School of Computing

University of Eastern Finland

Monika Machunik

Ontology-aided Business Document Classification

IMPIT program

Thesis supervisor: Tuomo Kakkonen

24th November 2012, Joensuu

Abstract

Business Intelligence (BI) systems provide information and assistance for business decision-makers, aiding in increasing company revenue, reducing the costs, and lowering the risks. *Text classification* (TC) is one of tasks performed by such systems. In this thesis we give an overview of current TC methods. We focus in particular on ontology-aided methods. This is due to the nature of the target domain in which the results of this thesis will be applied. The term ontology-aided refers to the fact that these methods utilise information about the target domain contained in an ontology. A method relying on information about category hierarchy is called a hierarchical categorisation method, as opposed to flat categorisation where no hierarchy information is used. In an extreme case a method based on ontology may not require any training data, which is referred to as training-less classification.

We compared the performance of different methods in the domain of business news articles and an ontology for storing BI. The results of the research were implemented in a TC module that was integrated into an existing BI system called *Data Analysis and Visualisation aId for Decision-making* (DAVID). The ontology used in classification is the *Company, Product and Event* (CoProE) ontology, developed for the DAVID system. Both the system and the ontology are developed at the University of Eastern Finland.

We applied the classification to a set of 405 manually collected and annotated business documents. The documents were divided into 14 categories defined by the CoProE ontology. We found out that categorisation performed with respect to ontology hierarchy resulted in 7% worse accuracy than in the case of flat categorisation. As for flat categorisation, the most accurate results were achieved by applying the k-Nearest Neighbour method (k=12), which produced 73% macro accuracy, followed by Support Vector Machine and Naive Bayes algorithms, which yielded the macro accuracies of 68% and 43%, respectively. Generating n-grams for n=8 instead of extracting words from text improved the results by almost 8%, in agreement with the literature.

ACM classification of the research topic

H.3 INFORMATION STORAGE AND RETRIEVAL

- H.3.1 Content Analysis and Indexing
- H.3.3 Information Search and Retrieval

I.2 ARTIFICIAL INTELLIGENCE

- I.2.7 Natural Language Processing

J.1 ADMINISTRATIVE DATA PROCESSING

List of Abbreviations

AI	Artificial intelligence [page 1]
BI	Business Intelligence [page 1]
BOW	Bag-of-words [page 23]
CoProE	Company, Product and Event [page 3]
DAVID	Data Analysis and Visualisation aId for Decision-making [page 2]
DM	Data mining [page 1]
FN	False negative [page 45]
FP	False positive [page 45]
GUI	Graphical user interface [page 50]
ICT	Information and communication technologies [page 1]
IDF	Inverse document frequency [page 24]
IPC	International Patent Classification [page 42]
K-NN	K-Nearest Neighbour [page 19]
MAP	Maximum a posteriori [page 28]
ML	Machine learning [page 1]

NLP	Natural language processing [page 1]
OWL	Web Ontology Language [page 9]
PR	Pattern recognition [page 2]
RDF	Resource Description Framework [page 9]
SVM	Support Vector Machine [page 18]
SW	Semantic web [page 1]
TC	Text classification [page 2]
TC	Text categorisation [page 2]
TF	Term frequency [page 24]
TN	True negative [page 45]
TP	True positive [page 45]
UNSPSC	United Nations Standard Products and Services Code [page 3]
URI	Uniform Resource Identifier [page 9]
W3C	World Wide Web Consortium [page 9]

Acknowledgements

I would like to thank my supervisor Dr Tuomo Kakkonen for providing information and support during writing this thesis.

This work was supported by the „Towards e-leadership: higher profitability through innovative management and leadership systems” project, which is funded by the European Regional Development Fund and Tekes – the Finnish Funding Agency for Technology and Innovation.

Contents

1. Introduction

Information and communication technologies (ICT) are present in almost all aspects of our lives. ICT is a constantly evolving domain that aims at automating and supporting various human activities. Nowadays, not only manual human labour is being replaced by computer systems, but also intellectual work can be simulated by computers. Current trends in Computer Science include automating tasks requiring assessment, reasoning and knowledge which originally could be done only by human being. Branches of Computer Science such as *artificial intelligence* (AI), *machine learning* (ML), *data mining* (DM), *natural language processing* (NLP), as well as *semantic web* (SW), provide methods which can support carrying out intellectual tasks. Many applications of the above-mentioned technologies exist, starting from computer games, where machines can nowadays beat human players, through speech recognition, computer vision, understanding natural language, and finishing on expert systems, e.g. medical diagnosis systems or credit card transaction systems (McCarthy 2007).

Furthermore, AI, DM and related methods have been adopted for the use in business and e-business. Their application areas include marketing, customer relationship management and fraud detection, product development, process planning and monitoring, information extraction and risk analysis (Soares et al. 2008). Comprehensive BI systems have emerged, called *Business Intelligence* (BI) systems, whose task is to assist enterprise executives and analysts in the decision making process. Such software can increase profitability of a company by analysing customer data, identify the activities within the company that have ineffective performance, determine the factors influencing employee retention, just to mention a few examples (Ericsson 2004). In order to organise and analyse data, BI systems make an extensive use of AI and DM methods. One of aspects of such NLP data processing applied in BI systems is *document classification*, which is also referred to as *text classification* or *text categorisation* (TC).

Sebastiani (2005) defines TC as ,,the task of automatically sorting a set of documents into categories from a predefined set". The documents are assigned categories based on their content. TC has a wide range of applications, out of which the most widely known is spam filtering. Other examples include automated scientific document indexing based on predefined thesauri of technical terms, authorship attribution, or survey coding. In an organisation such automated TC can bring vast benefits, freeing it from a lot of manual work connected to organising its document bases, it can also be used as part of a BI system.

TC methods most widely used nowadays are utilising statistical *Pattern recognition* (PR) learning algorithms. These methods rely on comparing the word frequencies across the categories. In most cases their performance is satisfactory, however there exist application domains where more advanced approaches are needed. In the age of rapidly evolving AI and SW, it seems natural to look for solutions which would involve utilising text semantics for document categorisation. One of the novel approaches to solving TC task is using information contained in *ontologies.* Ontology is a term related to SW and will be described more in detail in Section 2.1.2. For now it is sufficient to describe it as an abstract representation of knowledge about some particular domain. This can serve as background information for classification process. Ontologies have been successfully used to improve accuracy of the classification task Janik & Kochut (2008), Seddiqui & Aono (2008). More detailed review of such efforts will appear in the latter part of the thesis.

1.1 Research problem and questions

DAVID system

Data Analysis and Visualisation aId for Decision-making (DAVID) system is an example of BI software. At the time of writing this thesis (November 2012) the software is being developed at the University of Eastern Finland as a part of the ,,Towards e-leadership: towards higher profitability through innovative leadership and management systems" project that is funded by the European Regional Development Fund (*European Regional Development Fund* 2005) and TEKES – the Finnish Funding Agency for Technology and Innovation (*Finnish Funding Agency for Technology and Innovation* 2012). According to Kakkonen (2010), DAVID aims at supporting corporate decision making by using ,,text mining, semantic web and natural language processing technologies for collecting and analysing source documents". The information about collected data is visualised and communicated to the user, which makes the system useful for tasks such as competitor analysis, analysis of customer feedback and opinions, or education and training of leaders, including support business decision making as the ultimate goal.

CoProE ontology

The structure for holding knowledge base in the DAVID system is the *Company, Product and Event* (CoProE) ontology. This ontology has been created especially for DAVID, in the way of reuse of existing ontologies. Quoting Kakkonen & Mufti (2011), CoProE is based on the *newsEvents* ontology (Lösch & Nikitina 2009) and an RDF Schema presentation of the *United Nations Standard Products and Services Code* (UNSPSC) (Ramakrishnan 2012). The ontology was created with the intent to be capable of supporting the *business news* domain knowledge needs of DAVID system. The *newsEvents* ontology aims at modelling information about news and their relevance to the user, while the UNSPSC codes classify the products and segments of industries.

Research questions

DAVID system collects business news documents automatically, from various web sources. This data needs to be processed and analysed in order to become a valuable piece of business knowledge. One step of processing is TC, where the input documents are the news articles from DAVID business document database and the categories to which these documents are classified are the ontology classes of CoProE. The module should be able to determine the level of relevance of a document to each category. This allows business decision-makers to focus their reading efforts on those documents that are relevant to their current decision-making task.

The purpose of the research described in the thesis is to find accurate and efficient ways of performing ontology-aided TC for DAVID system. The categorisation is implemented on the basis of already existing TC methods, and it utilises the domain knowledge contained in CoProE ontology.

We formulated the following research questions:

1. What are the existing ways of performing TC and ontology-aided TC? What are their merits and flaws?

2. Which of these methods suits the DAVID system best?

3. Can these methods be improved and how?

1.2 Research methods

The research is conducted in a number of steps. First we review what kind of TC methods are currently used and which are yielding the best performance in what kind of applications. We pay special attention to the ones utilising ontologies. This is going to be the answer to the first research question.

To answer the second question, we proceed with implementing a number of the existing TC methods, comparing the results and selecting those that show the best performance. For this step we also needed to prepare manually a set of relevant documents for the ML based methods – we tell more about this process in Section 5.1.

The last phase of the research, corresponding to the third research question, is applying a range of modifications to the methods that we implemented in the previous phase, and finding the ones that yield the best accuracy for our case. Finally, the best performing TC method is integrated into the TCModule of the DAVID system.

1.3 Structure of the thesis

The thesis is organised as follows: in Chapter 1 we introduce the background of this thesis, which is DAVID BI system and CoProE ontology, state the research questions and briefly introduce the research methods used.

In the Background chapter we present the RapidMiner application which we used for creating the training and classification processes, as well as tell more about the DAVID system and the process of establishing of the CoProE ontology. We also describe more in detail the concepts of SW and ontologies.

In Chapter 3 we describe the TC problem and the existing standard ways of approaching it. We review the previous work in the research fields of TC and ontology-aided TC.

Chapter 4 presents the TCModule software, which was implemented as the result of this thesis. We introduce the architecture of the module itself, as well as describe the underlying RapidMiner classification processes.

In Chapter 5 we specify the experiment settings, the test set and the course of the experiments. At the end of the chapter we summarise the experimental results.

Chapter 6 contains our conclusions drawn from the experiments as well as an attempt to justify the obtained results. In the end we consider what further work should be done on the classification module and how it can be improved in the future.

2. Background

In this chapter we give a closer look at the DAVID system and the CoProE ontology that were mentioned in Section 1.1. The subsequent sections give an overview of the DAVID system architecture and the process of creation of the CoProE ontology. In second part of the chapter we present the RapidMiner application which was the core software used for creating the training and classification processes. We describe its basic concepts and the way it cooperates with TCModule.

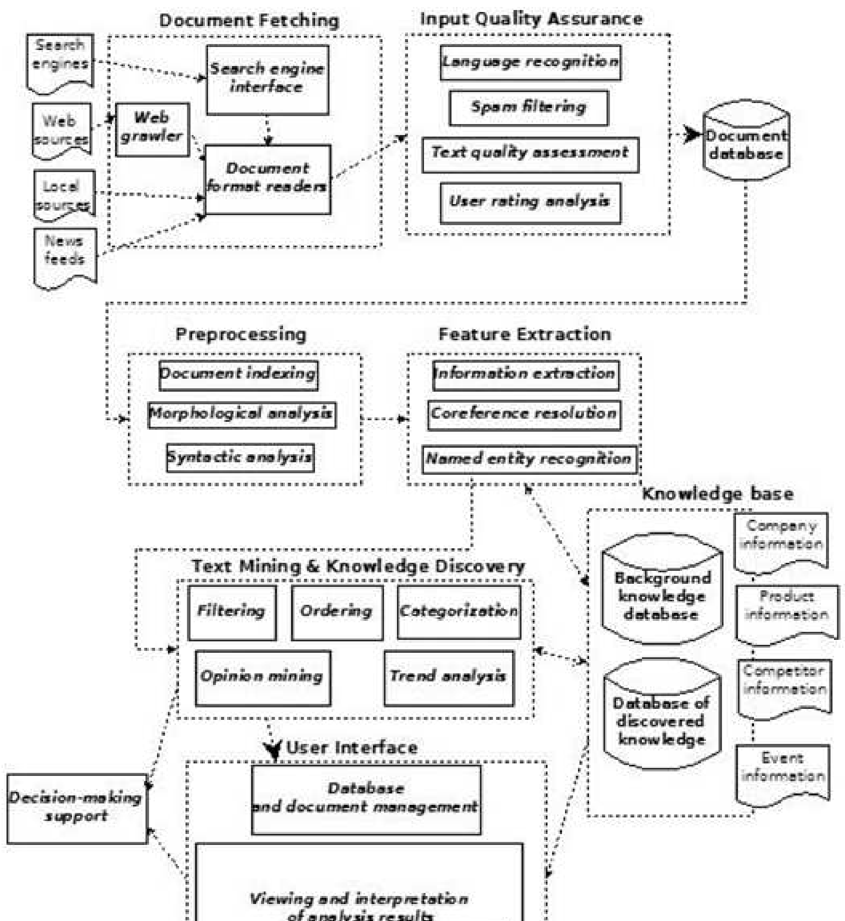
2.1 DAVID system and CoProE ontology

This section provides more information on the architecture of the DAVID BI system (Section 2.1.1) and the process of creating the CoProE ontology (Section 2.1.2).

2.1.1 DAVID system

Based on the information published in the project website (Kakkonen 2010), DAVID performs analysis on news documents related to business. It maintains an internal database of these documents and a knowledge base (i.e. the CoProE ontology), which holds both known and new inferred information. Before inserting new documents to the database, the input data is filtered to assure its quality and relevance. The knowledge base is used as semantic input in the process of latter document analysis and is constantly being updated with the analysis results. In this way the system is able to evolve and learn based on current trends in the business field that is being analysed.

Figure 2.1: Architecture of DAVID system



In Figure 2.1 is depicted the architecture of DAVID system, consisting of seven main modules. The focus of this thesis is concentrated around the Document Categorisation sub-module, which belongs to the Text Mining & Knowledge discovery part of the system.

2.1.2 CoProE ontology

In Section 1.1, we introduced the CoProE ontology which holds the knowledge base for DAVID system. We also mentioned that the ontology has been created by reusing existing ontologies. Below we explain the general concept of what an ontology is, and later we explicate the ontologies constituting the CoProE ontology.

Ontologies and Semantic Web

According to Siegel (2011), SW is a concept where Web content will be made understandable to machines, which will therefore be able to process it automatically. It is an „overhaul of our information infrastructure", based on the following principles:

- information will become unambiguous,
- data will be linked, it will be accessible from one place and represented in a standardised way (interoperable),
- the systems will be flexible, and they will be using the real-time data available on-line.

In order to achieve these goals, the data needs to be represented in a standardised way on the Web. There are two ways of data representation that serve for this purpose: *taxonomies* and *ontologies.*

Taxonomy describes the hierarchy of concepts within some particular domain. It is sufficient for encapsulating knowledge for specialised systems, without getting too complex to be maintained. *Ontology*, on the other hand, is designed at a higher level of abstraction. It consists of statements (triples, assertions). Each statement consists of three parts, which are: [*subject*], [*verb*] and [*object*] (also referred to as *class-property-value*). In this way it is possible to describe any piece of knowledge, e.g. ,,[dark chocolate] [contains] [cacao liquor]", or ,,[chocolate] [may contain] [dark chocolate]" [1]. Ontologies are capable of expressing more complex knowledge than taxonomies, but are also more difficult to create.

There exist numerous publicly available ontologies, that deal with various areas of expertise, for example Gene Ontology for genomics (Consortium 2000), PRO ontology for proteins (Natale et al. 2011), or WordNet lexical reference system (Fellbaum 1998) (which is not in fact an ontology, but can be interpreted as one), just to name a few. Ontologies are encoded using markup languages. The most common ones used for this purpose are *Web Ontology Language* (OWL) (*W3C Web Ontology Language Specifications* 2012) and *Resource Description Framework* (RDF) (*W3C Resource Description Framework Specifications* 2012). RDF is XML-based and it is a *World Wide Web Consortium* (W3C) specification for describing Web resources. In RDF the resources are referred to by their *Uniform Resource Identifier* (URI) (*RFC 2396* 1998). OWL is built on top of RDF, and is also endorsed by W3C. The purpose of the OWL language is to process information on the web. It has stronger syntax and larger vocabulary than RDF.

The goal of publishing different ontologies is to be able to connect and use them as one vast source of knowledge. The concept of reusing existing knowledge is crucial for the ,,vision" of Semantic Web. The CoProE ontology described in this thesis is an example of ontology reuse, as it is composed of existing *newsEvents* ontology (Lösch & Nikitina 2009) and UNSPSC code taxonomy (Ramakrishnan 2012). However, since many of the existing ontologies are created in-house by single companies and not always compatible with each other, the interoperability between the majority of them remains a future goal.

[1]It is important to disambiguate between ontology classes (e.g. [chocolate] or [dark chocolate]) and ontology instances of those classes (e.g. [Fazer chocolate]). Ontology may, but does not have to, contain class instances.

The newsEvents ontology

The *newsEvents* ontology is has been reused for creating the CoProE ontology. Quoting the information published by Lösch & Nikitina (2009), the *newsEvents* ontology was designed in the way to be capable of answering the following *competency questions*:

1. ,,Related to the history of an event: Is there any information about the event already in the knowledge base? In which order and in which time frame was the information about the event published?"

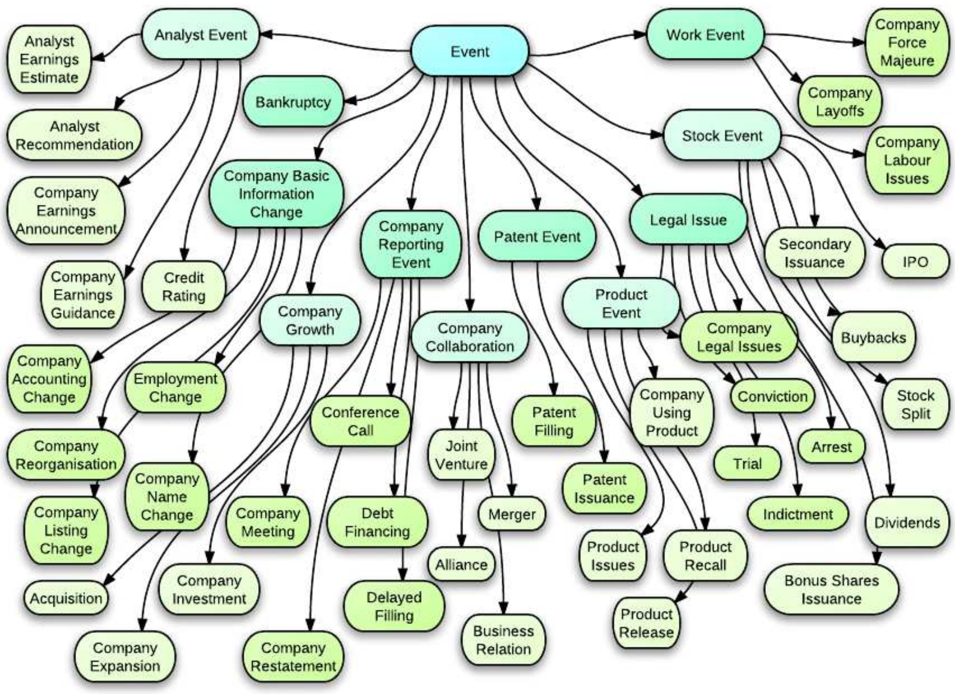
2. ,,Related to the assessment of similarity: How similar are two events? How similar are two entities (i.e. companies, authorities, etc.)?"

3. ,,Related to relations between entities: Which products does a company produce? Which industry does a company belong to? Where is a company located?"

Here are a few example assertions from the *newsEvents* ontology:

- [EmploymentChange] [affectsPosition] [Position]
- [Bankruptcy] [hasBankruptCompany] [BankruptCompany]
- [CompanyNameChange] [hasNewCompanyName] [CompanyNewName]
- [CompanyNameChange] [hasOldCompanyName] [CompanyOldName]

Figure 2.2: Event hierarchy of CoProE. Top level event categories are Analyst Event, Company Basic Information Change, Company Reporting Event, and so forth. Each of these, with the exception of Bankruptcy is further divided into event types.



In Figure 2.2 is presented the event hierarchy of CoProE ontology. It consists of categories that form two-level hierarchy. All these categories are used for the categorisation in this thesis.

The United Nations Standard Products and Services Code (UNSPSC) codes

The UNSPSC collection, as opposed to *newsEvents*, does not contain any assertions. It is a taxonomy of products and services.

Figure 2.3: An example part of UNSPSC taxonomy that describes types of furniture

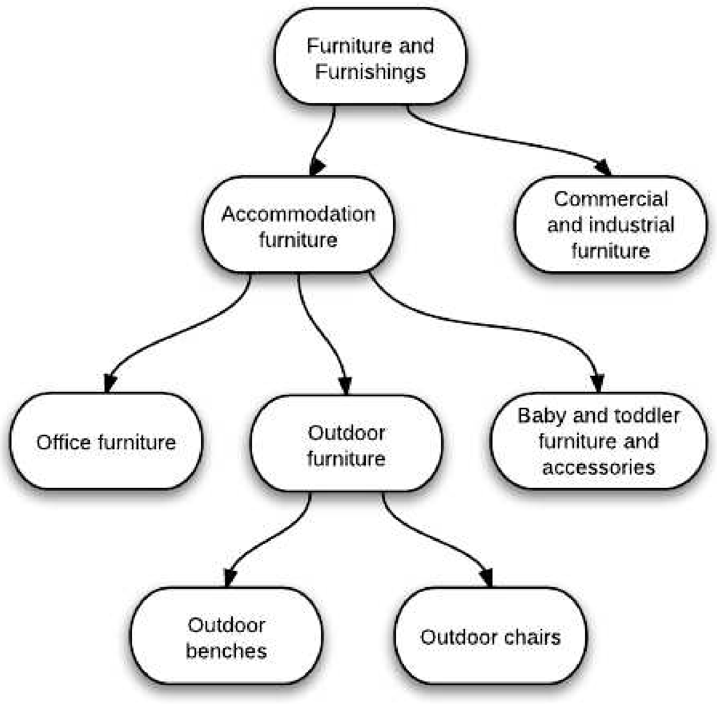

Furniture and Furnishings
Accommodation furniture
Commercial and industrial furniture
Office furniture
Outdoor furniture
Baby and toddler furniture and accessories
Outdoor benches
Outdoor chairs

Figure 2.3 shows an example part of UNSPSC taxonomy. It presents a part of product and services classification: furniture classification.

Populating CoProE ontology with data

After combining the *newsEvents* ontology and the *UNSPSC* taxonomy, the CoProE ontology was manually filled with data about relevant companies, products and events, as described in Mufti (2012). CoProE provides means of storing collected business information, such as ,,business events, companies, their products and services on offer as well as relationship with competitors companies and products" (Mufti 2012).

2.2 RapidMiner

2.2.1 Introduction

RapidMiner is a powerful open-source system for DM and analysis. It is developed by the Rapid-I company, in Dortmund, Germany. It contains a number of built-in implementations of DM algorithms, ready structures for handling data and routines for collaboration with external resources (such as disk storage or relational databases). RapidMiner allows the user to build complex DM systems by visually composing the data flow out of a number of stand-alone components. The user can set the properties of these components and connect them in order to create processing pipelines.

We chose RapidMiner as the platform for implementing the TC software for this research due to the ease of visually creating DM processes and good availability of learning materials on the Web. Moreover, the standard RapidMiner 5 Text Processing extension offers additional operators for handling text documents, such as document tokenization, word stemming, and other text transformations. In addition to that, the application provides possibilities for extending it by writing own extensions and plug-ins. Such extensibility possibilities are beneficial for future development and extending of the *Document Classification* module.

2.2.2 Fundamental concepts and terms

In this section, we introduce a few basic concepts and terms used across RapidMiner that are relevant to this thesis. *Attribute* is the characteristics of the data which is to be classified. The attribute value can be of several different types. For example, some attributes may take textual values (`text` data type) and other may take only numerical values (`numerical` data type). The distinction between attribute data types is important for application of certain DM algorithms – the accepted data types vary between algorithms. Defining the data types accepted by each algorithm as inputs reduces the risk of generating incorrect output resulting from inappropriate data representation.

In RapidMiner there are two kinds of attributes: *regular attributes* and *special attributes*. In TC, *regular attributes* correspond to the features extracted from a document. These are the ones that are used for the calculations in DM algorithms, and most commonly they correspond to the token frequencies in the document. *Special attributes* are not used for calculations, but for identifying the data. They can adopt a *role*. *Target attribute* is a special attribute whose role is the `label` role. *Target attribute* is used for storing the name of the document category. Another essential role is the `id` role, which denotes the attribute uniquely identifying a particular document.

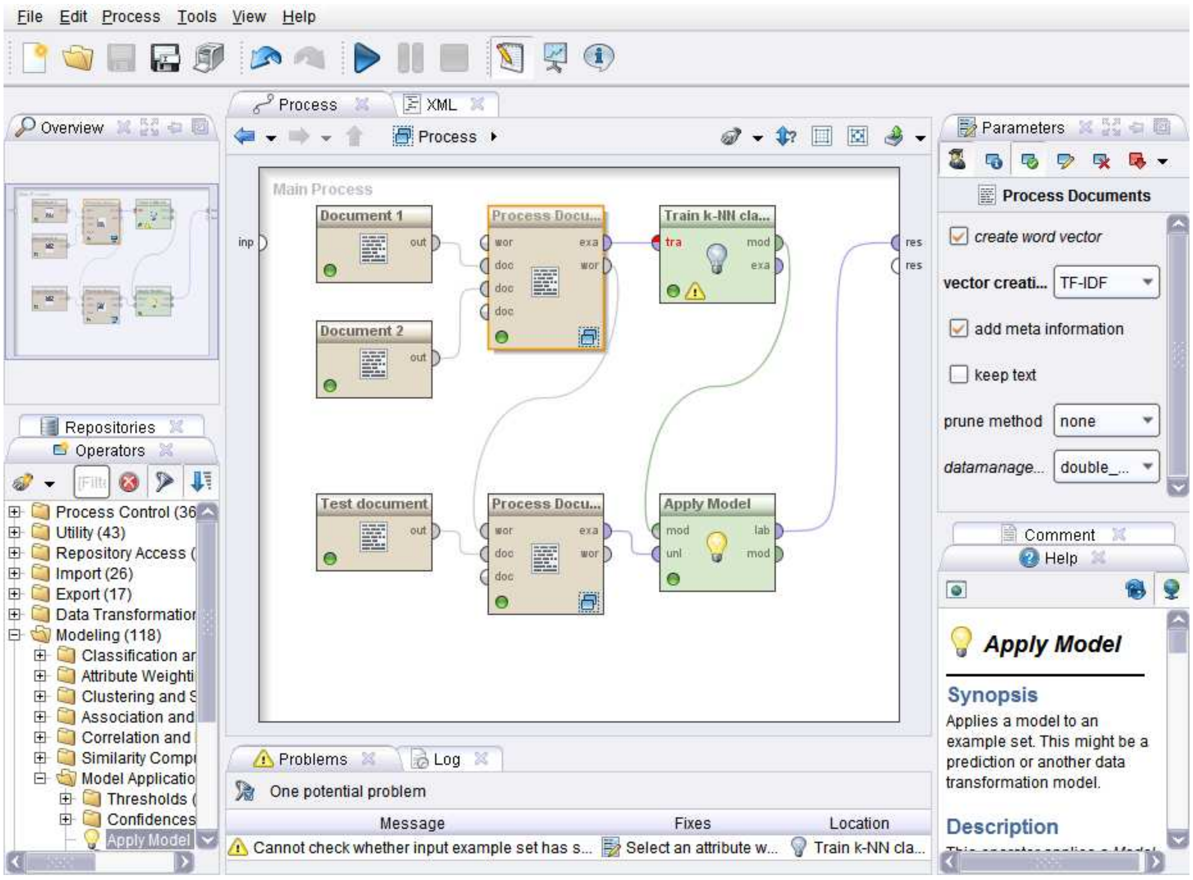
In RapidMinder, an *example* describes a single data entity. It is defined by a set of attributes (both special and regular) and their values. A set of examples defined by single set of attributes is called an *example set*. In RapidMiner an example set is usually visualised in a form of a grid, where the columns are values of subsequent attributes and each row corresponds to an example from the example set. An example set that has been assigned with predicted labels (e.g. after the classification model has been applied to it) is referred to as a *labelled example set*.

The following two subsections describe the basic concepts of RapidMiner. The information is based on RapidMider 5 User Manual (*RapidMiner 5.0 User Manual* 2010).

2.2.3 Operators and processes

An *analysis process* (or just *process*) can be defined as the data flow which performs a particular DM task. Internally each process is stored in XML format. A process can be is built and modified in the visual GUI of RapidMiner.

Figure 2.4: *Main RapidMiner window – flow design. In the central part there are the operators that perform basic TC. The three Document operators define the content of the training and test documents, the Process Documents operators perform feature extraction, Train k-NN classifier trains a classifier based on the training data, and Apply model classifies the test document with the trained classifier. On the right it is possible to see the parameters of currently selected Process Documents operator.*



Process Documents

☑ create word vector

vector creati... TF-IDF

☑ add meta information

☐ keep text

prune method none

datamanage... double_...

Apply Model

Synopsis

Applies a model to an example set. This might be a prediction or another data transformation model.

Description

One potential problem

Message	Fixes	Location
⚠ Cannot check whether input example set has s...	Select an attribute w...	Train k-NN cla...

Figure 2.4 shows the main window of RapidMiner with an example process. Each process in RapidMiner owns a number of *sinks* (process inputs) and *sources* (process outputs), as well as a number of *operators*. In Figure 2.4, the main process sink is visible on the left of the process window, marked with `inp` label, and the two sources are on the right side of the process window, marked with `res` label. The more sinks or sources the user connects, the more new available slots appear.

An *operator* is an independent computation unit that can collaborate with other operators. Each operator has a number of *input* and *output ports*, which allow it to provide and consume data. The number of ports, the required data type arriving at the particular port, and mandatority of supplying the data depends on the particular operator. Besides ports, each operator holds a number of *parameters* that can be altered by the user.

By connecting output and input ports of various operators and adjusting their properties, the user can build complex DM systems. The data that arrives at any of the sources of the root process becomes the output of the process. To be able to deliver any results after the process is run, at least one source should be connected to an output port of one of the process operators. It is possible to deliver arbitrary number of output data objects.

Obviously, the most crucial for DM are the operators that perform various classification, clustering and data transformation operations. In addition to that there is a number of operators that provide the means to define loops and conditional branch execution. It is also possible to group a number of operators into a single sub-process. Another group of operators are the ones that are designed to make certain common DM tasks easier by encapsulating their logic and exposing only placeholders for supplying the operators of specific type. An example of such operator is the *X-Validation* operator, that performs cross validation. Such operators are also called operator *containers* – they may contain other operators, just as the root process contains its operators. Similarly to the root process, such operators own their sources and sinks and can deliver the data they computed to the upper level operators. This allows the user to create hierarchical data flows, with multiple indent levels, and brings even more flexibility and possibilities to them.

After the process has been built and configured it is possible to run it. In RapidMiner interface the user can watch the data travelling through the operators by defining breakpoints on the operators. Whenever a breakpoint is reached the application displays the data at all the operator's input or output ports. This makes it easy to examine the data flow in greatest detail, as well as precisely locate possible anomalies or errors.

2.2.4 Connectivity

RapidMiner supports a range of file formats, to which it can write or from which it can read the data from. These include Excel, Microsoft Access, XML and simple CSV files, just to name a few. RapidMiner also allows to fetch the data directly from a connected database. Usually such data retrieval is performed by defining an SQL query as a parameter of a certain operator. The database connection has to be specified upfront. RapidMiner supports a range of popular database systems, such as Oracle, IBM DB2, Microsoft SQL Server, MySQL, and PostgreSQL. For the purpose of this thesis a PostgreSQL database was used. Nevertheless, due to the variety of database systems supported by RapidMiner it is possible to switch to a different database that will be needed by DAVID system.

3. Text Classification

In this chapter, we review the work that has been done on the subject of TC. We focus in particular on TC techniques that utilise ontologies. We start with the problem definition (Section 3.2) and a short overview of the early work on TC (Section 3.1). Then we move on to describing the standard TC methodology which consists of document indexing, classifier training and classifier evaluation. In Section 3.3 we describe the document indexing process, point out the problems that may arise during performing TC in certain cases, and review the solutions proposed in the literature. Section 3.4 presents Naive Bayes, K-nn and *Support Vector Machine* (SVM) classification algorithms. In Section 3.5 we inspect alternative approaches to TC, including ontology assisted solutions. Finally, in Section 3.6, we describe ways of assessing the classification quality and performance, as well as introduce the basic concepts and terms that will be used in the latter part of the thesis.

3.1 Early work on text classification

As reported by Sebastiani (2005), TC dates back to 1960s. Starting as a niche research field, until late 1980s it evolved into widely used set of methods which has proved to be efficient and useful in the numerous real-world applications. The starting point for tackling TC problem was ML, which was an increasingly developing research field at that time. Because the main focus of ML was PR, the TC problem was approached with PR methods, which in turn rely mainly on statistical analysis of pre-classified data (the *training set*). The task was solved with PR methods, such as *Naive Bayes* or *K-Nearest*

Neighbour (K-NN). A text classifier was trained on the training set, and later was used for classifying new documents. Such solution turned out to perform with good and satisfying results, and it is successfully and widely applied nowadays.

3.2 Problem definition

Sebastiani (2005) defines TC as ,,the task of automatically sorting a set of documents into categories from a predefined set". These categories are labels which are defined prior to performing the categorisation. This kind of TC is a type of *supervised learning.* In contrast to supervised learning, in *unsupervised learning* the categories are not defined upfront. Such problem is usually tackled with applying clustering methods. In the case of the TC system implemented in this thesis, the categories are defined by CoProE ontology and constitute a predefined set. Therefore, the focus of this thesis is supervised learning. We give the formal definition of supervised TC based on Manning et al. (2008):

> Let D be the *document space*, $d \in D$ be the *description* of a document, and $C = \{c_1, c_2, ..., c_j\}$ a fixed set of *categories.* We are given a *training set* T of labelled documents $[d, c]$, where $[d, c] \in D \times C$. We want to approximate the mapping $\gamma : D \rightarrow C$ such that for each $[d, c] \in T$, $\gamma(d) = c$. The approximated mapping $\hat{\gamma} : D \rightarrow C$ we call the *classification function.* In order to find the classification function, we are looking for a *learning algorithm* (or *learning method*) $\Gamma(T)$, which will learn the classification function $\hat{\gamma}$, $\Gamma(T) = \hat{\gamma}$.

Dozens of learning algorithms exist that can be employed for solving TC problems. Nevertheless, these algorithms have wider application than solely TC. They can learn functions that classify not only textual documents, but any kind of data, as long as it is represented in the way that is suitable for the algorithm (this will be described in more detail in Section 3.3). In this thesis, we focus on the following three commonly used algorithms: *Naive Bayes*, *K-Nearest Neighbour* and *Support Vector Machine.* These algorithms will be will be discussed in Sections 3.4.1, 3.4.2 and 3.4.3, respectively.

3.2.1 Single-label vs multi-label classification

TC can be divided into two types of tasks, based on the relation between the categories. According to Sebastiani (2005), we are dealing with a *single-label* categorisation task when each document d belongs to exactly one category c. A special case of single-label task is *binary* TC, where each document d is assigned one of two labels c or $\overline{c}$ („*not* c”) – the document either belongs to category c or not. A *multi-label* task is a classification task where each document d may be assigned to multiple categories c_i (or no category). In other words, in multi-label classification task the categories may overlap; it is also possible that they do not cover the whole classification space. This thesis focuses on a multi-label classification task. The business articles that DAVID system deals with can belong to multiple categories, as they describe various events occurring in the market and actions taken by the entrepreneurs.

Multi-label classification algorithms tend to be complex and costly. Furthermore, they have the tendency to overfit the training data (see Section 3.3.2) (Koller & Sahami 1997). In practise it is common to take a simplified approach, which is training $|C|$ binary classifiers for the category set C. Each of the binary classifiers identifies data belonging to one single category. The $|C|$ binary classifiers act together as one multi-label classifier. A similar technique has also been used in this thesis in the case of SVM classification. Since the used SVM algorithm cannot operate on multi-labelled data, as many SVMs was trained as there are categories. For each training run the document set was divided into two subsets: the documents that belong to the category and the document that do not. See Section 5.1 for a more detailed description.

3.2.2 Overlapping categories

It may happen that while training a single-label classifier the feature vectors for a number of documents that belong to two different categories will have very similar values (i.e. they will partially overlap). Such a problem often occurs for example in character recognition, as reported by Liu (2008). In such a case the documents will be highly probable to be classified incorrectly, as the categories they belong to are characterised by almost identical feature vectors. It is interesting to note that the described situation in fact corresponds to a multi-label classification task. This shows that determining whether we are dealing with a single or multi-label classification task is not always trivial, and our intuitive understanding does not always correspond to the properties exposed by the extracted features. In fact, the choice between single and multi-label classification is based on the abstract background knowledge we have about the domain, and it only expresses our desire for training a certain model type. Even though we may encounter category overlaps in our single-label task, we may still prefer to perform a single-label classification, as such may be more reasonable for our application domain.

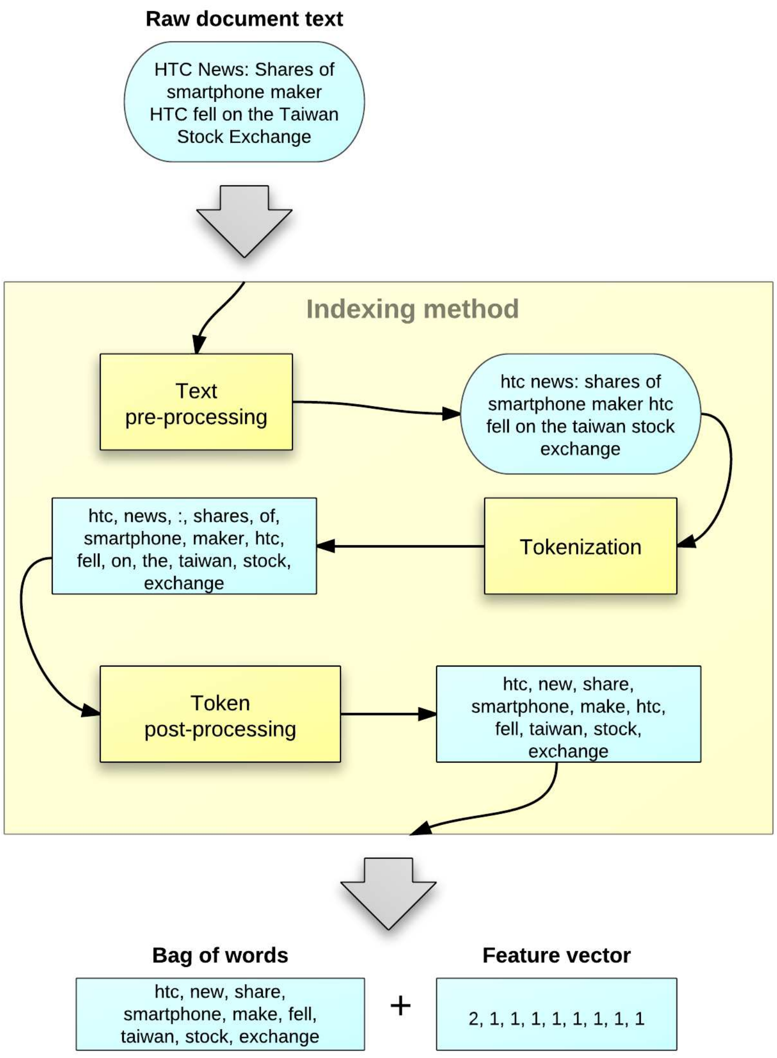
In fact, there exist various ways of tackling this problem without applying multi-label classification. One of them can be used when the categories form a hierarchy, i.e. each category is either a subcategory or supercategory of another one. The classification can then be performed separately at each category level, breaking the classification problem into smaller chunks, and in this way utilising the hierarchy information. More about *hierarchical classification* is explained in Section 3.5.1.

3.3 Document indexing

As mentioned in Section 3.2, we need a unite and unified representation of the content of documents in order to be able to classify them. *Document indexing* is the process of mapping the document content to a representation (i.e. a set of features), which can be directly interpreted by a classification algorithm (Sebastiani 2005).

Following the Theodoridis & Koutroumbas (2008), classification algorithms operate on data vectors, called *feature vectors*. Each feature vector corresponds to an *example* which is to be classified, and represents a sequence of *features*. Each component of a feature vector corresponds to the level of presence of a certain feature in the example[1]. The construction of the feature vector set is called *document indexing* in TC.

Figure 3.1: Document indexing



Raw document text
HTC News: Shares of smartphone maker HTC fell on the Taiwan Stock Exchange
Indexing method
Text pre-processing
htc news: shares of smartphone maker htc fell on the taiwan stock exchange
Tokenization
htc, news, :, shares, of, smartphone, maker, htc, fell, on, the, taiwan, stock, exchange
Token post-processing
htc, new, share, smartphone, make, htc, fell, taiwan, stock, exchange
Bag of words
htc, new, share, smartphone, make, fell, taiwan, stock, exchange
+
Feature vector
2, 1, 1, 1, 1, 1, 1, 1, 1

Figure 3.1 presents an overall document indexing process applied to an example document. As shown in the figure, typical pre-processing pipeline in TC consists of text pre-processing, document tokenisation, and token post-processing. This results in a bag-of-words presentation (see Section3.3.1) and a feature vector representing the frequencies of the tokens. The *indexing method* defines what the term is and how its weight is calcu-

[1]Note that all the feature vectors belonging to a particular training set are of the same length.

lated. In the traditional TC approach terms are usually document words, or their *stems* (morphological roots). They are obtained in the process of document *tokenization*, i.e. chopping the text into smaller parts called *tokens* and applying a number of linguistic routines (Manning et al. 2008, chapt. 2). Such processing may include word stemming, letter case conversion, filtering out common, topic-neutral words – so-called *stop words* (e.g. prepositions, articles), etc. Apart from utilising document words, it is also possible to include semantically meaningful word phrases in the word set.

Another approach is to use so-called *n-gram* language model which treats strings of arbitrary length n as tokens (Ifrim et al. 2008, Peng et al. 2003, Tenenboim et al. 2008). Peng et al. (2003) presents a language-independent text classification by using n-grams at a character level. Another author, Ifrim et al. (2008), notices that performing TC on a word level is determined historically, due to efficiency issues, which today are in fact solved to some degree. If a n-gram model was applied to the document presented in Figure 3.1 instead of word tokenisation, the feature set would contain strings like: *htc_, tc_n, ews:, ws:_, s:_s, :_sh, _sha, ...*, where of $n = 4$ and _ denotes the space.

3.3.1 Terms and term weighting

Document indexing methods that are used in TC are usually borrowed from IR. According to Sebastiani (2005), the features are the *terms* that occur in the document. A document d_j is represented as a vector of *term weights* $\vec{d_j} =< \omega_1 j, \omega_2 j, ..., \omega_{|\tau|j} >$. $\tau = (t_1, t_2, ..., t_{|\tau|})$ is the set of all the terms occurring in at least k training documents and is called the *dictionary.* $0 < \omega_k j < 1$ corresponds to the importance of dictionary term t_k in characterising the semantics of document d_j.

There exist a number of ways to calculate term weights. A binary, or *Boolean* approach assigns to the weight $\omega_k j$ value either 0 or 1, depending on whether term t_k appeared in document d_j or not. The more sophisticated methods operate on so-called *bag-of-words* (BOW) model (aka. *unigram model* (Ifrim et al. 2008)), which contains a set of document terms together with the number of their occurrences (Manning et al. 2008, sec. 6.2). In TC, the most frequently used term weighting method is the $tf * idf$ function. It measures the statistical frequency of the terms appearing in the document by combining *term frequency* (TF) measure and *inverse document frequency* (IDF) measure components. Term frequency tells about how often the term appears in a particular document, and inverse document frequency is inversely proportional to how often a term appears in all the documents. The values of $tf * idf$ are normalised in order to acknowledge the difference between document lengths.

3.3.2 Feature extraction and selection

The terms (features) are extracted collectively for the whole training set, in order for the feature vector components to correspond to the same terms across all the documents. As pointed out by Sebastiani (2005), this generates huge number of features, as not only each document contains many words, but also the words vary between the documents. Training a classifier on a data with numerous features carries the risk of *overfitting*, which will be described in Section 3.3.2. Another issue connected with large feature set is the lower classification efficiency. For the non-linear classifier training algorithms the computational cost of such task can be prohibitive.

These problems lead to the need of reducing the feature set size, which is also referred to as *dimensionality reduction*. Sebastiani (2005) mentions two ways of reducing the dimensionality of a feature set. One way is to apply a *scoring function*, which assigns each feature a score, depending on how much it is correlated with each category, and then choose only the features with highest scores. This is referred to as *feature selection*. Another method is *feature extraction*, which is constructing a smaller feature set by generating ,,artificial" terms based on the initial feature set. Both feature selection and feature extraction can limit the feature set size from thousands to hundreds elements.

Overfitting and overtraining

As explained by Theodoridis & Koutroumbas (2008), *generalisation* is the ability of a classifier to classify correctly new unseen data based on the information that it has learnt from the training data set. *Training data overfitting* happens when the classifier tries to adapt to particular details of the training data set instead of increasing its generalisation ability. As a result, the classifier has high tendency to perform better on training data set than on previously unseen data.

Overfitting may occur as a result of *overtraining*. Supplying a training set that has too little data is going to produce classifiers with very low accuracy - simply because there was not enough data to perform the correct generalisation. However, when training data set is too large, and the opposite problem may occur, which is known as *overtraining* the classifier. If the training set supplied is too numerous, the classifier learns to perfectly classify the training data, but fails on new data.

Similarly as in case of data set size, another reason for overfitting is too large set of features. In order to avoid overfitting, both the training data set and the feature set should be kept not too small but also not too large. The optimal numbers can be found empirically for a particular classification case, by testing different data sizes and evaluating corresponding classifiers. In this thesis such testing has been performed in order to find the optimal token frequency thresholds. The tokens occurring less frequently than the lower threshold and more often than the upper threshold were removed from the feature set.

In this section, we have described the standard process of preparing text documents to be interpreted by a classification algorithm. The same routine applies both to training set documents and the documents which should be classified (i.e. the test set). In the following sections we describe the basic classification algorithms that are used in TC.

3.4 Training the classifier

In this section we present a number of classification algorithms that can be applied to TC. We chose the following commonly used methods for this research:

- Naive Bayes (Section 3.4.1) represents family of Bayesian classifiers, which rely on probability calculations. In the same section we provide a concrete example of Naive Bayes algorithm application.
- K-nn rule (Section 3.4.2) is a straight-forward algorithm which performs learning and document classification at the same time.
- SVM algorithm (Section 3.4.3) is a member of the family of linear algorithms, which by making a naive assumption produce yet low classification error rate.

Finally, in Section 3.5, we outline a number of improvements that can be applied to traditional TC by including category hierarchy information, or even more complex information about category structure, which can be described by an ontology.

3.4.1 Naive Bayes

Naive Bayes classification algorithm is one of the simplest but yet efficient ones. The fact that it has linear time complexity has made it a popular classification method (Manning et al. 2008). The basis for Naive Bayes algorithm is the Bayes' theorem defining the elementary statistics formula about conditional probability, explicated in Everitt (2002, p. 33):

Let $P(X)$ denote the probability of event X, and $P(X|Y)$ denote the probability of event X occurring under the condition that event Y has occurred. Then the following is true:

$$P(B_j|A) = \frac{P(A|B_j)P(B_j)}{\sum_{i=1}^{k} P(A|B_i)P(B_i)} \tag{3.1}$$

provided events B_i are mutually exclusive ($\forall_{B_i,B_j} P(B_i \cap B_j) = 0$) and exhaustive ($\bigcup_i B_i = \sigma$, where σ is any σ-algebra on Ω).

The term $P(B_j)$ in Equation (3.1) is referred to as *prior probability* of the event B_j occurring. In TC, the A event corresponds to the document being described by the feature vector a, and the B_j events correspond to the document belonging to the category

b_j. The unknown $P(B_j|A)$ is called the *posterior probability* of event B_j occurring given the information that event A occurred. The $P(A|B_j)$ term is often referred to as the *likelihood*, which can be described as the probability of event A occurring under the condition that event B_j has occurred. The $\sum_{i=1}^{k} P(A|B_i)P(B_i)$ in our case is equal to $P(A)$ and it corresponds the overall probability of event A occurring.

Bayes formula lets us utilise additional information we have about an event while calculating its probability. When we know the conditions under which the event occurred, and the probability of such event occurring under such conditions, we can approximate the probability of the event better.

Naive Bayes in TC

As described by Manning et al. (2008, sec. 13.2), in case of TC we start with document d which is represented by feature vector $t =< t_1, t_2, ..., t_{dim(t)} >$. Based on our training data set, we want to calculate the probability that document d belongs to category c. The basic rule used by naive Bayes classifier is derived from the Bayes formula:

$$P(c|d) = \frac{P(c)P(t|c)}{P(t)} \propto P(c)P(t|c) = P(c)\prod_i P(t_i|c) \tag{3.2}$$

$$P(c|d) \propto P(c)\prod_i P(t_i|c) \tag{3.3}$$

where $P(c|d)$ stands for posterior probability of document d belonging to category c, $P(c)$ is the prior probability any document belonging to this category and $P(t|c)$ is the likelihood of a document represented by a feature vector t occurring in category c.

Components $P(c)$ and $P(t|c)$ are calculated based on the feature vectors of the training documents. Component $P(t)$, which is the probability of any document being characterised by vector t, can be omitted as it does not depend on the category. Another simplification of the initial formula is achieved by replacing the $P(t|c)$ element with $\prod_i P(t_i|c)$. $P(t_i|c)$ denotes the likelihood of a term t_i occurring in a document of category c.

It is important to note that d acts here as a multivariate random variable and is represented by a sequence of single random variables, and each of them is corresponding to one of terms t_i that occur in the document. Therefore, the substitution defined in Equation (3.2), we assume that the multivariate random variable components are *independent*[2], which implies that the probability of a word occurring in a document is independent of its context and position in the text (Næss 2007). This assumption is obviously not true for text data, and is the reason of the algorithm to be called ,,naive". However, naive Bayes classifiers perform surprisingly well in practice despite this simplifying assumption.

Determining the category

The simplest method to determine the category c of document d is to choose the one for which we obtain the highest posterior probability (so-called *maximum a posteriori* (MAP) category). In order to do so, we compare the $P(c|d)$ values obtained for each category c. This can be expressed as (Manning et al. 2008, sec. 13.2):

$$c_{map} = \arg\max_c P(c|d) = \arg\max_c P(c) \prod_i P(t_i|c) \tag{3.4}$$

Because all the probability values calculated in Naive Bayes formulae are in range $< 0, 1 >$, the important issue to take into account while multiplying them is the risk of *floating point underflow*[3]. To address the problem a logarithm function $log()$ is usually applied to both sides of the (3.3) equation[4], resulting in:

$$log(P(c|d)) \propto log(P(c) \prod_i P(t_i|c)) = log(P(c)) + \sum_i log(P(t_i|c)) \tag{3.5}$$

$$log(P(c|d)) \propto log(P(c)) + \sum_i log(P(t_i|c)) \tag{3.6}$$

[2]Two random variables A, B are independent iff $P(A \cap B) = P(A)P(B)$.

[3]Underflow occurs when a very small number fails to be represented in computer memory; this can occur when performing calculations on numbers that are very close to 0 – in such case the result may be falsely 0.

[4]According to the logarithm function property $log(ab) = log(a) + log(b)$.

The proportional relation has been preserved as logarithm function is *monotonically increasing*[5] (Manning et al. 2008, sec. 13.3), therefore we can write:

$$c_{map} = \arg\max_c log(P(c|d)) = \arg\max_c log(P(c)) + \sum_i log(P(t_i|c)) \tag{3.7}$$

$$c_{map} = \arg\max_c log(P(c)) + \sum_i log(P(t_i|c)) \tag{3.8}$$

, which is the final formula used by a basic naive Bayes classifier for finding the c_{map} category of a document represented by $t =< t_1, t_2, ..., t_{dim(t)} >$ feature vector. The probabilities are calculated based on the training set. The interpretation of Equation (3.8) can be intuitively described in the following simple way. Each of the $P(t_i|c)$ components express the relative occurrence frequency of each term in other documents of a particular category. The more frequent the terms are in that category, the more likely it is that the document belongs there. Similarly, for the $P(c)$ component, the more often the category occurs in the training set, the more probable the document will be to be assigned to that category.

It is quite possible that while classifying a new document one of its terms will not occur in the training set at all. In such a case the $P(t_i|c)$ result will be 0, which together with applying the Equation (3.4) will result in the posterior probability equal to 0, regardless of other term frequencies (as we multiply the conditional probabilities for all terms). Such situation is, of course, undesirable. The solution that is usually applied is *Laplace smoothing*, which adds one to every count:

$$P(t_i|c) = \frac{N_{t_i,c} + 1}{\sum_{t_i}(N_{t_i,c} + 1)} = \frac{N_{t_i,c} + 1}{\sum_{t_i} N_{t_i,c} + dim(t)} \tag{3.9}$$

Let us consider an example of applying the Naive Bayes algorithm in its simplest form. Let's assume that we are given the following set of categories and amounts of training

[5] It is such function f which for any $x_1 < x_2$ satisfies the inequality $f(x_1) < f(x_2)$.

documents belonging to them:

1. Cooking (C_1) – four documents (d_1, d_2, d_3, d_4)
2. Gardening (C_2) – three documents (d_5, d_6, d_7)
3. Arts (C_3) – two documents (d_8, d_9)

At this point, we are able to calculate the probabilities of each category in the data set (the more documents there are in a category, the more probable it is to occur):

1. $P(C_1) = \frac{4}{9}$
2. $P(C_2) = \frac{3}{9}$
3. $P(C_3) = \frac{2}{9}$

We have also stored a list of all the frequent tokens appearing across the training set, as well as for each of the training documents we have stored the corresponding feature vector representing occurrences of each word from the list. Table 3.1 shows the resulting BOW model.

Table 3.1: BOW model of the training data set

word	Occurences											
	d_1	d_2	d_3	d_4	C_1	d_5	d_6	d_7	C_2	d_8	d_9	C_3
cook (t_1)	3	2	4	5	14	1	1	1	3	2	1	3
house (t_2)	2	3	1	1	7	3	2	1	6	1	1	2
garden (t_3)	1	1	1	1	4	3	5	4	12	1	2	3

Imagine we have a new document d_n that needs to be classified to one of these categories. We tokenize this document and store the intersection of the list of its tokens and the training BOW list. The word list of the new document is presented in Table 3.2.

Table 3.2: BOW model of the training data set

word
cook (t_1)
garden (t_3)

To classify the probabilities of document d belonging to each of the categories C_1, C_2, and C_3, we calculate the following:

1. $P(C_1|d) \propto P(C_1)\prod_i P(t_i|C_1) = P(C_1)P(t_1|C_1)P(t_2|C_1)P(t_3|C_1) = \frac{4}{9}\cdot\frac{14}{24}\cdot\frac{7}{24}\cdot\frac{4}{24} = \frac{1456}{124416} = 0,0117$
2. $P(C_2|d) \propto P(C_2)\prod_i P(t_i|C_2) = P(C_2)P(t_1|C_2)P(t_2|C_2)P(t_3|C_2) = \frac{3}{9}\cdot\frac{3}{21}\cdot\frac{6}{21}\cdot\frac{12}{21} = \frac{648}{83349} = 0,008$
3. $P(C_3|d) \propto P(C_3)\prod_i P(t_i|C_3) = P(C_3)P(t_1|C_3)P(t_2|C_3)P(t_3|C_3) = \frac{2}{9}\cdot\frac{3}{8}\cdot\frac{2}{8}\cdot\frac{3}{8} = \frac{36}{4608} = 0,009$

Because the probability $P(C_1|d)$ scored the highest value, we can assume that the document d belongs to category C_1. Even though document contains words from both C_1 and C_3, C_1 appeared more frequently in training document set. This is the reason why this category is chosen as the most probable one for the document d.

3.4.2 K-Nearest Neighbour

A straight-forward and simple K-nn rule has been proven to yield fairly good results when applied to TC. Let us assume that we have a set of N training vectors extracted from the document set. Then, according to Theodoridis & Koutroumbas (2008), we perform the following steps. Given feature vector x and distance measure $d(x_i, x_j)$:

1. From the training vector set, choose k vectors that are closest to the vector x in respect to the distance measure d. K should be odd for two class problem, and in case of multi-class problem it should not be a multiplier of the number of classes M.

2. Assign the vector x to a category to which the majority of chosen k vectors belong to.

The most frequently used distance metrics in K-nn are the *Euclidean* and *Mahalanobis* distance. Because in TC our feature vectors consist of numbers denoting the term frequencies, it is possible to treat them as numeric vectors.

Figure 3.2: K-Nearest Neighbour

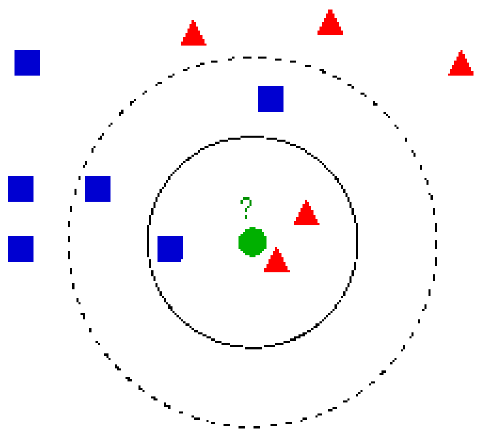


Figure 3.2 depicts the algorithm at the stage when a new element (the green circle) is to be classified. If $k = 3$, the element would be classified as a red triangle. In contrast, if $k = 5$ was chosen, the new element would be classified as a blue square.

Error estimation

Despite it simplicity, the K-nn algorithm exhibits good practical performance. In fact, it can be proved that its classification error probability in case of a two class problem:

$$P_B \leq P_{kNN} \leq \frac{1}{\sqrt{ke}}$$

when $N \to \infty$. P_B is the optimal error of Naive Bayes classifier. For small Bayesian errors the following approximations hold:

$$P_{NN} \approx 2P_B$$

$$P_{3NN} \approx P_B + 3P_B^2$$

which is reflected by the fact that k-NN classifier tends to Bayes optimal classifier when k increases (Theodoridis & Koutroumbas 2008).

According to Theodoridis & Koutroumbas (2008), K-nn algorithm performs relatively well especially if a large number of training vectors is available. However, in case of a small training data set its performance can drop dramatically. Another drawback is the need of calculating the distance between vectors, which can be computationally demanding in case of high vector space dimensionality (which is exactly the case of TC).

3.4.3 Support Vector Machine

So far we have been focusing on utilising probability theory in design of the classifier. Now we introduce one of the algorithms belonging to the family of *linear classifiers*. SVMs are among the most commonly used methods in TC and in ML in general. As explained by Theodoridis & Koutroumbas (2008), linear classifiers are a subgroup of classifiers where the discriminant functions (see Section 3.4.3) are linear, and therefore their decision surfaces are hyperplanes. A hyperplane a plane in 3-dimensional space generalised into n dimensions, where one coordinate is fixed. Discriminant functions used by a number of Naive Bayes classifiers could also be linear, however, in case of linear classifiers the linearity is the basic classifier property on which deriving the classifier building algorithm is based. The main advantage of linear classifiers is their simplicity and computational attractiveness. The SVM algorithm has been originally proposed by Cortes & Vapnik (1995). Its key feature among other linear classifiers is that it attempts to find the hyperplane which has the greatest *generalisation* potential, i.e. the one that is the most probable to classify future data correctly.

Discriminant functions and decision surfaces

Let us recall the transformations we made in Section 3.4.1, when describing the Naive Bayes algorithm. In Equations (3.5) and (3.6) of we replaced the $P(c|d)$ function by $log(P(c|d))$ function. We call the $log(P(c|d))$ function a *discriminant function.* In fact, according to Theodoridis & Koutroumbas (2008), any function $g(d) = f(P(c|d))$ where f is a monotonically increasing function is a discriminant function. Discriminant functions are often used instead of working on the probabilities directly, as they can be more convenient to use from mathematical point of view.

Each discriminant function describes a *decision surface.* We expressed in Equation (3.4) the statement that the category should be determined by choosing the one that has the maximum value of $P(c|d)$. If we interpret a category as s sub-region of the multidimensional feature space, then for each two contiguous sub-regions c_i and c_j there exists a decision surface g_{ij} that separates these two regions.

The surface is defined by the following equation: $P(c_i|d) - P(c_j|d) = 0$. For documents belonging to one of the regions the expression $P(c_i|d) - P(c_j|d)$ will be negative and for the other region it will be positive. If we apply a discriminant function g to our probability measures, the decision surface can be expressed as: $g_{ij}(d) = g_i(d) - g_j(d) = 0$. In case of linear classifiers the decision surfaces are hyperplanes. Because of that, we can write the equation of $g_{ij}(d)$ as:

$$g_{ij}(x) = \omega^T x + \omega_0 = 0 \tag{3.10}$$

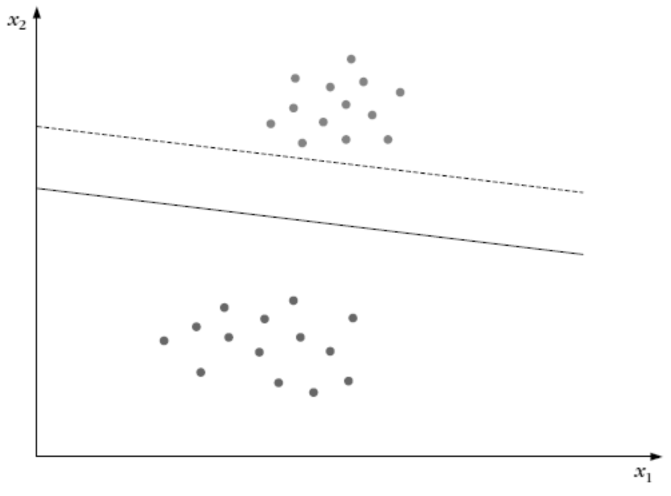
where $\omega = [\omega_1, \omega_2, ..., \omega_t]^T$ is called *weight vector*, and ω_0 is known as *threshold.* Note that the weight vector is orthogonal to the decision hyperplane, and $|g(x)|$ defines the distance of point x from the hyperplane. For a number of examples $g(x)$ will be positive, and for others negative – this provides a way of classifying the examples into two category sets.

Because the decision surface is a hyperplane, we need to assume that the data is linearly separable. We will design the classifier for this simple case, and then extend its capabilities for the more general case, by finding an *optimal classifier.*

Linearly separable classes

For a linearly separable data set there exist a number of hyperplanes, each of which define a different classifier that will classify the data correctly. A geometrical illustration of this fact can be seen in Figure 3.3.

Figure 3.3: An example of two class categorisation problem and two possible classifiers (Theodoridis & Koutroumbas 2008)



As mentioned in the introduction of Section 3.4.3, the SVM algorithm attempts to find the hyperplane of the greatest *generalisation* potential. Looking at the simple two dimensional case illustration in Figure 3.3, it can be noticed that full line will be a better classifier than the dotted line. The most suitable hyperplane corresponds to a line that lies exactly in the middle of the gap between two point groups.

The distance between the hyperplane and the closest point from the point group representing a category is called the *margin* between the hyperplane and the category. SVM algorithm finds such a hyperplane that maximises the margin between it and both categories. Applying the general formula for distance between a point and a hyperplane, the margin between the category ω and the hyperplane $g(x)$ can be defined as:

$$z = \frac{|g(x)|}{||\omega||}$$

We also define the *class indicators*, denoted by y_i:

$$y_i = \begin{cases} 1, x_i \in \omega_1 \\ -1, x_i \in \omega_2 \end{cases}$$

If we now apply a number of optimisation theory and convex programming methods we end up with the following equation describing components of the optimal ω of the separating hyperplane:

$$\omega = \sum_{i=1}^{N_s} \lambda_i y_i x_i \tag{3.11}$$

The Equation (3.11) describes *support vectors*, which are training vectors that are critical for finding the optimal hyperplane. The hyperplane can be found by constructing any linear combination of $N_s \leq N$ support vectors where all $\lambda_i \neq 0$. The optimal hyperplane is called the *support vector machine.*

$\lambda = [\lambda_1, \lambda_2, ..., \lambda_N]$ in the Equation (3.11) is the vector of *Lagrangian multipliers* λ_i, which are found by maximising the following function:

$$\max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{ij} \lambda_i \lambda_j y_i x_i^T x_j \right)$$

subject to

$$\sum_{i=1}^{N} \lambda_i y_i = 0$$
$$\lambda \geq 0$$

The ω_0 coefficient can be obtained implicitly from the following condition:

$$\lambda_i [y_i(\omega^T x_i + \omega_0) - 1] = 0, i = 1, 2, ..., N$$

There can be several possible λ_i combinations satisfying the above equations, however all of them will represent the same hyperplane $g(x)$.

Nonseparable classes

When data is not linearly separable it is not possible to find a hyperplane that will correctly divide the space into two distinct category data sets. Each hyperplane will classify a number of points incorrectly. We express this number of incorrectly classified points by introducing additional factors in our equation. Let us note that for x being a support vector, the following condition holds:

$$g(x) = \omega^T x + \omega_0 = \pm 1$$

Therefore, for all the vectors lying inside the band the following is true:

$$-1 < y_i[\omega^T x + \omega_0] < 1$$

We now introduce *slack variables* ξ_i in the following way.

$$y_i[\omega^T x + \omega_0] \leq 1 - \xi_i$$

For ξ values that are > 1 the equation describes the misclassified vectors. For $0 < \xi \leq 1$ it describes the vectors that are classified correctly. $\xi = 0$ corresponds to correctly classified vectors that lie outside the band.

In case of nonseparable classes, our goal is not only to maximise the margin but also to minimise the number of $\xi_i < 0$. After a number of transformations (refer to Theodoridis & Koutroumbas (2008)), we obtain the following set of equations, differing from the separable case ones only by the C coefficient.

$$\max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{ij} \lambda_i \lambda_j y_i x_i^T x_j \right)$$

subject to

$$\sum_{i=1}^{N} \lambda_i y_i = 0$$

$$0 \leq \lambda \leq C$$

where C is a constant that controls the relative influence of these two factors (maximising the margin and minimising the number of vectors inside the band). The linearly separable case corresponds to $C \to \infty$.

Multiclass case

The above equations can be applied in case of two class classification problem. What happens when we deal with multiple classes at the same time? According to Theodoridis & Koutroumbas (2008), there is a number of approaches which constitute a workaround to that problem.

- one-against-all: M binary classifiers are trained; for each of such classifiers we consider one category as one set, and all other categories as second set; the problem in the case of this approach may be that the categories are not balanced, especially in case of big data sets
- one-against-one: $M(M-1)/2$ binary classifiers are trained, one per every category pair; the disadvantage is the big amount of classifiers that needs to be trained

The main disadvantage of SVM classification algorithm is its computational complexity (both at training and test stage), which reaches $O(n^3)$ in case of naive implementation. Various attempts at optimising this number have been made, however they did not make the maximum complexity drop below $O(n^2)$ (Theodoridis & Koutroumbas 2008).

3.5 Utilising category structure

In this section, we present information about how category structure information can be utilised in the process of TC. We describe hierarchical classification that uses category hierarchy, and then move on to more complex ontology-assisted solutions.

3.5.1 Hierarchical classification

As opposed to *flat categorisation*, we are dealing with *hierarchical categorisation* when the categorisation utilises some particular category hierarchy. Sun & Lim (2001) gives good overview of different variants of hierarchical TC. According to these authors, the two main types of hierarchical category structure is *category tree* and *virtual category tree.* The difference between these two is that in the first case we assign the documents to one of all the categories occurring in the tree, while in the latter case every document can only be assigned to one of leaf nodes of the category tree.

As illustrated by Sun & Lim (2001), Tenenboim et al. (2008), Koller & Sahami (1997), in case of hierarchical classification a separate classifier is trained at each level of the category hierarchy. Later, while classifying a document we start from the category tree root and move deeper in the direction determined by the category assigned at each level. Depending on whether we use category trees, or virtual category trees, we return the last certain category before the classification confidence drops below certain threshold, or continue until one of the leaf categories is reached.

Figure 3.4: Example of a category tree for hierarchical classification

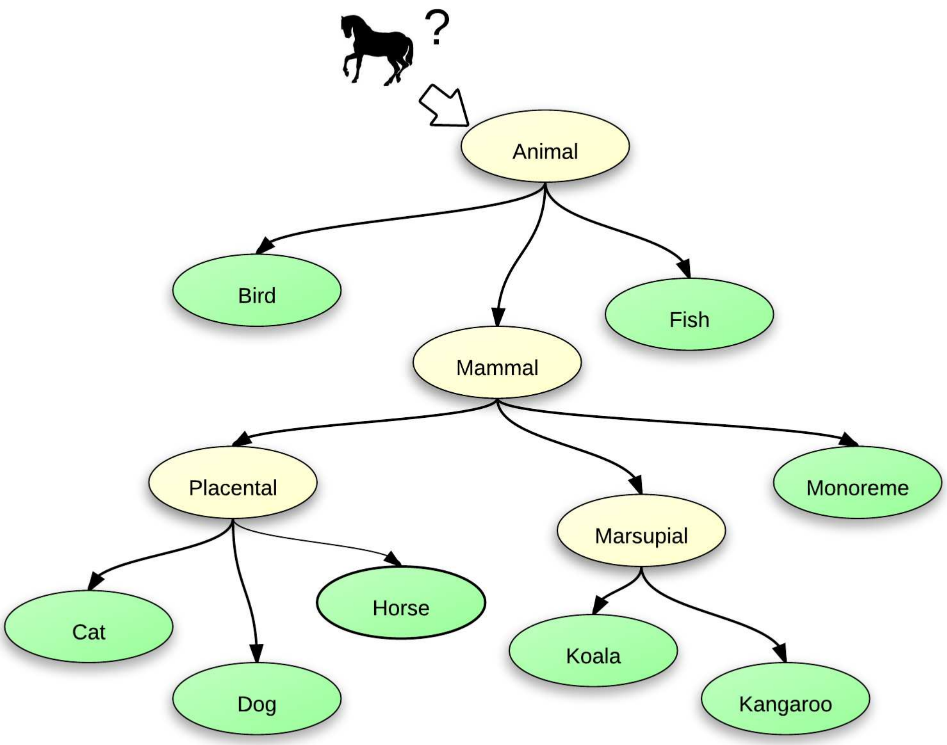


Figure 3.4 presents an example category tree. The green nodes are the leaf nodes, to which the document to be classified is going to be assigned. Each yellow node corresponds to a classifier that distinguishes between its child categories. We would therefore have to train four different classifiers in order to perform classification for the category hierarchy presented in Figure 3.4. Depending on whether the inner nodes can also be considered as final categories, the tree could serve as either category tree or virtual category tree.

Using such an approach breaks down the classification problem into smaller chunks, which means that at each level a different set of features may be considered. This is potentially beneficial for the classification, as for different sub-categories different features may prove to be relevant. A flat classifier, in contrast, tries to extract features collectively for all the categories, which may limit its capabilities (Sun & Lim 2001). The reduction of the feature space dimensionality at each level potentially improves the accuracy of hierarchical classification and reduces the risk of overfitting (Koller & Sahami 1997).

Hierarchical classification has another advantage over the other methods, namely that reducing the feature space size decreases the time needed for training and applying a classifier. Moreover, at least in theory, this can further increase the accuracy, as due to reduced feature space we can afford using more costly and more accurate classification algorithms (Koller & Sahami 1997).

This method also has its flaws, however. Multiple classifiers require larger amount of training data for accomplishing results that could be achieved in flat classification with less training data. Also the lower-level category assignment options depend on the of the top-level assignment. Moreover, a misclassification at an upper level implies incorrect assignment to at the lower level, therefore the accuracy may drop significantly if inadequate amount of training data is available (Sun & Lim 2001).

3.5.2 Ontology-aided Text Classification

In Section 2.1.2 we explained the concept of ontologies and we described an existing ontology. Ontologies contain information about the target domain. In TC this information can be used as a context for categorisation. The complexity of ontologies makes it difficult to design such a TC method based on them that would utilise an arbitrary ontology as a whole. Hence, when applying ontologies in TC, a subset of the ontology structure and/or some part of the informaton stored in the ontology is considered. In most cases, the ontology as a whole would be too complex and would contain too much information for being practical for TC.

If we establish a relation between ontology classes and TC category set, we can extract the information that describes the structure of the categories. In the simplest case such a structure forms a hierarchy, in more complex cases we can represent it as a graph. In hierarchy case, we can extract the category taxonomy directly from the ontology. Such an approach has been presented by Tenenboim et al. (2008), where the authors used news taxonomy for applying hierarchical classification of news documents arriving to the *ePaper* mobile news reading service. Seddiqui & Aono (2008) also made use of ontology class taxonomy, but in a slightly more sophisticated way. Their task was to classify patent document abstracts using the *International Patent Classification* (IPC) ontology. Besides extracting the category taxonomy from the ontology, they generate a taxonomy of BOW for the training set in the following way:

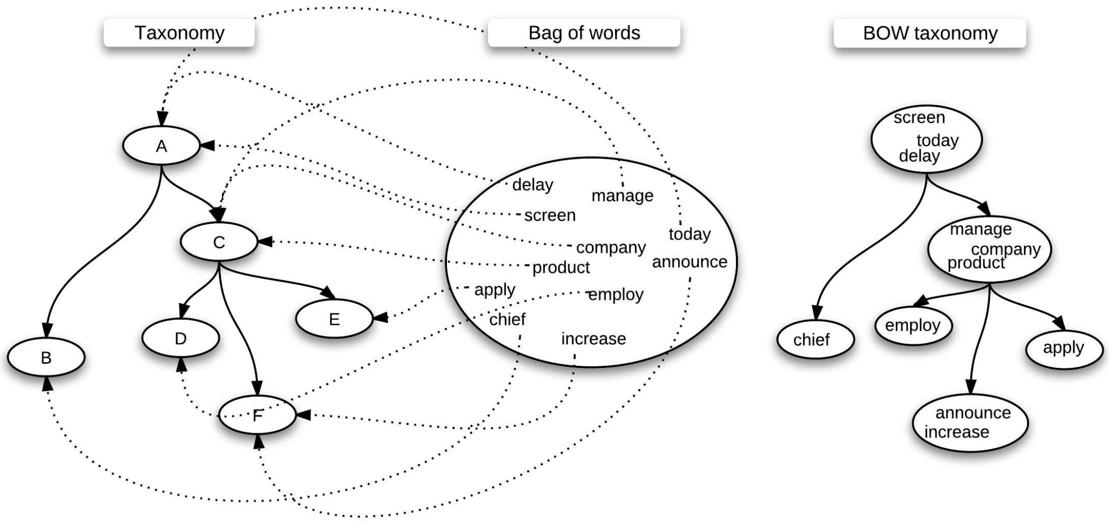
1. For each training document its BOW is extracted, and then the relevance of each word in respect to each category is captured.

2. Each word is placed in the node of the category hierarchy that contains the category the word was the most relevant to. In this way a word hierarchy is being formed.

3. Finally, a BOW taxonomy is created, with topology reflecting the topology of the hierarchy of categories.

For a new document to be classified, its BOW presentation is matched with the BOW taxonomy. The aim is to find the most suitable place for it. Because the nodes in the BOW taxonomy correspond to category hierarchy nodes, such assignment determines a category to which the new document belongs to. In this approach it is important to note the application domain of the method. Patent documents often contain new terms and their authors try to use attractive and novel vocabulary. Therefore, traditional methods based on BOW only may not be the most favourable choice.

Figure 3.5: Extracting BOW taxonomy from category taxonomy and document BOW



An illustration of this three-step process can be viewed in Figure 3.5. Each word of the BOW model is mapped to a category and then placed in an appropriate place in the BOW taxonomy. Note that relation between the words in the BOW taxonomy is not based on the word meaning but the relevance to a category in the category taxonomy. All these small taxonomies were then combined into one BOW taxonomy using statistical methods.

A more sophisticated solution was presented by Janik & Kochut (2008). The authors describe a novel fully ontology-based TC method, which does not require a training set. They utilise ontology class structure as an arbitrary graph that describes category relations. Then, for each document a semantic graph is created. The semantic graph represents the entities referenced the document: names, places and objects, and the relationships between them. Next, the semantic graph is compared with the graph describing the category structure extracted from the ontology. By finding the best fit for the graph inside the category structure they locate the most suitable category for a document. This approach was tested on a RDF ontology constructed from the full English version of Wikipedia (*Wikipedia website* 2012). The test documents consisted of CNN news articles, which were mapped to Wikipedia articles. The method showed good accuracy of 80%, which was 7% better than the Naive Bayes classifier trained on Wikipedia articles, and 14% worse than NB classifier trained on CNN articles.

It is arguable whether ontologies can in fact provide means of TC that will overperform the traditional TC. Nevertheless, the advantage in utilising an ontology is that the it can serve as an user-adjusted input to the classifier training process. This is in particular relevant for interactive information systems, as the user can modify the category hierarchy, adjusting it to the current data structure and information needs, as presented by Seddiqui & Aono (2008). A drawback resulting from the same fact as the above advantage is that in case the ontology does not reflect the document structure well, classification accuracy will be significantly decreased.

A method for answering the opposite question, namely how well does the ontology match the described data, is presented by Netzer et al. (2009). There, similar approach is taken in order to evaluate the accuracy of the ontology structure. First, a set of documents is classified in a traditional way and then by the same documents are categorised using the taxonomy. Comparing the convergence of both result sets delivers an ontology quality measure.

3.6 Classifier evaluation

According to Sebastiani (2005), there are three factors that constitute the classifier quality. The *training efficiency* corresponds to the average time that is needed to build the classifier. *Classification efficiency* is the average time used for classifying the document. Finally, the *classification effectiveness* is the accuracy of classification results. We will describe these measures more in detail in the following sections.

3.6.1 Effectiveness

When talking about classifier effectiveness, or in other words classification accuracy, we need to introduce some standard terminology. The *training set* is the set of pre-classfied documents on which the learner builds the classifier. Each document from this set has the correct category assigned, and this information is used by the learner. The *test set* is the set of documents which the classifier will classify in order to evaluate the classification accuracy (Sebastiani 2005). The test documents also have correct categories assigned but this information is not visible to the classifier. By comparing the categories assigned by the classifier and the categories that the test documents in fact belong to it is possible to draw conclusions about classification accuracy.
For a category c_i we say that a classified document is a:

- *true positive* (TP) if it has been correctly classified as belonging to a category it in fact belongs to
- *true negative* (TN) if that has been correctly classified as not belonging to a category
- *false positive* (FP) if it has been incorrectly assigned to a category to which is does not belong to
- *false negative* (FN) if it has not been recognised as belonging to the category that it in fact belongs to

We also define *precision* and *recall* accuracy measures. According to Sebastiani (2005), *precision* is ,,the percentage of documents deemed to belong to c_i that in fact belong to it":

$$\pi_i = \frac{TP_i}{TP_i + FP_i}$$

In other words, precision is the percentage of classified documents that are relevant to category. *Recall* is ,,the percentage of documents belonging to c_i that are in fact deemed to belong to it":

$$\rho_i = \frac{TP_i}{TP_i + FN_i}$$

Recall is the percentage of documents relevant to a category that were in fact classified as belonging to it.

Figure 3.6: Precision and recall

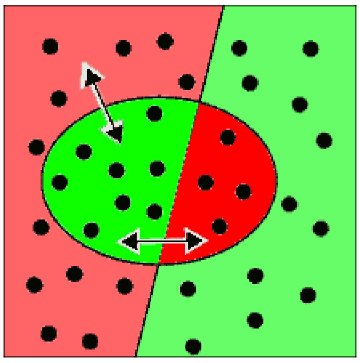


Figure 3.6 shows graphical interpretation of precision and recall measures. Red regions represent incorrectly classified examples. On the left they are FNs and on the right they are FPs. Precision is the proportion of the left green area to the area of the whole oval (horizontal arrow), and recall is the proportion of the left green area to the whole left region (diagonal arrow).
In case of multi-label TC, precision and recall values are calculated for each of the categories separately. In order to provide an overall accuracy result, a certain way of averaging these values must be adopted. In TC there are two main types of accuracy averaging, as stated by Sebastiani (2005):

- *microaveraging* - when ,,categories count proportionally to the number of their positive training examples", so the focus is on the examples

- *macroaveraging* - when ,,all categories count the same", so the focus is on categories

Table 3.3: Formulae for calculating macro- and microaverages (Sebastiani 2005)

	Microaveraging	Macroaveraging
Precision (π)	$\pi = \frac{\sum_i TP_i}{\sum_i TP_i + FP_i}$	$\pi = \frac{\sum_i \pi_i}{\lvert C\rvert} = \frac{\sum_i \frac{TP_i}{TP_i+FP_i}}{\lvert C\rvert}$
Recall (ρ)	$\rho = \frac{\sum_i TP_i}{\sum_i TP_i + FN_i}$	$\rho = \frac{\sum_i \rho_i}{\lvert C\rvert} = \frac{\sum_i \frac{TP_i}{TP_i+FN_i}}{\lvert C\rvert}$

Table 3.3 presents the exact formulae for calculating macro- and microaverages for precision and recall values. The first accuracy measure favours classifiers that perform well mainly on heavily populated categories, while the second measure emphasises classifiers that perform well also on categories that have less documents. The choice between the two measures depends on the concrete evaluation case. Macroaveraging, however, is often considered as the more informative measure.

We introduced above the ways of calculating the overall precision and recall, which is needed in case of multi-label classification. It makes sense to ask what should be the single accuracy measure that characterises a classifier. Because it is possible to tune many classifiers to perform with better precision at the cost of recall effectiveness, and vice versa, both of these measures need to be taken into account. According to Sebastiani (2005), the most common way of calculating the overall accuracy measure is using the following formula: $F_\beta = \frac{(\beta^2+1)\pi\rho}{\beta^2\pi+\rho}$, $0 \leq \beta \leq \infty$. Usually $\beta = 1$, and then the formula becomes the harmonic mean of π and ρ values: $F_1 = \frac{2\pi\rho}{\pi+\rho}$. Making the β coefficient bigger than 1 would emphasise the importance of recall, while taking $0 \leq \beta < 1$ would correspond to paying more attention to precision. It is therefore possible to adjust the accuracy measure to the particular classification case we are dealing with, depending on which behaviour of the classifier is more desirable.

3.6.2 Efficiency

According to Sebastiani (2005), we can speak of two kinds of classifier efficiency. *Training efficiency* of a classification method is the average time required to build a classifier.

Classification efficiency is the average time needed for classifying a new document. The importance of these measures depends on the area of application. For example in case when the application should maintain a real-time interaction with the user, long classification times are not acceptable. On the other hand, if the training document database is not going to be updated frequently, training efficiency is usually not an important factor. For example, even though SVM has a much lower training efficiency than Naive Bayes or K-nn (Manning et al. 2008, Sec. 13.6), it is used more often than the faster but less accurate Naive Bayes algorithm – because in most applications training time does not play such a big role.

Efficiency is an important measure of the quality of a particular classifier. Efficiency of a classifier depends on a number of volatile parameters, such as software platform and hardware configuration of a particular machine. That is why it is not very reliable for evaluation and comparison between different classification methods run on different machines. It can however give some indications. Nevertheless, currently the hardware capabilities are constantly increasing, which results in classifier efficiency being more and more satisfactory for nowadays classification software.

3.7 Summary

ML-based TC methods that are widely employed nowadays seem to have many limitations. Preparing the training set, based on which the text classifier will be trained upfront, is a significant amount of work. Moreover, in some application domains such a training set is not available, for example in case of patent classification (Seddiqui & Aono 2008). Another shortcoming of the traditional TC approach is the fact that the document content is analysed in a BOW manner, which means that there is no actual understanding of the meaning of the document content. In fact, the feature extraction step becomes crucial for the final classification accuracy. Once good features are selected, any reasonable classification algorithm will display a reasonable effectiveness (Peng et al. 2003, Scott & Matwin 1999).

Despite all these shortcomings, today's TC methods display high robustness and are successfully and broadly applied across different domains. There have been many attempts made to perform the classification in more semantics-aware manner, e.g. based on relations between words or using ontologies (Tenenboim et al. 2008, Janik & Kochut 2008, Seddiqui & Aono 2008), however they perform better than traditional TC only in a number of specific domains.

One of the reasons of such superiority of traditional TC methods could be the note made by Janik & Kochut (2008), who suggested that the objective document category resulting purely from its text semantics does not always correspond to the subjective category that an average reader would assign to the document. The authors give as an example an article about ,,cardiovascular health problems of a certain politician". According to the content of the article it would be classified as belonging to *health* category, but in the news website it would more likely appear under *politics* category. That could be the main reason why classifying methods that use a training set are performing better than semantically based methods in real-life domains. The conclusion about document category is drawn not by reasoning, but just by analysing the previous results.

4. TCModule

In this chapter we present the overview of the GUI and the implementation of TCModule. The data model used by the module is described in Section 4.3. In Section 4.4 we describe the design and development of the underlying RapidMiner processes, which form the core of classifier training and evaluation in TCModule. Next, we talk about the user interface and present a brief code overview (Section 4.5). Finally, in Section 4.8 we present the *document parser*, a tool that parses a file structure and automatically fills the database with the documents and information about their category.

4.1 Introduction

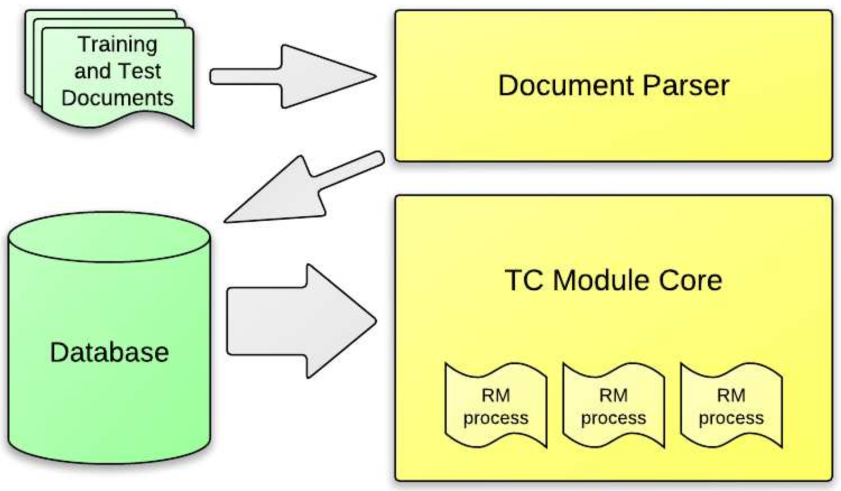
Text Classification Module (TCModule) is a stand-alone application that is able to train a text classifier on a set of documents and then classify new documents. It is designed to work with a database which is the document source. At the same time the module can be easily integrated into the DAVID system. In the following sections we describe the tools and libraries used, the data model represented by the database, the user interface, the connection to RapidMiner and the implementation details.

4.2 Tools and libraries used

TCModule is written in Java and is compiled using version 1.6 of JDK. It uses the standard Java Swing library for *graphical user interface* (GUI). The tools used for development were:

- NetBeans IDE[1] - an open source Java development environment that includes a visual GUI builder,
- PostgreSQL Admin Tools[2] - PostgreSQL visual database management software,
- Protégé[3] - an open source ontology editor and knowledge-base framework.

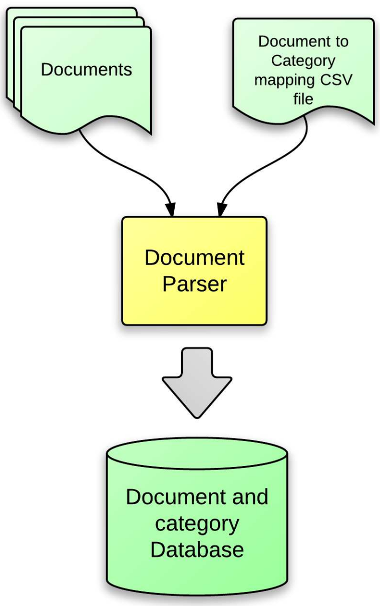
Figure 4.1: The overall architecture of TCModule



TCModule utilizes a relational database introduced in Section 4.3 for storing training and test documents. TCModule is built on top of RapidMiner 5 Java API (*RapidMiner 5 API Documentation* 2009) that was introduced in Section 2.2. The database system used for testing the application was PostgreSQL (*PostreSQL offical web site* 2012). PostgreSQL is an open-source, object-relational database management system. Figure 4.1 shows the overall architecture of the system.

[1]`www.netbeans.org`
[2]`www.postgresql.org`
[3]`http://protege.stanford.edu/`

Figure 4.2: Document fetching done by the parser


Documents
Document to Category mapping CSV file
Document Parser
Document and category Database

Apart from the code responsible for training the model and classifying the data, TC-Module comes with a parser which automatically reads files that contain text documents, as well as the `csv` files that contain the document-category relation information. The process of document fetching is presented in Figure 4.2.

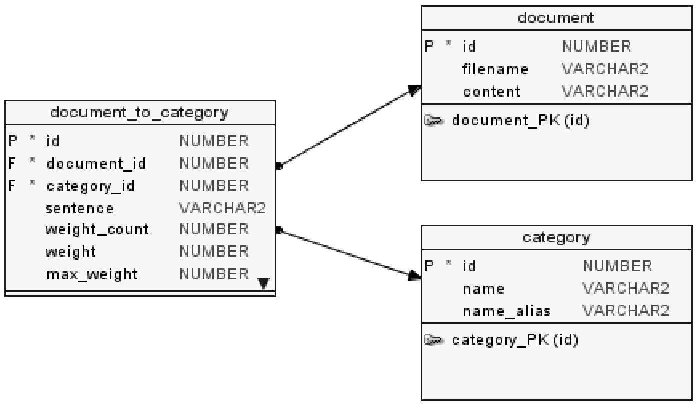
4.3 Data model

The database model is presented in Figure 4.3. Each *document* has a property `content`, which stores the document raw text, and an `id`, which is unique for each document. The *category*'s `name` property is unique for each category. The `name_alias` property stores an abbreviated version of category name.

Because not only one category can contain many documents, but also one document can belong to more than one category, we defined the `document_to_category` table. This table stores ids of corresponding documents and categories. The <`document_id`, `category_id`> pairs must be unique in the table.

There are three special columns in the `document_to_category` relation: numeric at-

Figure 4.3: Database schema for storing the documents and their categories

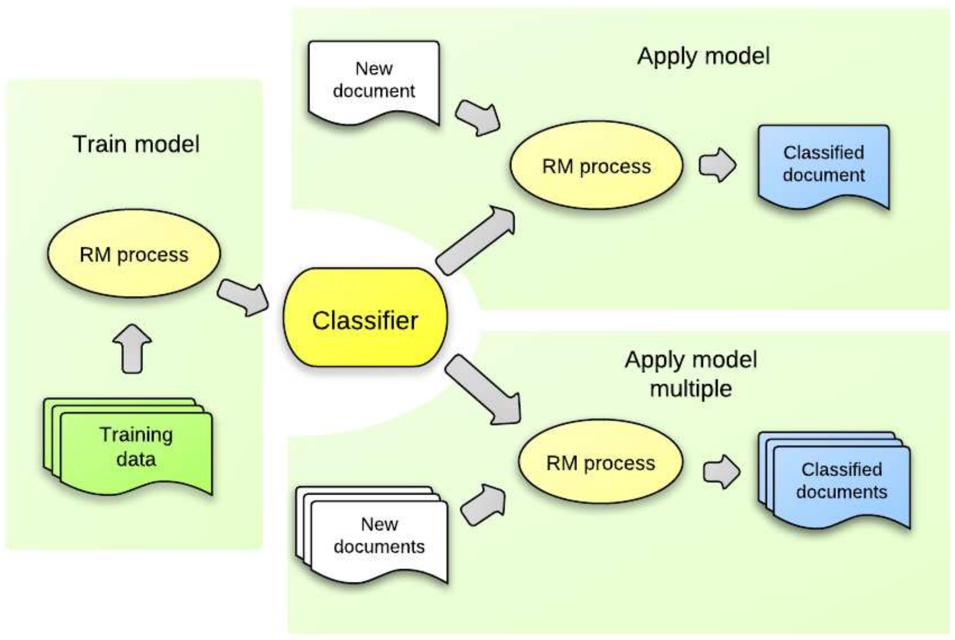


tributes `weight` and `weight_count`, as well as the boolean attribute `max_weight`. The `weight` column defines the level of relevance of a document to the relevant category. Next, `max_count` specifies how many document-category relationships there are with the same weight. Finally, the `max_weight` attribute determines whether the given record defines the category with the highest weight for the document, i.e. it defines if the current category is considered as the most probable category for the document that is being classified. The `max_weight` attribute is true only for the records whose `max_count` value is equal to one. The information stored in these three attributes can be used for constructing SQL queries for retrieving the documents. See Section 4.8 for more information about the logic that was applied in TCModule.

4.4 Underlying RapidMiner processes

TCModule is a wrapper for RapidMiner processes that were developed by using the RapidMiner GUI. TCModule is needed in order to be able to utilize the RapidMiner TC pipeline from the DAVID system. The RapidMiner processes perform the actual model training and document classification. They were created in RapidMiner and exported to an `xml` file. They can also be run in RapidMiner, independently from TCModule.

Figure 4.4: TCModule use cases



In the Figure 4.4 are presented the three use cases corresponding to the following three processes:

- *Train model*, for training the model,
- *Apply model*, for applying the model to a single document entered manually by the user,
- *Apply model multiple*, for applying the model to a batch of documents fetched from database.

RapidMiner's *macro* mechanism is utilised in order to provide means of communication between the processes and the wrapping Java application. In each RapidMiner process it is possible to define a macro together with its corresponding value, and later refer to this value only the name of the macro. All of the RapidMiner processes included in TCModule contain a *Set Parameters* operator, which allows to define a number of macros and their values. The remaining operators refer to these values.

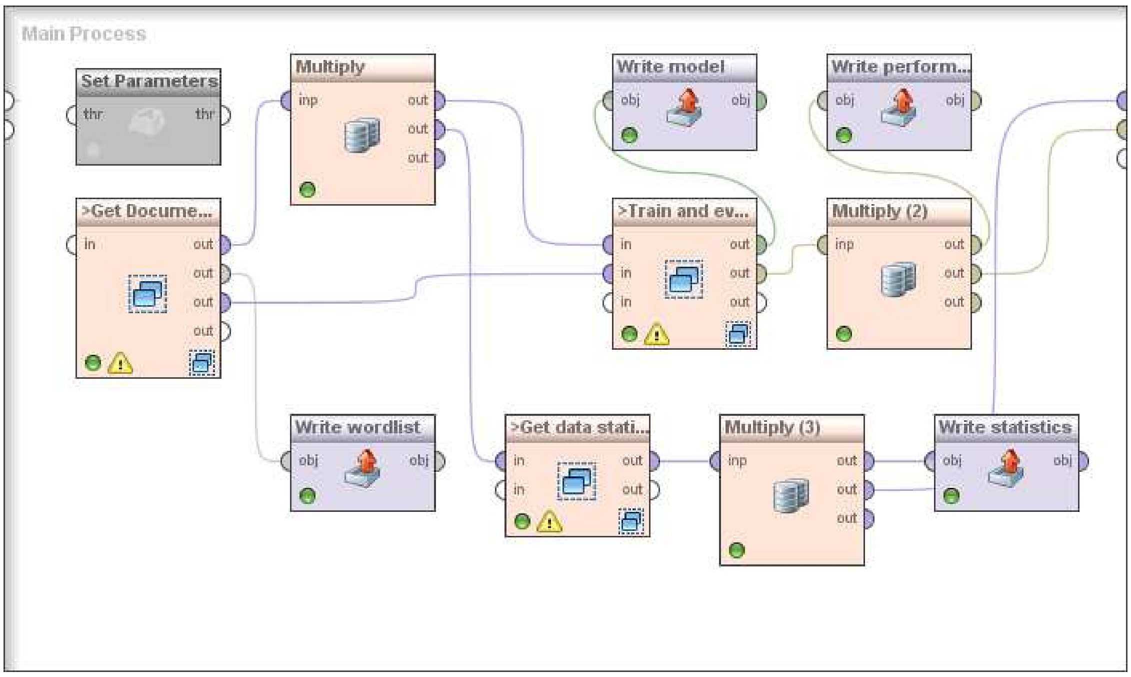
In each of the processes, the *Set Parameters* operator is disabled by default. This is done because the macro values will be input from TCModule at run-time. However, if the user wishes to run the processes directly from RapidMiner, this operator should be enabled and appropriate macro values supplied manually.

We will now describe in detail each of the three processes that train the model and classify the documents. The following operator naming convention is used throughout the processes: the operators whose names start with '>' character contain nested sub-processes.

4.4.1 Train model process

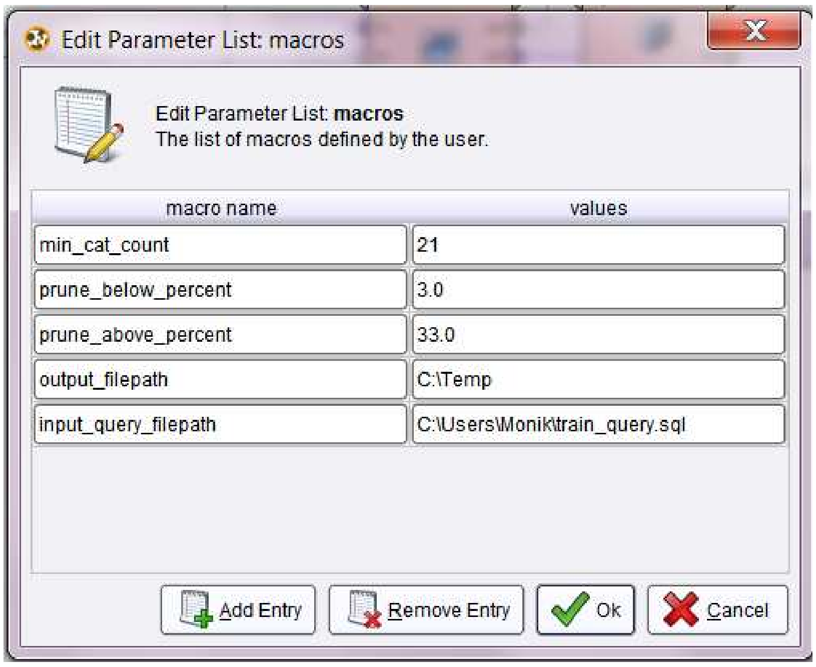
This process trains a TC model based on documents fetched from the database. Apart from training the model, it performs a number of additional computations, which is predicting expected model accuracy and extracting information about categories with respect to the number of documents that they contain.

Figure 4.5: Overall view of *Train model* process



The *Train model* process is depicted in Figure 4.5. Let us describe the data flow. As it can be seen in the figure, the first operator is *Set Parameters* operator, which allows to specify the classification parameters [4].

Figure 4.6: Setting the properties for *Train model* process



Edit Parameter List: **macros**
The list of macros defined by the user.

macro name	values
min_cat_count	21
prune_below_percent	3.0
prune_above_percent	33.0
output_filepath	C:\Temp
input_query_filepath	C:\Users\Monik\train_query.sql

In Figure 4.6 is depicted the parameters setting view for the *Set Parameters* operator. As explained in Section 4.5, parameter `output_filepath` specifies the file path where the model files will be saved, and `input_query_filepath` specifies the file from which the SQL query that fetches the training documents will be read. The other parameters will be introduced as needed.

- >*Get Documents* operator is responsible for fetching the documents from the database and for extracting the features from the input documents. It also prunes the documents and words that do not satisfy threshold criteria defined by *Set Parameters* operator.

[4]The *Set Parameters* operator in the Figure 4.13 is disabled. This is because the process is prepared to be run by the TCModule application.

- *Multiply* operator copies the data, so that the same data can be further used in multiple places. The data is copied by reference, which makes the operation cheaper and more robust [5].

- *Write wordlist* operator saves the extracted features as a word list, in an XML file [6]. The word list that specifies the feature set of the training data that has been used to train the model is needed when applying the model to new unseen data. This is due to the fact that the training and new document data should have identical set of attributes.

- >*Get data statistics* operator aggregates information about documents and their categories, creating a list of category-number of document entries. This list is then delivered both to the process output source and saved in the file `output_filepath`\statistics.xml.

- >*Train and evaluate the model* operator performs the main goal of the process. It trains the classification model and calculates expected model accuracy. The results are then delivered both to the process output source and saved on disk in zipped XML format, in the files `output_filepath`\model.zip and `output_filepath`\performance.zip.

Get Documents and Train and evaluate the model operators

We will now take a closer look at the two key operators: >*Get Documents* and >*Train and evaluate the model.*

[5]At the same time, however, the data should not be modified in more than one place, as the modifications of one copy influence all the other copies.

[6]The file is saved under `output_filepath\wordlist.xml` file path.

Figure 4.7: The sub-process of >*Get Documents* operator

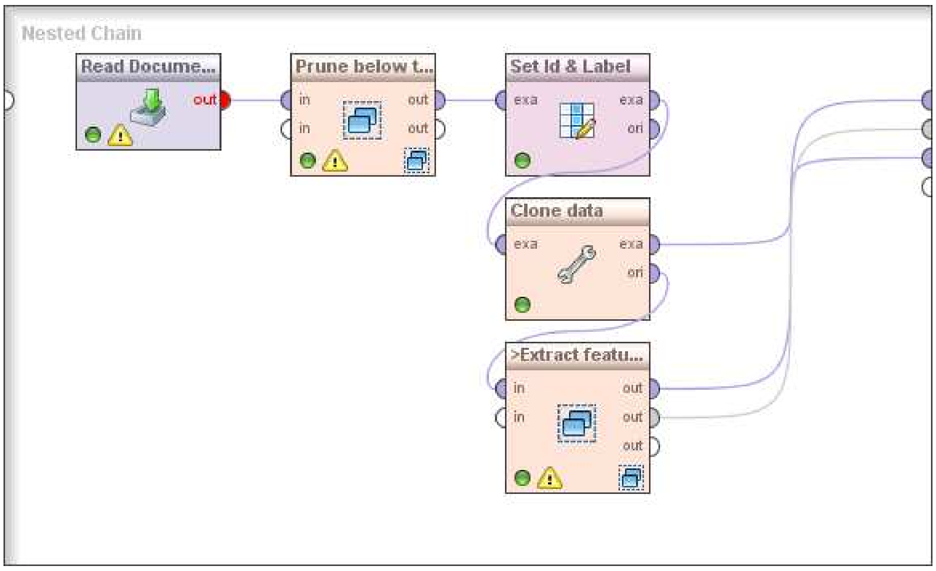
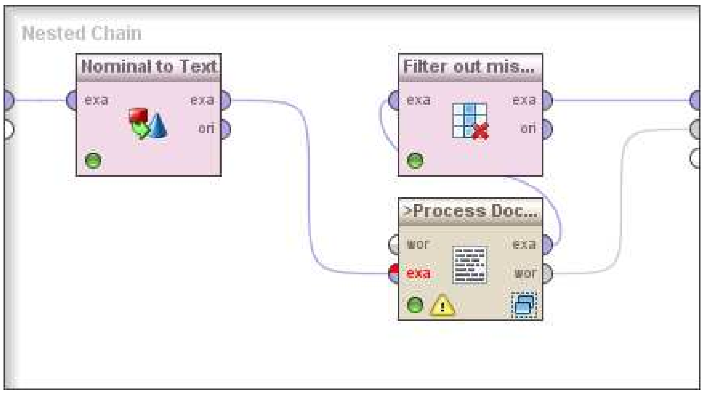


Figure 4.7 shows the inner sub-process of >*Get Documents* operator. It consists of the following elements:

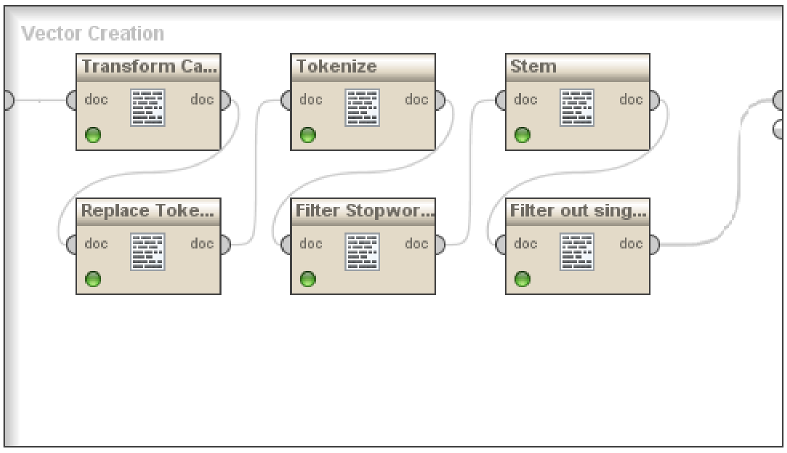
- *Read documents* fetches the data from database by executing the query from file specified by the `input_query_filepath` macro parameter.
- *Prune below threshold* removes the documents that belong to categories that have number of documents less that the value defined by `min_cat_count` macro parameter.
- *Set Id & Label* operator defines the id and label roles for `id` and `category` attributes, respectively.
- *Clone data* performs deep copy of example set, so that later it can be safely used both for training the model and model cross validation.
- >*Extract features* operator performs the actual feature extraction. Figure 4.8 illustrates the sub-process of this operator.

Figure 4.8: The sub-process of >*Extract features* operator



In Figure 4.8 is presented the sub-process of >*Extract features* operator. *Nominal to Text* is a compulsory operator which specifies the data type of the `content` attribute as `text`. This is needed for the consecutive >*Process Documents from Data* operator, in order to determine the attributes it should extract the tokens from. *Filter out missing values* removes the attributes which have no value specified. This is performed as a safety check; it should not occur in our case.

Figure 4.9: The sub-process of >*Process documents from data* operator



Going deeper into nested operator hierarchy we will now look into operator >*Process Documents from Data*. Its inner operators are depicted in Figure 4.9. This operator creates the word vector out of training documents. For each document it performs text tokenization and applies certain token processing routines. In the end it calculates the frequencies of tokens for the whole example set collectively.

As described in Section 3.3.1, various term weighting schemes can be applied in TC. One of them is the idf measure, which is exactly the measure used by >*Process Documents from Data* operator here. Besides extracting the tokens and their frequencies the operator removes tokens appearing in less than `prune_below_percent` percent of all documents, as well as tokens appearing in more than `prune_above_percent` percent of all documents.

The content of the *Process Document* sub-process is presented in Figure 4.9. The sub-process is run for each document in the example set (i.e. training set). The role of each component of the sub-process is described in Table 4.1.

Table 4.1: Text pre-processing operators

Operator	Purpose
Transform Cases	Transforms the document's text into lowercase. This is done in order to avoid differentiating between words that vary only by letter case.
Replace Tokens	Defines a mapping which is then used for replacing certain phrases with another ones. This is done in order to normalize phrases that have the same meaning. Another reason for using the mapping is preventing filtering out currency symbols, or other one-letter symbols that can be relevant to the article content. The replacement list used in the current version of TCModule is presented in Table 4.2.
Tokenize	Cuts the text into tokens, based on a specified separator. Currently the separator is any character that is not a letter. *Filter Stopwords* operator filters out so-called *stop words*, which are prepositions, articles, and other commonly used English words that are not relevant for classification.
Stem	Stems the words, so that different forms of the same word would be treated equally. For example, all the words: 'fish', 'fisher' and 'fishing' would be reduced to the root word 'fish'.
Filter out single letters	Removes single letter tokens, as they are not relevant to the classification.

Table 4.2: Word replacements of *Replace tokens* operator

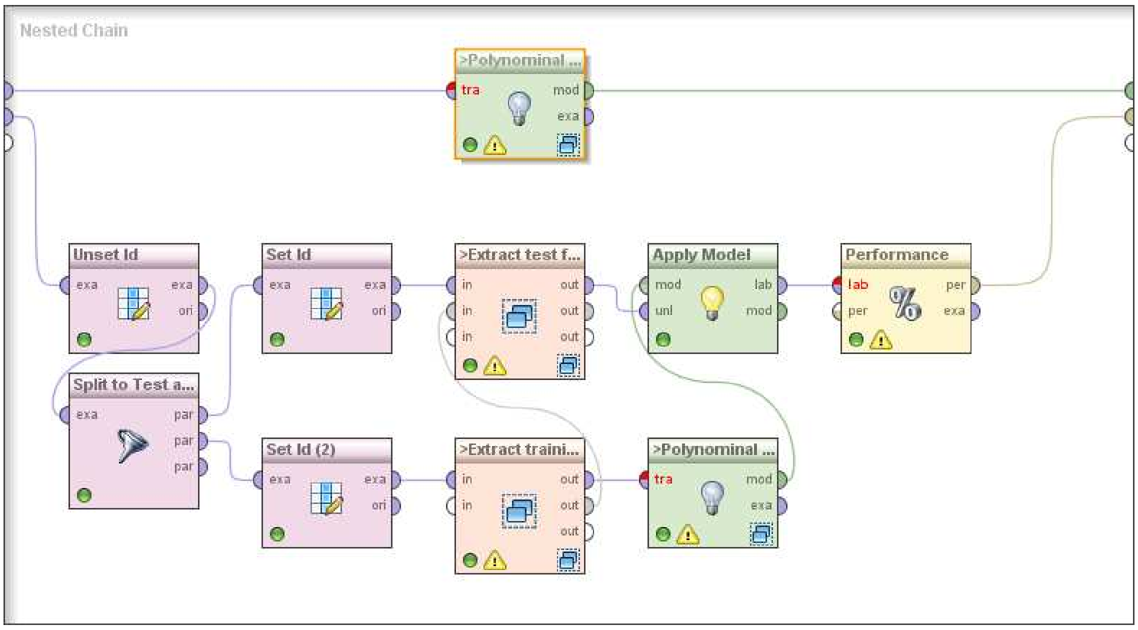
word	replacement
labor	labour
joint venture	jv
initial public offering	ipo
$	dollar
€	euro
£	pound
¥	yen
§	paragraph

We have just described in detail the process of feature extraction. Let us now move to the >*Train and evaluate the model* operator. Its sub-process is presented in Figure 4.10. This operator performs two parallel computations, each of which is performed on separate copy of the training data set.

The first operation trains the model on all the available training examples, and it is performed by >*Polynominal by Binomial Classification* operator, visible in the very top row of the process view in Figure 4.10. >*Polynominal by Binomial Classification* is a wrapper for an operator that executes the SVM classification algorithm. This operator is needed because SVM requires the input data to have binomial labels [7]. >*Polynominal by Binomial Classification* simulates such behaviour by running the SVM algorithm multiple times, each time dividing the training set into two subsets: examples having label of certain value and not having label of this particular value. After training a number of models for each of such divisions, they can be applied collectively as one model. In fact, the model returned by >*Polynominal by Binomial Classification* operator is just a single

[7]That is caused by the fact that the SVM algorithm divides the feature space into two separate subsets. Because of that the training data arriving at the input of such operator should always have only two kind of labels (binomial labels).

Figure 4.10: The sub-process of >*Train and evaluate the model* operator



model instance.

The second operation performed by the >*Train and evaluate the model* operator is calculating the expected accuracy of the model:

- *Split to Test and Training* operator divides the input data into two subsets: test and training data sets. Splitting the data is done ratio 1:4, respectively.
- *Unset id* removes the `label` role from the `label` attribute, so that the splitting does not take it into account.
- *Set id* sets back the `label` role on the `label` attribute.

After the data has been divided into two sets, feature extraction is performed on each of these subsets separately. This part is important, because extracting the features prior to data division would lead to the training data carrying some information extracted from test data. This should never be the case, because the training and the set data have to be kept independent for the evaluation to be reliable. The word list that was generated by training data is the used as one of the inputs for feature extraction from the test data. This is to make sure that the tokens not occurring in the test data will still be present in feature set, with term weights set to 0. It is also important to note that the term weighting in test documents is performed not by using idf but only tf measure (the distribution of tokens across test documents should not affect the classification of each of them separately). Moreover, in order to ensure that no relevant tokens will be skipped, no token pruning is done while extracting features from the test set.

Training the model for evaluation is performed in an identical way as in the case of training the actual model. The only difference is that only a part of available data is used. In the end the trained model is applied to the test data by the *Apply model* operator, and the performance is calculated by the *Performance* operator. The result of applying the model is a set of new special attributes added to each example. The attribute we are interested in is the `prediction(category)` attribute, that holds the name of the category that has been predicted for the document. The performance is then calculated based on conformity between the `label` and `prediction(category)` attributes.

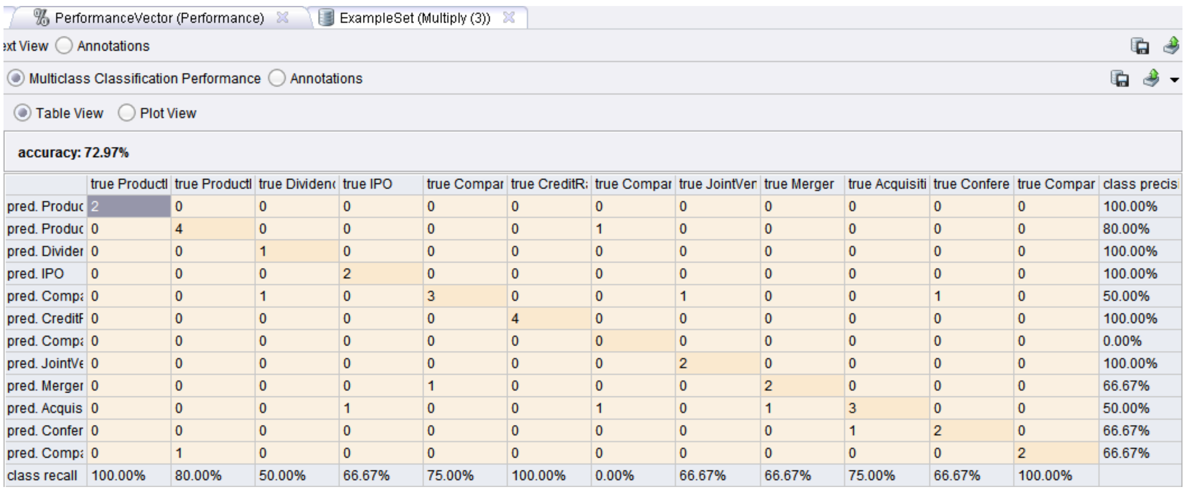
4.4.2 Estimated evaluation accuracy

We have just gone through the process of training the model and calculating its expected accuracy, as well as extracting the statistics about document distribution across categories. While viewing the model output will not give much readable information, the two other results can be viewed and are delivered to the output source of the *Train model* process.

Figure 4.11 shows the classification results. The **accuracy** measure displays the overall percentage of the documents that were correctly classified in respect to the total number of documents. The overall accuracy measure in respect to category c_i is calculated with the following formula: $accuracy = \sum_i \frac{TP_i}{TP_i+FP_i+FN_i}$, and corresponds to overall F_β accuracy measure for $\beta = 1$ (see Section 3.6.1).

The table appearing below the accuracy measure is called the *confusion matrix* and presents more detailed view of how many documents were assigned to each category correctly (i.e. *TPs*), as well as the number of *FPs* and *FNs*. The bottom row of the

Figure 4.11: Result view of expected model performance



PerformanceVector (Performance) | ExampleSet (Multiply (3))

ext View | Annotations

Multiclass Classification Performance | Annotations

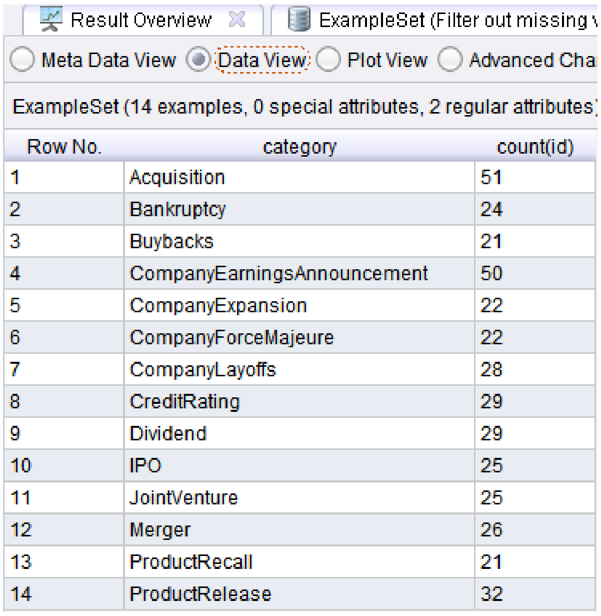
Table View | Plot View

accuracy: 72.97%

	true Productl	true Productl	true Dividen	true IPO	true Compar	true CreditR	true Compar	true JointVen	true Merger	true Acquisiti	true Confere	true Compar	class precisi
pred. Produc	2	0	0	0	0	0	0	0	0	0	0	0	100.00%
pred. Produc	0	4	0	0	0	0	1	0	0	0	0	0	80.00%
pred. Divider	0	0	1	0	0	0	0	0	0	0	0	0	100.00%
pred. IPO	0	0	0	2	0	0	0	0	0	0	0	0	100.00%
pred. Comp	0	0	1	0	3	0	0	1	0	0	1	0	50.00%
pred. CreditF	0	0	0	0	0	4	0	0	0	0	0	0	100.00%
pred. Comp	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
pred. JointVe	0	0	0	0	0	0	0	2	0	0	0	0	100.00%
pred. Merger	0	0	0	0	1	0	0	0	2	0	0	0	66.67%
pred. Acquis	0	0	0	1	0	0	1	0	1	3	0	0	50.00%
pred. Confer	0	0	0	0	0	0	0	0	0	1	2	0	66.67%
pred. Comp	0	1	0	0	0	0	0	0	0	0	0	2	66.67%
class recall	100.00%	80.00%	50.00%	66.67%	75.00%	100.00%	0.00%	66.67%	66.67%	75.00%	66.67%	100.00%	

table and the right-most column show the recall and precision, for each category. Macro- and microaverage values can be calculated based on the confusion matrix. This process is performed later by TCModule wrapper application.

Figure 4.12: Result view of document distribution across categories



Result Overview | ExampleSet (Filter out missing v

Meta Data View | Data View | Plot View | Advanced Cha

ExampleSet (14 examples, 0 special attributes, 2 regular attributes)

Row No.	category	count(id)
1	Acquisition	51
2	Bankruptcy	24
3	Buybacks	21
4	CompanyEarningsAnnouncement	50
5	CompanyExpansion	22
6	CompanyForceMajeure	22
7	CompanyLayoffs	28
8	CreditRating	29
9	Dividend	29
10	IPO	25
11	JointVenture	25
12	Merger	26
13	ProductRecall	21
14	ProductRelease	32

Figure 4.12 depicts the resulting document distribution of documents across categories. The purpose of this view is only informative.

4.4.3 Apply model process

Apply model process applies previously saved model to an arbitrary document entered by the user.

Figure 4.13: Overall view of *Apply model* process

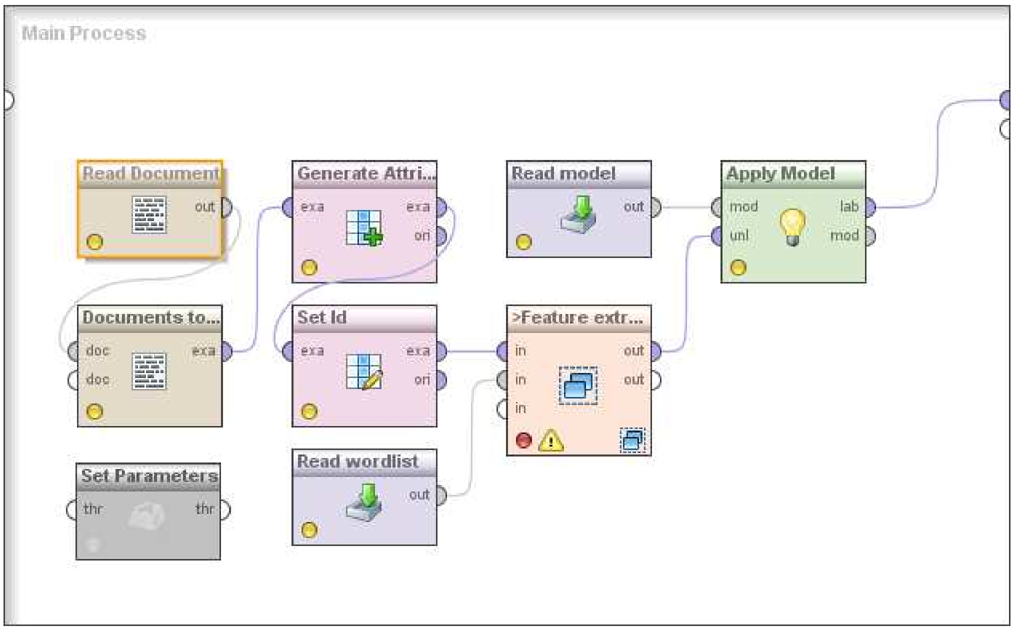


Figure 4.13 shows the overall process view. There are only two parameters to be set for the *Set parameters* operator, `input_filepath`, and `input_document`[8]. Below we describe the remaining operators and their function:

- *Read document* reads the content of a document from the specified `input_document` path.
- *Documents to Data* converts the document content to a RapidMiner Document. I.e. together with *Read document* it acts like the database reading operator of *Train model* process (Section 4.4.1).
- *Generate Attributes* generates an `id` attribute for the document.

[8]Similarly to the case of *Train model* process (Figure 4.5), the *Set parameters* operator is disabled, to collaborate with TCModule application which will set the parameters itself. If the process is to be run inside RapidMiner this operator should be enabled first.

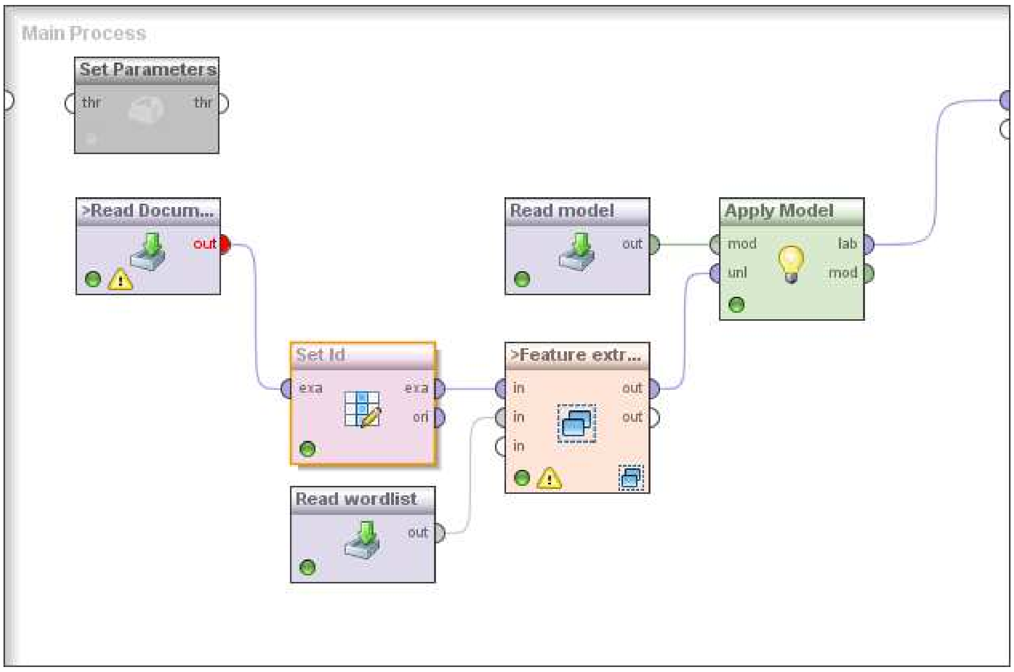
- *Set Id* assigns the `id` role to the `id` attribute.
- *Read wordlist* reads the stored word list.
- *Read model* reads the stored classification model.
- >*Feature extraction* extracts the features from document. The process is identical to the test data extraction used by *Train model* (tf measure instead of itf measure used for term weighting, and no token pruning). Also, analogically to *Train model* process, the word list generated based on the training data is used as an additional input for test data feature extraction (in order to make sure that the tokens that did not occur in the test data will still be present in the final feature set).
- *Apply model* applies the classification morel model to the test document.

The output of the classification is a classified example. It contains the original attributes of the test document and a number of additional special attributes, generated by *Apply model* operator. The most important is the `prediction(category)` attribute which tells which category has been assigned to the document. A number of `confidence(<category_name>)` attributes stores numerical values which correspond to the level of certainty that the document would belong to each of the categories.

4.4.4 Apply model multiple process

Apply model multiple process works analogically to the *Apply model* process, with the only difference that instead of single document read from file, it classifies a number of documents fetched by an SQL query. Figure 4.14 presents the process view.

Figure 4.14: Overall view of *Apply model multiple* process

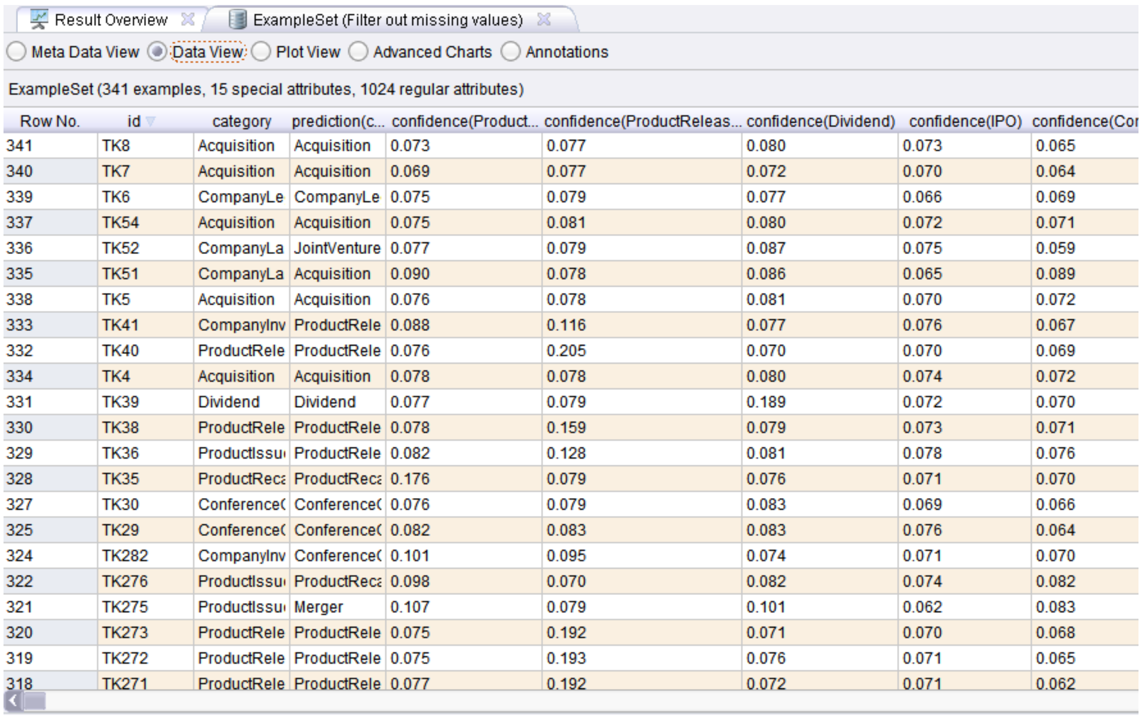


>*Read Documents* fetches the documents from database, in exactly the same manner as the operator with the same name in *Train model* process (described in Section 4.4.1). The file path for reading the SQL query is specified by `input_query_filepath` macro parameter.

The Figure 4.15 illustrates the result view of the process, which is labelled example set that contains multiple examples. Each of them has an additional set of special attributes: `prediction(category)` and a number of `confidence(<category>)`[9]. For instance, Row 340 in Figure 4.15 shows the results for the document with `id` *TK7*. The correct category of the document is *Acquisition*. The model has predicted correctly for it to belong to the same category. The confidence values for the document belonging to the categories *Product recall* and *Product release* are 0.69 and 0.77, respectively. The confidence value for *Acquisition* category was the highest and that is why the document has been assigned to it. On the other hand, the correct category of the document with `id` *TK41* (Row 333) should be *Company investment*, while it was assigned to the *Product release* category.

[9]Compare with the description of *Apply model* process in Section 4.4.3.

Figure 4.15: The result of *Apply model multiple* process



ExampleSet (341 examples, 15 special attributes, 1024 regular attributes)

Row No.	id	category	prediction(c...	confidence(Product...	confidence(ProductReleas...	confidence(Dividend)	confidence(IPO)	confidence(Cor
341	TK8	Acquisition	Acquisition	0.073	0.077	0.080	0.073	0.065
340	TK7	Acquisition	Acquisition	0.069	0.077	0.072	0.070	0.064
339	TK6	CompanyLe	CompanyLe	0.075	0.079	0.077	0.066	0.069
337	TK54	Acquisition	Acquisition	0.075	0.081	0.080	0.072	0.071
336	TK52	CompanyLa	JointVenture	0.077	0.079	0.087	0.075	0.059
335	TK51	CompanyLa	Acquisition	0.090	0.078	0.086	0.065	0.089
338	TK5	Acquisition	Acquisition	0.076	0.078	0.081	0.070	0.072
333	TK41	CompanyInv	ProductRele	0.088	0.116	0.077	0.076	0.067
332	TK40	ProductRele	ProductRele	0.076	0.205	0.070	0.070	0.069
334	TK4	Acquisition	Acquisition	0.078	0.078	0.080	0.074	0.072
331	TK39	Dividend	Dividend	0.077	0.079	0.189	0.072	0.070
330	TK38	ProductRele	ProductRele	0.078	0.159	0.079	0.073	0.071
329	TK36	ProductIssu	ProductRele	0.082	0.128	0.081	0.078	0.076
328	TK35	ProductRec:	ProductRec:	0.176	0.079	0.076	0.071	0.070
327	TK30	Conference(	Conference(	0.076	0.079	0.083	0.069	0.066
325	TK29	Conference(	Conference(	0.082	0.083	0.083	0.076	0.064
324	TK282	CompanyInv	Conference(	0.101	0.095	0.074	0.071	0.070
322	TK276	ProductIssu	ProductRec:	0.098	0.070	0.082	0.074	0.082
321	TK275	ProductIssu	Merger	0.107	0.079	0.101	0.062	0.083
320	TK273	ProductRele	ProductRele	0.075	0.192	0.071	0.070	0.068
319	TK272	ProductRele	ProductRele	0.075	0.193	0.076	0.071	0.065
318	TK271	ProductRele	ProductRele	0.077	0.192	0.072	0.071	0.062

This misclassification occurred because the confidence value for *Product release* category was higher than the confidence for the *Company investment* category.

4.5 User interface

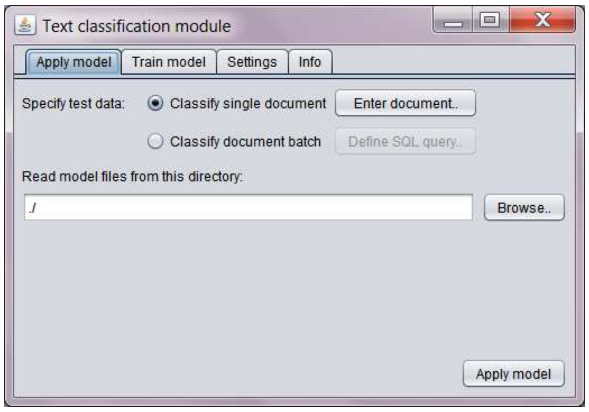
TCModule provides means for modifying, running and fetching results of the process through a GUI. Internally, RapidMiner process interface is separated from the graphical interface, to ensure enough portability of the software.

The main application window consists of four tabs:

1. *Apply model* – here the user can apply a model that has been previously stored on the hard drive.

   The user selects the directory where the model has been saved. In order to provide the data to be classified, the user can either copy-paste the content of a single

Figure 4.16: *Apply model* tab of TCModule



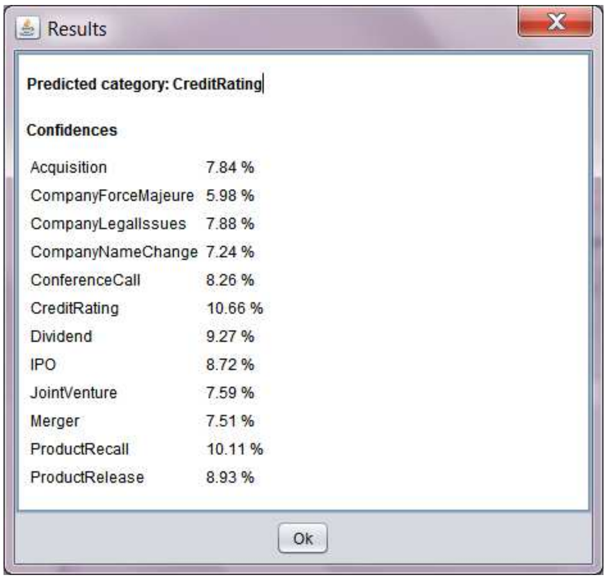
Specify test data: [✓ Classify single document] Classify document batch

Read model files from this directory:

./

document, or specify an SQL query that will fetch a batch of documents from the database. The *Apply model* tab is depicted in Figure 4.16.

Figure 4.17: Model application results



Predicted category: CreditRating

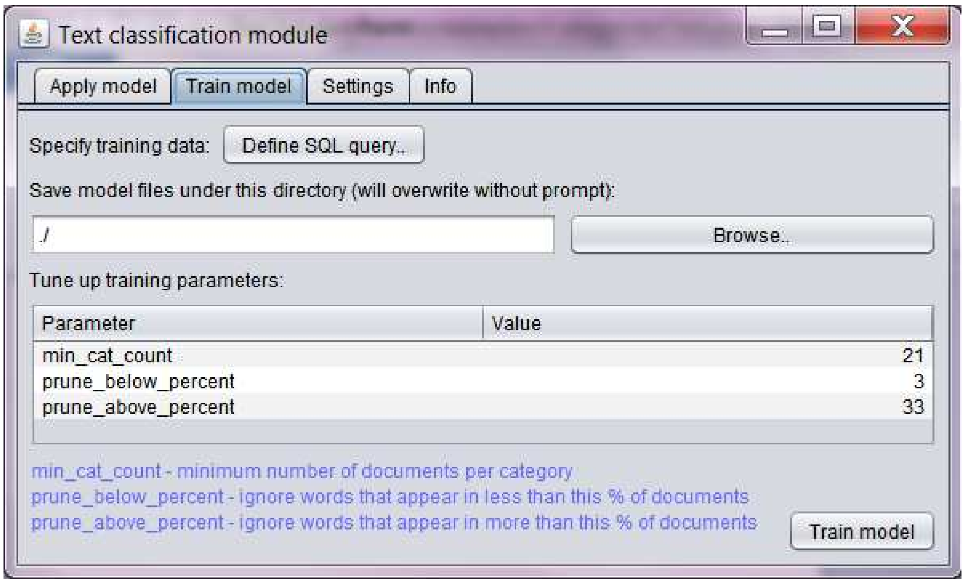
Confidences

Acquisition	7.84 %
CompanyForceMajeure	5.98 %
CompanyLegalIssues	7.88 %
CompanyNameChange	7.24 %
ConferenceCall	8.26 %
CreditRating	10.66 %
Dividend	9.27 %
IPO	8.72 %
JointVenture	7.59 %
Merger	7.51 %
ProductRecall	10.11 %
ProductRelease	8.93 %

As a result of model application, the user is displayed a dialog with information about the confidence values for each of the categories in the model, as well as the category name that is most probable for given document. If more than one document was fetched for classification, the software will only show a list of document `id` values and the most probable categories for each document. The document classification result dialog is shown in Figure 4.17.

2. *Train model* – here the user can re-train the model by specifying an SQL query that will fetch the documents.

Figure 4.18: *Train model* tab of TCModule



Specify training data:

Save model files under this directory (will overwrite without prompt):

./

Tune up training parameters:

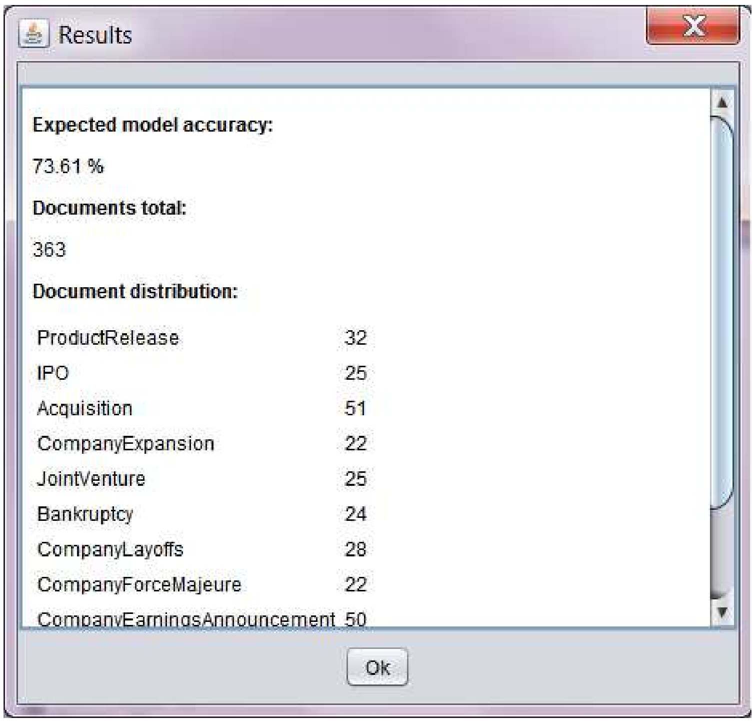
Parameter	Value
min_cat_count	21
prune_below_percent	3
prune_above_percent	33

min_cat_count - minimum number of documents per category
prune_below_percent - ignore words that appear in less than this % of documents
prune_above_percent - ignore words that appear in more than this % of documents

The user can adjust three threshold parameter values:
– `min_cat_count` – specifies the minimum number of documents per category. If a category has less documents than specified by this parameter, the category will not be included in the model. Categories with very few training documents are not likely to be recognised correctly for new data.
– `prune_below_percent` – specifies how many percent of least frequent words should be ignored. The words that occur in only one or two documents are probably not meaningful to any of the categories.

– `prune_above_percent` – specifies how many percent of most frequent words should be ignored. Words that appear in all or most documents are not useful for distinguishing between the categories. The Figure 4.18 shows the *Train model* tab.

Figure 4.19: Model training results



Expected model accuracy:

73.61 %

Documents total:

363

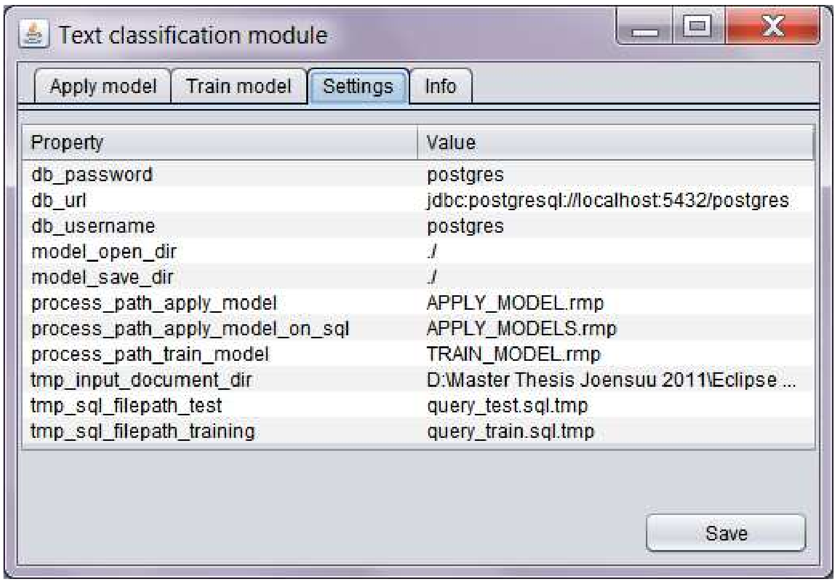
Document distribution:

ProductRelease	32
IPO	25
Acquisition	51
CompanyExpansion	22
JointVenture	25
Bankruptcy	24
CompanyLayoffs	28
CompanyForceMajeure	22
CompanyEarningsAnnouncement	50

After the model has been learnt, the user is shown the expected accuracy of the model. This accuracy is based on training another model with the same routine on a part of the data and then testing it on the remaining part (test data). This results is not the precise accuracy of the trained model, but rather an estimate that can be used for adjusting the threshold parameters. Besides the model accuracy, the software displays the categories and the number of documents per category. Figure 4.19 presents the training model results dialog.

3. *Settings* – here the user can alter the settings of TCModule. These include database connection settings and file paths from which the application looks for the model files and stores temporary files by default. After pressing the ,,Save" button, the settings are saved to a file, and will be loaded with the next launch of the application.

Figure 4.20: *Settings* tab of TCModule



Property	Value
db_password	postgres
db_url	jdbc:postgresql://localhost:5432/postgres
db_username	postgres
model_open_dir	./
model_save_dir	./
process_path_apply_model	APPLY_MODEL.rmp
process_path_apply_model_on_sql	APPLY_MODELS.rmp
process_path_train_model	TRAIN_MODEL.rmp
tmp_input_document_dir	D:\Master Thesis Joensuu 2011\Eclipse ...
tmp_sql_filepath_test	query_test.sql.tmp
tmp_sql_filepath_training	query_train.sql.tmp

The *Settings* tab is depicted in Figure 4.20. Below we present an overview of the settings options and their meanings:

- `db_url` is the URL of the database
- `db_username` is the database user name
- `db_password` is the database user password
- `model_open_dir` is the path with the classification model and the word list
- `model_save_dir` is the path where the model and the word list will be stored after running the training process
- `process_path_apply_model` is the path to the *Apply model* process file
- `process_path_apply_model_on_sql` is the path to the *Apply model multiple* process file
- `process_path_train_model` is the path to the *Train model* process file
- `tmp_sql_filepath_training` is the path relative to the process file, where the temporary text file containing the query for fetching training documents will be stored

- `tmp_sql_filepath_test` is the path relative to the process file, where the temporary text file containing the query for fetching test documents will be stored
- `tmp_input_document_dir` is the path relative to the process file, where the temporary text file containing the single document for classification will be stored

4. *Info* – a simple panel with basic information about the software.

4.6 Communication between TCModule and RapidMiner

4.6.1 Process input

The RapidMiner processes need to receive input from the Java application they are wrapped in. Such communication is accomplished by using RapidMiner *macro* mechanism. Macros are also referred to as *parameters* in TCModule. As mentioned in Section 4.4, each of the underlying RapidMiner processes expects a set of macros being defined before running it. In this way the SQL queries, the file paths, and the threshold values are passed to the processes.

This mechanism is transparent for the GUI user. The user simply enters the text and the application will automatically store it in a corresponding temporary file. The paths to the temporary files can be modified in the *Settings* tab.

The input that deserves special attention is the SLR queries input. As described in the previous paragraph, the value of the macro providing an SQL query is the path of the file where the SQL query is stored in text format. The query itself needs to conform to certain criteria: it should return fields `id` and `content`. Id is the unique document identifier, and content is the document text, which later will be tokenized and processed.

In case of a query that is defining training documents, also a `category` field should be returned. It stores the category of a document. Furthermore, it is required that each row returned by the query has unique `id` value.

The database connection settings need to be defined upfront. In RapidMiner connection settings are specified as parameters of the *Read Database* operator, which does the actual data fetching. As an exception from macro-based input, TCModule modifies those operator parameters directly, according to the DB settings entered by the user in the *Settings* tab.

4.6.2 Fetching the process output

All the RapidMiner processes utilized by TCModule deliver an output of particular type to its source at a particular index. The specific number of output objects and their type depends on the process. Depending on the process it is running, TCModule fetches the relevant output and presents it in its GUI.

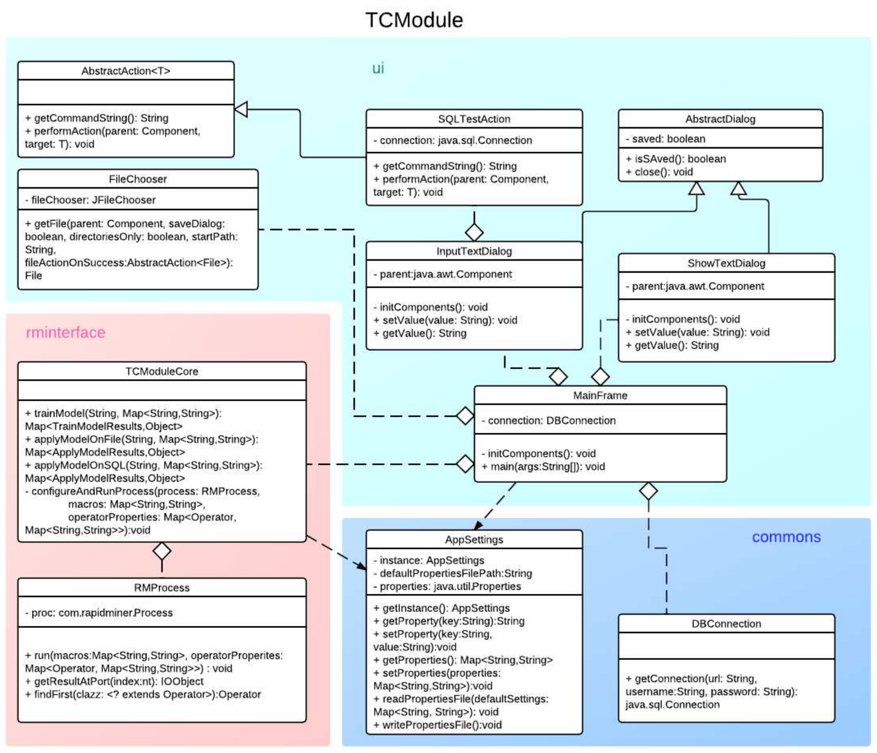
4.7 Implementation details

In this section, we present very brief overview of the code of TCModule.

Figure 4.21 depicts the UML class diagram of the TCModule. The system consists of three packages:

- `commons` - contains helper and general purpose classes. `AppSettings` reads, stores and writes application settings, and `DBConnection` encapsulates a method for creating a `java.sql.Connection` with appropriate connection settings.
- `ui` - contains the Swing component and related classes, as well as the main entry point to the application.

Figure 4.21: UML class diagram of TCModule


TCModule
ui
AbstractAction<T>
+ getCommandString(): String
+ performAction(parent: Component, target: T): void
SQLTestAction
- connection: java.sql.Connection
+ getCommandString(): String
+ performAction(parent: Component, target: T): void
AbstractDialog
- saved: boolean
+ isSAved(): boolean
+ close(): void
FileChooser
- fileChooser: JFileChooser
+ getFile(parent: Component, saveDialog: boolean, directoriesOnly: boolean, startPath: String, fileActionOnSuccess:AbstractAction<File>): File
InputTextDialog
- parent:java.awt.Component
- initComponents(): void
+ setValue(value: String): void
+ getValue(): String
ShowTextDialog
- parent:java.awt.Component
- initComponents(): void
+ setValue(value: String): void
+ getValue(): String
rminterface
TCModuleCore
+ trainModel(String, Map<String,String>): Map<TrainModelResults,Object>
+ applyModelOnFile(String, Map<String,String>): Map<ApplyModelResults,Object>
+ applyModelOnSQL(String, Map<String,String>): Map<ApplyModelResults,Object>
- configureAndRunProcess(process: RMProcess, macros: Map<String,String>, operatorProperties: Map<Operator, Map<String,String>>):void
MainFrame
- connection: DBConnection
- initComponents(): void
+ main(args:String[]): void
RMProcess
- proc: com.rapidminer.Process
+ run(macros:Map<String,String>, operatorProperites: Map<Operator, Map<String,String>>) : void
+ getResultAtPort(index:nt): IOObject
+ findFirst(clazz: <? extends Operator>):Operator
AppSettings
- instance: AppSettings
- defaultPropertiesFilePath:String
- properties: java.util.Properties
+ getInstance(): AppSettings
+ getProperty(key:String):String
+ setProperty(key:String, value:String):void
+ getProperties(): Map<String,String>
+ setProperties(properties: Map<String,String>):void
+ readPropertiesFile(defaultSettings: Map<String, String>): void
+ writePropertiesFile():void
commons
DBConnection
+ getConnection(url: String, username:String, password: String): java.sql.Connection

- `rminterface` - the interface to the RapidMiner API. It contains the following classes:

  `TCModuleCore` - the main class for training and applying the classification model. It contains methods for running each of the three RapidMiner processes. All these methods take as the input the file path of the process and a set of macro values. They return either a `TrainModelResults` → or `ApplyModelResults` → `Object` mapping. The type of mapped value is assumed based on the mapping key. The methods of the `TCModuleCore` class are the following:

  – `Map<TrainModelResults,Object> trainModel()` trains a classifer on a set of documents retrieved from database. It returns a set of `TrainModelResults`-`Object` pairs, and each pair is a piece of information about the trained model:

    ∗ `MODEL_ACCURACY` is the expected model performance, presented as a floating point number from the $[0, 1]$ interval,

    ∗ `DOCUMENTS_TOTAL` defines the total number of training documents,

    - `DOCUMENT_DISTRIBUTION` is the category distribution across training documents. It maps the category name to the number of documents belonging to it.
  - `Map<ApplyModelResults,Object> applyModelOnFile()` applies the previously trained model on a single file. It returns the same information as the `applyModelOnSQL()` method, except that only one document is included in the results.
  - `Map<ApplyModelResults,Object> applyModelOnSQL()` applies the previously trained model on a set of documents retrieved from database. It returns the following information:
    - `CONFIDENCE_MAPPING` hash table that maps each document id into another hash table, which maps category names to their confidence values,
    - `CLASSIFICATION_MAPPING` hash table that maps document id to its predicted category name.

`RMProcessWrapper` is the wrapper for a RapidMiner process.

  - runs the `xml` process with a set of predefined macros. The `run()` method expects the following input variables:
    - `Map<String,String> macros` - set of macros to be set the process before running it. The set of macros depends on the process, and each such set was described in Sections 4.4.1, 4.4.3 and 4.4.4.
    - `Map<Operator, Map<String,String>> operatorProperties` - a table that maps an operator to set of properties to be set on it before running the process. For example, this allows defining the database settings on the `DatabaseDataReader` operator.
  - retrieves the results at the port of the given index.

4.8 Input document parser

As mentioned in Section 4.2, TCModule is accompanied by a parser which automatically fetches the documents and categories into the database from a predefined file structure.

The parser is located in the `dbparser` package. The main class is the `ParserMain` class. It contains four constants:

- `String documentsPath` points to the input document files,
- `String csvDocumentToCategoryPath` points to the `csv` file containing information about files and categories,
- `boolean replaceOld` defines whether the previous data in the database should be replaced before inserting the new one,
- `boolean skipAmbigious` defines if during setting the `max_weight` field of `document_to_category` relation, the program should skip documents which have equal maximum weight in regards to more than one category;

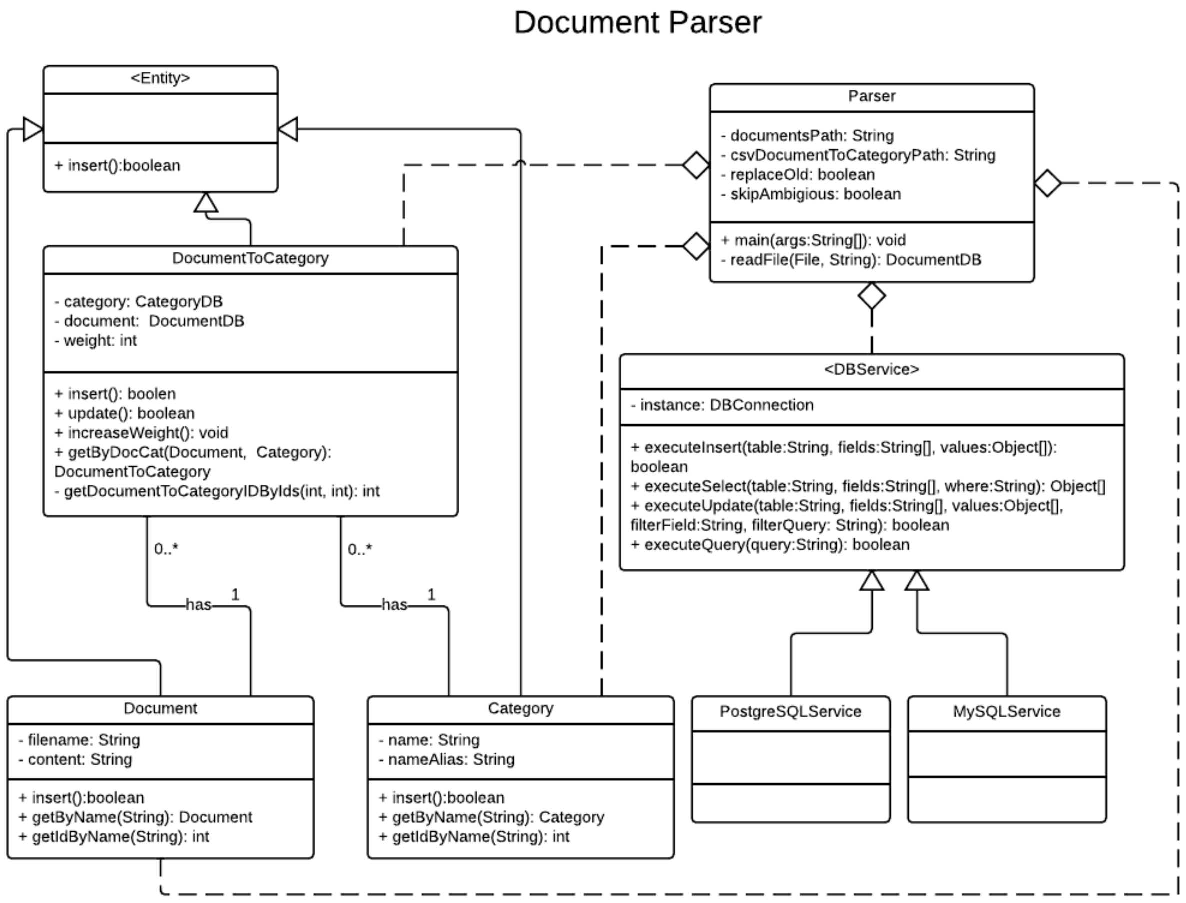
Figure 4.22 presents the UML class diagram of the document parser. The parser can be run by launching the `static void main` method of `ParserMain` class.

4.8.1 Expected file structure

The parser expects the input files to be in plain text format and have the `txt` extension. The `csv` file contains information about each document-category relation. Expected file structure is as follows:

```
document_id;category_name;classifying_sentence;
document_id;category_name;classifying_sentence;
document_id;category_name;classifying_sentence;
document_id;category_name;classifying_sentence;
```

Figure 4.22: Document parser UML class diagram


Document Parser
<Entity>
+ insert():boolean
Parser
- documentsPath: String
- csvDocumentToCategoryPath: String
- replaceOld: boolean
- skipAmbigious: boolean
+ main(args:String[]): void
- readFile(File, String): DocumentDB
DocumentToCategory
- category: CategoryDB
- document: DocumentDB
- weight: int
+ insert(): boolen
+ update(): boolean
+ increaseWeight(): void
+ getByDocCat(Document, Category): DocumentToCategory
- getDocumentToCategoryIDByIds(int, int): int
<DBService>
- instance: DBConnection
+ executeInsert(table:String, fields:String[], values:Object[]): boolean
+ executeSelect(table:String, fields:String[], where:String): Object[]
+ executeUpdate(table:String, fields:String[], values:Object[], filterField:String, filterQuery: String): boolean
+ executeQuery(query:String): boolean
0..*
has
1
0..*
has
1
Document
- filename: String
- content: String
+ insert():boolean
+ getByName(String): Document
+ getIdByName(String): int
Category
- name: String
- nameAlias: String
+ insert():boolean
+ getByName(String): Category
+ getIdByName(String): int
PostgreSQLService
MySQLService

`document_id` is the unique document identifier, and it corresponds to document file name [10]. `Category_name` is the name of the category the document has been assigned to. Due to the properties of the training data used in the experiments in this thesis, each document may appear in the list more than once and hence can be assigned to multiple categories.

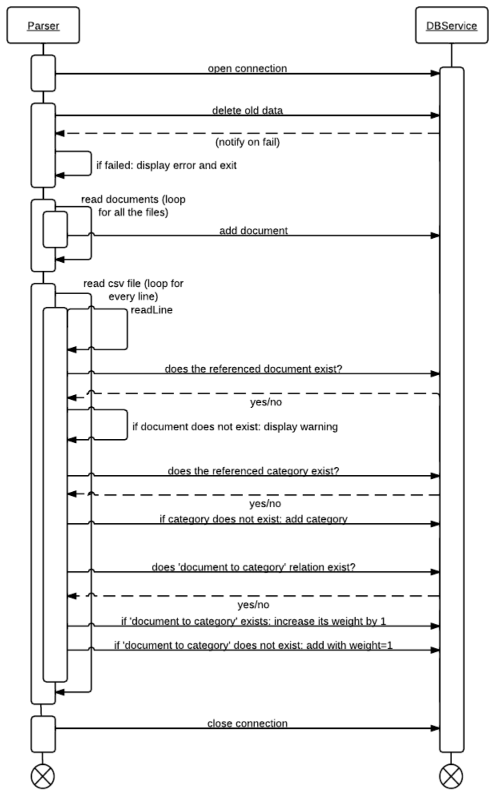
As described in Section 5.1, we assumed that a document is related to a category if it contains a sentence that is relevant to this category. The `classifying_sentence` field contains that related sentence, though the presence of this field in `csv` file is optional. In

[10]If the file name contains an underscore, the file name part after the last underscore will be skipped. This allows to add version flags to the document names without changing their `document_id` in the `csv` input file.

the end, nothing prevents the user from using the tool on data in which no classifying sentences are specified.

4.8.2 Execution logic

Figure 4.23: Document parser UML sequence diagram


Parser
DBService
open connection
delete old data
(notify on fail)
if failed: display error and exit
read documents (loop for all the files)
add document
read csv file (loop for every line)
readLine
does the referenced document exist?
yes/no
if document does not exist: display warning
does the referenced category exist?
yes/no
if category does not exist: add category
does 'document to category' relation exist?
yes/no
if 'document to category' exists: increase its weight by 1
if 'document to category' does not exist: add with weight=1
close connection

In Figure 4.23 is presented the UML sequence diagram of the document parser. The program parses the `csv` file, looking for valid input in the `documentsPath` directory. The documents that are present in the `csv` file but not found in the input folder are skipped and a warning is displayed. The same action is taken if any of the compulsory `csv` fields is missing from a record. The `document` table is populated with the valid `document_id` and `content` values. Simultaneously, the `category` table is filled with all the category names that are found in the `csv` file.

The parser also counts the number of sentences in the document that were relevant to a category, and stores it under the `weight` property. If certain document-category relation does not exist in the `document_to_category` table, a new one with `weight`$= 1$ is inserted. Otherwise the `weight` property of existing record is increased by 1.

The following method is employed to make sure that each document is assigned to exactly one category: the `document_to_category` records that received the highest `weight` score in regards to a particular category are going to have their `max_weight` property set to `true`. However, if the class constant `skipAmbigious` is set to `true`, and there is more than one document-category entity that got the same highest score for particular document, all of such document-category entities will have their `max_weight` property set to `false`. This is done in order to prevent ambiguous documents from being marked as belonging to more than one category. Such weight-based approach makes it possible to easily query the documents together with their most relevant categories, and be ensured that each document has only one category assigned.

5. Experiments

The current chapter describes the data set used in experimenting with the system and its collection process (Section 5.1), as well as the experiment settings (Section 5.2). It also describes the experiments performed on the collected data. In Section 5.3 we present and compare the results achieved by various classification routines.

5.1 Experiment data

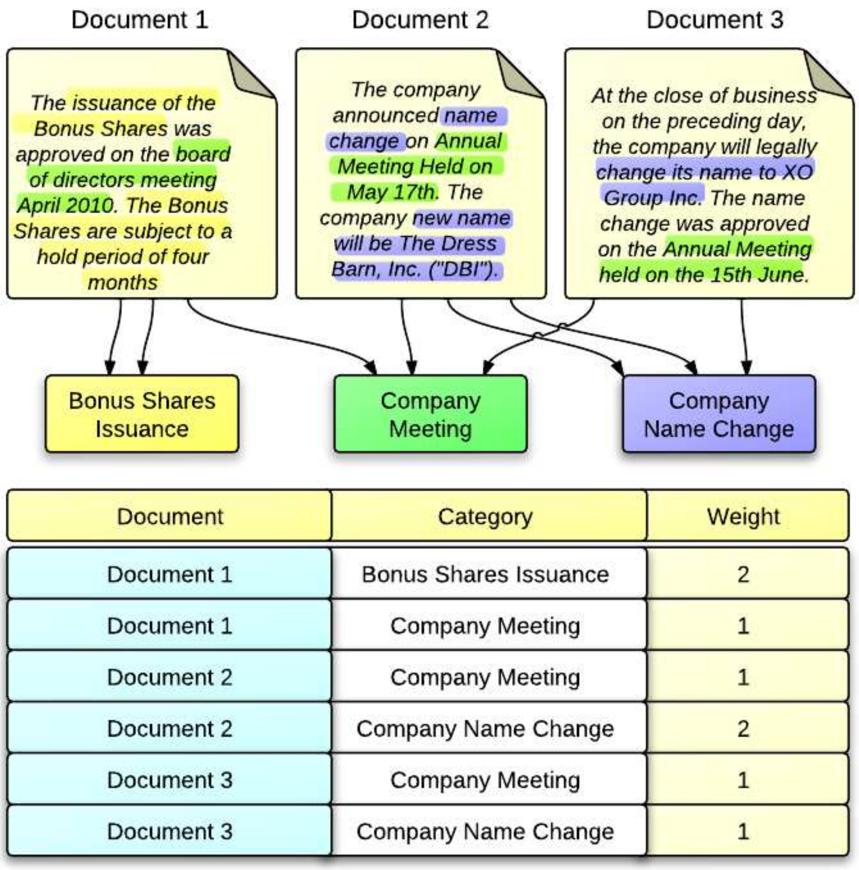
5.1.1 Data collection

The documents used as the training and test data have been collected manually by five partcipants of the e-leadership project. The data collection group consisted of two doctoral researchers, one PhD student and the author of the current thesis. The data was collected from various news resources on the web, such as REUTERS [1], The New York Times [2], YAHOO! Finance [3], as well as press releases and news concerning some particular companies, such as VALTRA [4], John Deere [5], Microsoft [6], Apple [7] and Samsung [8]. We skipped the articles containing news summaries, as they potentially combine data from various sources. The articles have been cleaned from advertisements or other irrelevant

[1]http://www.reuters.com/
[2]http://www.nytimes.com/
[3]http://finance.yahoo.com/news
[4]http://www.valtra.com/news/25.asp
[5]http://search.deere.com/DDC/en_US/News/
[6]http://www.microsoft.com/presspass/press/NewsArchive.mspx?cmbContentType=PressRelease
[7]http://www.apple.com/hotnews/
[8]http://www.samsung.com/us/news/newsList.do?gltype=globalnews

information. Only the sentences belonging to the actual news article or report were taken into account, i.e. adverts, links, copyright notices and other such text fragments were removed.

Figure 5.1: Usage of document to category relevance weights



The issuance of the Bonus Shares was approved on the board of directors meeting April 2010. The Bonus Shares are subject to a hold period of four months

The company announced name change on Annual Meeting Held on May 17th. The company new name will be The Dress Barn, Inc. ("DBI").

At the close of business on the preceding day, the company will legally change its name to XO Group Inc. The name change was approved on the Annual Meeting held on the 15th June.

Document	Category	Weight
Document 1	Bonus Shares Issuance	2
Document 1	Company Meeting	1
Document 2	Company Meeting	1
Document 2	Company Name Change	2
Document 3	Company Meeting	1
Document 3	Company Name Change	1

Each document in the data set was annotated as follows: Each sentence containing an event type defined in the CoProE ontology was tagged with the corresponding event type. For each document-category pair, a category *relevance weight* was specified based on the number of sentences appearing in the article that are relevant to that category. I.e. the category relevance weight for each category equals to the frequency of sentences that contain an event described by that event type category. Figure 5.1 presents how the document-category mapping is represented by using the relevance weights.

5.1.2 Document database

At the time of writing the thesis (November 2012), the data set contained 840 documents, out of which 679 are not ambiguous as to which category they belong to (their highest weight score favours one category). Out of the total of 42 categories, 38 contain at least one unambiguous document. There are 14 categories that contain more than 20 unambiguous documents, and there are 405 unambiguous documents belonging to those 14 categories. The Table 5.1 shows the document-category distribution for these 405 documents.

Table 5.1: Document-category distribution for unambiguous documents

Category	Number of documents
Acquisition	51
Company earnings announcement	50
Product release	32
Credit tating	29
Dividend	29
Company layoffs	28
Merger	26
Joint venture	25
IPO	25
Bankruptcy	24
Company force majeure	22
Company expansion	22
Buybacks	21
Product recall	21

5.2 Test settings

We compared the classification accuracies and efficiencies of a number of different classification approaches on our data set of business documents described in Section 5.1, using a number of different approaches. The experiments were run by using the processes implemented as part of TCModule. In order to generate comparable evaluation data, the same classification and evaluation routine was used for each TC method. The classification described in the thesis was performed on data set with two-level hierarchical category structure. This structure information was, however, utilised only in the hierarchical categorisation method.

For classification we used a subset of the documents that are not ambiguous and belong to a category that contains more than 20 documents. Moreover, for categories that have more than 22 documents we excluded the excess of documents with the lowest weights. These threshold numbers were chosen experimentally, in a way that there is as many documents included as possible without drastically decreasing the classification accuracies. Removing extra documents was done in order to compensate for number of documents that vary significantly across categories. Unbalanced data set would affect the evaluation: a majority of documents would belong to one category, assigning every documents to that category would be a good classifier. Moreover, different classification algorithms have varying tolerance to unbalanced data set. Therefore balancing the data helps in establishing an uniform classification routine, which would not favour one algorithm over the other. Such upper limit reduces both the training set size and overall classification accuracy, therefore it was applied only for running the comparison tests and is not going to be used in TCModule training process.

The data was further divided into test and training sets with the ratio 1:4, respectively. Ratios 1:9 and 1:4 are the ones that are the most often used in the relevant research literature. The 1:9 ratio was not suitable in this case, due to the relatively small size of the data set. The random seed was hard-coded to ensure that the differences between method performances are caused by the differences between algorithms, not the division of the data set.

The text pre-processing method included lowercase conversion, text tokenization, stopwords removal, word stemming, and filtering out tokens shorter than two letters. This routine is identical as the one applied in TCModule.

The baseline method used for comparison was SVM algorithm with *one-vs-many* category division strategy. The reason for choosing SVM as the baseline was that it is currently one of the best performing TC methods in means of the classification accuracy.

5.3 Results

In order to determine which methods perform the best on the document set we collected, the DAVID document database, we compared a number of TC models trained with different approaches and algorithms. We also attempted at introducing improvements and modifications in order to increase the classification accuracy.

Below we present comparison of the evaluated methods, their accuracies and average time of execution on a single machine. If not stated otherwise, the precision and recall measures are macro averages, which were described in Section 3.6.1. The macro accuracy is the harmonic average of precision and recall. The execution time is the time needed to perform feature extraction and run all the before mentioned cross-validation rounds. Execution time includes both the training and classification efficiencies.

5.3.1 Comparison of classification algorithms

In the this section we compare the performance of the three classification algorithms: k-NN, Naive Bayes and SVM, as well as their modifications.

Table 5.2: Comparison between the three classification algorithms

Method	Recall(%)	Precision(%)	Macro(%)	Exec.time(s)
SVM	67.46	67.97	67.71	8.0
Naive Bayes	39.37	47.45	43.04	0.8
K-NN (k=12)	72.90	73.60	73.25	1.0

The Table 5.2 summarizes the accuracy and efficiency of the three classification algorithms: SVM, Naive Bayes, and k-NN. The algorithms used are the standard ones that are implemented in RapidMiner. The best overall accuracy was presented by k-NN algorithm for $k = 12$, which scored 73.25% macro accuracy.

Naive Bayes yielded relatively low accuracy, which may be caused by not having enough training data. Because there was not enough documents in the training set, many words appeared only in few of them, and feature vectors contained a lot of zero values. Due to its nature, Naive Bayes is sensitive to such sparse featured data.

SVM yielded the best accuracy. It also performed the slowest, which is mainly because the classifier used by it is a binary classifier. It had to be adopted for a multi-category case by dividing the training data set into number of binary subclasses and training a separate classifier for each of these divisions.

Table 5.3: K-NN classification algorithm for different k values

Method	Recall(%)	Precision(%)	Macro(%)	Exec.time(s)
k=1	58.18	57.25	57.71	1.0
k=3	59.65	61.69	60.65	1.0
k=5	65.63	66.92	66.27	1.0
k=7	66.26	65.23	65.74	1.0
k=8	69.16	68.18	68.67	1.0
k=9	69.43	69.67	69.55	1.0
k=10	70.99	71.55	71.27	1.0
k=11	71.36	71.85	71.6	1.0
k=12	72.90	73.60	73.25	1.0

In Table 5.3 we can see comparison between the results scored by k-NN classifiers for various k values. The higher the k value was the more accurate result we achieved. Also longer execution time was expected for higher k values, however the k-NN appeared to be very roboust, despite the high space dimensionality and regardless of the number of neighbours. The highest accuracy was scored for $k = 12$. Please note that $k = 7$ is not valid as it is a divisor of number of classes (which is 14) yet still gives good results.

Figure 5.2: K-NN classification algorithm for different k

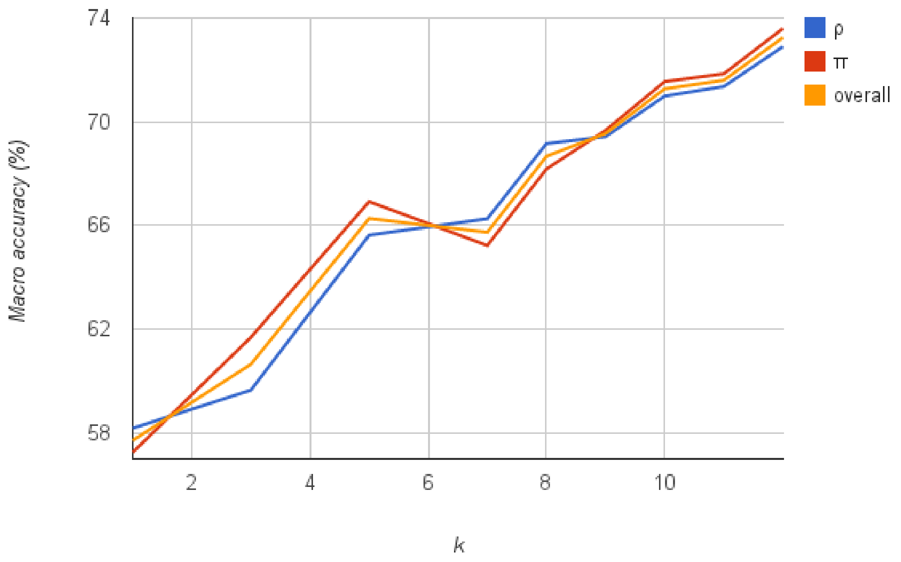


Figure 5.2 presents the accuracies for different k. The recall overperformed the precision for $k = 1$, and then from $k = 6$ until $k = 9$.

Table 5.4: SVM classification algorithm for different category set division strategy

Method	Recall(%)	Precision(%)	Macro(%)	Exec.time(s)
one-against-all	67.46	67.97	67.71	8.0
one-against-one	44.26	46.37	45.29	8.0

As it can be observed from Table 5.4, one-against-all strategy performed significantly worse for SVM classification. This has happened due to the limited training data size (one-to-one strategy divides training data into larger number of subsets). The execution times were similar, as in this case there was more data sets but less documents in each set.

Table 5.5: Flat vs. hierarchical classification results

Method	Recall(%)	Precision(%)	Overall(%)	Exec.time(s)
Macro				
Flat categorisation	67.46	67.97	67.71	8.0
Hierarchical categorisation	59.29	61.64	60.44	7.8
Top category categorisation	35.29	36.54	35.90	8.3
Micro				
Flat categorisation	81.92	79.18	67.41	
Hierarchical categorisation	73.91	74.80	59.18	
Top category categorisation	83.03	83.33	71.20	

5.3.2 Flat vs. hierarchical classification

Table 5.5 presents comparison between the flat and hierarchical categorisation methods. For both classification methods we used our baseline algorithm, SVM. It can be seen that hierarchical categorisation performed worse than the simple flat approach. Lower accuracy may have been caused by the limited training data set size, which is important factor in hierarchical categorisation. The execution time was expected to be longer, as a number of different classifiers were involved. The execution times are, however comparable. This can be explained by the fact that by applying a number of classifiers to two hierarchy levels we limited the data set on each level, which increased the speed of feature extraction and document classification.

The third and the last row of the Table 5.5 present the accuracy measures only for the top level of the category hierarchy, which was depicted in Figure 2.2. It is interesting to note that while macro and micro measures for the flat and full hierarchical categorisation are similar, the micro accuracy for the top category level is much higher than its macro accuracy. This reflects the fact that micro averaging focuses on the classified documents, while macro accuracy focuses on the categories. This is why micro averaging is strongly influenced by the difference in category sizes. Such difference must have occurred at the top level, because hierarchical categorisation we balanced the category sizes only at the lowest category level.

5.3.3 Comparison of text pre-processing routines

In this section we present a number of pre-processing techniques that we applied in an attempt to improve the classification accuracy and efficiency. One of them was the *n-gram* approach, mentioned in Section 3.3, as an alternative to dividing the document into tokens by words.

Table 5.6: N-gram classification for different n values

Method	Recall(%)	Precision(%)	Macro(%)	Exec.time(s)
$n = 2$	43.91	41.95	42.91	2.1
$n = 3$	67.33	68.58	67.95	20.7
$n = 4$	74.85	75.87	75.36	53.6
$n = 5$	75.41	75.25	75.33	51.9

Table 5.6 shows results for n-gram classification for different n values. N-grams were generated from text that has had stop-words removed upfront. The best overall result was achieved for $n = 4$, however the recall was slightly lower than for $n = 5$. It is interesting to note that processing time started to decrease after $n = 4$. This may be due to the fact that for relatively small n, the ,,grams" were shorter and therefore more likely to repeat, on the other hand for high n, the number of ,,grams" was smaller.

As suggested by Ifrim et al. (2008), using n-grams instead of words did not decrease the classification accuracy. Moreover, the effectiveness is slightly better than the results of the so-far winning k-NN algorithm. The efficiency is, on the other hand, much lower.

Figure 5.3 shows the accuracies for different n values in form of a chart. The precision was smaller than recall for $n = 2$, to exceed it at $n = 3$ and $n = 4$, and become very close for $n = 5$.

Figure 5.3: Accuracy of different classification routines

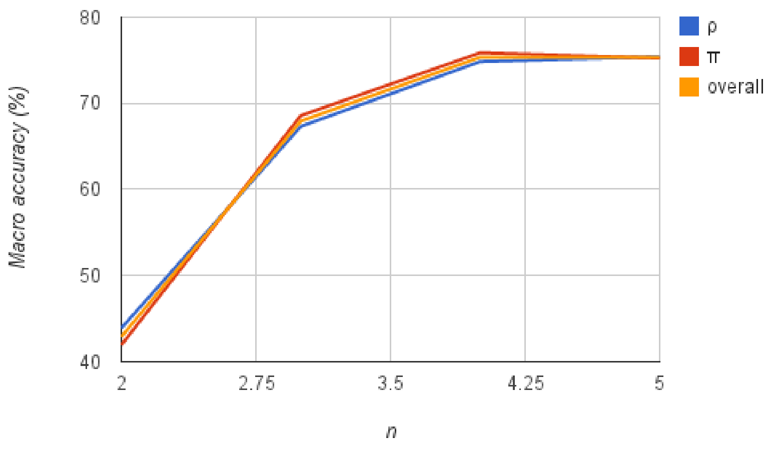


Table 5.7: Different text pre-processing routines comparison

Method	Recall(%)	Precision(%)	Macro(%)	Exec.time(s)
Full text pre-processing	67.46	67.97	67.71	8.0
Tf instead of *tf*idf*	63.95	65.53	64.73	6.4
Without stemming	66.31	67.02	66.66	7.6
N-gram $n = 5$	75.41	75.25	75.33	51.9

In Table 5.7 we can see a summary of different text pre-processing routines. As expected, using *term frequency* weighting decreased the classification accuracy. This is due to the fact that in that case term popularity across the document set is not taken into account, as explained in Section 3.3.1. As expected, skipping word stemming decreased (and speeded-up) the classification. The difference is not that big, however. This can be explained by the fact that keywords determining categories containing words such as ,,announcement" or ,,earning" after stemming become "announc" and "earn", which occur quite often in any business related article. N-gram accuracy is higher than the one of the word-based approach. However, the efficiency is significantly worse.

5.4 Discussion

Section 5.4.1 summarises the results of the accuracy evaluation. In Section 5.4.2 we briefly describe the obtained results in regards to the efficiency of the evaluated classification methods. The overall evaluation is presented in Section 5.4.3.

5.4.1 Accuracy

Figure 5.4: Accuracy of different classification routines

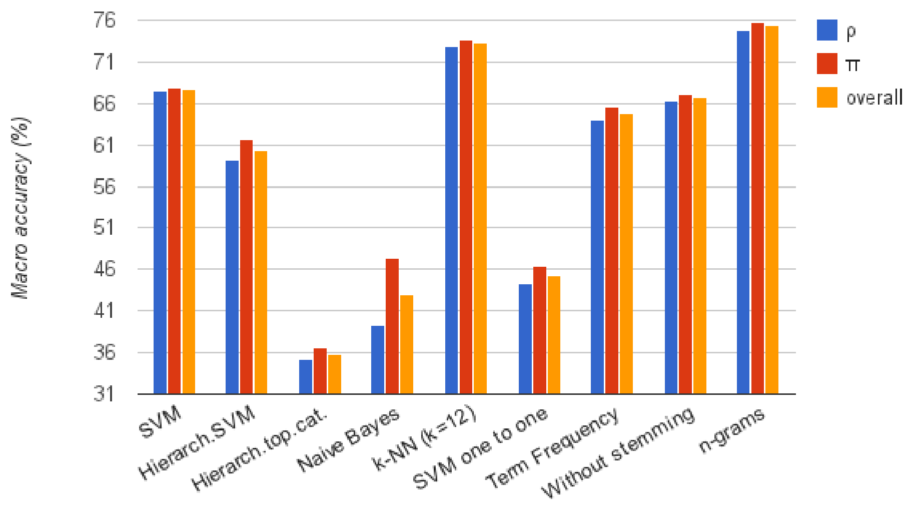


Figure 5.4 presents an overview of the accuracies of the evaluated classification algorithms. The list below presents the comparison between the accuracies of the evaluated methods:

- Hierarchical categorisation did not yield satisfactory results on our test data; it performed slightly worse than the simple flat categorisation. This could be due to the fact that our training data set is of a relatively limited size. As we explained in Section 3.5.1, hierarchical categorisation methods need a large number of training examples to perform accurately.

- Naive Bayes performed with very low accuracy. This can be explained by the sparsity of data in case of TC, which was even intensified by the fact of using small amount of training data. As mentioned before, many words appeared only in few documents, and the variances of corresponding feature vector components were close to zero.

The Naive Bayes yielded the largest difference between precision and recall measures. The precision tended to be higher than recall which means the algorithm did not perform well in finding all the documents that belongs to a category, but did not make that many wrong guesses.

- The highest accuracy was achieved by the k-NN algorithm when n was set to 12. These results differ from the results presented by Manning et al. (2008, Sec.13.6) (Table 5.8), where the SVM and k-NN algorithms showed similar macro accuracies. The difference is, however, small, which is understandable as we were using different data set.
- One-to-one SVM strategy performed with significantly worse accuracy than one-to-many. This was probably caused by the limited size of the training data set.
- Full text pre-processing resulted in the highest accuracy. Not using word stemming decreased the accuracy slightly. Using tf instead of idf lowered the accuracy, which was expected, as term frequency does not take into consideration the difference between the length of documents (i.e. the term frequencies are not normalized). Generating n-grams from word tokens yielded the best results for $n = 4$, and they outperformed the SVM classification.

It is interesting to compare achieved results with the ones reported in research literature. Manning et al. (2008, Sec.13.6) presented the effectiveness comparison between Naive Bayes, k-NN and SVM. The classification was done for 90 classes classification of ModApte split of Reuters-21578 (*Reuters-21578 Test Collection,* 2012), the widely used test collection for TC research.

Table 5.8 presents comparison between the three TC methods that were analysed in this thesis. According to the results reported by Manning et al. (2008, Sec.13.6), SVM

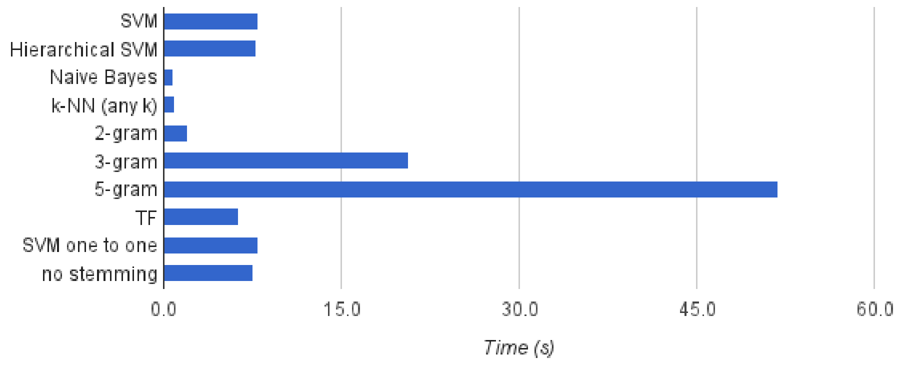
Table 5.8: Comparison of NB, k-NN and SVM classification accuracies for Reuters-21578 data set, according to Manning et al. (2008)

Accuracy measure	NB (%)	k-NN (%)	SVM (%)
Micro accuracy	80	86	89
Macro accuracy	47	60	60

method achieved results comparable to k-NN. This does not correspond to our results, where k-NN yielded significantly higher accuracy than SVM. However, the macro accuracy for Naive Bayes was much lower than for the other two methods in both cases.

5.4.2 Efficiency

Figure 5.5: Efficiency comparison



In addition to the changes in classification accuracy we observed the efficiencies of the evaluated classification methods. Figure 5.5 shows the comparison between execution times of each of them. Below we describe the findings:

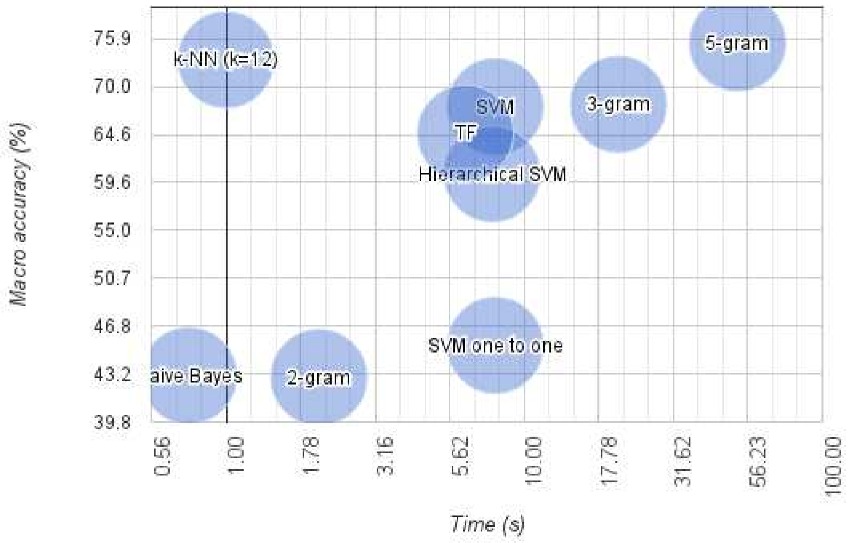
- Hierarchical categorisation took similar amount of time as the regular one (7.8–8.0 seconds). Even though more classifiers had to be trained, the number of documents classified by each of them was lower, therefore the training and classification time was not longer.

- The efficiencies of both Naive Bayes and K-nn methods (0.8 and 1.0 seconds, respectively) was much better than the efficiency of SVM (8.0 seconds). The reason for that is the fact that for SVM the data had to be converted to data with binomial labels (only two categories), which means running the algorithm several times for each category.
- Extracting n-grams for different n values slows down the classification significantly. The time of execution of training and classification phase for the n of the best accuracy took 53.6 seconds, which is almost seven times as long as the classification with using words as tokens.
- Using words that are not stemmed improved the efficiency by only 0.4 seconds. This is caused by the fact that the stemming takes slightly more time than the time that is saved by limiting the size of feature vectors. Also using tf measure for token weighting speeded up the process by 1.6 seconds, however it decreased the accuracy.
- Applying one-to-one SVM strategy took the same amount of time as one-to-many approach. We explained it by the fact that dividing the category sets into bigger number of subsets increased number of training and classification rounds, but also decreased number of documents per training and classification.

5.4.3 Overall evaluation

Figure 5.6 presents the overview of all the evaluated methods, comparing their accuracies and execution times. The k-NN algorithm performed best, both in terms of efficiency and effectiveness. The n-gram approach showed slightly better accuracy, however the efficiency was much worse than the k-NN algorithm. Different SVM approaches yielded similar results, while Naive Bayes performed with shortest execution time, however also the worst accuracy.

Figure 5.6: The overview of the evaluated methods



6. Conclusions

6.1 Summary

In this thesis we have described the current state of TC research, as well as introduced the innovative ontology-aided approach and training-less TC, i.e. the one for which we do not need to provide a training set. We performed comparison of three basic TC methods: Naive Bayes, K-nn and SVM. The results of the research were combined into a software system called TCModule, which is targeted at business documents. The system is able to train a classification model and then apply it to a set of documents. TCModule uses a relational database from where it retrieves the documents. The implementation of TCModule is based on classification processes that were developed using RapidMiner GUI.

We sought answers to the following research questions (introduced in Section 1.1):

1. What are the existing ways of performing TC and ontology-aided TC? What are their merits and flaws?
2. Which of these methods suits the DAVID system best?
3. Can these methods be improved and how?

We gave the answer to the first research question in Chapter 3. We presented and analysed Naive Bayes, K-nn and SVM algorithms. We found out that these traditional TC methods are widely employed and they yield good results. The feature extraction step, however, is crucial for high classification accuracy. We also introduced hierarchical classification, which allows to use information about category structure in process of classification. Nevertheless, its main drawback is the fact that it requires more training data than categorisation with flat category structure. Another type of TC we reviewed were ontology-aided solutions. This is due to the fact that the DAVID system relies heavily on a purpose-built ontology for storing and processing business information. In one of the approaches a taxonomy of tokens was created from the BOW model of a document and compared with the category structure. Such method was expected to perform well in the domain of patent documents, which contain a lot of novel vocabulary. Another ontology-aided solution – the train-less TC yielded results comparable to traditional TC, but it did not require any training documents.

In Section 5.3 we answered the second and the third research questions. We evaluated how well each of the TC methods performs on the data prepared for the DAVID system by running experiments with each of the selected TC methods. We discovered that hierarchical categorisation did not perform well in our application domain and test set. We speculated that this was caused by the limited training set. The K-nn and SVM algorithms showed similar accuracies, while Naive Bayes performed significantly worse. The execution times of Naive Bayes and K-nn were similar (around 1 second), while training a classifier with SVM algorithm and running it on test data took much longer (8 seconds).

In the latter part of the Section 5.3, we compared the accuracy and efficiency of different classification methods, which provided answers to the third research question. We observed that using n-grams of the length of eight words instead of single words improved the accuracy by more than 7%. Removing word stemming step from feature extraction process resulted in decreasing the accuracy by more than 1%.

6.2 Limitations and future work

Thesis leaves a lot of room for future modifications and improvements. Firstly, more training data is going to be available in the future for the DAVID system, and therefore for TCModule. With the new data, it will be possible to revisit hierarchical classification methods, as well as re-evaluate the other methods covered in this thesis. Currently our database contains only 14 categories than include more than 20 documents, which is too few to perform TC for all the events present in the test set and in the CoProE ontology. Moreover, there is still a potential in hierarchical classification, which performed poorly due to limited training data set.

An interesting question is how the classification accuracy would be influenced by applying classification based entirely on an ontology, in the manner presented by Janik & Kochut (2008) (see Section 3.5.2).

Possible improvements could also be gained by applying more advanced pre-processing routines. One of them could be adding more domain-specific words to the replacement list presented in Table 4.2 in the TCModule chapter. This would prevent the classifier from differentiating between terms that have the same meaning but are sometimes referred e.g. by an abbreviation. In the future, new words can be determined by manually analysing the content of incoming training documents. In addition, the data about particular companies could be extracted from the ontology and used as extra input for the replacement list (for example replacing name of each company with the word 'company'). This would enable the classifier to treat all references to companies in an uniform way, not differentiating between single companies.

Bibliography

Consortium, T. G. O. (2000), ‘Gene ontology: tool for the unification of biology’. http://www.geneontology.org/GO_nature_genetics_2000.pdf, accessed on 22nd August 2012.

Cortes, C. & Vapnik, V. (1995), ‘Support-vector networks’, *Machine Learning* **20**(3), 273–297. Kluwer Academic Publishers Hingham, MA, USA.

Ericsson, R. (2004), *Building Business Intelligence Applications with .NET*, Charles River Media, Inc. Rockland, MA, USA.

European Regional Development Fund (2005). http://europa.eu/legislation_summaries/employment_and_social_policy/job_creation_measures/l60015_en.htm, accessed on 14th April 2012.

Everitt, B. (2002), *Cambridge Dictionary of Statistics*, Cambridge University Press. Institute of Psychiatry, Kings College London.

Fellbaum, C. (1998), ‘Wordnet: An electronic lexical database’. Cambridge, MA: MIT Press. http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=8106, accessed on 22nd August 2012.

Finnish Funding Agency for Technology and Innovation (2012). http://www.tekes.fi/en/, accessed on 14th April 2012.

Ifrim, G., Bakýr, G. & Weikum, G. (2008), ‘Fast logistic regression for text categorization with variable-length n-grams’, *’08 Proceedings of the 14th ACM*

SIGKDD international conference on Knowledge discovery and data mining . Las Vegas, Nevada, USA.

Janik, M. & Kochut, K. (2008), 'Training-less ontology-based text categorization', *In Workshop on Exploiting Semantic Annotations in Information Retrieval (ESAIR 2008) at the 30th European Conference on Information Retrieval (ECIR'08)* . Glasgow, Scotland.

Kakkonen, T. (2010), 'Towards e-leadership project'. `http://cs.joensuu.fi/~tkakkone/eleadership/research.html`, accessed on 14th April 2012.

Kakkonen, T. & Mufti, T. (2011), 'Developing and applying a company, product and business event ontology for text mining', *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies* . Graz, Austria.

Koller, D. & Sahami, M. (1997), 'Hierarchically classifying documents using very few words', *'97 Proceedings of the Fourteenth International Conference on Machine Learning* . San Francisco, CA, USA.

Liu, C.-L. (2008), 'Partial discriminative training for classification of overlapping classes in document analysis', *International Journal on Document Analysis and Recognition* **11**(2), 53–62.

Lösch, U. & Nikitina, N. (2009), 'The newsevents ontology - an ontology for describing business events', *Proceedings of the 8th International Semantic Web Conference, 1st Workshop on Ontology Design Patterns* . Washington DC, USA.

Manning, C. D., Raghavan, P. & Schtze, H. (2008), *Introduction to Information Retrieval*, Cambridge University Press. Stanford University, California, USA.

McCarthy, J. (2007), 'What is artificial intelligence?'. Stanford University, Computer Science Department, California, USA.

http://www-formal.stanford.edu/jmc/whatisai/whatisai.html, accessed on 22nd August 2012.

Mufti, T. (2012), Populating and implementing an api for coproe - an ontology for text mining for competitive intelligence, Master's thesis. School of Computing, University of Eastern Finland, Joensuu, Finland.

Næss, A. B. (2007), Bayesian text categorization, Master's thesis. Norwegian University of Science and Technology, Trondheim, Norway.

Natale, D. A., Arighi, C. N., Barker, W. C., Blake, J. A., Bult, C. J., Caudy, M., Drabkin, H. J., DEustachio, P., Evsikov, A. V., Huang, H., Nchoutmboube, J., Roberts, N. V., Smith, B., Zhang, J. & Wu, C. H. (2011), 'The protein ontology: a structured representation of protein forms and complexes', *Nucleic Acids Research* **39**. Published online, http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3013777/?tool=pubmed, accessed on 22nd August 2012.

Netzer, Y., Gabay, D., Adler, M., Goldberg, Y. & Elhadad, M. (2009), 'Ontology evaluation through text classification'. Department of Computer Science Ben Gurion University of the Negev, Israel.

Peng, F., Schuurmans, D. & ShaojunWang (2003), 'Language and task independent text categorization with simple language models', *Proceedings of HLT-NAACL 2003, Main Papers, Stroudsburg, PA, USA* .

PostreSQL offical web site (2012).
URL: *http://www.postgresql.org/about/, accessed on 14th April 2012*

Ramakrishnan, A. (2012), 'Leveraging the power of unspsc for business intelligence', *Satyam White Paper* . http://www.unspsc.org/AdminFolder/Documents/Leveraging_the_Power_of_UNSPC.pdf, accessed on 22nd August 2012.

RapidMiner 5.0 User Manual (2010).
`http://netcologne.dl.sourceforge.net/project/rapidminer/1.RapidMiner/5.0/rapidminer-5.0-manual-english_v1.0.pdf`, accessed on 14th April 2012.

RapidMiner 5 API Documentation (2009).
`http://rapid-i.com/api/rapidminer-5.1/overview-summary.html`, accessed on 14th April 2012.

Reuters-21578 Test Collection, (2012).
`http://www.daviddlewis.com/resources/testcollections/reuters21578/`, accessed on 14th April 2012.

RFC 2396 (1998). `http://tools.ietf.org/html/rfc2396`, accessed on 20th August 2012.

Scott, S. & Matwin, S. (1999), 'Feature engineering for text classification', *Proceedings of the Sixteenth International Conference on Machine Learning (ICML99)* . University of Ottawa, Ottawa, Canada.

Sebastiani, F. (2005), 'Text categorization', *Alessandro Zanasi (ed.), Text Mining and its Applications* . Southampton, UK.

Seddiqui, M. H. & Aono, M. (2008), 'Use of ontology in text classification', *19th Technical Committee on Semantic Web and Ontology* . Toyohashi University of Technology, Aichi, Japan.

Siegel, D. (2011), 'The semantic web and the power of pull'.
`http://thepowerofpull.com/what/introduction`, accessed on 15th December 2011.

Soares, C., Peng, Y., Meng, J., Washio, T. & Zhou, Z.-H. (2008), 'Applications of data mining in e-business and finance: Introduction', *Frontiers in Artificial Intelligence and Applications, IOS Press Amsterdam, The Netherlands* . Amsterdam, The Netherlands.

Sun, A. & Lim, E.-P. (2001), ‘Hierarchical text classification and evaluation’, *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM 2001)* . California, USA.

Tenenboim, L., Shapira, B. & Shoval, P. (2008), ‘Ontology-based classification of news in an electronic newspaper’, *Book 2 Advanced Research in Artificial Intelligence* . Varna, Bulgaria.

Theodoridis, E. & Koutroumbas, K. (2008), *Pattern Recognition*, iv edn, Academic Press. San Diego, California, USA.

The World Wide Web Consortium (2012). `http://www.w3.org/`, accessed on April 14th 2012.

W3C Resource Description Framework Specifications (2012). `http://www.w3.org/RDF/`, accessed on 14th April 2012.

W3C Web Ontology Language Specifications (2012). `http://www.w3.org/TR/owl2-overview/`, accessed on 14th April 2012.

Wikipedia website (2012). `http://www.wikipedia.org`, accessed on 14th April 2012.

World Intellectual Property Organisation website (2012). `http://www.wipo.int`, accessed on 14th April 2012.