crossover, while the tournament selection is used to keep the fitness biased. After the output set is completed, each individual must have a good fitness on the validation set as well as the training set and the final score is calculated by the following equation to take into account the quality on the validation set as well as the main fitness function:

$$Score(I_j) = F(I_j) + MAP(I_j, V) \tag{4.6.1}$$

where I_j stands for an individual, $F(I_j)$ is the fitness of I_j and $MAP(I_j, V)$ donates the MAP of I_j on the validation set.

[Yeh et al., 2007] used an open source GP toolkit (LAGEP) which implements layer architecture of multi-population genetic programming in conjunction with a method for adaptive mutation rate tuning. If all the individuals in the generation have similar fitness values it is appropriate to increase the mutation rate to increase the number of new individuals. Three evaluation measures are used:

- *P@n* (Precision at Position n)
- *MAP* (Mean Average Precision)
- *NDCG* (Normalised Discount Cumulative Gain)

The above evaluation measures, for 5-fold cross validation, are used to evaluate the system's effectiveness. One non-learning-based method BM25 and two learning-based methods, *Ranking SVM* [Herbrich et al., 2000] and *RankBoost* [Freund et al., 2003] are used by them as baselines to compare two evaluation results of the RankGP method. The main problem of this approach is the higher computational cost compared to other learning methods that are used in the experiments, which is the common problem of genetic programming based solutions.

[de Almeida et al., 2007] have tried a genetic programming approach to optimise MAP for a specific collection. In this study, terminals as non-primitive parts of available term-weighting schemes are used. Also complex features from different parts of the term weighting scheme such as term-discrimination, term-frequency and normalisation components are considered. However, due to omitting the primitive measures for terminals there is the danger of ignoring a large area of search space. Also the complex nature of the used terminals has made it difficult to analyse the produced functions. As a result it has become difficult to understand the nature of the ad-hoc retrieval. The learning method in this study is a combination and aggregation of existing retrieval functions. This approach relies on the GP's advantage to produce a symbolic representation of a possible general solution. As a conclusion this study has used the GP abilities and advantages to develop term-weighting schemes, but it has analysed neither the produced solutions nor the ranked lists that are results of each solution.

Another learning to rank investigation based on genetic programming is done by [Trotman, 2005]. The author evaluates the two best performing learned ranking functions by genetic programming to several traditional ranking functions including Okapi BM25 [Robertson et al., 1995], probability measure [Robertson and Jones, 1976; Harman, 1992a], inner product [Witten et al., 1999], Cosine measure [Harman, 1992a] and one proposed by [Oren, 2002a]. Similar to other learning to rank methods, variety of terminals as evidence were considered and the genetic programming process filters those that are effective in enhancing the fitness function, which in this case is mean average precision. The experimental results show that on all collections, the GP learned ranking functions perform at least as well as BM25 and in some cases outperform it by 10%. On the other hand, the generated ranking functions are quite complicated and difficult to understand, which is a drawback in using them on new collections.

4.7 Classification

Genetic programming has been used to improve all the aspect of classification, from preprocessing and inducing classifiers to combining classifiers. For the preprocessing stage of classification, genetic programming is used to induce new features, select the features or weight the features. In [Muharram and Smith, 2005] four fitness function based on common feature selection methods are designed to produce new features to later be used by decision tree and perceptron classifiers.

[Koza, 1991; Folino et al., 1999] is generating decision trees for classification by using the classification accuracy as the fitness function for genetic programming and the evolved decision trees are used to classify new documents. Other researches have combined the classification accuracy with tree size to produce smaller trees [Kuo et al., 2007].

Rule-based classifiers are also producible by genetic programming. For example, in a binary rule-based classifier, the rule returns true or false for each document against each class. [De Falco et al., 2001] use genetic programming to discover rules for classification. The fitness function consists of a component that estimates the accuracy of the classifier by considering both true positives and false positives and also a component that forces the evolution process to produce simpler individuals. The functions are logical operators {AND, OR, NOT } and relational operators $\{<, \leq, =, \geq, >\}$. Each function allows three arguments, which include constants and attributes of the domain of the problem. Discriminant functions are another way of performing classification, which can be learned using genetic programming. [Zhang et al., 2005] compares the performance of classification by the discriminant functions learned using genetic programming to two successful statistical classification techniques, SVM and artificial neural networks and report equal or better performance. In addition, they found that the training times are also substantially lower for the genetic programming based method, which is very important since computation time is one of the major drawbacks of the genetic programming based techniques.

Finally, genetic programming can also be used to combine the predictions of other classifiers and produce a final prediction. The base classifiers can be based on genetic programming [Imamura et al., 2003] or can be other methods of classification [Langdon et al., 2002].

A thorough and comprehensive survey of all the applications of genetic programming in classification is presented in [Espejo et al., 2010].

4.8 Summary

In this chapter, we overviewed the use of evolutionary algorithms in information retrieval, natural language processing and data mining and knowledge discovery. Similar to many other machine learning techniques, evolutionary algorithms and specifically genetic programming can be applied to variety of problems and it can be effective in many domains.

Most of the previous research of using genetic programming in information retrieval are dealing with query expansion, learning ranking functions or weighting schemes. The main issue in most of the studies is reported to be the computation cost of genetic programming. Therefore, the proposed methods tried to deal with the this issue by simplifying the structure of the individuals or choosing a fitness function that is easy and fast to compute. Although genetic programming has been shown to be effective in finding high quality ranking functions, there is a drawback in using them for this problem. The evolved ranking function, even though might use simple functions, can be very complicated to understand and justify and this makes the traditional retrieval models more appealing in using on new collections and domains.

In the next chapters, we try to address both problems in our proposed approach. One of the main aims of this work is producing readable representatives for sets of documents. In Chapter 6, we present functions and terminals for the structure of the genetic programming trees that produce readable and easy to understand individuals. In addition, we add a component to the fitness function that punishes the individuals with more functions over those with less. In regarding the computational cost of the genetic programming, we focus on improving the speed of the fitness computation. In our implementation, we use a high performance indexing system for calculating the similarities between the representatives and the documents. Additionally, the representatives are used as logical expressions to exclude most of the documents from the similarity computation process.

CHAPTER 5

Classification by Genetic Programming

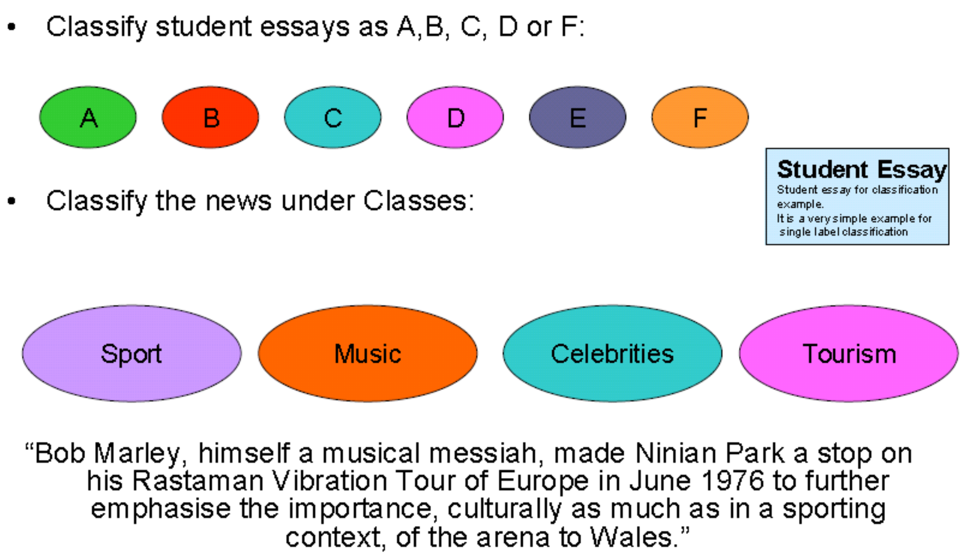
5.1 Introduction

So far, we have introduced information retrieval and some challenges for large data collections. Chapter 2 discussed some of the approaches of representing documents and calculating similarities. This chapter introduces the problem of classification in information retrieval and proposes a method based on *Genetic Programming (GP)* to solve the problem.

In this chapter we will look at the fundamental steps to solve the document classification problem step by step and discuss how we address each step in our method. Then we will introduce the bases of our approach based on GP and explain the nature of the rules or the class representatives in our work. Finally, we discuss the evaluation of classification methods and report the results of our baseline on a standard data set.

5.2 Classification

Classification can be defined as mapping the documents $\{\mathbf{d} \in \mathcal{D}\}$ to one or multiple classes $\{c \in \mathcal{C}\}$. Figure 5.1 shows an example of a classification problem. There is a set of predefined classes, when a new document arrives, then the task is to find the most



- Classify student essays as A,B, C, D or F:
- Classify the news under Classes:

Student essay for classification example.
It is a very simple example for single label classification

"Bob Marley, himself a musical messiah, made Ninian Park a stop on his Rastaman Vibration Tour of Europe in June 1976 to further emphasise the importance, culturally as much as in a sporting context, of the arena to Wales."

Figure 5.1: Classification

relevant class or classes for the new document.

Classification can be performed by many different methods and approaches. In this thesis a machine learning approach called genetic programming (GP) technique is used to do the classification. Evolutionary based techniques and particularly GP, try to solve complex problems such as classification by using the idea and some of the terminologies of biological evolution. Similar to other machine learning methods, there is a collection of training documents for each predefined class. A *rule* which is a representative of that class is built based on the training set. The rule is learnt or chosen from a practically infinite number of possible rules based on its quality to classify the training data. The selected rule for each class is called the *representative* for that class. A representative is a string that summarises the content of the class it represents. By investigating the representative, instead of looking at each document in the class, we get an idea of the topic of the class. Clearly, the rule must be able to filter future unseen documents to be classified under the class it represents.

To classify new documents, we use the representative of each class to play the role of the centroid of the class and estimate the relevance of the new document with respect to the class. When a new document arrives, the similarity between the new document and the representative of the class is calculated. Then the new document is labelled by the class which, for the new document, has the most similar representative.

The basic idea of the GP approach is to generate human-readable rules to perform the classification. For each class a rule is learned based on the training data provided for that class and will be used to predict the category of the future documents. Similar to any other classification method, this method has different parts and algorithms that cooperate with each other to constitute the whole method.

Training documents go through the pre-process stages as described before (see Section 2.3). Different rules are learnt to represent the documents and classes, based on their quality to classify the training data the best one is chosen as the classifier. When a new document arrives, the similarity between the new document and each class representative is calculated, then the new document is labelled by the class which has the most similar representative to the new document.

It is a very straightforward process to compare the new document with the whole class and decide about the new document's label, however it is not practical, where the size of classes or the number of new documents are high. It is not only very expensive but also very time consuming, it is neither cost effective and nor time effective. To deal with this problem, instead of comparing the new document with each one of the documents in the class, the class representative can be used.

A class representative includes the discriminant terms which are useful in finding relevant documents to that class. All the terms that may be common between the new document and the class are considered. This increases the similarity measures. A good representative expresses the best view of the class, where the complexity and the volume of similarity calculation decreases. To find the representative for a specific class, we start with training data. Different features form these training data are extracted. The genetic programming search strategies help to find the best possible rule which represents the class. These required steps are: Extract discriminative features, Combine extracted feature to construct a rule and evaluate the candidate solutions against the training data.

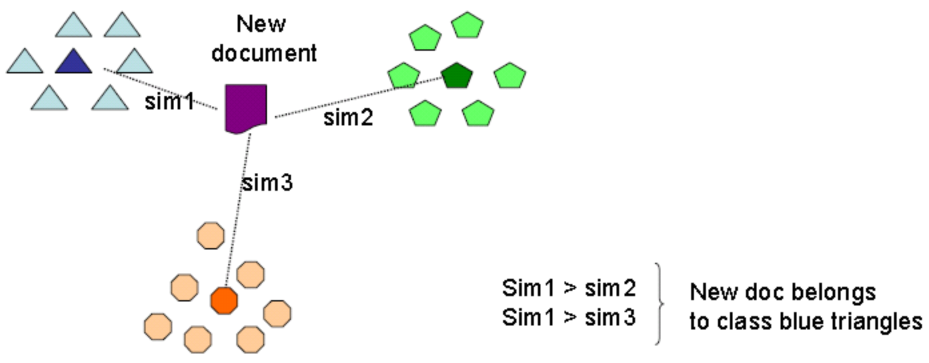


Figure 5.2: GP Classification

Figure 5.2 demonstrates a scenario where there are three different classes, triangles, pentagons and octagons. The class representative for each of them is generated. When a new document arrives, it is compared to each class representative and similarities between them are calculated. Based on the calculated similarities the new document is labelled to the class that is most similar.

5.3 Main Steps in Classification

Classification can be performed by lots of different techniques, however most of the approaches consist of the following main steps:

- Document indexing
- Feature selection
- Choosing a classifier
- Training and testing the classifier
- Evaluating the effectiveness of the classifier

Document indexing prepares the documents by transforming them from raw text to a more comprehensible presentation for the system. During document indexing all documents go through the pre-processing which has been explained in Section 2.3.1. These pre-processing steps remove all the unnecessary parts by applying stop-words removal, stemming and normalisation. For each document a list of terms along with some of their attributes such as their frequencies are generated.

These terms carry different information about the document they are extracted from. Therefore it is important to select those terms which represent the documents better, terms with discriminant power. In our approach, selecting the features and combining them to construct a class representative for a particular class is done simultaneously. While GP selects different terms to generate the tree-shape individuals, it tends to select the most discriminant terms to distinguish relevant documents from non-relevant ones. In addition, the document representative that is generated will be used to make classification decisions.

A similarity measure based on the retrieval functions introduced in Chapter 2 will be chosen to be the classifier that estimates the similarity between the documents and the class representative. The similarity score is used to decide about the classification results of the documents.

For training the classifier, which is optimising the class representative, a fitness functions is used to score the individuals, with respect to their ability to classify the training documents under the correct category. The final step is evaluating the classifier by classifying the documents in the test collection and estimating the classification metrics described in Section 2.7.

In the next sections, we describe the above steps in more detail and sketch the overall picture of the classification performed by our GP-based approach.

5.4 Document Indexing

All words do not have equal value to represent the semantic of a text, some words are more significant than others. In documents in traditional format all words (terms) are encoded in so many different ways that a classifier or classifier-building algorithm can not interpret them. To make a document ready for the classifier some preprocessing is required (see Section 2.3). These preprocessing steps not only generate a set of terms, as an index of discriminant terms in the document, but also reduce the size of the search space that the classifier is dealing with, therefore less time is spent for non-discriminant terms.

While the number of terms increases the model becomes more complex and the number of parameters that algorithm needs to include and deal with goes up. Apart from the speed issues that the huge number of parameters causes, unnecessary terms with no discrimination value introduce noise to the process. The aim of this step is to represent each document with a vector of features (such as words) to simplify the next steps of the algorithm. However, the loss of discriminate terms can be very costly regarding the loss of indicative features in the representative vector[Moravec, 2005; Barresi et al., 2008; Nouri and Littman, 2010].

5.5 Feature Selection

The classification task requires a specific manner regarding the time and the cost and especially the importance of features with significant values, therefore a subset of existing terms is selected. This process is known as *feature selection*. During the feature selection, the size of available vocabulary decreases which improves the time and cost efficiency.

During the training process, there is another problem or error which may occur and end up building a classifier with low performance over unseen data. This can happen where too much unnecessary detail about the training set is collected and entered in the document representatives, especially when irrelevant terms are collected by accident, instead of fundamental and discriminant terms. For instance consider a very rare term such as "arachnocentric" which has no information about a specific class such as sport, repeated quite frequently in training documents and is associated with one of the classes by mistake. When the classifier is applied to test (unseen) data, this association can decrease and damage the classifier accuracy. In text classification process, when a classifier has a very high accuracy over the training set while it performs poorly over the test set, this shows that a classifier has collected superfluous information about training set and is known as *overfitting*.

The main intention of feature selection is to reduce or prevent these errors occurring. [Parr et al., 2008; Yan et al., 2009]

5.5.1 Class Representative

The main part of our approach to classification by GP is to learn a set of rules in which each one represents one of the predefined classes in the class collection $\mathcal{C}$. The GP learning algorithm receives a set of features as input and combines them in a way that represent a particular class. The genetic programming algorithm searches (see Chapter 3) among a large set of candidate solutions to find the fittest among them. In our problem, the fitness is computed against the training data. In other words, the performance of the candidate solution on the training data determines its fitness and hence the possibility of its selection as the final representation of the class.

Definition 1. ***Representative*** *is a string of different features which are logically related to each other and represents a particular class.*

Algorithm 2 GP based algorithm to generate the representative of a class.

```
Require: class $c$ and the training data set
Require: parameters of GP such as: $popSize$, operators rate, ...
1:  $P \leftarrow$ randomly generate an initial population
2:  repeat
3:     for all $p_i \in P$ do
4:        assessFitness($p_i$)
5:        $best \leftarrow$ findNewBest($best$, $P$)
6:     end for
7:     $P' \leftarrow \{\}$
8:     while $|P'| < popSize$ do
9:        $c \leftarrow$ generateNewChild($P$)
10:       $P' \leftarrow P' \cup \{c\}$
11:    end while
12:    $P \leftarrow P'$
13: until termination criteria met
14: return $best$
```

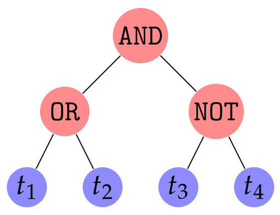
First we need to define the structure of this string and the components that compose it then, we need a method to evaluate each generated string to find the fittest one.

Algorithm 2 sketches the main process of generating a representative for a class. Firstly, an initial population is generated randomly based on the settings of the genetic programming engine. Then, the quality of each candidate solution, which is called fitness of the individual, is calculated. The fitness for a class representative can be described as a measure of how good it represents the corresponding class. A representative that finds most of the relevant documents and very least of non-relevant documents of the future documents is ideal.

If such a representative is generated then the termination condition is met. Where the termination condition is not fulfilled, the reproduction methods are used to generate a new population with new individuals and the process will be repeated until an ideal representative is found, or we meet the termination condition. To implement this algorithm, we need to define the following subroutines:

```
assessFitness, findNewBest, generateNewChild
```

First we describe the structure of the individuals or candidate representatives, so we can discuss estimating their fitness and generating new generations later.

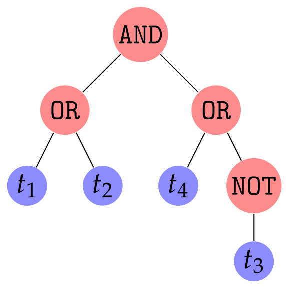


$(t_1$ OR $t_2)$ AND $(t_3$ NOT$(t_4))$

Figure 5.3: Tree-shaped structure

5.5.2 Terminal and Function Nodes

The representation of individuals in genetic programming is discussed in Chapter 3. A tree shaped structure has *functions* represented by non-leaf nodes and *terminals* represented by leaf nodes, since we want to generate a representative that can be used to select and de-select documents. A Boolean expression consisting of Boolean operators and terms extracted from the collection's vocabulary is used to shape the structure. Figure 5.3 shows an example of such a tree.



$(t_1$ `OR` $t_2)$ `AND` $(t_4$ `OR` `NOT`$(t_3))$

Figure 5.4: An example of the structure of the class representatives. Functions are `AND`, `OR` and `NOT`, terminals are t_1, t_2, t_3 and t_4.

Non-leaf nodes which are called functions include:

- `AND` joins its children and implies that all of them must be present in the document to satisfy the condition of the representative.
- `OR` joins its children and means either of the branches can be present in the document to satisfy the representative.
- `NOT` is a one child operand and enforces the absence of the child from the document.

Terminals are terms provided as input to the learning algorithm. We will discuss about the selection and types of the terms in the next section. To be able to verify the structure of the tree and make sure we build valid trees, we use *strongly-typed genetic programming* (see Section 3.3.3). Another benefit, is tuning the usage of each type to guide the learning algorithm in finding better class representatives. Strongly-typed genetic programming enforces the random generation and the genetic operators to produce valid individuals according to the type declarations. Table 5.1 defines each function along with the type of operand, the number of terminals and the type of children which are accepted by the function.

	Operand type	Number of children	Type of children
`AND`	`Query`	2	`Query`
`OR`	`Query`	2	`Query`
`NOT`	`Query`	1	`Term`

Table 5.1: Type of the functions and their accepted children

A `Query` type is a tree or subtree which is accepted as a representative. `Term` is the type of the terms and is also a `Query`, therefore, even one single term is acceptable as a representative. In general, `AND` and `OR` accept a subtree which is a `Query` or a term, however, `NOT` only accepts terms as their child. This is to avoid the generation of very complex trees that have a whole branch under a `NOT` operand. Since, these representatives are used to filter and score text documents, combining `NOT`s with complex subtrees is not going to improve the way we select the documents.

5.5.3 Positive and Negative Terms

The aim of a class representative is to return as many relevant documents and least number of non-relevant documents. During training the classifier, to create the class representative, choosing appropriate terms is very important. Apart from the structure of the tree, the identity of each term in the leaf nodes determines the performance of a particular representative and in total our classifier. Considering documents in their traditional format, the system faces a very vast number of terms with different features (or parameters). This huge space ends up in a very long and difficult search which results very poorly. To overcome this problem the collection goes under a series of processes known as *pre-processing* and *filtering*, which consist of stop-word removal, stemming, etc (see Section 2.3). Pre-processing reduces the size of search space drastically, however it is a very sensitive and wise task to choose right terms, that specifically distinguish the relevant documents.

On the other hand, while it is important to pick discriminant positive terms to increase the representative performance and quality, it is equally crucial not to include those terms which attract non-relevant documents. While positive terms are supposed to expose relevant documents, negative terms are discriminant to return those documents that are not relevant. As non-relevant documents are not desired to be returned as results, these negative terms are used in the form of negative operands for negative operators in the learning algorithm. Their presence as negative operands means documents that contain these terms, with high discrimination power, are not desired and should not be considered as results.

We classify these terms, that are collected from the labelled documents provided as the training data, into two categories: Terms such that their presence indicates the relevancy of that document to the class and those where their absence does so. Therefore, we define these two important concepts as:

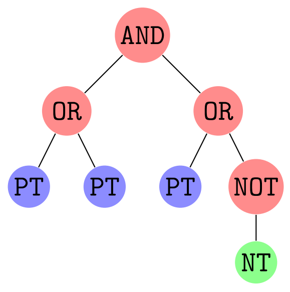
Definition 2. *A **positive term** is a term such that its occurrence in the class representative guides the classifier to select documents that include the term. The **positive list** of a class is a sorted list of positive terms that have been selected to be used in the learning algorithm.*

Definition 3. *A **negative term** is a term such that its occurrence in the class representative guides the classifier to avoid selecting the documents that include the term. The **negative list** of a class is a sorted list of negative terms that have been selected to be used in the learning algorithm.*

5.5.4 Selection of the Terms

As mentioned previously, we are using strongly-typed genetic programming (see Section 5.5.2) to control the structure of the class representatives generated by the learning algorithm. Using typed genetic programming requires identifying the complete relationships between parents and children, the number of children for each parent, the type of children each parent can accept. We explained the type of the functions and mentioned there are two types of nodes as terminals: positive terms and negative terms. Each tree includes operators with leaf nodes chosen from positive or negative lists according to the type of operator. For an example, consider Figure 5.5.

There are three operators `AND`, `OR` and `NOT`. Operator `AND` accepts two children, which in this example has accepted another two operators as its operands. It can also accept



(PT OR PT) AND (PT OR NOT(NT))

Figure 5.5: An example of using positive terms PT and negative term NT.

positive terms as its operands. Operator OR operates exactly like operator AND, it accepts other function nodes as well as positive terms. Operator NOT is different. It only accepts one negative term as its child. Operator NOT does not take other functions as its children. We have not used operators such as NAND and NOR for two reasons. Firstly, there is no logical expression which contains NAND or NOR that can not be rewritten using only NOT. Secondly, even though NAND or NOR are useful in other contexts, for our purpose readability is negatively affected while no noticeable improvements are gained as the result. Therefore, we argue that a simpler operator such as NOT is capable of expressing the requirements of the representatives, without hurting their readability.

Obviously a symbolic tree has got no use until these terminal nodes, PTs on one hand and NT on the other hand, are replaced by actual terms from positive and negative term lists. This symbolic tree structure is a kind of a pattern and not a unique class representative, unless its instances replace the real positive and negative terms instead of terminal nodes and turn it to a unique representative of a specific class.

Term selection for the terminal nodes is a very essential part of the training process. For each terminal node, we generate a random number that indicates the index of the term in the positive or negative list. Every individual in the population must be evaluated to produce a logical string so its fitness can be calculated. Evaluating a tree is a recursive top-down approach. The root node is asked to evaluate itself. It asks its children to evaluate themselves and combines the result. This process recursively visits all the nodes in the tree until it reaches the leaf nodes that do not have children. As described before, the leaf nodes are term nodes. To evaluate a term node a random number is generated that corresponds to a term in the lists. Each random number indicates the index of a term in the positive or negative lists. In other words, for each terminal node that represents a positive or negative term, a random number is generated to replace the terminal with an actual term from the positive or negative lists.

In previous work, [Hirsch, 2005], the index is generated by defining extra terminals and functions, including addition, multiplication and constant numbers between 0 and 9. Each terminal node is a subtree of an arithmetic operation that can be evaluated to return a number indicating the index of the term. This approach has a practical benefit, which is essential. If a tree is evaluated and its terminal nodes are assigned numeric values and the tree or a part of it that contains a terminal is evaluated again (for example in the next generation), the terminal must return the same numeric value as before. By using the subtree of arithmetic operators, it is guaranteed that terminals return unique values anytime they are evaluated. However, there are a few disadvantages in this approach. Firstly, it is more expensive to calculate a subtree than generating a random number. Secondly, having more terminals and functions makes the trees more complex without adding more complex representatives to the search space. The subtrees that find the indices are not part of the representative itself. Therefore, we might have two completely different trees with the same representative after evaluation.

To overcome these issues and keeping the terminals unique across generations, we use a technique called *Ephemeral Random Constants*[1]. An ERC node is a node that is initialised for the first time, for example by some random number, and always returns the initial value from then on. The ERC node returns the initial value even if it is crossed over into other individuals. With this approach, we not only keep the initial values of the terminal nodes, we avoid the use of arithmetic subtrees and their complexity.

When a tree-shaped structure becomes a unique and specified representative then its quality or *fitness* can be calculated regarding the required class. There might be many different trees such as Figure 5.5 in the genetic programming search space, but as long as the terminal nodes are replaced with different terms from the positive and negative

[1]Its use in genetic programming first described in [Koza, 1992].

list, they will make legitimate candidate solutions.

Algorithm 2 requires a collection of positive terms for each given class as input. During the tree-shape individual generation, one term is selected randomly for each positive terminal node `PT` from positive term list. Equally, there is a collection of negative terms for the given class, those terms that are not desired to appear as positive features of the class representative, as they tend to return non-relevant documents. When positive terminal nodes and negative terminal nodes are replaced by real terms from the provided lists, each instance of the structure becomes a unique and identifiable candidate representative for the given class.

To build the lists, the only source of evidence that we have is the training data. In the training data, there are relevant documents for each class and the rest of the documents in the training set are considered to be non-relevant to that class. Since all the training documents have been processed and indexed, we can easily collect all the terms in the relevant documents and sort them according to a specific measure. Although sorting the terms is not a must do, it has two advantages. Firstly, in large collections with a large vocabulary, we might need to discard some of the terms to be excluded, from being used in the learning algorithm. Having a measure helps us to decide which terms are more likely to be useful in discriminating between documents. Secondly, while we choose terms to replace the `PT` nodes in the trees, we might want to select more discriminant terms more than others to shape the tree.

Term frequency has been used in many information retrieval problems as a measure of estimating the discriminant power of terms [Salton and Buckley, 1988; van Rijsbergen, 1979; Nigam et al., 1999]. All the terms in the relevant documents are collected and sorted according to their accumulated term frequency within the relevant documents, which are called TF_R. Depending on the collection size, a threshold will be defined and all the terms with TF_R higher than the defined threshold are added to the positive terms list of the given class. Another way to discard the terms will be by defining a threshold based on the size of the list. In other words, we select the top k terms collected from the relevant documents and discard the rest of them.

Applying the same approach for the negative list is not practical, because the number of terms in the non-relevant documents are very high, recalling that all the other documents in the collection[2] are non-relevant to the class that we are training. Additionally, high value of term frequency within non-relevant documents is not necessarily a good measure to consider a term as a negative feature.

Considering the definition of positive term for a class, we can deduce that a positive term for class c can be a negative term for class c', if it is not already in its positive list. If a term does not retrieve relevant documents for class c', it retrieves documents belonging to another class, we need to add it to the negative list of class c', in order to reject the non-relevant documents which belong to class c.

5.5.5 Choosing a Classifier

Each classification approach needs a classifier at some point in order to separate the relevant and non-relevant documents. There are different types of classifiers based on their complexity. Here, we briefly introduce two types of classifiers: *linear* and *nonlinear* classifiers which are commonly used in different approaches of classification. Later in this section, examples of both types are provided to clarify the application of classifiers in classification.

In order to look at classifiers in a more tangible way, only binary classifiers are considered. The functional form $w_1x_1 + w_2x_2 = b$ represents a linear classifier where the document set should be divided into two classes c and $\bar{c}$. In this formula, (x_1, x_2) is the vector representative of document $\mathbf{d}$, w_i is the parameter vector which in conjunction with b represent the *decision boundaries*. The document $\mathbf{d}$ belongs to c if $w_1x_1 + w_2x_2 > b$ and to $\bar{c}$ if $w_1x_1 + w_2x_2 \leq b$. So in fact a linear classifier is a line between documents that belong to two specified classes. The linear classifiers seem easy to implement at the first glance, as they do not have a very complicated formula and lots of parameters, however there are some major problems with linear classifiers.

Figure 5.6 shows an example of a linear classifier that divides two sets of documents. The line indicates a decision boundary to determine the class of each document. The

[2]We may refer to the training set as the collection while we discuss the training phase.

first difficulty is related to the training methods for the linear classifiers. How to determine the parameters w_i and b based on the training set (seen data), where the quality of learning method, which is what we are looking for, is evaluated by its result on unseen data [Duda et al., 2000].

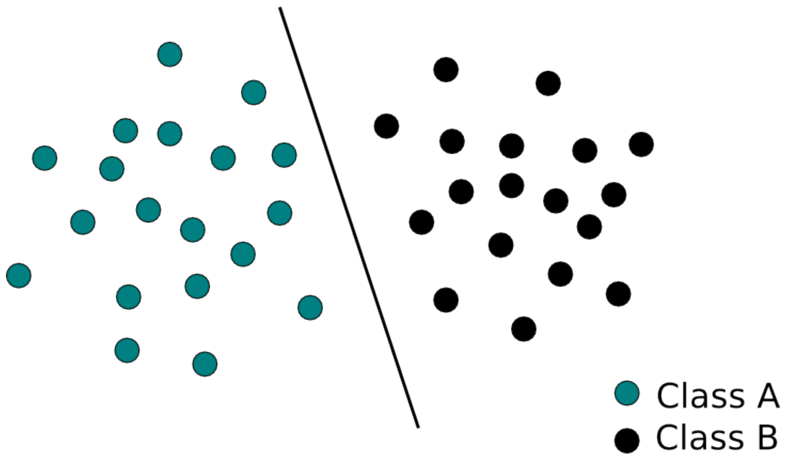


Figure 5.6: An example of a linear classifier. The line indicates the boundary between the classes. The future documents will be classified based on their position with respect to the line.

The second difficulty concerns *noise documents*. A noise document is a document that does not match the general distribution of classes and has misleading effects on the classification process. All features from noise documents are known as *noise features* [Manning et al., 2008]. So when a noise document exists in the training set it increases the classification error and when a noise feature appears in the document representation it increases the average of classification error.

The linear classifiers is a line to separate documents, therefore there is an infinite number of lines that can represent the classifier and it is a big challenge to choose a suitable training method for linear classifiers. Naive Bayes [Zhang, 2004] and Rocchio [Carter, 2007] are two very well-known linear classifiers.

Nonlinear classifiers are used where the distribution of documents is more complex and does not fit in the linear classifiers' boundaries. They are more complicated and require more parameters. One of the popular nonlinear classifiers is *kNN*, which is highly accurate for those problems that can not be classified with linear classifiers. However,

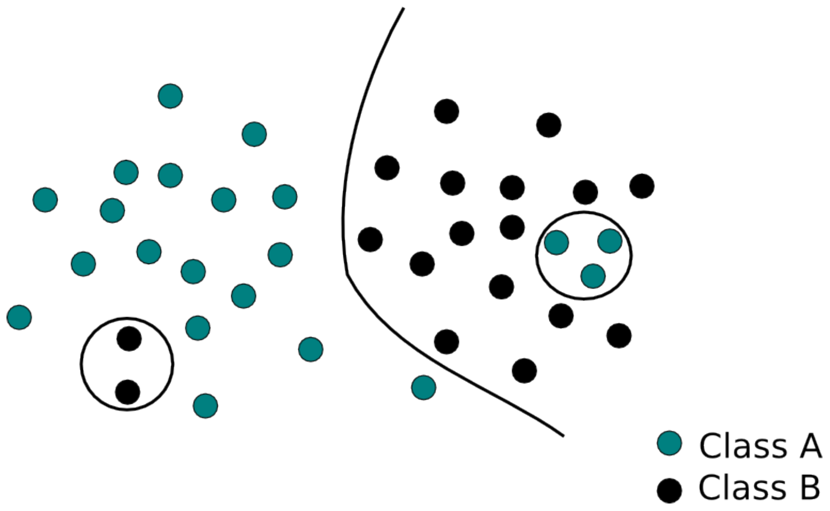


Figure 5.7: An example of a nonlinear classifier. The distribution of the documents are too complex for a line to properly discriminate them, therefore a more complex model is needed to classify these documents.

nonlinear classifiers require a large training set, which can be considered as one of the challenges for this approach. Figure 5.7 shows an example of a nonlinear classifier.

To clarify the distinction between the classifiers and also explain some of the techniques which are directly related to our proposed approach, here we shall describe two different classifiers, namely Naive Bayes and *k*NN.

Naive Bayes

Naive Bayes is a probabilistic model based on the Bayes's theorem. It is called "naive", because of the strong assumption of independence of features. Assume document **d** is represented by a feature vector $(x_1, x_2, \ldots, x_n)$ and we want to find the most probable $c \in \mathcal{C}$ given these features:

$$P(c|\mathbf{d}) = P(c|x_1, x_2, \ldots, x_n) \tag{5.5.1}$$

$$= \frac{P(c)P(x_1, x_2, \ldots, x_n|c)}{P(x_1, x_2, \ldots, x_n)} \tag{5.5.2}$$

$$= \frac{1}{Z} P(c) \prod_{i=1}^{n} P(x_i|c) \tag{5.5.3}$$

in which, $Z = P(x_1, x_2, \ldots, x_n)$ is a constant for all $c \in \mathcal{C}$ and does not change the assignment of the classes. Therefore, the decision rule is:

$$\arg\max_{c} P(c) \prod_{i=1}^{n} P(x_i|c) \tag{5.5.4}$$

For example, in case of two classes c and $\bar{c}$, $\mathbf{d}$ is classified under the class c if and only if:

$$classify_b(d) = \frac{P(c|\mathbf{d})}{P(\bar{c}|\mathbf{d})} \geqslant 1 \tag{5.5.5}$$

where $classify_b$ is called Bayesian classifier. In spite of its simplicity and the independence assumption, Naive Bayes is one of the most effective classification algorithms in many real world applications [Zhang, 2004].

k-NN k-Nearest Neighbour

In a kNN classifier for a given document d the k closest neighbours are considered and the document will be assigned to the class which contains the majority of those neighbour, while k is a parameter. The rational is related to "contiguity hypothesis", where a test document d is expected to belong to the same class that majority of training documents surrounding it belong to.

In 1NN the closest neighbour is considered and the new document is labelled by that document label. However, 1NN is not a valid decision as the classification decision is dependent on one single training document, which can be mislabelled or be an unusual example. kNN where $k > 1$ is more robust, it relies on more training documents, so more samples reduce the probability of mislabelled document or unusual samples.

There are two main versions of kNN classification algorithm:

The first one is probabilistic version where the probability of one document d to be labelled by a class c based on the proportion of its k closest neighbours is estimated.

The cosine similarity version is the second approach, where a document's score of belonging to one class c is calculated as:

$$score(c,d) = \sum_{d' \in S_k(d)} I_c(d') cos(\overrightarrow{v}(d'), \overrightarrow{v}(d))$$

$S_k(d)$ is the set of documents that are the d' k nearest neighbours, and $I_c(d')$ will be 1 if document d' belongs to class c and 0 if not. The document with the highest score is assigned to the class. This approach is more accurate, there may be more than one class with an equal number of the majority of the nearest neighbours in the top k. In this case the class with more similar neighbours will be the selected one.

Training a kNN classifier is a two-step process, the first step is determining k and the second step is preprocessing documents (see Section 2.3.1). As discussed before, it makes more sense to do all the preprocessing once at the beginning for all the training documents instead of repeating it each time a new test document needs to be classified. Regardless of the number of classes, the kNN classifier computes the distance between the new document and all the training documents. In the next step, the classifier deals with the k nearest documents to the new document, instead of including all classes and their documents, for deciding about the class of the new document.

Test time for kNN is calculated by $\Theta(|D| M_{ave} M_a)$. In a simple description kNN memorises all the training documents and compares them with each new test document, that is why it is also known as *memory-based learning* or *instance-based learning*. In machine learning, it is desirable to have as big a training set as possible, however large training sets cause the kNN classifier to have a *severe efficiency penalty*.

There are two approaches that may improve the kNN testing efficiency. There are fast *k*NN algorithms that deal with small dimensionality M. For large M, there are approximations that give error bounds to gain specific efficiency. There has been little testing done to prove that approximation efficiency is better that $\Theta(|D|)$ without a great accuracy lost.

It has been claimed [Manning et al., 2008], that the two problems of finding the nearest neighbours for one test document and *ad hoc* retrieval are both *k*NN problems with different densities. *Inverted index* is very efficient for ad hoc retrieval, so it can be applied to *k*NN as a powerful tool as well. In *k*NN, inverted index can be efficient if the test document does not share too many terms with many of the training documents.

*k*NN is one of the most accurate classifiers among learning methods in text classification [Joachims, 2002]. *Bayes error rate* is a test that measures the quality of a learning method. For a specific problem, it measures the average error rate of the classifier. *k*NN does not perform well for problems with a non-zero Bayes error rate. Considering the error rate for 1NN tends to double the Bayes error rate, so if the optimal classifier's error rate is shown by x then the error rate for 1NN is going to be $2x$. For those classifiers with a zero error rate, the error rate for most of the number of k tends to be zero despite the increase of the training set's size.

5.6 Training and Testing the Classifier

The aim of text classification is to classify all documents $\mathbf{d}$ in a document set or document space $\mathcal{D}$, under classes $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$, which may be called labels or categories. The document set is a high-dimensional space, while the classes are defined by users for their specific application.

To build a classifier, there is a *training set* D, which is a set of manually labelled documents $(\mathbf{d}, c)$ by human specialists, where $(\mathbf{d}, c) \in \mathcal{D} \times \mathcal{C}$. Each training set includes some typical samples for each class.

Recalling Section 3.2 in Chapter 3, the process of mapping documents to classes is done by a *learning method*. The main family of models that is used to deal with classification problems is called *supervised models*. Each supervised learning method takes a training set as the input and returns a learned classification function or simply a *classifier*. A supervised learning model needs a set of training data to infer a model that is applied to the test data. Another family of models is *unsupervised models* that do not rely on the training data to infer the model and the learning and the application of the model rely on the test data.

Once the learned classification function is ready, it should be applied to a new and unseen set of data, known as *test set*, to map them to the classes and finally examine the overall performance. The goal in text classification is high accuracy on new and unseen documents that are in the test set.

Sometimes before evaluating the learning process on the test set, another set called *validation set* is used. It divides the original training set into two separated sets: the new training set D_{tr}, which is a smaller subset of the original training set D, and the validation set D_{val}, which is the remaining documents of the original training set, this means $\{D_{tr} \cup D_{val}\} = D$. The learning method uses the new training set to produce the learned classification function and validation set to evaluate the accuracy of the classifier. Although the validation set is a subset of the original training set that is labelled by the user, therefore it is known to the learning process supervisor, it is not used in the training process so can be considered as unseen data to the learning method to evaluate the learning accuracy.

5.6.1 Calculating the Fitness

An important component of the learning algorithm is calculating the fitness of each individual in order to not only find the best individual, but also to have a measurement in order to select parents for breeding. A good class representative is the one that retrieves all the relevant documents and no non-relevant document in the test set. Since, in real world problems, we do not know the actual relevant and non-relevant documents in our test set, we evaluate the fitness of our individuals on the training set with the assumption that a good class representative on the training set is very likely to be a good class representative on the test set[3]. Algorithm 3, shows the fitness calculation procedure.

Algorithm 3 Calculating the fitness for an individual

```
Require: class $c$ and the training data set $D$
Require: individual $p$
1: $R_c \leftarrow$ documents belonging to $c$ in $D$
2: $Q \leftarrow \{\}$ {predicted relevant documents will be added to $Q$}
3: for all $d_i \in D$ do
4:    if $d_i$ satisfies the logical expression $p$ then
5:       $Q \leftarrow Q \cup d_i$
6:    end if
7: end for
8: $fitness \leftarrow$ computeMeasure($R_c$,$Q$) {computes the evaluation measure, based on the
   retrieved documents and relevant documents for the given class}
9: return $fitness$
```

For each individual, all the documents that satisfy the logical requirements of the individual are collected and labelled as relevant. Then, they are compared to the actual relevant documents of the given class and a score is computed as the fitness of the individual. To compute the score, we need to examine different measures of evaluating classifiers in general.

5.6.2 Evaluating a Classifier

In Chapter 2 Section 2.7, we discussed different approaches of evaluating an information retrieval system. In general, the same concepts are applied to measure the quality of a classification system. Similar to document retrieval, we can define precision and recall for each of the results of each class. Considering the values of TP_i, FP_i, FN_i and TN_i according to the Table 2.1, precision and recall for category $c_i \in \mathcal{C}$ are defined as:

$$P_i = \frac{TP_i}{TP_i + FP_i} \tag{5.6.1}$$

$$R_i = \frac{TP_i}{TP_i + FN_i} \tag{5.6.2}$$

[3]Of course, this assumption requires that the training set and test set are from one domain and are not very different in nature.

where P_i is the precision for c_i and R_i is the recall for c_i. Since, these measures are estimated with respect to one class, we can employ two different methods to estimate the overall precision and recall of the results:

- *microaveraging* is estimated by summing over all the individual judgements:

$$P = \frac{\sum_{i=1}^{|\mathcal{C}|} TP_i}{\sum_{i=1}^{|\mathcal{C}|} TP_i + FP_i} \tag{5.6.3}$$

$$R = \frac{\sum_{i=1}^{|\mathcal{C}|} TP_i}{\sum_{i=1}^{|\mathcal{C}|} TP_i + FN_i} \tag{5.6.4}$$

  where $|\mathcal{C}|$ indicates the number of classes.

- *macroaveraging*, on the other hand, is estimated by averaging over the precision and recall of each category:

$$P = \frac{\sum_{i=1}^{|\mathcal{C}|} P_i}{|\mathcal{C}|} \tag{5.6.5}$$

$$R = \frac{\sum_{i=1}^{|\mathcal{C}|} R_i}{|\mathcal{C}|} \tag{5.6.6}$$

  where P_i and R_i indicate the precision and recall of ith class (c_i) respectively. It must be mentioned that the methods explained above can give different results. Therefore, it should be clear which one is used during the experiments.

Similar to the discussion in Chapter 2 Section 2.7, there are other methods to evaluate the result of a classification system. Sometimes a single measure is required to trade off the precision vs. recall. The weighted harmonic mean of precision and recall, F-measure, is used for this objective. F-measure is a very common measure which we will mainly use in this thesis and is calculated as:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \tag{5.6.7}$$

where $\beta^2 = \frac{1-\alpha}{\alpha}$ while $\alpha \in [0,1]$ and $\beta^2 \in [0,\infty]$. The *balanced* F-measure that weights precision and recall equally, also known as F_1 means $\alpha = \frac{1}{2}$ and $\beta = 1$. In this case the formula is simplified to :

$$F_{\beta=1} = \frac{2PR}{P+R} \tag{5.6.8}$$

Though the balanced F-measure is not the only option, it is a trade off between precision and recall. Different values for β shows the focus on either precision or recall, based on the application purpose. We have that $\beta < 1$ emphasises precision and $\beta > 1$ emphasises recall. The values for precision, recall and F-measure are between 0 and 1. However they are commonly measured in percentage too. From Equation 5.6.7 it is clear that harmonic mean is the reciprocal of the arithmetic mean of the reciprocals ($\alpha = 1/2$). In the case of evaluating a classifier, harmonic mean makes more sense than arithmetic mean, because harmonic mean tends to be closer to the smaller value of the two compared to arithmetic mean. In other words, while harmonic mean mitigates the large value of one of precision or recall, it aggravates a small value of one of them. Harmonic mean punishes a failure in precision or recall more than rewarding a success of either one of them.

5.6.3 Performing Classification on Test Documents

The learning algorithm provides a string as a representative for each class. To classify the documents in the test set, we perform a procedure similar to the fitness calculation for each class c. Algorithm 4, sketches this procedure.

Algorithm 4 Classifying the test documents

```
Require: test data set, $S$
Require: class to representative map, $reps$
1: for all $c_i \in \mathcal{C}$ do
2:     $rep_i \leftarrow reps(c_i)$
3:     for all $d_j \in S$ do
4:         if $d_j$ satisfies the logical expression $rep_i$ then
5:             label $d_j$ as $c_i$
6:         end if
7:     end for
8: end for
```

In the next section, we perform the described learning algorithm and the classification provided above on a standard document collection and report and discuss the results.

5.7 Experiments

5.7.1 Data Collection

Standard and widely used text-based data collections have been a considerable aid in developing and testing information retrieval algorithms [Jones and van Rijsbergen, 1976]. They are very useful in developing, debugging and testing algorithms and also allowing comparison of different algorithms. In this chapter and the next, we will be using a very common text classification collection called "Reuters-21578" [Lewis, 1992c,a,b]. All the documents in this collection appeared on Reuters news-wire in 1987. All the documents were classified and labelled by personnel from Reuters Ltd [Lewis, 1992c]. There are 115[4] economic categories such as "coconut", "palm-oil" and "money-supply". We use the "ModeApté" split of training and testing documents [Apté et al., 1994a,b], which is commonly used for text classification evaluation [Sebastiani, 2002].

Table 5.2 shows some of the statistics of "Reuters-21578" collection.

Statistics	**Value**
Number of documents	12,902
Number of training document	9,603
Number of test documents	3,299
Size of the vocabulary before preprocessing	69,104
Size of the vocabulary after preprocessing	48,446
Number of tokens in the collection	1,227,955
Number of categories	90
Average number of training documents per category	106.7
Maximum document length	731
Average document length	68.609

Table 5.2: Statistics of "Reuters-21578" collection.

5.8 Development

The classification system is implemented in Java programming language. We have used the following open-source libraries in developing the system:

[4]There are 135 categories in the dataset, but 20 of them do not have any positive training document attached to them, so they are not used by researchers.

- **ECJ** which is an open-source evolutionary computation library with support of almost all evolutionary algorithms written entirely in Java. ECJ supports many different representations, selection and reproduction operators. It has a complete package for writing genetic programming based applications.
- **Apache Lucene** is a high performance text search engine. We have customised Lucene to perform all the information retrieval tasks needed for our system.
- **Apache Commons** is a set of re-usable Java components that are created for small purposes. Using well-designed, well-tested components like Apache Commons, improves the quality of the software and the speed of development.

5.9 Results

In this section, we provide the results of the classification system described in this chapter on the "Reuters-21578" collection. Both micro-average and macro-average for f-measure, precision and recall, are reported. In addition, the results for the top 10 categories of the collection (R10) are reported in detail. Figure 5.8 shows an example of a query learned for category `corn`. Figure 5.9 shows a part of a sample document from the training data of that category that shows how the learned representative matches that document.

Measure	**Value**
μF**-measure**	0.2718
MF**-measure**	0.4599
μ**-Average precision**	0.3148
μ**-Average recall**	0.2391
M**-Average precision**	0.6068
M**-Average recall**	0.4356
Average number of terms	11.4444
Average number of functions	11.7111

Table 5.3: Results of the classification system for all the categories of the collection on the training set.

Table 5.5 and Table 5.6 show the results of the classification system for the top 10 cat-

Measure	**Value**
$\mu - F$**-measure**	0.2064
$M - F$**-measure**	0.1348
μ**-Average precision**	0.2398
μ**-Average recall**	0.1811
M**-Average precision**	0.2025
M**-Average recall**	0.1324
Average number of terms	11.4444
Average number of functions	11.7111

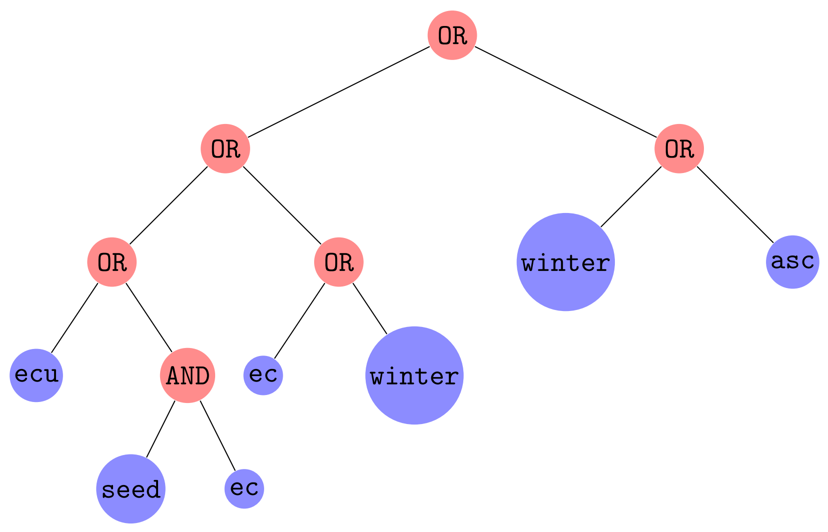
Table 5.4: Results of the classification system for all the categories of the collection on the test set.

	Category	F**-measure**	**No. Terms**	**No. Functions**
1	acq	0.4686	18	19
2	corn	0.1671	8	8
3	crude	0.1696	14	14
4	earn	0.2811	14	15
5	grain	0.4127	1	0
6	interest	0.1509	10	9
7	money-fx	0.1711	10	10
8	ship	0.1522	14	15
9	trade	0.2601	8	11
10	wheat	0.2232	16	15
	μ**-Avg precision**	0.4141	M**-Avg precision**	0.3417
	μ**-Avg recall**	0.2252	M**-Avg recall**	0.2059
	$\mu - F$**-measure**	0.2918	$M - F$**-measure**	0.2457

Table 5.5: Results of the classification system for the top 10 categories of the collection on the training set.

egories of the collection. Apart from the quality measures, which are F-measure and precision and recall, the number of terminals (leaf nodes) and the number of functions (non-leaf nodes) used in generating each representative are also reported. These numbers can show the simplicity of the representatives generated. Results are reported on both train and test set. Although the aim is to improve the results on the test data, reporting quality of the train data can help us to identify overfitting.

Table 5.3 and Table 5.4 show the overall performance of the system on the collection. Results of both train and test have been shown.



((PT AND PT) OR PT) OR (PT OR PT) OR (PT OR PT)

Figure 5.8: The representative learned for category `corn`.

Compared to other classification algorithms, the results reported in this section are very low. The main reason is that we have used the simplest methods in different parts of the learning algorithm. Our term selection method is working completely randomly and different terms have a similar chance of selection. On the other hand, we do not take benefit from phrases or negative terms in the current experiment.

In the next chapter, we aim to explore different components of the GP-based classification system and improve the quality of classification. In addition, we run some experiments to evaluate the readability of the representatives generated by the GP engine.

```
...It said crop conditions were better than earlier expected following
the extreme dry conditions last fall and the prolonged winter
temperatures this spring.  However, in general plant development was
at least three weeks or more behind normal this spring, and conditions
varied greatly by regions, the report said.  Fields seeded during the
optimum period last fall, and especially those receiving supplemental
irrigation water (about 65 pct of the fields observed), appeared to
be in good condition, with little evidence of winterkill, while others
varied considerably, the report said.  ...
```

Figure 5.9: A part of a sample document retrieved by the representative learned for `corn`, which is shown in Figure 5.8.

	Category	***F*-measure**	**No. Terms**	**No. Functions**
1	acq	0.512	18	19
2	corn	0.1653	8	8
3	crude	0.145	14	14
4	earn	0.158	14	15
5	grain	0.4071	1	0
6	interest	0.1117	10	9
7	money-fx	0.1486	10	10
8	ship	0.0816	14	15
9	trade	0.3034	8	11
10	wheat	0.2597	16	15
	μ-Avg precision	0.3775	**M-Avg precision**	0.3139
	μ-Avg recall	0.1995	**M-Avg recall**	0.1894
	$\mu - F$-measure	0.261	**$M - F$-measure**	0.2292

Table 5.6: Results of the classification system for the top 10 categories of the collection on the test set.

During the experiments in the next chapter, we use the results in Tables 5.5, Tables 5.6, Tables 5.3 and Tables 5.4 as our baseline and will improve step by step the quality of the system.

5.10 Summary

In this chapter we explained the main steps and concepts of our approach for classification based on class representatives learned by genetic programming.

Similar to other classification approaches, we started with indexing the documents. The indexing process, which is mainly transforming raw data into a list of terms and their attributes, is explained in detail.

The unique feature selection method based on GP is described and we showed how the generation of class representatives through terminal and function nodes is coupled with feature selection in our method. The role of positive and negative terms in achieving high quality class representatives is also discussed.

The training process is the evolution of class representatives to generate a high quality representative that is able to retrieve all the relevant documents and discriminate against non-relevant ones. The aim in this process is to optimise the classifier by selecting similar documents which have been identified as relevant in order to add or replace terms with more discrimination power with those already in the classifier. The fitness function we have used for the learning process in this chapter is F-measure. The testing set provides the possibility to evaluate the classifier on unseen documents. While the accuracy of the classifier is measured by fitness function. Micro-average and micro-average F-measure, precision and recall are used to evaluate the classifier.

CHAPTER 6

Improvements and Experiments

6.1 Introduction

In the previous chapter, we described a classification system based on genetic programming. In the training phase, we generate representatives for each category and use those representatives to classify the future documents in the testing phase, the main method of generating the representatives has been explained. In addition, our approach to classify documents with the use of class representatives has been provided (see Section 5.6.3). In this chapter, we improve almost all the aspects of the system described in the previous chapter, which will be called the baseline, and report the results.

There are two aspects of the baseline that we want to improve. Firstly, the quality of the classification on the test set, which is the aim of any classification systems. Secondly, the readability or parsimony of the representatives. Therefore we need to improve the quality of the classification without making the representatives very complex. On the other hand, simple representatives that do not perform well enough, are not desired. To improve the system in both directions, we examine different methods and improvements to all the components of the classification system. The modifications discussed in this chapter are categorised as follows:

- Learning the representatives
    - Parameters of the genetic programming engine
    - Definitions of the functions and the terminals in the GP tree

- Feature and term selection
- Fitness measure
  - Different evaluation measures that capture the quality better than the baseline
  - Similarity methods used in computing the similarity between the documents and the representatives
  - Addressing overfitting by adding a validation set in fitness computation
  - Incorporating a parsimony measure in the fitness

Each category listed above is explained and their results are reported. We discuss the results and at the end of the chapter provide a summary of the methods that empirically performed the best.

6.2 Learning the Representatives

An evolutionary computation-based algorithm has a set of parameters that substantially affect the quality and the speed of the algorithm. We consider a set of parameters that can change the genetic programming engine's outcome in a way which is beneficial to our classification task. Modifications and optimisations of these parameters have two aspects. Firstly, improving the quality of the results and secondly optimising the speed of the system. There has always been a trade off between these two aspects, therefore in this chapter we focus on how to improve the quality while keeping the system response time short.

To begin with, we examine the effects of the genetic programming engine parameters on the quality and afterwards investigate different tree structures and more complex functions.

6.2.1 Genetic Programming Parameters

Table 6.1 shows the set of parameters with their values used in the baseline[1]. For some of the parameters such as the *generation method*, there is only a small set of options and for some of them such as *number of generations*, there is a broad range of numeric values that can be tested. We report the effect of each parameter on the quality of the classification system separately.

	Parameter	**Value**
1	Number of generations	20
2	Population size	100
3	Generation method	Ramped half and half builder
4	Selection method	Tournament selection with size 7
5	Mutation probability	0.1
6	Reproduction probability	0.1
7	Crossover probability	0.7
8	ERC mutation probability	0.1
9	Number of elites	0
10	Max tree depth	6
11	Number of Subpopulations	1

Table 6.1: Parameters of GP used in the baseline.

Number of Generations and Population Size

The number of cycles that the GP engine assesses the fitness of the individuals and breeds new off-springs and assembles a new population is the definition for number of generations parameter. More generation increases the time of learning the representatives and may or may not improve the quality of the best individual. On the other hand, a low number for this parameter can prevent the GP engine to produce less than enough individuals and fail to learn a good representative.

Figure 6.1, Figure 6.2 show the effect of changing the number of generations for sample categories. A decent number of generations improves the fitness intensely at the start, however after a certain number of generations this increase gradually fades away and does not have any positive effects of classifier ability. The population size follows the

[1]For the definition of most of the parameters see Chapter 3.

same routine, though the quality of classifier and the final performance are directly affected. However, there is a peak point which after that it not only may be ineffective but also degrading the quality.

As our experiments show a good point can be selected to have a reasonable response time without degrading the performance of the system, however, there is a trade off between the quality and speed. The amount of physical memory, which directly depends on the population size is also another limiting factor. Figure 6.1, Figure 6.2 show the diagrams of population size changing from small numbers to very large populations.

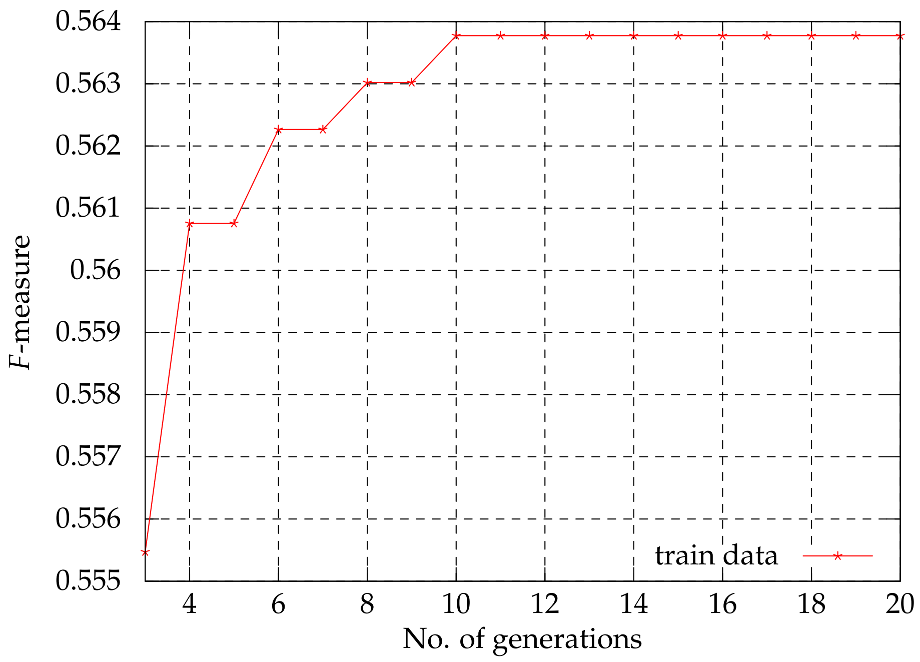


Figure 6.1: Change in the performance of the baseline on the train data based on the increase in the number of generations.

6.2.2 Tree Structure and Different Functions

Regarding GP, each representative is generated in the form of a tree shape structure, therefore the tree structure and its depth are very important. It is very important to note that the representative human readability has a direct relationship with the tree structure and more closely is affected by its depth, something that has not been considered in the baseline.

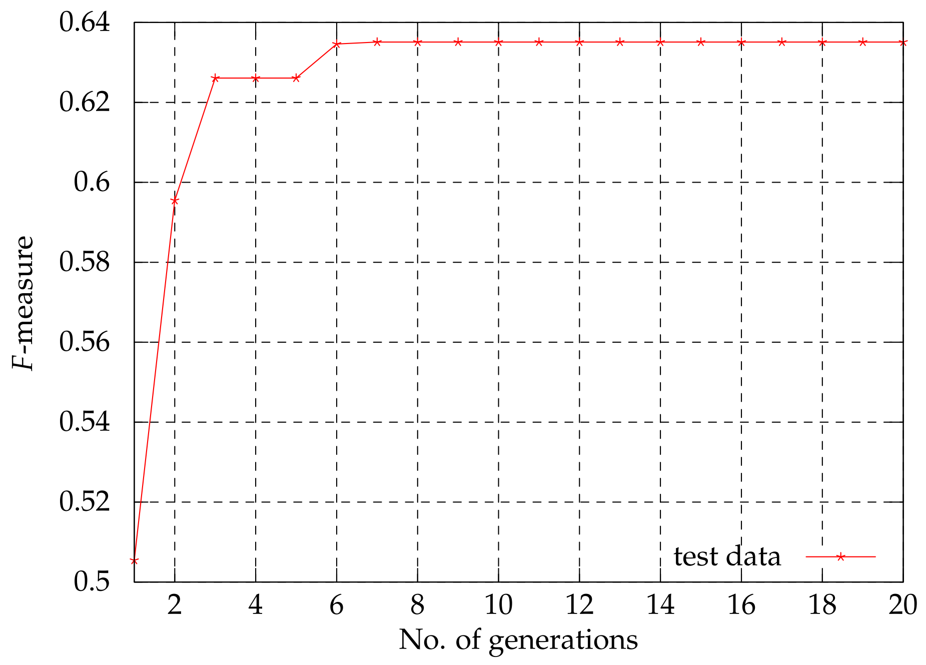


Figure 6.2: Change in the performance of the baseline on the test data based on the increase in the number of generations.

One issue in the baseline that should be tackled is having different individuals in the population with the same quality and with the same discrimination power. This makes the search algorithm incapable of exploring the search space as much as it should. The algorithm is busy investigating individuals which are similar to each other. During a search process in order to find the best representative for a class based on the training data, due to time and computing resources restrictions, only a limited set of individuals can be investigated. A search process is more effective when it investigates a variety of individuals with different terms and structures rather than similar individuals.

To address this problem, we have defined functions that are more complex than the ones in the baseline, however in the end, they lead to generate simpler trees. Table 6.2 shows a list of the new functions that are used to build the individuals.

The main difference between functions such as `AND` and `AND5` is the number of children that they accept. While both functions have the same effect, forcing the representative to select documents that include all the child terms, having more children and less depth makes the latter more readable on one hand and examines more positive terms

Function	Operand type	Number of children	Type of children
AND2	Query	2	Query
AND3	Query	3	Query
AND4	Query	4	Query
AND5	Query	5	Query
OR2	Query	2	Query
OR3	Query	3	Query
OR4	Query	4	Query
OR5	Query	5	Query
NOT	Query	1	Term

Table 6.2: List of the new functions and their accepted children

to be included in the tree on the other hand. In general, using the new functions has two advantages:

- Simpler trees with less depth. Readability is not the only benefit of this simplicity. Trees with less functions are less likely to be similar in quality and different in structure. In other words, we reduce the complexity of the tree structure to make it less prone to redundant individual generation.
- Each node needs to select more terms to fill the children. Therefore the genetic programming engine needs to query the terms in the positive and negative list more than the baseline, which consequently makes the engine to try more features and explore a larger part of the search space.

Figure 6.3 and Figure 6.5 show the difference between the baseline `OR` and the new `OR`.

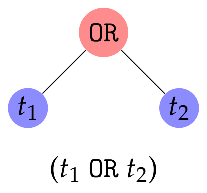
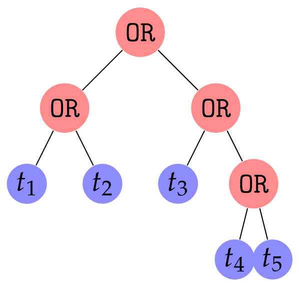
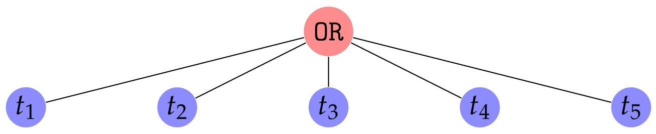


Figure 6.3: Function `OR` of the baseline with only two children.



$(((t_1 \texttt{ OR } t_2)) \texttt{ OR } (t_3 \texttt{ OR } (t_4 \texttt{ OR } t_5)))$

Figure 6.4: Function `OR` of the baseline with only two children. To construct a tree with five positive terms, at least three levels of `OR` are needed (four function nodes).



$(t_1 \texttt{ OR } t_2 \texttt{ OR } t_3 \texttt{ OR } t_4 \texttt{ OR } t_5\ )$

Figure 6.5: New function `OR5` with five children. A node with five children is more readable than a binary tree with at least three levels of depth, such as the one in Figure 6.4 .

6.2.3 Phrases

The use of phrases in text retrieval has been common in the information retrieval community [Smeaton and Kelledy, 1998; Pickens and Croft, 2000]. A phrase can contain more information than its constituents' terms [Smeaton and Kelledy, 1998]. There are two types of approaches to use phrase in text retrieval:

- Statistical phrases which are any pair of words that occur contiguously frequently enough in the documents is a phrase [Mitra et al., 1997; Caropreso et al., 2001].
- Syntactic phrases which are any group of words that constitute a syntactic structure, usually nouns for information retrieval [Kraaij and Pohlmann, 1998; Mitra et al., 1997].

In our work, we use statistical phrases, which means any pair of words can be considered as a phrase and there is no syntactic constraint on the phrases. We assign each phrase a weight which indicates the discrimination power and hence the importance of the phrase. Number of possible phrases in a collection is huge and considering all of them is neither practical, nor beneficial. Therefore, each phrase is assigned a weight and only phrases with a certain discrimination power are involved in the search process.

Methods to assign the weights to phrases are explained in a later section (6.3.1). In the GP engine phrases are treated like the positive terms. So, the type of a phrase is a `Term` and a phrase alone can constituent a class representative.

6.2.4 Experimental Results

In this section, we investigate the effect of the GP parameters (Section 6.2.1), new functions (Section 6.2.2) and phrases (Section 6.2.3) in quality of the GP classifier. The results reported in Section 5.9, Table 5.5 were produced using the baseline GP classifier with default parameters (see Table 6.1). Table 6.3 shows the contribution of each parameter change in the quality of the classifier for class `corn`.

	Run	**F1**	**Improvements**	**Percentage**
1	*baseline*	0.1671	0	0
2	*no. of generations = 50*	0.2322	0.0651	38.95
3	*population = 500*	0.2804	0.0482	20.75
4	*population = 1000*	0.5821	0.3071	109.5
5	*phrases*	0.7847	0.2026	34.80
6	*new functions*	0.8731	0.0884	11.26

Table 6.3: Step by step changes to the structure of the representatives over the baseline and contribution of each change to the quality of the classifier for an example class `corn`. For a description of the data collection, please refer to Section 5.7.1.

Table 6.3 shows the improved results of F_1 for the baseline after some changes. These results are produced on the same collection that was described in Section 5.7.1. The claim is by the increase of the number of generations the fitness measure should increase, when the number of generations increases to 50, then F_1 improves by 0.0651. While the number of generations is set to 50 the population size is increased. Growth of population size will improve the quality of classifier. However the main improvement happens at population size 1000 which is 0.3071. Though a decent population size is one of the key factors to the classifier accuracy and the quality of performance but after a certain point this effect gradually stops and increasing the population size does not improve the results any more.

At this stage considering phrases and employing *new functions* improve the results by 0.2026 and 0.0884 . The achieved results show that our claims regarding the number of generations, population size, use of phrases and new functions improve the quality and accuracy of a classifier on a given topic.

6.3 Feature and Term Selection

As discussed before, each document is represented by a set of features that are selected with the aim to represent a document in the process as fully as possible, while discarding non-informative features of the document.

The most important features in information retrieval and text classification are terms. Therefore, an important part of the feature selection process is to select a set of terms that are informative in discriminating between documents.

In the previous chapter, we described the GP algorithm that selects terms and combines them to generate a tree-shape structure. In this section, we explore ways to provide the GP algorithm with a better list of features to select from and a higher chance of selecting more discriminant terms.

6.3.1 Feature Weights and Term Weights

To generate an accurate representative with a high fitness measure, it is crucial to choose terms that reflect the best presentation of the required class. Referring to Chapter 2, terms with high discrimination power are desired to be used in representative generation. Terms that can discriminate between documents can be defined in two ways. First, high power to identify the relevant documents and retrieve them, which are defined as *positive terms*. Second, high power to identify non-relevant documents and avoid retrieving them, which are known as *negative terms*.

We explained the role of positive and negative terms in our classification method in the previous chapter. Here, we improve the selection method of the terms and show that this selection is very important to improve the quality of the classification system, because after all, this system is based on features and the better the features are, the better the system works.

6.3.2 Positive Terms

It was explained that terms may have different values and effects. To identify the most relevant documents it is important what terms are used during the query generation process. Terms with high discrimination power identify more relevant documents (more in both sense of numbers and relevancy). The best way to identify and choose these terms is after pre-processing (see Section 2.3). During the pre-processing, a set of processes changes the original structure of document into an internal representative in the form of a set of indexed features. These indexed features are terms that have been extracted from documents along with their specific parameters. Preprocessing is one of the main processes in information retrieval and there are different methods, all known as feature selection methods. These methods work by eliminating non-informative terms which reduces the noise features and keeping informative terms improves the classifier's accuracy.

Unlike document retrieval, that the rest of the terms in the collection can efficiently be considered to rank the documents, in text classification the use of all terms is neither efficient nor effective for most classification algorithms. To overcome the issue of large parameter space (a large set of distinct terms in the collection), a selection method is used to remove a set of the terms from the parameter space. This process is called filtering by term selection and leads to a reduced set of terms to be employed in the learning process and eventually in generating the class representatives.

There are many term selection methods[2] that have been investigated for different classification algorithms. These methods work by eliminating non-informative terms to reduce the noise features and keeping informative terms to improve the accuracy of the classifier by avoiding over-fitting.

Feature selection methods are ways to find terms with high discrimination power. So, they can be used in the positive list of the terms and be selected to be included in the class representative. To improve the outcome and optimise the quality of selected features, we investigate some of the feature selection methods used in the literature:

Term Frequency

In this method terms with high frequency, which are the most common terms in the class, are selected. When the number of selected features is high, this performs well and can be considered as a good alternative to other methods, however in ordinary feature selection with far fewer selected features it does a lot worse than other methods.

The frequency-based feature selection can be done in different ways:

- *Document frequency*: the number of training documents in the class that term t has occurred in them. We show this value as: $\#_d(t,c)$
- *Collection frequency*: the number of times term t has occurred in documents of class c. We show the value as: $\#_t(t,c)$

Mutual Information

Mutual information measures how much information the presence or the absence of a term in one document can provide, when the classifier decides about that document to be labelled by a specific class and it is measured as [Sebastiani, 2002; Manning et al., 2008]:

$$MI(t,c) = \log_2 \frac{P(t,c)}{P(t)P(c)} \tag{6.3.1}$$

[2]Also called term space reduction methods or feature selection methods

where, $P(t,c)$ is the probability of t being in a document that belongs to c. Mutual information can be estimated as [Yang and Pedersen, 1997]:

$$MI(t,c) \approx \log_2 \frac{\#_d(t,c) \times |D|}{\#_d(t,\bar{c}) \times \#_d(\bar{t},c)} \tag{6.3.2}$$

where, $|D|$ is the total number of documents, $\#_d(t,\bar{c})$ is the number of times t occurs without c and $\#_d(\bar{t},c)$ is the number of times c occurs without t. The highest value for this measure is $MI(t,c) = 1$ and shows the presence of the term can guarantee that document belongs to the given class.

χ^2– Chi-Square

χ^2 is a statistical test that measures the independence of two events, which are term occurrence and the class in feature selection for classification. Terms are ranked by the estimation of the following equation of their independence from each class [Yang and Pedersen, 1997]:

$$\chi^2(t,c) \approx \frac{|D| \times (\#_d(t,c) \cdot \#_d(\bar{t},\bar{c}) - \#_d(\bar{t},c) \cdot \#_d(t,\bar{c}))^2}{\#_d(t) \cdot \#_d(\bar{t}) \cdot \#_d(c) \cdot \#_d(\bar{c})} \tag{6.3.3}$$

where, $\#_d(\bar{t},\bar{c})$ is the number of documents that t does not occur and does not belong to c, and $\#_d(t)$ is the number of times that t occurs and the rest are similar to the notation in Equation 6.3.2.

Information Gain

The idea to use information gain for feature selection is to select terms that reveal the most about the classes. The information gain measure for feature selection is computed as [Dasgupta et al., 2007]:

$$IG(t,c) = \sum_{c_i \in \{c,\bar{c}\}} \sum_{t_j \in \{t,\bar{t}\}} \#_d(t_j,c_i) \times \log_2 \frac{\#_d(t_j,c_i)}{\#_d(t_j,\bar{c_i}) \times \#_d(\bar{t_j},c_i)} \tag{6.3.4}$$

Feature Scoring

We use the above method to select a set of features that are used to represent the documents. Although all these methods estimate a score for each feature and hence assign a weight to each term, there are other ways to score the terms and weight them which are different from the selection process and are shown to be effective [Sebastiani, 2002]. One of these methods that we use in this work is *tfidf*, where tf is the term frequency within the document and idf is the inverse document frequency of the term.

tfidf combines two sources of evidence: the importance of the term within the document and the discrimination power of the term in the collection. The more often it occurs in a document, the more it represents that document and the less documents it occurs in, the more discriminant it is. *tfidf* can be estimated in different ways and has been widely used for document retrieval [Salton and Buckley, 1988].

Table 6.4 shows the effect of different feature selection methods on the quality of the classifier on the training and test sets for one class, which is `corn`.

The results show that mutual information and χ^2 are performing very well and all the feature selection methods are performing better than the baseline, where the selection of the terms is completely random and the probability distribution of selecting the terms is uniform. These results also reveal that a feature selection method such as χ^2 is not a very good candidate for weighting the terms, but if it is combined with a weighting scheme such as $tfidf$ it performs very well.

6.3.3 Negative Terms

Positive terms have been described and also the reasons are given that why they are important and how to select them, however it is equally important to identify terms which may mislead the system by attracting non-relevant documents and decrease the accuracy of retrieval process. These terms are known as *negative terms* (see Section 5.5.3 in Chapter 5). These terms may be collected form relevant or non-relevant documents, however their importance to us is to attract non-relevant documents.

Selecting positive terms seems straightforward and sensible to do, however negative

	FS Method	**Weighting**	**No of Features**	**F-measure (Train)**	**F-measure (Test)**
1	None	None	∞	0.308	0.392
2	tf	tf	∞	0.749	0.750
3	MI	MI	∞	**0.873**	**0.903**
8	IG	IG	∞	0.749	0.750
4	χ^2	χ^2	∞	0.781	0.741
5	χ^2	χ^2	2,000	0.337	0.380
6	χ^2	χ^2	10,000	0.788	0.761
7	χ^2	tf-idf	2,000	**0.873**	**0.903**

Table 6.4: The effect of different feature selection method on the performance of the classifier on the testing and training data for class `corn`. The number of generations for all the runs of this experiment was 20 and the population size was 1000.

term collection is a bit tricky, as they may be in relevant documents and not necessarily from non-relevant documents. We have considered three main characteristics for terms to fall into negative terms collections:

- The first group is the group of terms with low discrimination power, they do not bring that much value to the class representative to attract relevant documents, though they exist in the relevant documents due to low discrimination power they may lead choosing non-relevant documents as well as relevant ones. By considering them as negative terms, the case of retrieving non-relevant documents will be eliminated, however it is possible to lose highly relevant documents just because they include one or more of these terms with low discrimination power. Also it is a very time consuming process as the system should go through all of the terms and calculate the discrimination power for each and every one of those terms, then compare the discrimination power to a predefined threshold, and in case of a smaller value than the threshold, identify the term as a negative term.

- The second group is the group of terms with high discrimination power that does not exist in the required class training documents. Sometimes it is possible after doing all the preparation processes (see Section 2.3.1) for some terms with high discrimination power to stand out. They belong to the training documents however they do not represent the specific required class. These terms mislead the class representative and result poorly to find relevant documents. These documents may or may not represent other classes. These terms with high discrimination power can influence the system drastically while they do not improve the results, so considering them as negative terms may reduce the chance of retrieving non-relevant documents.

- The third group is the group of terms with high discrimination power that represent other classes but not the required class. These terms probably are the positive terms for other classes whereas they are totally misleading for the required class. Though the system should identify and calculate the positive terms for all required classes, overall it does not increase the time of the whole classification

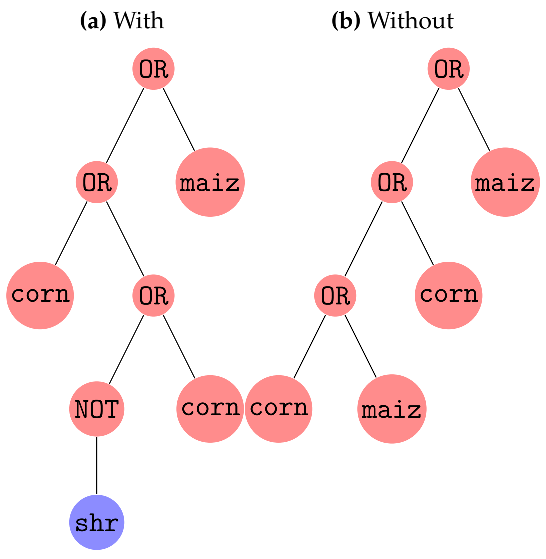


Figure 6.6: An example of a class representative learned with and without a negative term terminal and a `NOT` function.

task, as it saves time by identifying positive terms for all classes and calculates their discrimination powers right at the beginning and uses each class positive terms as negative terms for another. Of course it is essential that some refinements and eliminations should be done before using them.

Figure 6.6 shows examples of class representative for the same class with the same settings apart from the fact that one of them has an extra function `NOT`, which takes exactly one child that is also chosen from the negative list including a set of negative terms.

6.3.4 Term Selection Procedure

In the baseline system described in the previous chapter, positive and negative terms were chosen based on a random selection from their lists. This method may be the first choice as it is random and all terms get the equal chance to be selected, but as it has been described and explained different terms have different values in discriminating between relevant and non-relevant documents. Therefore, their chances should increase or decrease based on their discrimination power or some other features and characteristics. In this section, we present two algorithms that have more authority over choosing terms based on their different features.

Ranked-based

In this algorithm terms are sorted based on their scores, then the probability of selecting a term in the *i*th position among *n* terms will be calculated as follows:

$$P(\text{selection of term } i) = \frac{N - i + 1}{N(N+1)/2} \tag{6.3.5}$$

In this algorithm, the terms are selected based on their feature value, however the chance of selecting them during the process of learning only depends on their rank. In other words, it does not matter how close two feature values are, the probability of selecting them only depends on the distance of their ranking in the list.

Scored-base

This algorithm estimates the probability of one term to be selected regarding its "score" according to the features used for term selection. The first step is to calculate the score, then normalise the set of scores to make them a "probability distribution"[3]. Finally, the probability of term in the *i*th position is equal to its score.

Note that in both methods the selection is with replacement, which means a selected term can be selected again.

6.4 Fitness Improvements

To identify which class representative works better, there is a need to measure their fitness. Choosing a good fitness function is very crucial in the success of any evolutionary algorithm, because it is the procedure that guides the random-based evolutionary algorithm towards a solution. For an information retrieval task, such as the classification that we are dealing with, the fitness can be measured by the number of relevant

[3]i.e. $\sum_i^n score_i = 1$

documents the representative identifies. However the overall performance is affected by so many other details, such as the rank or score of relevant documents in the retrieved set or the number of non-relevant documents in the retrieved documents set. The baseline system described in Chapter 5 uses *F*-measure to evaluate a representative performance and estimate a fitness value. In this section, we investigate other fitness functions that take into account the results ranking and scoring to improve the learned representatives.

6.4.1 *F*-measure as the Fitness Function

The baseline system described in the previous chapter uses *F*-measure to estimate the fitness for each individual. The fitness is calculated as:

$$fitness(q_i, c) = \frac{1}{1 + \mathrm{eval}(q_i, c)} \tag{6.4.1}$$

where here $\mathrm{eval}()$ function is $F-$measure and better individuals have lower fitness value and vice versa. In this method lower $F-$measure produce higher fitness values.

Algorithm 4 shows how the classification is done by using a class representative. This algorithm is used while the fitness for an individual is estimated and also when the test documents are classified. One important point about this algorithm is, it labels all the documents that satisfy the class representative, however this leads to classify too many documents under a particular class, when the class representative is general.

There are several ways to select a subset of documents to be labelled under a class or to select a subset of classes that a document is labelled with. Among the following list of methods that are used in previous work, we choose **PCut**, which is more suitable for our classification algorithm [Yang, 2001; Tang et al., 2009]:

- **RCut:** Rank-based cut ranks the classes assigned to a document based on their scores and labels the document with the top *k* classes of the rank list. *k* is a parameter given by the user and usually estimated based on the training data.
- **PCut:** In proportion-based cut a ranked list of the documents is created for each

class and the class is assigned to the top n documents of the ranked list. n is estimated based on the training data.

- **SCut:** This approach is called score-based local optimisation and is similar to PCut. The only difference is that the threshold is tuned for each category and there is no one threshold for all, which is the case in PCut.

Since we learn a class representative for each class and estimate the quality of the individual representative, The PCut method is the best method to be used in our algorithm. Additionally, [Yang, 2001] have concluded that PCut is more stable than SCut.

The fitness function based on $F-$measure treats a ranked-list of classified documents exactly like an ordinary set, so in this case if a relevant document's score is less than non-relevant document's score the overall fitness is equal to the situation that relevant document's score is higher than non-relevant document's score. On the other hand, if there are two individual class representatives and the first one returns the relevant documents in the 4th, 7th and 10th positions and the second one returns the relevant documents in the 20th, 28th and 33rd positions there is no difference between their fitness values, whereas in reality the first representative is performing better and the fact that it returns correctly chosen relevant documents in higher ranks must be taken into consideration. Although $F-$measure is very straightforward and effective, it misses out many advanced features and details about the fitness.

There is a problem with this approach, after choosing top-k documents, the sequence of documents and their scores are ignored, in other words, the system is not optimised to push the relevant documents to the beginning of the list.

To address this problem, we investigate two other evaluation metrics that take into account the order of the classified documents:

- Average Precision
- Normalised Discounted Cumulative Gain

6.4.2 Average Precision

Precision and recall are two very straightforward measures. They are easy to use with a fixed range (0.0 to 1.0 or 0% to 100%), with the ability to compare different queries and collections for different systems. The best system increases the precision with an admissible recall, however there is always a trade off between these two. Precision shows how many of retrieved[4] documents are actually relevant, but does not show how many of relevant documents are retrieved and how many have been ignored, while the recall shows how many of relevant documents are retrieved but does not consider how many non-relevant have been retrieved as well as relevant documents. The current systems may have an acceptably high recall while the precision is so poor, it means though the system retrieves many relevant documents, and there are number of non-relevant documents among retrieved documents. To combine these two measures and have a better view of the system performance, the weighted harmonic mean of precision and recall known as $F-$measure is used. All these measures are set-based and represent the results in unordered lists of documents. However as we discussed in the previous section, a ranked list of results is much more desirable. To be able to provide such a result, we use another measure called *average precision*. Average precision shows at each step of retrieval process what is retrieved and how it has increased or decreased the system performance.

Recall measures how many of actual relevant documents in the collection get retrieved, so a new retrieved document does not change the recall value if it is not relevant. In other words, when a non-relevant document is retrieved recall is constant.

To take the order of appearing relevant documents into consideration we calculate the average for different precisions at the levels that recall changes. This value is known as average precision and is calculated as [Manning et al., 2008]:

$$\mathrm{AP}(q_i, c) = \frac{1}{m} \sum_{k=1}^{m} \mathrm{Precision}(R_k) \tag{6.4.2}$$

[4]In the discussions of this section, retrieved documents for each class are those labelled to be under this class and relevant documents are those which truly belong to that class.

where, $d_1, ..., d_m$ is the set of positive documents for class c, q_i is the class representative, R_k is the set of top k ranked retrieved documents and Precision(R_k) is calculating precision at the bottom of the ranked list R_k.

6.4.3 Normalised Discounted Cumulative Gain

An approach to evaluation which has been used increasingly recently is *normalised discounted cumulative gain (NDCG)* [Järvelin and Kekäläinen, 2002; Croft et al., 2009]. NDCG is particularly designed for retrieval tasks where there are more than two notions of relevance, however it can easily be adapted to binary situations (relevant vs non-relevant). The premise of the measure is relevant documents appearing in the top of the list should be more valued than those appearing at the lower ranks. So, the reward of finding relevant documents is reduced by going through the list. DCG_p, which is discounted cumulative gain at position p is calculated as:

$$DCG_p = \sum_{i=1}^{p} \frac{2^{R(i)} - 1}{\log_2(1+i)} \tag{6.4.3}$$

where $R(i) \in \{0, 1\}$ indicates whether ith document is relevant or not. $NDCG_p$ is a normalised version of DCG_p to give the perfect rank list value 1.0.

6.4.4 Validation Set

Machine learning approaches to classification rely on a set of data previously labelled as training data to learn and tune the classifiers to be evaluated on the test data. No information from the test data should be available to the learning algorithm and no decision should be made based on the results of the system against the test data. Otherwise, the results might unrealistically be very good, however fitted to that specific test data. In addition, reported results obtained this way do not have any scientific credibility.

On the other hand, optimising the classifier only on the training data and learning class representatives that perform the best on the training data might lead to over-fitting and a weak performing class representative on unseen examples. Therefore, we set aside a small set of training data as unseen examples and calculate the fitness based on a combined version of the fitness on the seen and the fitness on the unseen data. This held out set is called the validation set, which is used to avoid the learning algorithm to over-fit the training data and make the class representative able to generalise.

We change the fitness function in Equation 6.4.1 to include a similar component:

$$fitness(q_i, c) = \alpha \cdot \frac{1}{1 + \text{eval}_t(q_i, c)} + \beta \cdot \frac{1}{1 + \text{eval}_v(q_i, c)} \quad \text{and} \quad \alpha + \beta = 1 \tag{6.4.4}$$

where $\text{eval}_t()$ is estimating the evaluation metric on the training set and $\text{eval}_v()$ is estimating it on the validating set. α and β are weights assigned to each component and are tuned manually to adjust the importance of each component.

6.4.5 Parsimony and Readability

One of the main motivations of using a genetic programming algorithm to learn a class representative for classification in this thesis was the readability of the result. In other words, it is very important and has many applications to generate a class representative that can be read, understood and possibly modified by humans.

Although readability is subjective and can not automatically be calculated, we assume that there is a strong correlation between readability of the representatives and the number of terminals and functions used to construct them. In Section 6.2.2, we introduced new functions to simplify the tree-shape structure and easily combine several terms together. This modification leads to less functions and a flatter structure of the tree, that we believe makes it more readable. However, in addition to the change to the structure of the individuals, we add another component to the fitness function to favour simpler individuals and penalise individuals that have a high number of functions and terminals. To achieve this we modify the fitness function of Equation 6.4.4 to include a readability component:

$$fitness(q_i, c) = \alpha \cdot \frac{1}{1 + \text{eval}_t(q_i, c)} + \beta \cdot \frac{1}{1 + \text{eval}_v(q_i, c)} + \gamma \cdot \frac{\log(x)^2}{4} \tag{6.4.5}$$

where x is the number of functions used to construct q_i.

We use a logarithm square function to decrease the effect of adding one function to the tree. Also, this component is between zero and 1, if the number of functions are less than 100. Similar to Equation 6.4.4, we have $\alpha + \beta + \gamma = 1$.

6.5 Results

Table 6.5 and Table 6.6 show the results of the experiments on the test collection by running the system that has all the features combined. These results are produced by running the classification system on the test collection that is described in Section 5.7.1 of the previous chapter. For more experimental results on a different collection, please see Appendix C.

In Table 6.6, we start from the baseline and extend it with the features step by step. The highest numbers of each column are highlighted in **bold**. In Table 6.5, the improvements for each class of the top 10 category and the number of terminals and functions for each one of them are shown.

		***F*-measure**		**No. Terms**		**No. Functions**	
	Category	**Baseline**	**New**	**Baseline**	**New**	**Baseline**	**New**
1	acq	0.1977	0.7563	18	5	17	4
2	corn	0.3279	0.9032	3	2	2	1
3	crude	0.417	0.7278	14	5	17	4
4	earn	0.018	0.9267	16	4	16	3
5	grain	0.0699	0.8926	13	5	12	4
6	interest	0.1577	0.4092	8	4	8	3
7	money-fx	0.1574	0.5282	14	5	14	4
8	ship	0.3134	0.6986	10	5	10	4
9	trade	0.186	0.4903	16	5	16	4
10	wheat	0.25	0.8974	11	1	13	0

Table 6.5: The results of the classification for the top 10 categories of the collection for the baseline and the new run, which is a baseline combined with all the improvements discussed in this chapter.

	Run	Avg terminals	Avg functions	μF_1	MF_1	μ-Avg π	M-Avg π	μ-Avg ρ	M-Avg ρ
1	**Baseline**	10.577	10.866	0.167	0.169	0.205	0.222	0.140	0.188
2	**+New Structure**	2.055	0.400	0.315	0.274	0.301	0.342	0.329	0.298
3	**+Feature Weighting**	2.366	1.355	0.606	0.456	0.501	0.494	0.768	0.505
4	**+Feature Selection**	3.288	2.288	0.699	0.490	0.641	**0.547**	**0.768**	0.509
5	**+Readability Factor**	3.211	2.211	**0.726**	**0.501**	**0.697**	0.543	0.757	**0.519**

Table 6.6: The results of the classification for the all the 90 categories of the collection for the baseline and the new run with the breakdown of the contribution of each factor. μ is micro-averaging, M is macro-averaging, π is precision and ρ is recall.

The results show that using the new tree structure is improving the performance of the classifier, in addition to simplifying the representatives. The very small number of functions in the new structure indicates that the learning algorithm favours functions with more than two arguments compared to the baseline that all the functions have two arguments.

By adding feature weighting and feature selection to the algorithm the quality of classification substantially increases, however, this leads to use more functions and terminals in the class representatives. Adding the readability factor to the fitness function slightly improves the performance of the classifier and as it is expected decreases the number of functions and terminals used in the final class representatives.

Figure 6.7 shows an example of a representative learned by the new structure and the readability factor in the fitness function for class `money-fx`, which is quite simple and readable. On the other hand, the class representative for the same class that is learned by the baseline structure and without the readability factor in the fitness function is shown in Figure 6.8. The features are almost the same, but the structure of the latter representative is more complicated and the relationship between the terms is quite difficult to understand.

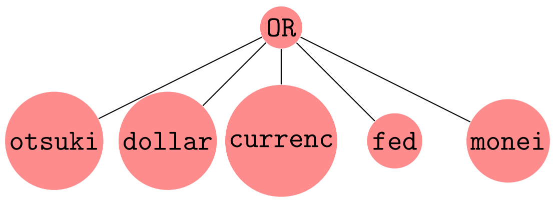


Figure 6.7: The class representative learned by the algorithm with the new tree structure and feature selection methods for class `money-fx`.

6.6 Summary

In this chapter we started to build upon our system we had introduced and explained in the previous chapter. Every aspect of the learning process was explored to achieve the aim of the thesis, which is generating a readable class representative that has a good

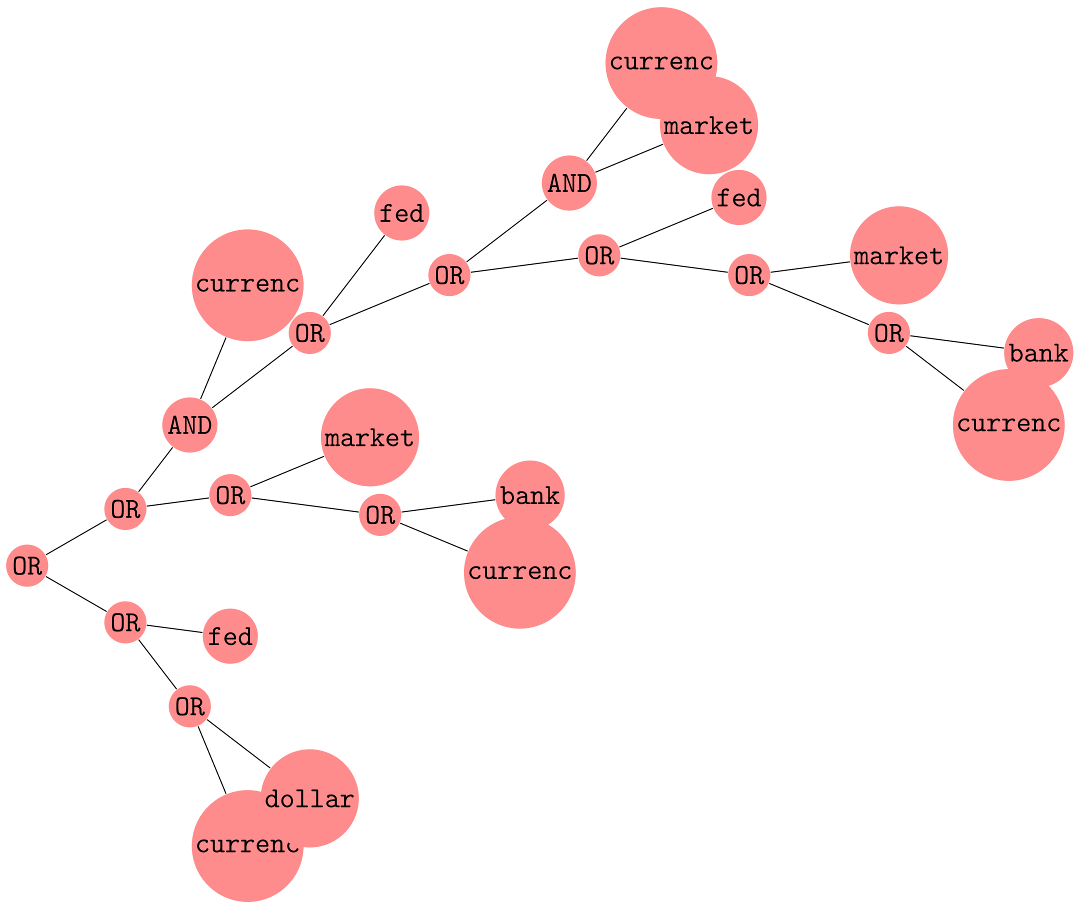


Figure 6.8: The class representative for class `money-fx`, learned by the baseline structure with feature selection.

classification performance.

First we examined the importance of the genetic programming parameters in the learning process and showed that higher population sizes or more evolving generations can lead to better representatives, however there is a limit in their ability to improve the quality.

We introduced a new tree structure, including several functions that simplified the shape of the individuals, while retaining the performance and even improving it by generating more general representatives.

To guide the learning process in selecting discriminant terms, we examined several feature selection methods and weighting schemes and concluded that almost all of them are relatively useful in helping the random feature selection of the GP to select the most informative terms. We also added a component to the fitness function that estimates the number of functions and terminals in an individual as a measure for its readability.

Finally, we combined all the factors that were discussed in this chapter to build a new system that substantially outperformed the baseline. The results were presented step by step to demonstrate the effect of each factor.

CHAPTER 7

Conclusions and Future Work

7.1 Introduction

This chapter concludes the thesis and outlines the main conclusions drawn from the research. It also provides a list of contributions made during this work. Additionally, some of the applications of the document representative generation in information retrieval are discussed. Finally new directions and possibilities for future research are mentioned.

7.2 Conclusions

The main focus of this thesis was designing and developing an evolutionary based algorithm to generate a representative for a set of documents that can later be used to classify, filter and retrieve similar documents. The aim of the research was to produce the representative in a readable form that can be read, understood and even modified by humans.

The algorithm was extended to perform the classification task and several novel methods were proposed to improve on different aspects of the algorithm. Experiments were carried out on each component and they were investigated to generate representatives with higher quality and better readability.

We first introduced information retrieval and evolutionary algorithms then reviewed previous work based on evolutionary algorithms in information retrieval and related fields. Most of the previous work has applied genetic programming to information retrieval to improve the learning to rank functions and combining weighting schemes. The learned ranking functions in most of these works are performing as well as or better than state-of-the-art ranking models. However, because of the complex structure of these ranking models they are difficult to understand and justify. This issue led us to concentrate on the simplicity of the generated representatives and optimise the system to learn representatives that can be read and understood by humans.

In Chapter 6, we investigated different parts of the system, including the genetic programming parameters, the feature selection method, the tree structure and the fitness estimation. Experiments are done and results are reported to find the best solution for each problem in the system. Based on our results, in order to learn more readable representatives, more complex functions that construct simpler trees are very useful. We discovered a good feature selection for positive terms is more effective in improving the quality of generating class representatives than adding negative terms or phrases. Adding a component to the fitness function that assesses the quality against a validation set can be very helpful on the generalisation of the representative and its performance on future unseen data. Apart from simplifying the structure of the tree that represents the individuals, an extra component is added to the fitness function in order to estimate the readability of the individual by counting the number of functions and terminals. The new component favours simpler and shallower trees over others, so the learning algorithm is guided to produce simpler representatives.

7.3 Contributions

In this thesis, considering the drawbacks of previous work and focusing on improving both the quality and parsimony, we have provided the following contributions:

1. A machine learning system based on GP is designed and developed to learn a representative for a set of documents. The algorithm is used to perform classification and other information retrieval tasks.

Document representatives are essential for all IR systems. These representatives are used during information retrieval tasks in place of the original documents. The quality of document representatives has a direct effect on the result of classification or any other task in this field. Therefore we have implemented this system to improve the quality of document representative generation.

In this method, terms are extracted from the documents based on their discrimination power, then the GP engine combines them in order to generate tree-shaped individuals as document representatives. These representatives are evaluated against the document collection and are ranked regarding how well they match relevant documents. The representative with the highest rank on the list is considered as the best representative for the given set of documents.

2. We have defined a new tree structure and added a component to the fitness function to improve the readability of the representatives.

   One of the most important advantages of using GP for representative generation is the outcome is likely to be human readable. This readability can be degraded drastically by complex structures and duplicated terms. In the baseline system [Hirsch et al., 2007] the generated representatives have very complex structures and include many duplicated terms, which defeats the purpose of choosing GP for human readable results.

   The new structure is a simple structure which is generated by complex functions. This new tree structure forces the representative generation process to monitor not only the terms it accepts but also its combination with the rest of the tree in order to avoid any complexity and duplication. The new component in the fitness function, adds an additional weight to individuals which are simpler. Based on the new component of the fitness function, shorter representatives are more preferable compare to those with the same classification accuracy. The proposed tree structure produces shorter representatives with the same number of terms in order to fill the search space with practically redundant individuals.

3. In the previous work, the feature selection mechanisms in the learning algorithm

is implemented by employing a part of the tree to be responsible for generating the index of the feature. Not only does this method make the process and individuals very complex by adding extra branches, functions and terminals, it also increases the possibility of producing two complex individuals which result in practically the same representative. In addition, a portion of the search time is spent on assessing individuals that are essentially the same, but are different in the details of the feature selection branch. In other words, the population of potential representatives is crowded with individuals which generate the same representatives.

We have used *Ephemeral Random Constants (ERC)*[1] to eliminate this problem, also to directly connect the probability of selecting a feature to its weight, which is estimated by the feature selection method. ERC replaces the need for that part of the tree responsible for the feature selection and makes the selection of features a non-uniform random number selection.

4. By combining the state-of-the-art feature selection strategies with the genetic programming algorithm, we have created a learning process that is in nature random, however it is guided to select features that are more likely to be helpful in the final representatives. In most of the other feature selection methods, a subset of all the features are selected to be involved in the classification process. The selection process in our algorithm occurs when the learning process randomly selects different features to be included in the tree. By making the random feature selection non-uniform, we make sure that more discriminant features, based on the weight given by the feature selection method, are more likely to be selected, however other features are not entirely discarded and they are tried and assessed during the process.

[1]see Chapter 5

7.4 Applications

In the previous chapters, we have demonstrated the use of our approach to learning document representatives by GP, in classification. In this section, we propose other applications of our approach in the field of information retrieval.

7.4.1 Query Expansion

Query Expansion is the process of expanding the query terms aiming to improve the performance of the retrieval process. Query expansion can be used to improve both recall and precision. There are two main categories of query expansion: Interactive query expansion; where the query is modified using terms from the user. Automatic query expansion; where the query is expanded automatically without user involvement [Ruthven and Lalmas, 2003].

In interactive query expansion a set of terms is automatically proposed to the user to add some of them to the query, however, in automatic query expansion the terms are added to the query based on the relevance feedback from the user or blind relevance feedback. Blind RF[2] or RF without relevance information is a method of improving the ranking before showing any result to the user. The system retrieves a ranked list of the documents based on the initial query and selects the top n documents of the rank list as relevant documents. In other words, without asking the user the system assumes the top n documents are relevant. The next step is selecting a set of terms from these documents and modify the initial query to improve the results. There have been several studies on selecting and modifying the query [Xu and Croft, 1996; Ogilvie et al., 2009; Chirita et al., 2007], including those that select terms from ontologies such as *WordNet* to add to the query [Bhogal et al., 2007].

In the last a few chapters, we described a GP-based approach to learn a representative for a set of documents. This approach can be applied to query expansion by considering the relevant documents as positive examples and non-relevant documents as negative examples. Our GP-based algorithm searches to find the best representation of the

[2] Also called: pseudo or ad-hoc relevance feedback.

relevant documents, either selected by the user or selected through blind Rf. The only difference here is adding the original query terms with high weights to the positive term list, so there will be a very high chance they will be in the generated representative, which will then be used as a query.

7.4.2 Routing, Content-Based Filtering

A filtering system monitors a stream of incoming information to find documents relevant to user's information need represented by profiles. During the filtering process documents arrive over time, however filtering is different from ad-hoc retrieval [Robertson and Callan, 2005]. In ad-hoc retrieval, the document collection remains, but the user's information need can change over time. In contrast, while filtering and routing, profiles are persistent and reflect a long-term information need.

The filtering/routing tasks can be seen as text categorisation and liable to use methods of machine learning. However, there are some significant differences between routing/filtering and text categorisation. In filtering, the system analyses the incoming documents and make a binary decision to retrieve or not to retrieve each document based on a profile. The routing system assigns retrieval scores to the incoming documents according to profiles and the final output is the top n ranked documents. In the filtering literature, the word *profile* refers to all of the information the system has obtained about a particular information need. Profiles are persistent which means a user may have the same information need for a period of time. With feedback from users, the filtering system can learn a better profile for particular information need [Robertson and Callan, 2005].

If we use the user's profile as our training set, we can learn a representative of the profile by our GP-based system and use it to assign scores to incoming documents. In filtering, based on the score, we make a binary decision of retrieving or not. For routing the score will be used to rank the documents and finally retrieve the top n documents.

7.5 Future Work

7.5.1 Combining Evidence

Modern information retrieval systems use multiple sources of evidence and document representative in performing the retrieval task [Tsikrika and Lalmas, 2004; Silva et al., 2009]. In structured document retrieval [Tsikrika and Lalmas, 2004], known-item finding [Ogilvie and Callan, 2003; Yahyaei and Monz, 2008] and web search engines [Silva et al., 2009]. Our method can be extended to combine evidence from different fields and parts of the document and use an additional function to assign weights to each source of evidence. This method can be applied to structured documents or web documents classification.

7.5.2 WordNet

WordNet[3] is a lexical database for English language [Miller, 1995]. Among many other features, it contains synonyms of words that can easily be found. By adding a function that finds a word from WordNet[4] to the structure of genetic programming individuals, we can improve the performance of our system by including words that are not in the training collection.

7.5.3 Regular Expressions and Stemming

In the current work, we focused on simple representatives which are readable and are good in generalisation. This work can be extended to include functions and terminals that use regular expressions to match the terms.

7.5.4 Other Languages and Collections

In general, the approach in this thesis can be applied to other collections and languages, however, testing the system on other collections and collections with languages other

[3]WordNet can be accessed from `http://wordnet.princeton.edu`

[4]It can select from a subset of synonyms of the current words in the positive list.

than English can show the effectiveness of the approach. For some languages that have a more complex morphology than English, adding terminals that match the stems of the words can be beneficial.

7.5.5 Novelty Detection and Topic Detection and Tracking Experiments

Among the problems that this work can be applied to are novelty detection [Allan et al., 1998b, 2000] and topic detection and tracking [Allan et al., 1998a; Allan, 2002]. Generated representatives can be used to make decision for the new events or finding the correct topic of the new documents, which are basically news stories.

APPENDIX A

Program Structure

The following figures show an overall design of the Java program to index, train and test the document representatives and perform the classification with them. Figures A.1, A.2 and A.4 show a static view of the main applications, genetic programming components and the feature selection methods and their hierarchy. Figure A.3 shows the structure of the indexing and feature extraction classes in addition to the way they interact with each other.

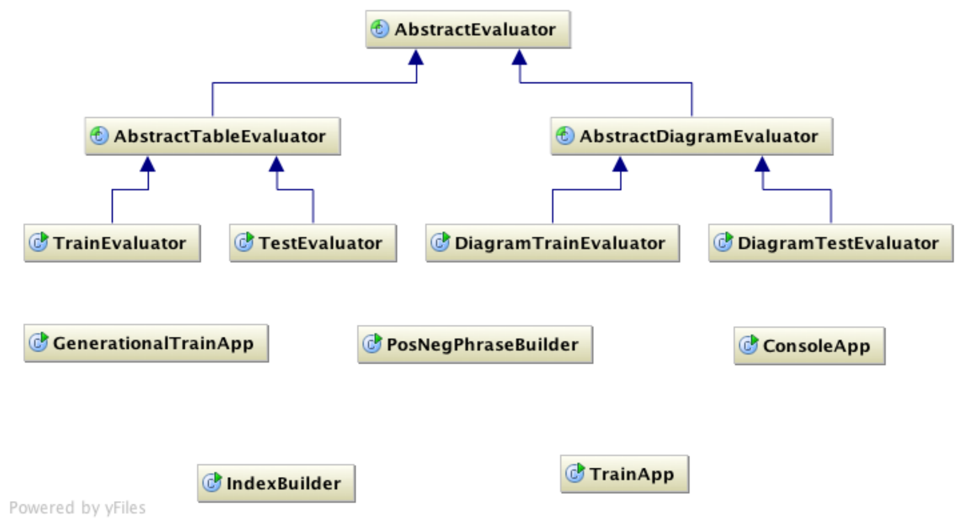


Figure A.1: Indexing, training and evaluation entry points of the application.

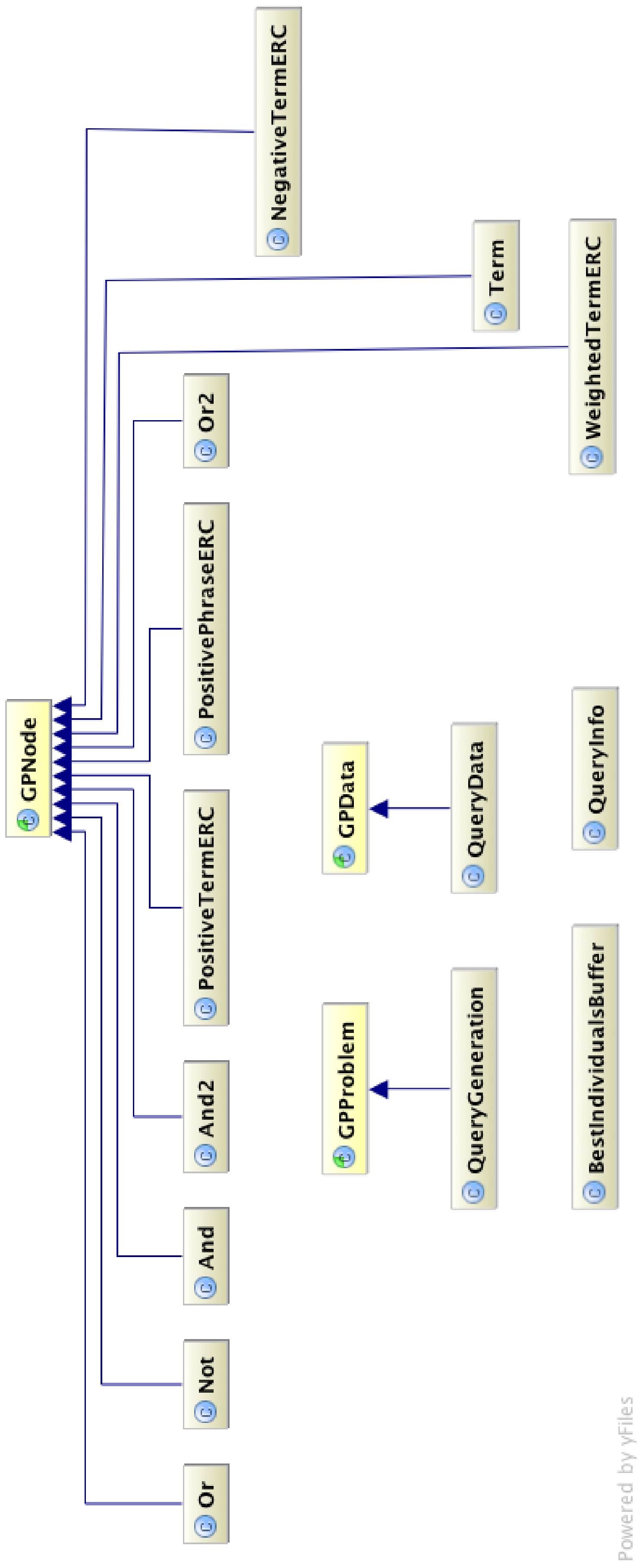


Figure A.2: Classes that define the functions, terminals and the GP problem.

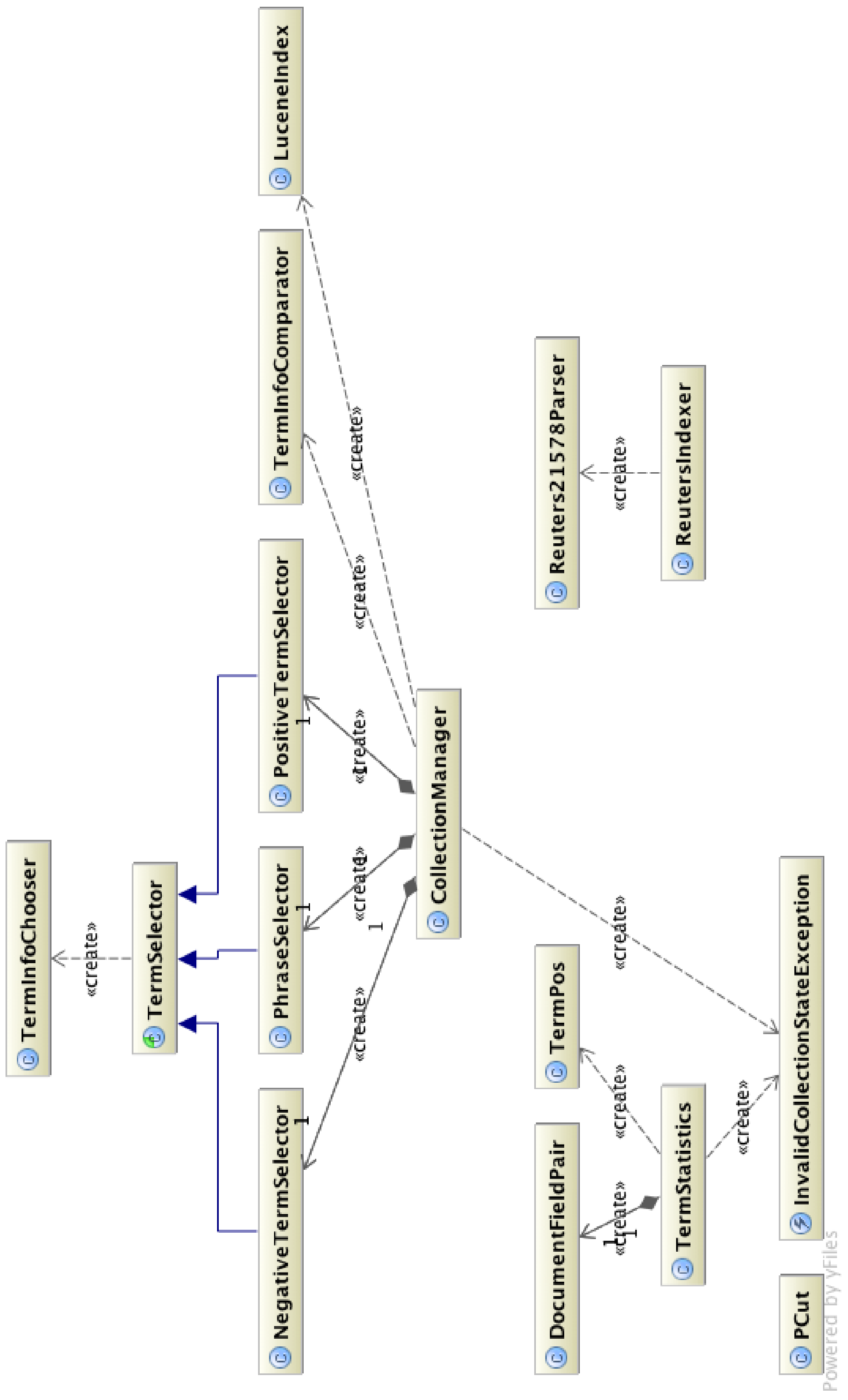


Figure A.3: Parsing, indexing main classes and general classes for feature selection.

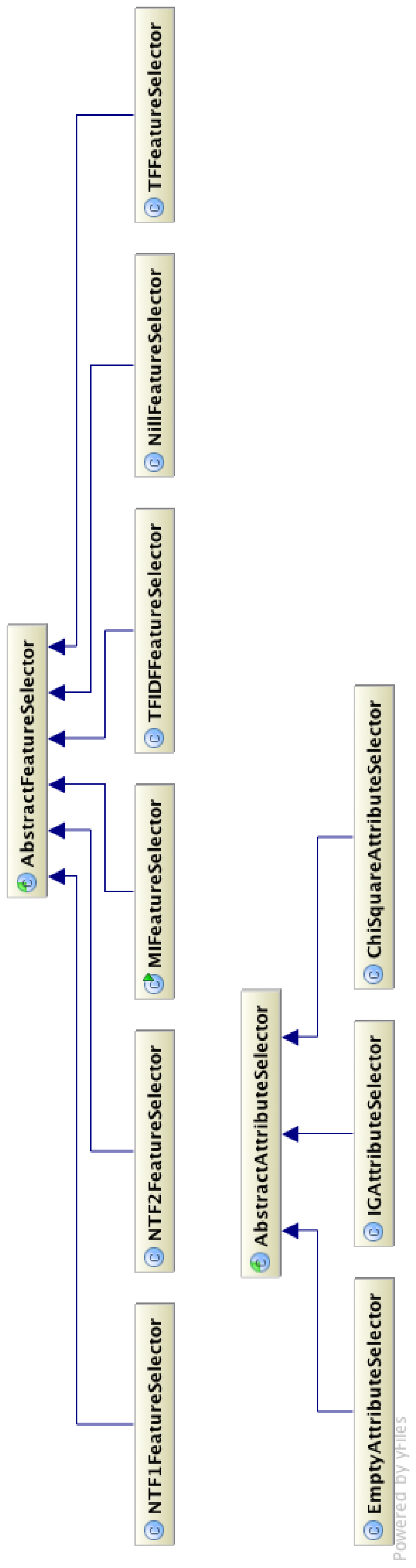


Figure A.4: Different feature selection and weighting methods.

APPENDIX B

Tools and Libraries

The classification system is implemented entirely in Java programming language, which makes it possible to be run on all major platforms. In developing the system, we used the following open-source libraries:

- **ECJ**[1] is an evolutionary computation library with support with almost all evolutionary algorithms. ECJ supports many different representations, selection and reproduction operators. It has a complete package for writing genetic programming based applications. It supports multi-threading, hierarchical parameter configuration and master/slave evaluation. ECJ is publicly available and can be downloaded from `http://www.cs.gmu.edu/~eclab/projects/ecj`. ECJ is licensed under Academic Free Licence, version 3.0 [2].

- **Apache Lucene** is a high performance, cross-platform text search engine. Lucene provides a lot of features such as high-performance indexing, field-searching and complex queries through a simple and well-designed API. There are many projects based on Lucene,extending its functionalities for crawling (*Nutch*) and search server (*Solr*). Lucene is supported by Apache Software Foundation[3] and can be downloaded from Apache's web-site. It is an open-source project and released under the Apache Software License[4].

[1] A Java-based Evolutionary Computation Research System
[2] `http://www.opensource.org/licenses/academic.php`
[3] `http://www.apache.org`
[4] `http://www.apache.org/licenses/LICENSE-2.0`

- **Apache Commons** is another project of Apache Software Foundation, which aims to provide reusable, open-source Java components. The packages of Apache Commons are independent and unrelated to any other package or product and are created for small purposes. Some of the components are:

  – *Collections*, which extends Java collection frameworks.

  – *IO*, which is a collection of I/O utilities.

  – *Lang*, which provides extra functionality for classes in `java.lang` package.

  – *Pool*, which is a generic object pooling component.

  Apache Commons is also released under Apache Software License and is available for download from `http://commons.apache.org`.

APPENDIX C

Experimental Results on the 20 Newsgroups Data Set

The *20 Newsgroups* data set is a collection of nearly 20,000 newsgroup documents, labelled by 20 categories [Lang, 1995]. It has been used widely for classification experiments. The names of the classes are shown in Table C.1 and there are 18,846 documents in our collection in the training and test sets. Table C.2 shows the results of the experiments explained in Section 6.5 on the 20 Newsgroup collection.

		F-measure		No. Terms		No. Functions	
	Category	**Baseline**	**New**	**Baseline**	**New**	**Baseline**	**New**
1	alt.atheism	0.2371	0.4306	11	5	12	4
2	comp.graphics	0.0955	0.4825	15	5	16	4
3	comp.os.ms-windows.misc	0.1698	0.5298	13	4	12	3
4	comp.sys.ibm.pc.hardware	0.0843	0.4110	12	5	11	4
5	comp.sys.mac.hardware	0.1483	0.5838	10	5	11	4
6	comp.windows.x	0.0648	0.6030	13	5	12	4
7	misc.forsale	0.0993	0.6884	17	2	21	1
8	rec.autos	0.0630	0.6303	10	3	9	2
9	rec.motorcycles	0.1536	0.8119	18	3	20	2
10	rec.sport.baseball	0.1818	0.7407	17	5	19	4
11	rec.sport.hockey	0.2787	0.6566	12	5	11	4
12	sci.crypt	0.1752	0.8286	15	5	18	4
13	sci.electronics	0.2016	0.3138	17	5	17	4
14	sci.med	0.1395	0.4816	9	5	8	4
15	sci.space	0.0868	0.6102	13	5	13	4
16	soc.religion.christian	0.1576	0.7154	19	5	18	4
17	talk.politics.guns	0.1930	0.6243	18	4	17	3
18	talk.politics.mideast	0.3715	0.6695	17	5	16	4
19	talk.politics.misc	0.0926	0.4221	10	5	10	4
20	talk.religion.misc	0.0657	0.3702	14	5	13	4

Table C.1: The results of the classification for all categories of the "20 Newsgroups" collection separately for the baseline and the new run, which is a baseline combined with all the improvements discussed in this thesis.

	Run	Avg terminals	Avg functions	μF_1	MF_1	μ-**Avg** π	M-**Avg** π	μ-**Avg** ρ	M-**Avg** ρ
1	**Baseline**	14.00	14.20	0.1443	0.1530	0.1220	0.1895	0.1766	0.1754
2	**+New Structure**	3.45	2.45	0.4925	0.4999	0.4429	0.4930	0.5547	0.5506
3	**+Feature Weighting**	4.10	3.10	0.5379	0.5476	0.4946	0.5552	**0.5895**	**0.5858**
4	**+Feature Selection**	5.00	4.00	**0.6052**	**0.5951**	**0.6318**	**0.6482**	0.5807	0.5738
5	**+Readability Factor**	4.55	3.55	0.5917	0.5802	0.6124	0.6210	0.5724	0.5648

Table C.2: The results of the classification for all the 20 categories of the "20 Newsgroups" collection for the baseline and the new run with the breakdown of the contribution of each factor. μ is micro-averaging, M is macro-averaging, π is precision and ρ is recall.

Appendix D

Example Representatives

Examples of representatives produced by the learning process described in this thesis. For each class the representatives learned by the baseline (described in Chapter 5, and denoted by Baseline) and the improved version (described in Chapter 6, denoted by Improved) are shown. The baseline examples are taken from the experiment that is explained in Section 5.9 and the improved representatives are taken from the experiments that are explained in Section 6.5.

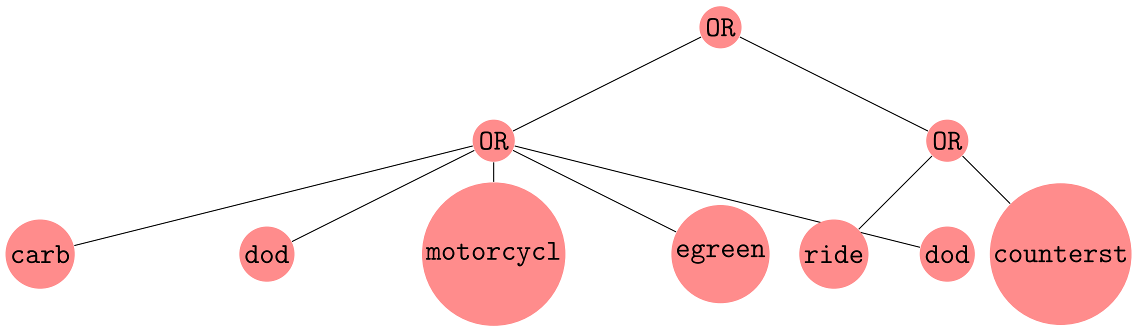


Figure D.1: Improved: The representative learned for category `rec.motorcycles`.

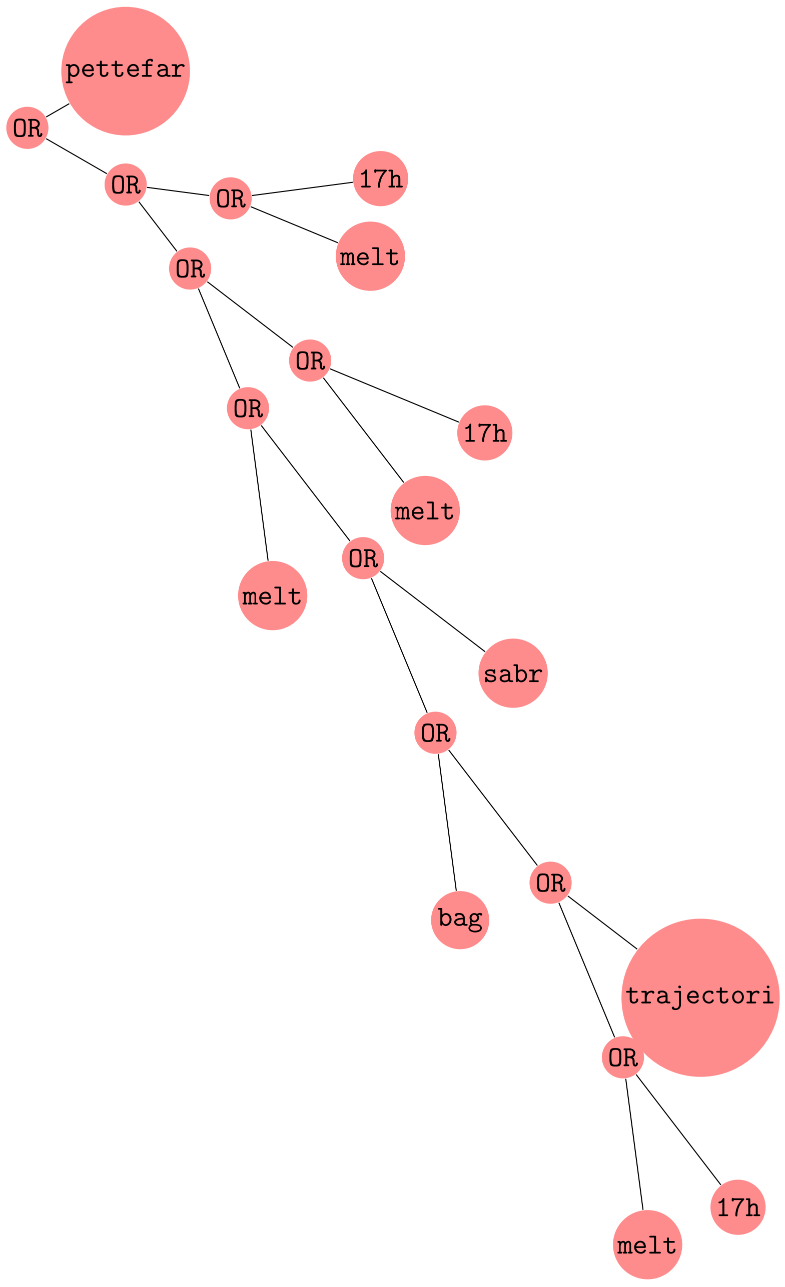


Figure D.2: Baseline: The representative learned for category `rec.motorcycles`.

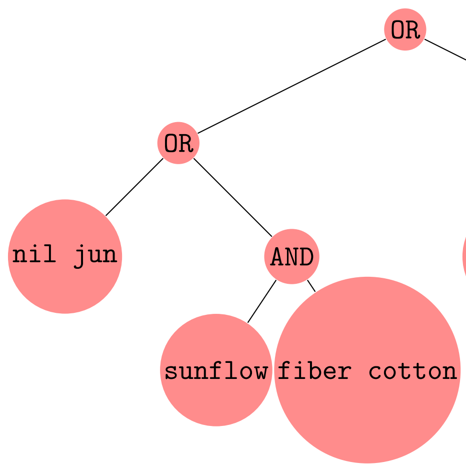


Figure D.3: IMPROVED: The representative learned for category sunseed.

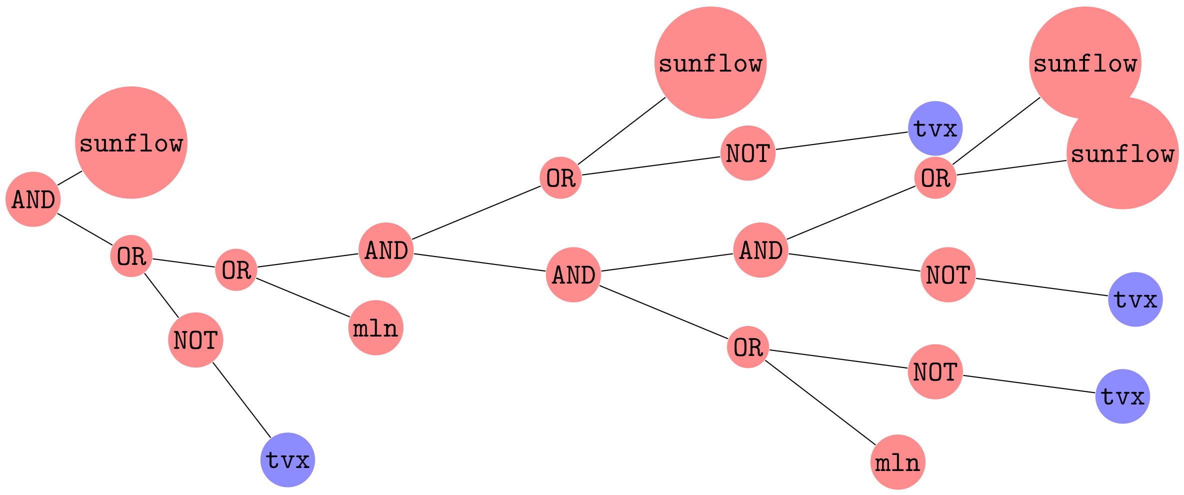


Figure D.4: BASELINE: The representative learned for category sunseed.

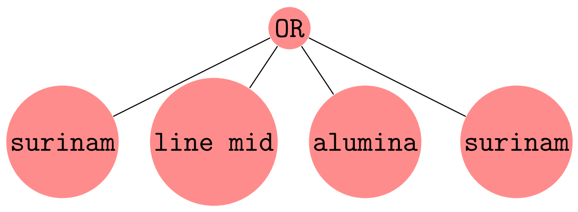


Figure D.5: IMPROVED: The representative learned for category `alum`.

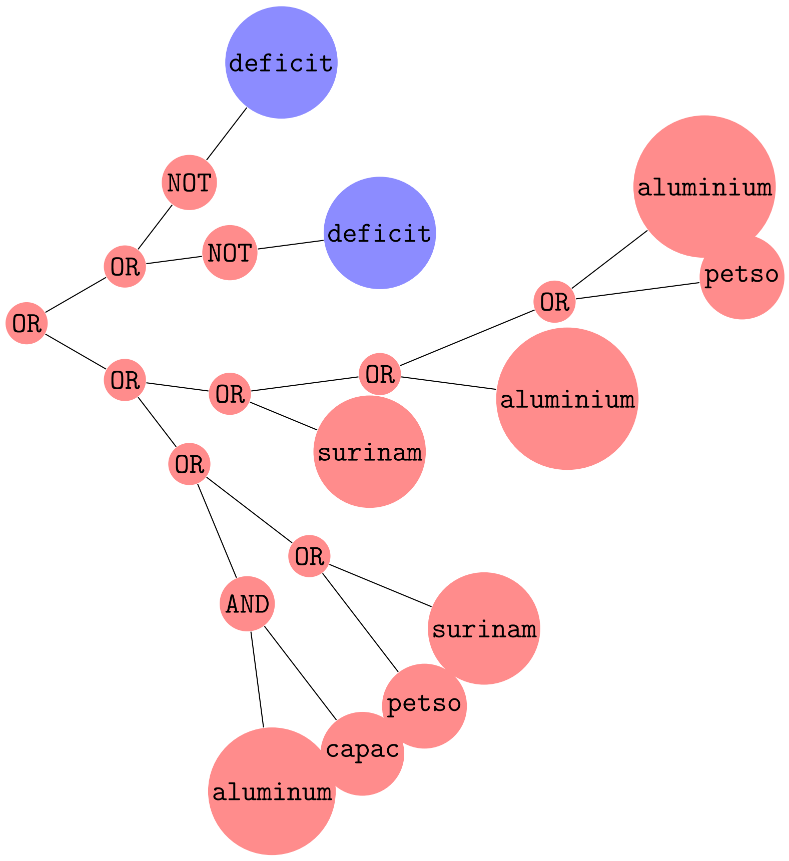


Figure D.6: BASELINE: The representative learned for category `alum`.

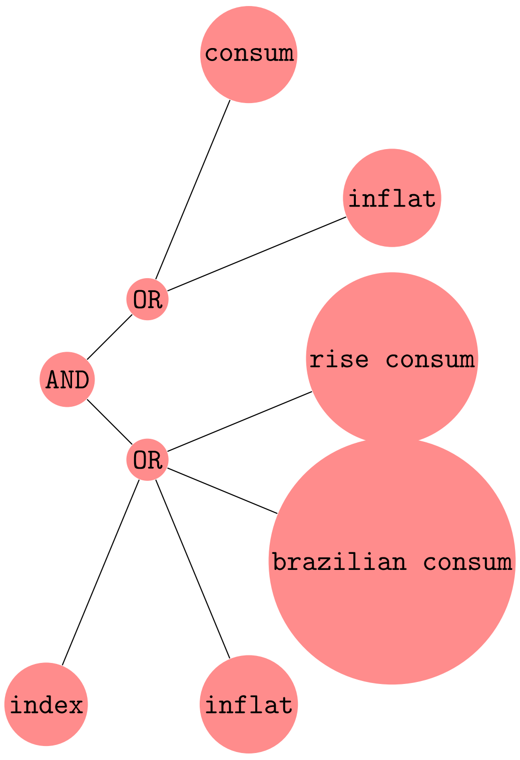


Figure D.7: IMPROVED: The representative learned for category `cpi`.

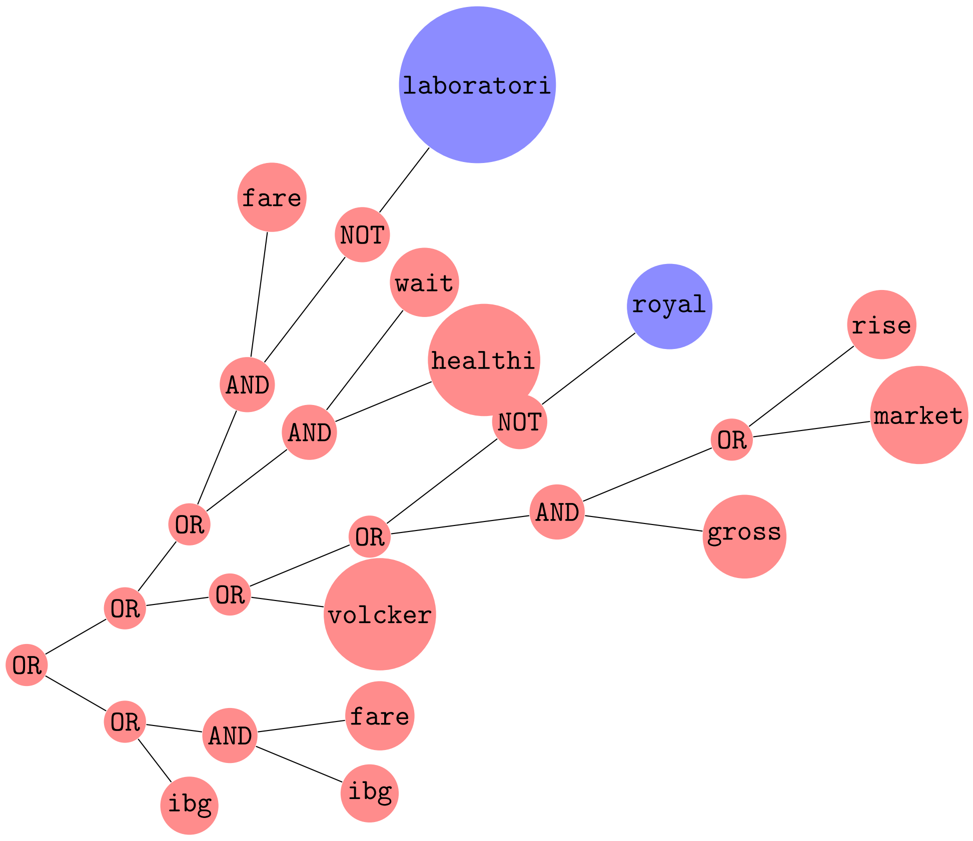


Figure D.8: BASELINE: The representative learned for category `cpi`.

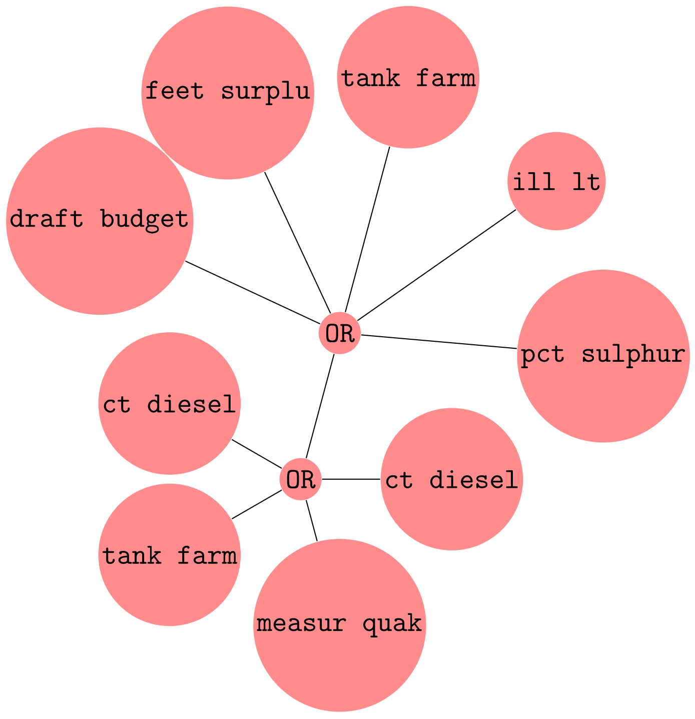


Figure D.9: Improved: The representative learned for category `fuel`.

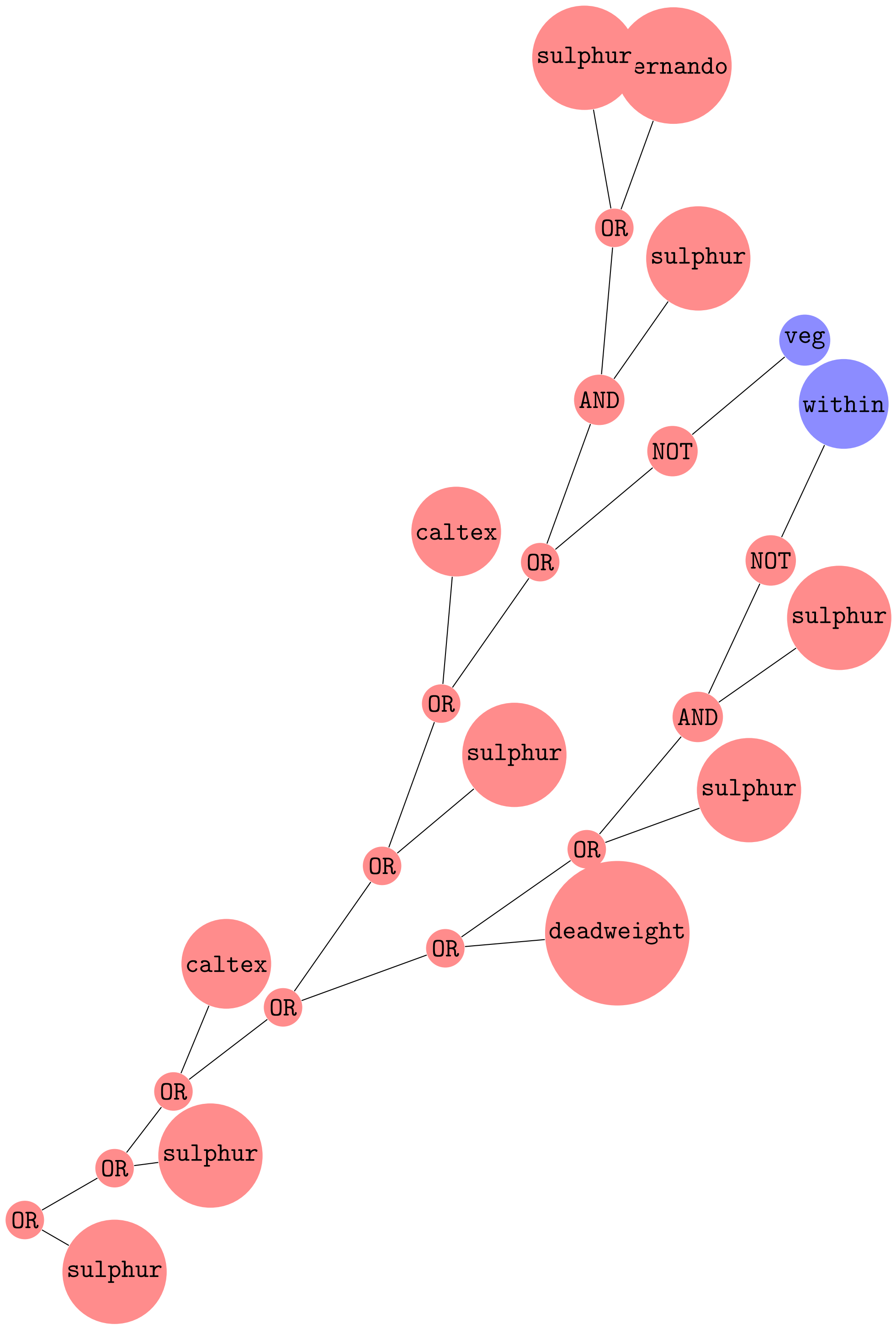


Figure D.10: BASELINE: The representative learned for category `fuel`.

Bibliography

J. Allan. *Introduction to topic detection and tracking*, pages 1–16. Kluwer Academic Publishers, Norwell, MA, USA, 2002. ISBN 0-7923-7664-1.

J. Allan, J. Carbonell, G. Doddington, J. Yamron, and Y. Yang. Topic detection and tracking pilot study: Final report. In *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, pages 194–218, 1998a.

J. Allan, R. Papka, and V. Lavrenko. On-line new event detection and tracking. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, pages 37–45, Melbourne, Australia, 1998b. ACM. ISBN 1-58113-015-5.

J. Allan, V. Lavrenko, and H. Jin. First story detection in TDT is hard. In *Proceedings of the ninth international conference on information and knowledge management*, CIKM '00, pages 374–381, McLean, Virginia, United States, 2000. ACM. ISBN 1-58113-320-0.

C. Apté, F. Damerau, and S. M. Weiss. Automated learning of decision rules for text categorization. *ACM Trans. Inf. Syst.*, 12(3):233–251, 1994a. ISSN 1046-8188.

C. Apté, F. Damerau, and S. M. Weiss. Towards language independent automated learning of text categorization models. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '94, pages 23–30, Dublin, Ireland, 1994b. Springer-Verlag New York, Inc. ISBN 0-387-19889-X.

L. Araujo. Multiobjective genetic programming for natural language parsing and tagging. In T. P. Runarsson, H.-G. Beyer, E. Burke, J. J. Merelo-Guervos, L. D. Whitley, and X. Yao, editors, *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *LNCS*, pages 433–442, Reykjavik, Iceland, 9-13 Sept. 2006. Springer-Verlag. ISBN 3-540-38990-3.

L. Araujo. How evolutionary algorithms are applied to statistical natural language processing. In *Artificial Intelligence Review*, volume 28, pages 275–303, December 2007.

L. Araujo and J. Pérez-Iglesias. Training a classifier for the selection of good query expansion terms with a genetic algorithm. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

L. Araujo, H. Zaragoza, J. R. Pérez-Agüera, and J. Pérez-Iglesias. Structure of morphologically expanded queries: A genetic algorithm approach. *Data Knowl. Eng.*, 69(3): 279–289, 2010.

T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Evolutionary Computation 2: Advanced Algorithms and Operators*. Institute of Physics Publishing, Bristol, 2000. contains excerpts from the Handbook of Evolutionary Computation.

R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, Boston, MA, USA, 1999. ISBN 0-201-39829-X.

Y. Bai. *Data Stream Processing and Query Optimization Techniques*. PhD thesis, University of California, Los Angeles, 2007.

P. Bailey, N. Craswell, I. Soboroff, and A. P. de Vries. The CSIRO enterprise search test collection. *SIGIR Forum*, 41(2):42–45, 2007.

S. Barresi, S. Nefti, and Y. Rezgui. A concept based indexing approach for document clustering. In *Proceedings of Second IEEE International Conference on Semantic Computing*, ICSC '08, pages 26–33, Santa Clara, CA, USA, August 2008.

S. Basu, I. Davidson, and K. L. Wagstaff, editors. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. CRC Press, 2008.

A. Bergström, P. Jaksetic, and P. Nordin. Enhancing information retrieval by automatic acquisition of textual relations using genetic programming. In *Proceedings of the 5th* *international conference on Intelligent user interfaces*, IUI '00, pages 29–32, New Orleans, Louisiana, United States, 2000. ACM.

J. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum, New York, 1981.

J. Bhogal, A. Macfarlane, and P. Smith. A review of ontology based query expansion. *Inf. Process. Manage.*, 43(4):866–886, 2007. ISSN 0306-4573.

C. C. Bojarczuk, H. S. Lopes, and A. A. Freitas. Discovering comprehensible classification rules by using genetic programming: a case study in a medical domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '99, pages 953–958, Orlando, Florida, USA, July 1999.

C. C. Bojarczuk, H. S. Lopes, and A. A. Freitas. Genetic programming for knowledge discovery in chest-pain diagnosis. *IEEE Engineering in Medicine and Biology Magazine*, 19(4):38–44, July-Aug. 2000.

C. L. Borgman, S. G. Hirsh, and J. Hiller. Rethinking online monitoring methods for information retrieval systems: from search product to search process. *J. Am. Soc. Inf. Sci.*, 47(7):568–583, 1996. ISSN 0002-8231.

P. F. Brown, J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. A statistical approach to machine translation. *Comput. Linguist.*, 16 (2):79–85, 1990. ISSN 0891-2017.

M. F. Caropreso, S. Matwin, and F. Sebastiani. A learner-independent evaluation of the usefulness of statistical phrases for automated text categorization. In *Text Databases and Document Management: Theory and Practice*, pages 78–102. IGI Publishing, Hershey, PA, USA, 2001. ISBN 1-878289-93-4.

P. H. Carter. The Rocchio classifier and second generation wavelets. In *Proceedings of the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6576, April 2007.

W. B. Cavnar and J. M. Trenkle. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Hierarchical classification: combining Bayes with SVM. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 177–184, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2.

P. A. Chirita, C. S. Firan, and W. Nejdl. Personalized query expansion for the web. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 7–14, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-597-7.

N. Craswell and D. Hawking. Overview of the TREC 2004 web track. In *Proceedings of Text REtrieval Conference (TREC)*, 2004.

B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison Wesley, 1 edition, February 2009. ISBN 0136072240.

R. Cummins and C. O'Riordan. Evolving local and global weighting schemes in information retrieval. *Inf. Retr.*, 9(3):311–330, 2006. ISSN 1386-4564.

R. Cummins and C. O'Riordan. Using genetic programming for information retrieval: local and global query expansion. In *Proceedings of Genetic and Evolutionary Computation Conference*, GECCO '07, page 2255, London, England, UK, July 2007.

A. Dasgupta, P. Drineas, B. Harb, V. Josifovski, and M. W. Mahoney. Feature selection methods for text classification. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 230–239, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-609-7.

B. D. Davison, T. Suel, N. Craswell, and B. Liu, editors. *WSDM '10: Proceedings of the third ACM international conference on Web search and data mining*, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-889-6. 618103.

H. M. de Almeida, M. A. Gonçalves, M. Cristo, and P. Calado. A combined component approach for finding collection-adapted ranking functions based on genetic programming. In *Proceedings of the 30th Annual International ACM SIGIR Conference*, pages 399–406, Amsterdam, July 2007.

I. De Falco, A. Della Cioppa, and E. Tarantino. Discovering interesting classification rules with genetic programming. *Applied Soft Computing*, 1(4):257–269, May 2001.

S. Dominich. *The Modern Algebra of Information Retrieval*. Springer, 2008.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000. ISBN 0471056693.

S. T. Dumais and H. Chen. Hierarchical classification of web content. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 256–263, Athens, Greece, July 2000.

J. Dunn. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3:32–57, 1973.

A. S. Ee-Peng, A. Sun, E. peng Lim, and W. keong Ng. Performance measurement framework for hierarchical text. *Journal of the American Society for Information Science and Technology*, 54:1014–1028, 2003.

J. L. Elsas and S. T. Dumais. Leveraging temporal dynamics of document content in relevance ranking. In *WSDM*, pages 1–10, 2010.

J. L. Elsas, V. R. Carvalho, and J. G. Carbonell. Fast learning of document ranking functions with the committee perceptron. In *Proceedings of First ACM International Conference on Web Search and Data Mining*, pages 55–64, Stanford, USA, February 2008.

G. Escudero, L. Màrquez, and G. Rigau. Boosting applied to word sense disambiguation. In *Proceedings of the 11th European Conference on Machine Learning*, pages 129–141, London, UK, 2000.

P. Espejo, S. Ventura, and F. Herrera. A survey on the application of genetic programming to classification. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(2):121 –144, march 2010. ISSN 1094-6977.

W. Fan, M. D. Gordon, and P. Pathak. Personalization of search engine services for effective retrieval and knowledge management. In *ICIS '00: Proceedings of the twenty first international conference on Information systems*, pages 20–34, Atlanta, GA, USA, 2000. Association for Information Systems. ISBN ICIS2000-X.

W. Fan, E. A. Fox, P. Pathak, and H. Wu. The effects of fitness functions on genetic programming-based ranking discovery for web search. *J. Am. Soc. Inf. Sci. Technol.*, 55:628–636, May 2004a.

W. Fan, M. D. Gordon, and P. Pathak. A generic ranking function discovery framework by genetic programming for information retrieval. *Inf. Process. Manage.*, 40(4):587–602, 2004b. ISSN 0306-4573.

A. Faraoun, K. M. Boukelif. Genetic programming approach for multi-category pattern classification applied to network intrusions detection. *International Journal of Computational Intelligence and Applications*, 6(1):77–100, March 2006.

B. Field. Towards automatic indexing: Automatic assignment of controlled-language indexing and classification from free indexing. *Journal of Documentation*, 31:246 – 265, 1975.

G. Folino, C. Pizzuti, and G. Spezzano. A cellular genetic programming approach to classification. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1015–1020, Orlando, Florida, USA, July 1999. Morgan Kaufmann.

W. B. Frakes and R. A. Baeza-Yates. *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, 1992. ISBN 0-13-463837-9.

J. Fransson. *Efficient Information Searching on the Web: A Handbook in the Art of Searching for Information*. jonasfransson.com, 2009.

A. A. Freitas. A review of evolutionary algorithms for data mining. In *Soft Computing for Knowledge Discovery and Data Mining*, pages 79–111. 2008.

Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, 2003. ISSN 1532-4435.

S. Gao, W. Wu, C.-H. Lee, and T.-S. Chua. A MFoM learning approach to robust multiclass multi-label text categorization. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 42, New York, NY, USA, 2004. ACM. ISBN 1-58113-828-5.

S. Gauch, A. Chandramouli, and S. Ranganathan. Training a hierarchical classifier using inter document relationships. *JASIST*, 60(1):47–58, 2009.

N. Ghamrawi and A. McCallum. Collective multi-label classification. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 195–200, New York, NY, USA, 2005. ACM. ISBN 1-59593-140-6.

J. Goldstein-Stewart and R. K. Winder. Designing a system for semi-automatic population of knowledge bases from unstructured text. In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development*, KEOD '09, pages 88–99, Funchal - Madeira, Portugal, October 2009.

A. D. Gordon. A review of hierarchical classification. *Journal of the Royal Statistical Society. Series A (General)*, 150(2):pp. 119–137, 1987. ISSN 00359238.

D. A. Grossman and O. Frieder. *Information retrieval: algorithms and heuristics*. Springer, 2004.

A. Gulli. *Clustering and Ranking for Web Information Retrieval*. VDM Verlag Dr. Mueller e.K., January 2008.

A. Göker and J. Davies, editors. *Information Retrieval: Searching in the 21st Century*. Wiley Publisher, 2009.

K. Halvorsen, T. Hart, D. Islam-Frénoy, J. Puopolo, P. Wessel, and C. Zeier. *Book of Search*. FAST, 2008.

D. Harman. Ranking algorithms. In *Information Retrieval: Data Structures & Algorithms*, pages 363–392. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992a. ISBN 0-13-463837-9.

D. Harman. Relevance feedback revisited. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 1–10, New York, NY, USA, 1992b. ACM. ISBN 0-89791-523-2.

K. Hatano. A simpler analysis of the multi-way branching decision tree boosting algorithm. In *ALT '01: Proceedings of the 12th International Conference on Algorithmic Learning Theory*, pages 77–91, London, UK, 2001. Springer-Verlag. ISBN 3-540-42875-5.

M. A. Hearst and H. Hirsh. AI's greatest trends and controversies. *IEEE Intelligent Systems*, 15(1):8–17, 2000. ISSN 1541-1672.

R. Herbrich, T. Graepel, and K. Obermayer. *Large margin rank boundaries for ordinal regression*. MIT Press, Cambridge, MA, 2000.

A. Herrero, E. Corchado, C. Redondo, and A. Alonso, editors. *Computational Intelligence in Security for Information Systems 2010*. Springer, 2010.

L. Hirsch. *An Evolutionary Approach to Text Classification*. PhD thesis, School of Management Royal Holloway University of London, April 20 2005.

L. Hirsch, R. Hirsch, and M. Saeedi. Evolving lucene search queries for text classification. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1604–1611, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-697-4.

J. H. Holland. *Adaptation in Natural and Artifical Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

A. Hotho, A. Nürnberger, and G. Paass. A brief survey of text mining. *LDV Forum*, 20 (1):19–62, 2005.

I. Iliopoulos, A. J. Enright, and C. A. Ouzounis. Textquest : Document clustering of medline abstracts for concept discovery in molecular biology. In *Pacific Symposium on Biocomputing 6*, pages 384–395, 2001.

K. Imamura, T. Soule, R. B. Heckendorn, and J. A. Foster. Behavioral diversity and a probabilistically optimal GP ensemble. *Genetic Programming and Evolvable Machines*, 4:235–253, September 2003. ISSN 1389-2576.

P. Jackson and I. Moulinier. *Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorization*. John Benjamins Pub Co, June 2007.

K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002. ISSN 1046-8188.

F. Jelinek. *Statistical methods for speech recognition*. MIT Press, Cambridge, MA, USA, 1997. ISBN 0-262-10066-5.

T. Joachims. *Learning to Classify Text Using Support Vector Machines – Methods, Theory, and Algorithms*. Kluwer/Springer, 2002.

S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 2:241–254, 1967.

K. S. Jones and C. J. van Rijsbergen. Information retrieval test collections. *Journal of Documentation*, 32(1):59–75, 1976.

K. S. Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: development and comparative experiments. *Inf. Process. Manage.*, 36:779–808, November 2000. ISSN 0306-4573.

D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition (2ed.)*. Prentice Hall, 2008.

B. Kessler, G. Nunberg, and H. Schütze. Automatic detection of text genre. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 32–38, Madrid, Spain, 1997.

J. R. Koza. Concept formation and decision tree induction using the genetic programming paradigm. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, volume 496 of *Lecture Notes in Computer Science*, pages 124–128, Dortmund, Germany, October 1991. Springer-Verlag.

J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-11170-5.

W. Kraaij and R. Pohlmann. Comparing the effect of syntactic vs. statistical phrase indexing strategies for Dutch. In *Proceedings of Research and Advanced Technology for Digital Libraries, Second European Conference*, pages 605–617, 1998.

P. Kro andmer, V. Sna ands andel, J. Platos and, and A. Abraham. Evolutionary improvement of search queries and its parameters. In *Hybrid Intelligent Systems (HIS), 2010 10th International Conference on*, pages 147 –152, aug. 2010.

H. Kuĉera and W. N. Francis. *Computational Analysis of Present-Day American English*. Brown University Press, Providence, RI, USA, 1967.

R. Kuhn. Speech recognition and the frequency of recently used words: a modified markov model for natural language. In *Proceedings of the 12th conference on Computational linguistics - Volume 1*, pages 348–350, Morristown, NJ, USA, 1988. Association for Computational Linguistics. ISBN 963 8431 56 3.

C.-S. Kuo, T.-P. Hong, and C.-L. Chen. Applying genetic programming technique in classification trees. *Soft Comput.*, 11:1165–1172, August 2007. ISSN 1432-7643.

K. Lang. NewsWeeder: learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pages 331–339. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1995.

W. B. Langdon, S. J. Barrett, and B. F. Buxton. Combining decision trees and neural networks for drug discovery. In *Proceedings of the 5th European Conference on Genetic Programming*, EuroGP '02, pages 60–70, London, UK, 2002. Springer-Verlag. ISBN 3-540-43378-3.

L. S. Larkey. Automatic essay grading using text categorization techniques. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 90–95, Melbourne, Australia, August 1998.

D. D. Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 37–50, New York, NY, USA, 1992a. ACM. ISBN 0-89791-523-2.

D. D. Lewis. Feature selection and feature extraction for text categorization. In *HLT '91: Proceedings of the workshop on Speech and Natural Language*, pages 212–217, Morristown, NJ, USA, 1992b. Association for Computational Linguistics. ISBN 1-55860-272-0.

D. D. Lewis. *Representation and Learning in Information Retrieval*. PhD thesis, Computer Science Dept.; Univ. of Massachusetts; Amherst, MA 01003, 1992c. Technical Report 91–93.

J. Li, D. Dou, S. Kim, H. Qin, and Y. Wang. On knowledge-based classification of abnormal BGP events. In *Third International Conference on Information Systems Security*, ICISS '07, pages 267–271, Delhi, India, December 2007.

J.-Y. Lin, H.-R. Ke, B.-C. Chien, and W.-P. Yang. Designing a classifier by a layered multi-population genetic programming approach. *Pattern Recogn.*, 40(8):2211–2225, 2007. ISSN 0031-3203.

R. Losee. Probabilistic retrieval and coordination level matching. *Journal of the American Society for Information Science*, 38(4):239–44, July 1987.

H. P. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research and Development*, 2(2), 1958.

S. Luke. *Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat*. PhD thesis, Department of Computer Science, University of Maryland, A. V. Williams Building, University of Maryland, College Park, MD 20742 USA, 2000.

S. Luke. *Essentials of Metaheuristics*. 2009. available at http://cs.gmu.edu/∼sean/book/metaheuristics/.

S. Luke and L. Spector. A comparison of crossover and mutation in genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1997: Proc. of the Second Annual Conf.*, pages 240–248, San Francisco, CA, 1997. Morgan Kaufmann.

J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

M. Magennis and C. J. van Rijsbergen. The potential and actual effectiveness of interactive query expansion. In *SIGIR '97: Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 324–332, New York, NY, USA, 1997. ACM. ISBN 0-89791-836-3.

C. D. Manning and H. Schuetze. *Foundations of Statistical Natural Language Processing*. The MIT Press, June 1999. ISBN 0262133601.

C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

M. E. Maron. Automatic indexing: An experimental inquiry. *J. ACM*, 8(3):404–417, 1961. ISSN 0004-5411.

B. Martins and M. J. Silva. Language identification in web pages. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, SAC '05, pages 764–768, Santa Fe, New Mexico, USA, March 2005.

C. T. Meadow, B. R. Boyce, D. H. Kraft, and C. L. Barry. *Text information retrieval systems*. Emerald Group Publishing, 2007.

G. A. Miller. WordNet: a lexical database for English. *Commun. ACM*, 38(11):39–41, 1995. ISSN 0001-0782.

T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. ISBN 978-0-07-042807-2.

M. Mitra, C. Buckley, A. Singhal, and C. Cardie. An analysis of statistical and syntactic phrases. In *Fifth RIAO Conference, Computer-Assisted Information Searching On the Internet*, pages 200–214, 1997.

D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2): 199–230, 1995.

P. Moravec. Testing dimension reduction methods for text retrieval. In *Proceedings of Databases, Texts, Specifications and Objects Workshop (DATESO)*, pages 113–124, 2005.

M. Muharram and G. D. Smith. Evolutionary constructive induction. *IEEE Trans. on Knowl. and Data Eng.*, 17:1518–1528, November 2005. ISSN 1041-4347.

K. Myers, M. J. Kearns, S. P. Singh, and M. A. Walker. A boosting approach to topic spotting on subdialogues. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 655–662, Stanford, CA, USA, June 2000.

U. Y. Nahm and R. J. Mooney. Mining soft-matching rules from textual data. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 979–986, Seattle, Washington, USA, August 2001.

C. D. Nguyen, T. A. Dung, and T. H. Cao. Text classification for DAG-structured categories. In *Proceedings of Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference*, pages 290–300, Hanoi, Vietnam, May 2005.

K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. In *IJCAI-99 Workshop on Machine Learning for Information Filtering*, pages 61–67, 1999.

A. Nouri and M. L. Littman. Dimension reduction and its application to model-based exploration in continuous spaces. *Machine Learning*, 81(1):85–98, 2010.

P. Ogilvie and J. Callan. Combining document representations for known-item search. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Re-*

search and development in informaion retrieval, pages 143–150, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-646-3.

P. Ogilvie, E. Voorhees, and J. Callan. On the number of terms used in automatic query expansion. *Inf. Retr.*, 12(6):666–679, 2009. ISSN 1386-4564.

H.-J. Oh, S.-H. Myaeng, and M.-H. Lee. A practical hypertext categorization method using links and incrementally available class information. In *SIGIR*, pages 264–271, 2000.

N. Oren. Improving the effectiveness of information retrieval with genetic programming. Master's thesis, University of the Witwatersand, Johannnesburg, 2002a.

N. Oren. Reexamining tf.idf based information retrieval with Genetic Programming. In *Proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, SAICSIT '02, pages 224–234, Port Elizabeth, South Africa, 2002b. South African Institute for Computer Scientists and Information Technologists.

R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of Twenty-Fifth International Conference on Machine Learning*, ICML '08, pages 752–759, Helsinki, Finland, July 2008.

D. Pavelec, L. S. Oliveira, E. J. R. Justino, F. D. N. Neto, and L. V. Batista. Author identification using compression models. In *Proceedings of the 10th International Conference on Document Analysis and Recognition*, ICDAR '09, pages 936–940, Barcelona, Spain, July 2009.

J. Pickens and W. B. Croft. An exploratory analysis of phrases in text retrieval. In *Proceedings of Computer-Assisted Information Retrieval (Recherche d'Information et ses Applications)*, RIAO '00, pages 1179–1195, College de France, France, April 2000.

R. K. Pon and A. F. Cárdenas. iScore: measuring the interestingness of articles in a limited user environment. In *IEEE Symposium on Computational Intelligence and Data Mining*, 2007.

J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281, New York, NY, USA, 1998. ACM. ISBN 1-58113-015-5.

M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130 –137, 1980.

M. Porter. *Readings in Information Retrieval*, chapter An algorithm for suffix stripping, pages 313–316. Morgan Kaufmann, 1997.

X. Qi and B. D. Davison. Web page classification: Features and algorithms. *ACM Comput. Surv.*, 41(2), 2009.

R. Rehurek and M. Kolkus. Language identification on the web: Extending the dictionary method. In *Proceedings of Computational Linguistics and Intelligent Text Processing, 10th International Conference*, CICLing '09, pages 357–368, Mexico City, Mexico, March 2009.

I. Robertson, Stephen. E. Soboroff. The TREC 2002 filtering track report. In *TREC*, 2002a.

P. Robertson, S.E. Harding. Probabilistic automatic indexing by learning from human indexers. *Journal of Documentation*, 40:264 – 270, 1984.

S. Robertson. Comparing the performance of adaptive filtering and ranked output systems. *Inf. Retr.*, 5(2-3):257–268, 2002b. ISSN 1386-4564.

S. Robertson and J. Callan. Routing and filtering. In E. Voorhees and D. Harman, editors, *TREC: Experiments and Evaluation in Information Retrieval*, pages 99–121. The MIT Press, 2005.

S. Robertson, S. Walker, M. M. Beaulieu, M. Gatford, and A. Payne. Okapi at TREC-4. In *NIST Special Publication 500-236: The Fourth Text REtrieval Conference (TREC-4)*, pages 73–96, Gaithersburg, MD, 1995.

S. E. Robertson and S. K. Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

S. E. Robertson, C. J. van Rijsbergen, and M. F. Porter. Probabilistic models of indexing and searching. In *Information Retrieval Research, Proc. Joint ACM/BCS Symposium in Information Storage and Retrieval*, SIGIR '80, pages 35–56, Cambridge, England, June 1980.

I. Ruthven and M. Lalmas. A survey on the use of relevance feedback for information access systems. *The Knowledge Engineering Review*, 18(2):95–145, 2003. ISSN 0269-8889.

I. Ruthven, A. Tombros, and J. M. Jose. A study on the use of summaries and summary-based query expansion for a question-answering task. In *Proceedings of the 23rd Annual BCS European Colloquium on Information Retrieval Research (ECIR 2001)*, pages 41–53, Darmstadt, Germany, 2001.

C. L. Sable and V. Hatzivassiloglou. Text-based approaches for non-topical image categorization. *Int. J. on Digital Libraries*, 3(3):261–275, 2000.

G. Salton. *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.

G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989. ISBN 0-201-12227-8.

G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.

G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986. ISBN 0070544840.

G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

R. E. Schapire and Y. Singer. BoosTexter: a boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

F. Sebastiani. Text categorization. In L. C. Rivero, J. H. Doorn, and V. E. Ferraggine, editors, *The Encyclopedia of Database Technologies and Applications*, pages 683–687. Idea Group Publishing, Hershey, US, 2005.

F. Sebastiani and C. N. D. Ricerche. Machine learning in automated text categorization. *ACM Computing Surveys*, 34:1–47, 2002.

M. Segond, C. Fonlupt, and D. Robilliard. Genetic programming for protein related text classification. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 1099–1106, 2009.

T. P. C. Silva, E. S. de Moura, J. M. B. Cavalcanti, A. S. da Silva, M. G. de Carvalho, and M. A. Gonçalves. An evolutionary approach for combining different sources of evidence in search engines. *Inf. Syst.*, 34(2):276–289, 2009.

S. E. Sim, S. M. Easterbrook, and R. C. Holt. Using benchmarking to advance research: A challenge to software engineering. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '05, pages 74–83, Portland, Oregon, USA, May 2003.

D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal. A survey of controlled experiments in software engineering. *IEEE Trans. Software Eng.*, 31(9):733–753, 2005.

A. F. Smeaton and F. Kelledy. User-chosen phrases in interactive query formulation for information retrieval. In *BCS-IRSG Annual Colloquium on IR Research*, 1998.

D. Soboroff, Ian. Harman. Novelty detection: The TREC experience. In *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, HLT/EMNLP '05, pages 105–112, Vancouver, British Columbia, Canada, October 2005.

K. Sparck-Jones. *Automatic keyword classification for information retrieval*. Archon Books, 1971.

K. Sparck-Jones and P. Willett. *Readings in Information Retrieval*. Morgan Kaufmann, 1997.

A. Spoerri. *InfoCrystal: A Visual Tool For Information Retrieval*. PhD thesis, MIT, 1995.

K. Sun, Y. Cao, X. Song, Y.-I. Song, X. Wang, and C.-Y. Lin. Learning to recommend questions based on user ratings. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*, pages 751–758, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-512-3.

L. Tang, S. Rajan, and V. K. Narayanan. Large scale multi-label classification via metalabeler. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 211–220, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4.

D. M. J. Tax and R. P. W. Duin. Using two-class classifiers for multiclass classification. In *Proceedings of 16th International Conference on Pattern Recognition*, ICPR '02, pages 124–127, Quebec, Canada, August 2002.

W. F. Tichy. Should computer scientists experiment more? *IEEE Computer*, 31(5):32–40, 1998.

A. Trotman. Learning to rank. *Information Retrieval*, 8(3):359–381, 2005. ISSN 1386-4564.

T. Tsikrika and M. Lalmas. Combining evidence from web retrieval using the inference network model - an experimental study. *Information Processing & Management, Special Issue in Bayesian Networks and Information Retrieval*, 40(5):751–772, September 2004. ISSN 0306-4573.

D. Tufis, R. Ion, and N. Ide. Fine-grained word sense disambiguation based on parallel corpora, word alignment, word clustering and aligned wordnets. *CoRR*, abs/cs/0503024, 2005.

K. Tzeras and S. Hartmann. Automatic indexing based on Bayesian inference networks. In *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 22–34, Pittsburgh, PA, USA, June-July 1993.

C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979. ISBN 0-408-70929-4.

E. M. Voorhees. Query expansion using lexical-semantic relations. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 61–69, New York, NY, USA, 1994. Springer-Verlag New York, Inc. ISBN 0-387-19889-X.

E. M. Voorhees. Overview of TREC 2005. In *Proceedings of the Fourteenth Text REtrieval Conference*, TREC '05, Gaithersburg, Maryland, November 2005.

D. Vrajitoru. Crossover improvement for the genetic algorithm in information retrieval. *Information Processing and Management*, 34:405–415, 1998.

S. Willie and P. Bruza. Users' models of the information space: the case for two search models. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 205–210, New York, NY, USA, 1995. ACM. ISBN 0-89791-714-6.

I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, October 1999. ISBN 1558605525.

I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

J. H. Wolfe. Object cluster analysis of social areas. Master's thesis, University of California, Berkeley., 1963.

S. K. M. Wong, W. Ziarko, and P. C. N. Wong. Generalized vector spaces model in information retrieval. In *SIGIR '85: Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 18–25, New York, NY, USA, 1985. ACM. ISBN 0-89791-159-8.

Y. Wu and D. W. Oard. Bilingual topic aspect classification with a few training examples. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 203–210, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-164-4.

J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research* *and development in information retrieval*, pages 4–11, New York, NY, USA, 1996. ACM. ISBN 0-89791-792-8.

S. Yahyaei and C. Monz. Applying maximum entropy to known-item email retrieval. In *Proceedings of the 30th European Conference on Information Retrieval*, ECIR '08, pages 406–413, Glasgow, UK, March-April 2008.

J. Yan, S. Yan, N. Liu, and Z. Chen. Straightforward feature selection for scalable latent semantic indexing. In *Proceedings of the SIAM International Conference on Data Mining*, SDM '09, pages 1159–1170, Sparks, Nevada, USA, April-May 2009.

Y. Yang. A study of thresholding strategies for text categorization. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, pages 137–145, New Orleans, Louisiana, United States, 2001. ACM. ISBN 1-58113-331-6.

Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-486-3.

Y. Yang, S. Slattery, and R. Ghani. A study of approaches to hypertext categorization. *J. Intell. Inf. Syst.*, 18(2-3):219–241, 2002.

Y. Yang, J. Zhang, and B. Kisiel. A scalability analysis of classifiers in text categorization. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '03, pages 96–103, Toronto, Canada, July-August 2003.

J.-Y. Yeh, J.-Y. Lin, H.-R. Ke, and W.-P. Yang. Learning to rank for information retrieval using genetic programming. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval (LR4IR 2007) (Jointly held with the 30th Annual International ACM SIGIR Conference)*, pages 41–48, Amsterdam, Netherlands, 2007.

D. Zhang. Evaluating user information needs through a library portal environment:

New opportunity and new challenges. *Anne Woodsworth (ed.) (Advances in Librarianship, Volume 29), Emerald Group Publishing Limited*, 29:247–268, 2005.

H. Zhang. The optimality of naive Bayes. In V. Barr and Z. Markov, editors, *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference*, Miami Beach, Florida, USA, 2004. AAAI Press.

L. Zhang, L. B. Jack, and A. K. Nandi. Fault detection using genetic programming. *Mechanical Systems and Signal Processing*, 19(2):271 – 289, 2005. ISSN 0888-3270.

L. Zhao, M. Zhang, and S. Ma. The nature of novelty detection. *Inf. Retr.*, 9(5):521–541, 2006.

A. Zimek, F. Buchwald, E. Frank, and S. Kramer. A study of hierarchical and flat classification of proteins. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 99(PrePrints), 2008. ISSN 1545-5963.

J. Zobel and A. Moffat. Exploring the similarity space. *SIGIR Forum*, 32(1):18–34, 1998. ISSN 0163-5840.