About the Exploration of Data Mining Techniques using Structured Features for Information Extraction

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Felix Jungermann

Dortmund

2012

Tag der mündlichen Prüfung: 05.06.2012

Dekanin:
Prof. Dr. Gabriele Kern-Isberner

Gutachter:
Prof. Dr. Katharina Morik
Prof. Dr. Dietmar Jannach

Acknowledgements

Als erstes möchte ich mich ganz herzlich bei meiner Betreuerin Prof. Dr. Katharina Morik bedanken. Sie hat mich nicht nur außerordentlich bei der Entwicklung dieser Arbeit bestärkt und unterstützt, sie hat mich zudem auch immer wieder fachlich aufs Neue beeindruckt. Die gemeinsame Arbeit mit ihr hat mich in meiner persönlichen Entwicklung nicht nur sehr geprägt sondern auch gestärkt, wofür ich ihr nochmals meinen größten Dank aussprechen möchte.

Mein Dank gilt zudem auch meinem Zweitbetreuer Prof. Dr. Dietmar Jannach, der – auch wenn oder gerade weil er nicht hauptsächlich im Gebiet des maschinellen Lernens aktiv ist – mir immer wieder hilfreiche Tipps und Anregungen zu meiner Arbeit geben konnte.

Des Weiteren möchte ich mich bei allen aktuellen und ehemaligen Mitarbeitern am LS8 für die tägliche Unterstützung, den Austausch und die Gespräche auf dem Flur bedanken. Bei Nico bedanke ich mich besonders für die Implementierung und Einweisung in CRFs auf GPUs. Herauszuheben seien weiterhin Anja, Sammy und Andreas, ohne die am LS8 wohl gar nichts ginge. Ein ganz besonderer Dank gilt Christian, Marco und Benjamin, die mehrere Jahre gemeinsam mit mir als WiMis am LS8 verbrachten.

Ich möchte hiermit auch der Deutschen Forschungsgesellschaft (DFG) meinen Dank aussprechen, da sie meine Arbeit durch die Finanzierung zweier Sonderforschungsbereiche teilweise mitfinanziert hat. Es handelt sich bei besagten SFBs um den SFB 475 ”Komplexitätsreduktion in multivariaten Datenstrukturen” und um den SFB 876 ”Verfügbarkeit von Information durch Analyse unter Ressourcenbeschränkung”.

Bedanken möchte ich mich auch bei meiner Familie, die nicht nur aus meinen Eltern sondern auch aus meinen Schwestern, Cousinen, Onkel, Tanten, Schwiegereltern, etc. besteht. Sie haben mich und meine Arbeit – auch wenn sie sie nicht verstanden haben sollten – immer unterstützt. Insbesondere bedanke ich mich bei meinen Eltern, die mein Studium finanziert haben. Dafür und für die Unterstützung in allen Lebenslagen bin ich sehr, sehr dankbar.

Abschließend jedoch möchte ich mich bei meiner geliebten Frau Simone und meinen beiden Kindern Rieke und Klaas bedanken. Ohne eure Hilfe in jedweder Form wäre diese Arbeit sicherlich noch nicht fertig.

Ich liebe euch!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The World Wide Web is a huge source of information. The amount of information being available in the World Wide Web becomes bigger and bigger every day. It is impossible to handle this amount of information by hand. Special techniques have to be used to deliver smaller excerpts of information which become manageable. Unfortunately, these techniques like search engines, for instance, just deliver a certain view of the information's original appearance. The delivered information is present in various types of files like websites, text documents, video clips, audio files and the like. The extraction of relevant and interesting pieces of information out of these files is very complex and time-consuming. Special techniques which allow for an automatic extraction of interesting informational units will be analyzed in this work. Such techniques are based on machine learning methods. In contrast to traditional machine learning tasks the processing of text documents in this context needs certain techniques. The structure of natural language contained in text document poses constraints which should be respected by the machine learning method. These constraints and the specially tuned methods respecting them are another important aspect in this work.

In this introduction we will define the research questions which we will solve in this work. We will give an intuitive example which shall be used during the whole introduction to identify the particular research questions. That certain example is sentiment analysis [Liu, 2010] (see Section 5.3.1). Sentiment analysis or opinion mining is used for the identification and extraction of opinions concerning particular entities. Those entities are products in most of the practical use-cases in which sentiment analysis is employed. Figure 1.1 shows an exemplary sentence which can be found somewhere

The computer is great but the hard drive is disgusting.

Figure 1.1: A sentence containing opinions concerning entities.

in the World Wide Web (WWW), e.g. in a forum. The goal of sentiment analysis is to extract the entities and the corresponding opinions. The presented example contains the opinion that a 'computer' (entity) is 'great' (sentiment), and the other opinion is

that the 'hard drive' (entity) is 'disgusting' (sentiment).

By using this simple example we will present our research goals in the following paragraphs. The particular fields we will focus on in this work are presented in Figure 1.2. Each block of the figure is presented in a certain section in this introduction. The short-

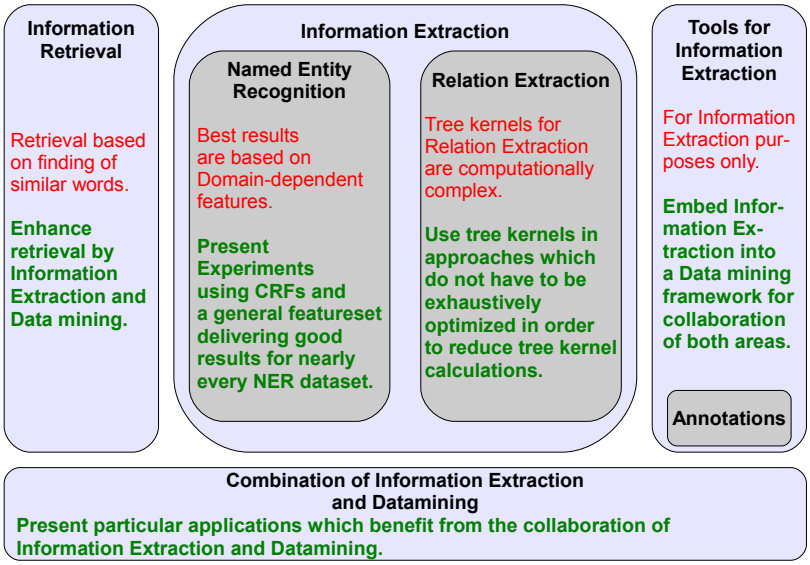


Figure 1.2: The particular fields we will focus on in this work (red: current shortcomings; green: our work).

comings of current approaches are written in red letters and our provided solutions are written in green letters.

1.1 Information Retrieval

Common search engines such as *Google* are efficiently making the data of the WWW accessible by compressing and indexing the original data. They deliver text documents, videos, pictures and other document types which are related to a previously given user-query.

Search engines have to crawl the WWW. After that, they have to efficiently index the information which is available in the WWW. Finally, they have to make the indexed information accessible in a very suitable manner. The field of scientific research addressing these needs of search engines is known as *Information Retrieval* [Van-Rijsbergen, 1979].

Information Retrieval has one shortcoming: it mostly delivers just an index-look-up for the user query. A common search engine requested by a user, given a certain query, will deliver documents, websites, videos and so on, containing the requested phrase. A user just gets a link to the files containing the whole request or parts of it. In most cases the user is not guided to the position in the delivered document or file the requested information is located at, and, no more additional pieces of information related to the request are given. No semantical knowledge is offered because it is only checked if the requested words occur in particular documents.

[Lancaster, 1968] defines *Information Retrieval* as follows: *An information retrieval system does not inform (i.e. change the knowledge of) the user on the subject of his inquiry. It merely informs on the existence (or non-existence) and whereabouts of documents relating to his request.*

If a user requests a common search engine using the terms *'great'* and *'hard drive'* a possible response will be a document containing the sentence presented in Figure 1.3. Although a document is presented which contains the requested search terms, the received result is the complete opposite of the result the user probably assumed to get. The user was probably looking for a *great hard drive* but the search result is a document mentioning a *great computer* and a *disgusting hard drive* in the same context.

The computer is great but the hard drive is disgusting.

Figure 1.3: A search result for the request *'great'* + *'hard drive'*.

The example above has shown that the task of sentiment analysis cannot be solved by traditional *Information Retrieval* because of the fact that semantical knowledge is neither used in the analysis of the request nor in the analysis of the documents which are delivered to the user. We will improve the task of *Information Retrieval* by the combination of *Information Retrieval* on the one hand and *Data Mining* and *Information Extraction* on the other hand. Our application representing the improvement of *Information Retrieval* are presented in Section 8.1.

1.2 Information Extraction

If a user does not only want to find trivial information like who was an actor in the movie called 'Star Wars', the user might be especially interested in deeper insights concerning the given request. If a user is searching for informational units concerning a famous person, for example, he will not want to read complete articles about the person. The user would probably like to get a condensed biography of the person. A user does not want to search the requested information in many delivered documents. A user wants to be directly guided to the interesting and relevant positions inside of the delivered documents. Additionally, a user wants the request to be interpreted. The

perfect result is an exact and condensed piece of information which answers the question the user asked by using a particular request. To create such condensed information the delivered files have to be shortened not to contain needless content. Although, the shortened version still must contain all relevant information. The interesting and relevant pieces of information have to be extracted. Delivering just a bunch of extracted relevant pieces of information would help a user, but if only the pieces of information are delivered without contextual information the user is more confused than aided. It has to be tested if some of the extracted pieces of information are related. Extracting such relations together with the informational units itself delivers further semantical knowledge which allows the user to gain knowledge without the necessity to study contextual information in detail.

Interesting and classifiable tokens in text documents are called named entities. The task of classifying tokens in text documents is called *Named Entity Recognition* (*NER*). The extraction of relations between formerly extracted named entities is called *Relation Extraction* (*RE*) [Grishman, 2003].

The extraction of such interesting or relevant information which is not achievable by standard *IR*-techniques is hard and can be very time-consuming if it is done by hand. *NER* and *RE* go far beyond *Information Retrieval* (*IR*). This field of research is called *Information Extraction* (*IE*).
By saying that *"Information Extraction is the automatic identification of selected types of entities, relations, or events in free text."* [Grishman, 2003] defines *Information Extraction* as the extraction of informational units of previously defined types. [Cardie, 1997] defines *Information Extraction* in a different way as she says *"An Information Extraction system takes as input a text and "summarizes" the text with respect to a previously specified topic or domain of interest: ..."*.

1.2.1 Named Entity Recognition

Named Entity Recognition (see Section 3.2) is the task to extract interesting and classified information out of documents. The fact that the extracted information is classified makes the task of *Named Entity Recognition* to a task which explicitly delivers semantical pieces of information concerning the document. In the task of sentiment analysis *Named Entity Recognition* is a preprocessing step for the extraction of the certain opinions and the entities which are connected to these opinions. Unfortunately, traditional *Named Entity Recognition* only focuses on the extraction on very trivial entities like person names, organization names, location names, and so on. The possibility to extract more complicated entities becomes more and more important in nowadays fields of research. In the biological domain, for instance, it is crucial that a particular system can extract gene names, names of amino acids, and so forth. For the task of sentiment analysis particular opinions and target entities like products, parties, and many more must be extracted.

Methods for *Named Entity Recognition* are very helpful for the application of sentiment analysis tasks. Figure 1.4 shows a sentence in which two types of entities have

been extracted. In the particular case one type of the entities is an opinion (red) and the other type is a computer-related product (blue). For this specific domain the entity-types to be extracted by the *Named Entity Recognition*-system are not trivial because they are from a special domain.

The computer is great but the hard drive is disgusting.

Figure 1.4: A sentence containing two types of annotated entities.

For such domains which are more complicated in case of *Named Entity Recognition* it is necessary that the used system relies on domain-dependent lexicons or gazetteers to achieve very good results. These domain-dependent resources are not easily to deliver for certain new tasks. We will focus on different *Named Entity Recognition* tasks, and we will only use domain-independent and orthographic features. We will show that current machine learning approaches only working on domain-independent features are delivering results which are nearly as good as the results achieved by approaches relying on domain-dependent features in Section 3.3. This allows the usage of one certain feature set directly for the application of different *Named Entity Recognition* tasks.

1.2.2 Relation Extraction

Relation Extraction (see Chapter 5) is the task which follows on *Named Entity Recognition*. After having extracted all entities of a particular sentence the possible relations between those entities are analyzed. For the task of sentiment analysis *Relation Extraction* can be used to determine whether a relation between two entities is present and which type of relation the two entities are part of. The example shown in Figure 1.5 presents two relations. The first is between the entities 'like' and 'computer' and the second one is between 'hard drive' and 'disgusting'. The types of the two relations is different because the first one is describing a positive sentiment and the second relation is describing a negative one.

Figure 1.5: A sentence containing dependency relations

The extraction of relations is not obvious because sometimes the relation is indicated by trigger-words like *great* or *disgusting* as shown in Figure 1.5. Sometimes the relation is just indicated by the construction of the sentence, and sometimes both, the

information about the appearing words as far as the structures are indicating particular relations. It is becoming apparent that the extraction of relations in contrast to the extraction of entities – which sometimes is just a look-up in a lexicon – again, is more complex and requires special techniques.

It is shown in Chapter 5 that for the analysis of relations the state-of-the-art techniques are based on structured features (see Section 2.1.4). Unfortunately, the analysis of structured features is computationally hard – especially for huge datasets. For sufficiently good results of sentiment analysis a huge amount of data has to be analyzed. We will show that approaches not relying on complex optimization can be used for the analysis of structured features. In addition, we show that those approaches on the one hand are delivering results which are nearly as good as results achieved by optimized methods, and on the other hand they are significantly faster compared to those methods.

1.3 The Tool for Information Extraction

For the extraction of interesting parts of information in text documents the documents are split into smaller parts of text [Grishman, 1997]. The sequential order of these parts is being conserved. This is important because on the one hand the later extraction of relations between the extracted interesting parts of the document needs the structural information, and on the other hand particular methods are directly processing a whole batch of such parts and the structural information defines these batches. The splitting is called tokenizing and the resulting parts are called tokens. A token is always just a fragment of the original document and it is not restricted to single words.

The single tokens have to be classified in the task of *Named Entity Recognition*, for instance. In the context of machine learning the particular tokens can be handled like data points in every usual *Data Mining* task. By following this assumption we are able to use *Data Mining* tools for the application of documents for *Information Extraction* purposes. Unfortunately, usual *Data Mining* approaches treat the data points in an independent manner. This means that dependencies between neighboring tokens, for instance, are usually not taken into account for the inference mechanisms.

It has been shown that traditional machine learning approaches are to be optimized for *Information Extraction* tasks. In NER, for instance, current approaches like conditional random fields (CRF) are taking the structure of the sentence, the word occurs in, into account [Lafferty et al., 2001, Sutton and McCallum, 2007, Tsochantaridis et al., 2005]. These approaches are achieving more accurate prediction performance than traditional approaches in this field. *Relation Extraction* in contrast profits from structured attributes which can be used by traditional approaches [Moschitti et al., 2008].

A tool is needed which allows to analyze documents in order to perform *Information Extraction*. This tool should enable the transformation of documents into a format which will be readable and processable by *Information Extraction* methods. In

addition, *Information Extraction* methods should be available for the analysis of the documents. To create sufficiently good analysis the document has to be enriched by additional features which aid the analyzing process. The tool should therefore provide preprocessing operators which enrich the documents. The features which are used for enriching the documents are manifold. The features can be numerical or nominal but they also can be of a structure. The tool should maintain those features for a sufficient feature set. The tool should be able to statistically evaluate and validate analysis in order to compare different learning methods for the same dataset. This comparison allows the selection of the best performing technique for particular datasets.

Usual frameworks for *Information Extraction* are constructed for the simple application of *Information Extraction* approaches. Unfortunately, these approaches are not based on the principles of *Data Mining* and machine learning. Frameworks for *Data Mining* are, for instance, offering methods for the evaluation and validation of significant results on data mining tasks. We will present an extension to the open-source *Data Mining* framework *RapidMiner* which combines on the one hand *Information Extraction* techniques and on the other hand all the particular infrastructure which is important for *Data Mining*. Our extension allows the comparison of traditional *Data Mining* and certain *Information Extraction* approaches in a statistically significant way. Additionally, our framework provides arbitrary features for the analysis process which, amongst others, allows the usage of tree-structured features. Finally, the embedding of *Information Extraction* into a *Data Mining* framework allows the easy analysis of extracted information by *Data Mining* methods to gain further insights which is not provided by any other *Information Extraction* framework.

1.3.1 Annotations

The creation of classified data is crucial for machine learning techniques. Due to the fact that every supervised machine learning method needs classified data those data has to be delivered. If the data is not yet classified, the data must be manually classified. In the context of *Information Extraction* classified datasets are called annotated. A sufficiently annotated dataset is needed for the application of machine learning methods on tokens which represent documents. To create such dataset it is needed that a sufficient amount of documents is manually annotated. The persons annotating texts are called annotators. The documents an annotator is processing should belong to one particular domain not to stress the annotator too much. If an annotator is familiar with such domain, he will have no problem to annotate the texts. Unfortunately, this work is very time-consuming especially if a huge amount of documents has to be processed. Additionally, if the annotator is not familiar with the domain he possibly will create mistakes during the annotation process.

Machine learning can help in this context to shorten the assignment of human annotators. Just the training set has to be annotated by hand. The amount of tokens which has to be manually tagged can be reduced by active learning [Tomanek, 2010]. The rest of the data can be annotated automatically by using a machine learning method

which was trained during a former training phase. The annotation process in the context of machine learning is called tagging because an additional information is tagged to the particular token in the text. As it is shown in Figure 1.5 some entity-types can be tagged only by using a lexicon access. More complicated or ambiguous words require more complex methods.

We will provide a graphical interface which allows users to easily annotate tokens. These tokens are later on used train *Information Extraction* methods for an automatic annotation process.

1.4 Combination of Information Extraction and Data Mining

The close collaboration of *Information Extraction* and *Data Mining* methods is a major achievment of our extension. Other approaches do not offer such close collaboration. *Information Extraction* tasks do not only benefit from *Data Mining* validation or evaluation methods. This collaboration furthermore allows the analysis of pieces of information which have been extracted by *Information Extraction* tasks. We will present particular real-world tasks which give reasons for the combination of *Information Extraction* and *Data Mining* methods. One of these tasks is the analysis of extracted relations.

A recent extension for the extraction of relations is the representation of these relations in a graph. This representation first of all offers an intuitive view on the extracted information which aids users to gain further insights. Additionally, the graph itself can be used to extract more information concerning the related entities. Although such graph can store relations which have been extracted by an *Information Extraction* system, the graph also can be a set of entities which are related because of some reasons.

In the WWW, for instance, such relation graphs already are presented in various ways. Well-known and popular examples of such relation graphs are social networks like *facebook* and *twitter*. These networks are offering a huge amount of entities on the one hand (for instance *users*, *messages*, *photos* and so on) and relations between these entities on the other hand (for instance *message from user A to user B*, *user A likes photo C* and so on). These relations and entities do not have to be extracted in a complicated way – they just have to be crawled. Figure 1.6 shows a message (called *Tweet*) from a user of the blogging platform *twitter*.

This *Tweet* contains a *tag* – indicated by the preceding # – and a *url*. By using these pieces of information we can construct a quaternary relation connecting the *author*, *tag*, *url* and the *tweet* itself. Extracting all or just some of the relations apparent in such networks allows us to create a relation graph. An analysis of entities in the graph without having information about the entity itself is possible by just focusing on the relations in the graph.

Figure 1.6: A message from the blogging-platform *twitter*. The message can be interpreted as a quaternary relation between *message*, *user*, *tag* and *url*.

Extracted entities and relations can be combined to form some kind of event like, for instance, the acceptance of a request which has been discussed in the parliament (see Section 8.1). If these *Information Extraction* task is finished the extracted events, again, can be used to extract additional knowledge.

The explicit collaboration of *Data Mining* and *Information Extraction* techniques is not possible in any of the popular frameworks for *Information Extraction*. We closed this gap by the development of an *Information Extraction* extension for the *Data Mining* framework *RapidMiner*. We will present particular real-world applications which exactly profit by such collaboration in Chapter 8.

1.5 Outline

The work is organized as follows.
Chapter 2 is about the definition of all needed formalisms of machine learning which are used in this work. Additionally, multiple approaches of machine learning applicable to the fields of *Information Extraction* are presented.

In Chapter 3 we describe the historical development from first approaches of *Information Extraction* over *Named Entity Recognition* to the point of *Relation Extraction*.

Chapter 4 gives an overview about the possibilities to use linguistic resources for the creation of feature sets for *Information Extraction* purposes.

In Chapter 5 we show how *Relation Extraction* is formally defined. We additionally show what kind of methods are used for *Relation Extraction* in machine learning.

In Chapter 6 we are focusing on *Relation Extraction* techniques which benefit on the one hand from minimum optimization and on the other hand from efficient data structure.

Most of our experiments and implementations were done using the open source framework for *Data Mining RapidMiner*. To apply this framework on IE tasks we developed an extension called *Information Extraction Plugin*. We describe the extension in Chapter 7. To give an overview of software to be used for IE which is slightly comparable

to our extension we present two software packages in Section 7.3.

Chapter 8 presents applications which explicitly benefit from the collaboration of *Data Mining* and *Information Extraction*. In Section 8.1 we present the usage of *Data Mining* techniques on pieces of information which have been formerly extracted by *Information Extraction* methods, and in Section 8.3 we present the analysis of graphs which contain extracted relations by only using the relational data instead of using information contained in the related entities itself.

We conclude our work in Chapter 9.

Chapter 2

Machine Learning for Information Extraction

In this chapter we will present the basic techniques and methods for the later chapters. The methods we are presenting here are used in most of the following chapters. Machine learning [Mitchell, 1997] describes the ability of computer programs to *learn* from data. *Learning* means that the certain computer programs are extracted or created out of some set of data. The resulting program should be general enough to make decisions on data not belonging to the set the model was created on. The data to be analyzed in this context is manifold. Typical application areas are *fraud detection*, *spam detection*, *marketing analysis* and *image analysis*. These areas of course are just a small excerpt of the wide range of applicable tasks. We will call the program a *model* in the following as it is modeling the data.

Definition 1. *A **model** is a program that is created from a set of data. It is used to make decisions based on the set used for the creation of the **model**.*

Definition 2. *The set of data used for creating a **model** is called **training set** because the **model** is trained according to this set.*

Following the definition of [Hastie et al., 2003] the field of machine (or statistical) learning is split into 2 subtasks: *supervised* and *unsupervised* learning. For both learning schemes the data is represented by a set of *features*. One data point and its particular assignment of the *features* is called an *example*. The data which contains many *examples* is called *example set*.

Definition 3. *A data point is called **example**. Lowercase and bold letters like* $\mathbf{x}$*, for instance, indicate **examples**.*

Definition 4. *An **example** contains attributes which are called **features**. **Features** are indicated by indexed lowercase (not bold) letters* x_i*.*

The characteristics of a feature are remarkable. We distinguish between *flat* and *structured features* in this work.

Definition 5. *A **feature** will be called **flat** if its value is of nominal or numerical type.*

Definition 6. *A **feature** will be called **structured** if it is not **flat** and if its value is a structured type like a tree, for instance.*

Definition 7. *A set of **examples** $\mathbf{x}$ is called **example set** $\mathcal{X}$.*

For *supervised* learning the learned program is used to predict a certain output variable (called *label*) given the feature assignments. The *labels* usually are *categorical* or *quantitative*. The program is created during a training phase. During the training phase an example set is used to tune the program. The examples of this set must contain a *label*. After creating the program it can be used to predict the *label* of examples using only the feature assignments. The latter phase is called testing phase.

Definition 8. ***Examples** can contain a special output variable indicating the class or category of the **example**. This output variable is called **label** and it is indicated by y.*

Definition 9. *The set of data a **model** is applied to is called **test set** because the already trained **model** is making predictions on this set without knowing the correct **labels** of the set.*

For *unsupervised* learning the resulting program is created by only using the feature assignments. Approaches of *unsupervised* learning, for instance, are used to categorize the example set into different groups whereas similar examples are grouped together.

2.1 Feature types

In this Section we want to define the different types of features which can be used for machine learning tasks. The following listing is not complete, certainly, but we try to show the differences and similarities between the various types of features. Especially the tree-structured features which are presented in Section 2.1.4 are often used in this work and therefore they are very important. Numerical, nominal (and date) features are *flat* features (see Definition 5) and tree-structured features are *structured* features (see Definition 6).

2.1.1 Numerical Features

Numerical features $x_i \in \mathbb{R}$ contain real-valued numbers which are advantageous for the case of machine learning. An example x containing numerical features, only, can mathematically be processed – using the dot-product, for instance. In addition, numerical features are naturally ordered which is sufficient for the calculation of most distance measures. A special case are numerical features representing categorical values $c \in \mathbb{N}$ which are not necessarily ordered. These features cannot directly be used for many vector calculations.

2.1.2 Nominal Features

Nominal features $x_i \in \mathcal{C}^n$ contain nominal values like Strings. $\mathcal{C}^n$ represents a sequence of characters. These features do not contain numbers which makes them not to be processed by vector calculus functions directly. The crucial task for nominal features is to convert them into a numerical representation to make them applicable by machine learning methods. The trivial approach to convert nominal features into a numerical representation is to exchange every individual value of the nominal feature by an individual number $m \in \mathbb{N}$. Unfortunately, this will end up in a categorical numerical feature having the disadvantage not to be ordered in a useful way (see Section 2.1.1). Another approach is to convert every nominal feature into multiple binary numerical features. We assume that the nominal feature x_i has k different values on the data set. k binary numerical features will be created, each for one of the nominal values. After that the values of the k features will be set to 1 or 0. A feature will be set to 1 if the feature x_i contained the particular nominal value. Otherwise, the feature will be set to 0. $k-1$ of the resulting k features will contain a 0 and one feature will contain a 1. Although the number of features will increase if this approach is used, the resulting representation of the examples can be used by vector calculation methods directly.

2.1.3 Date Type Features

A date is a special kind of nominal feature because on the one hand it is ordered and on the other hand this ordering cannot be used easily in the case of vector calculation methods. The date values have to be converted into real-valued numbers before a calculation using these values is possible. The date feature type can be neglected for this work, but it is mentioned here because numerical, nominal and date features are the three possible feature types in *RapidMiner* which is the software we are using for most of our experiments (see Chapter 7).

2.1.4 Tree-structured Features

A structured feature is more complex in contrast to the three formerly presented feature types. In this work we will only use structured features containing a tree structure. The complexity of the structured features becomes apparent in different facts. Like nominal features the structured features do not contain any trivial order. Additionally, tree-structured features mostly are representing great formations like sentences, websites and so on (see Section 2.1.4). This fact makes it unlikely for two equal trees to occur together in a dataset, and therefore it is not useful to convert a tree-structured feature into binary numerical features. If, for instance, a tree-structured feature x_i contains a unique tree for each example it is not helpful in its original form because it does not give any impact for the task of machine learning. Another important point according tree structures is their recursive nature. The root node of a tree (mostly) has children node which, again, are the root nodes of trees.

Definition 10. *A tree is a graph $T = (V, E)$ containing set of nodes V and a set of edges E. One special node $e_r \in E$ does not have parent-nodes. This node is called root-node. Every other node $e \in E | e \neq e_r$ has one parent-node. If a node has*

no children nodes it will be called leaf-node or terminal (node). A node which only contains terminal nodes as children nodes is called preterminal (node). Every node $e \in E$ is the root-node of a subtree T' of T, where $T' \subseteq T$.

Definition 11. *The production of a particular node $e \in E$ is a special representation of the node consisting of the node e itself and the corresponding children nodes of e.*

Tree structures

We will use several tree-structured features in this work. We are presenting three tree structures we are using in this work to give the reader an idea of the manifold occurrences of trees in machine learning datasets. The trees we are presenting are parse trees which are representing the syntactical analysis of a given token-sequence according to a grammar. Non-leaf-nodes are tagged as non-terminal symbols of the grammar, whereas leafs are tagged as terminal symbols. The methods used for creating parse trees are called parsers.

Constituent parse trees A constituent parse tree is a tree structure which splits sentences into phrases and word types according to a grammar of natural languages. To give the reader an example we parsed the sentence "Felix went to New York to visit the statue of liberty.". The resulting parse tree is shown in Figure 2.1. The sentence is split into a noun phrase (NP) "Felix" and into a verb phrase (VP) "went to New York...". According to grammatical rules for certain languages the phrases, which are represented as subtrees in the parse tree, are split recursively. The words of the sentence are present in the leafs of the constituent parse tree. The constituent parse tree is very helpful in the sense that groups of words, which syntactically belong together, are represented in parse trees.

Dependency parse trees The dependency parse tree contains the words of a sentence as nodes and labeled edges between the nodes are representing the dependencies which are present between the words. The same sentence we already used for visualizing the constituent parse tree is used to show an exemplary dependency parse tree in Figure 2.2. It becomes visible that the sentence is split into three major parts which depend from the word "went". A dependency parse tree can even better be used to extract semantical knowledge from the sentence. In Figure 2.2 the dependency parse tree easily can be used to answer the questions "Who 'went'?", "Where did someone 'go'?" and "Why?". The answers are given by the three subtrees "Felix 'went'!", "S.o. 'went' to New York!" and "To visit the statue of liberty!". In addition, the dependency parse tree is very helpful for specific languages containing long-distance relationships like for the German language [Kübler and Prokic, 2006].

Machine-readable language trees Machine-readable languages like *XML* and *HTML* can easily be converted into a tree representation. If the particular documents are well formed the tags between an opening and a closing tag will form a new subtree. Figure 2.3 shows the tree representation of an *HTML* document.
Other machine-readable languages like *SQL*, for instance, are parsed in a more compli-

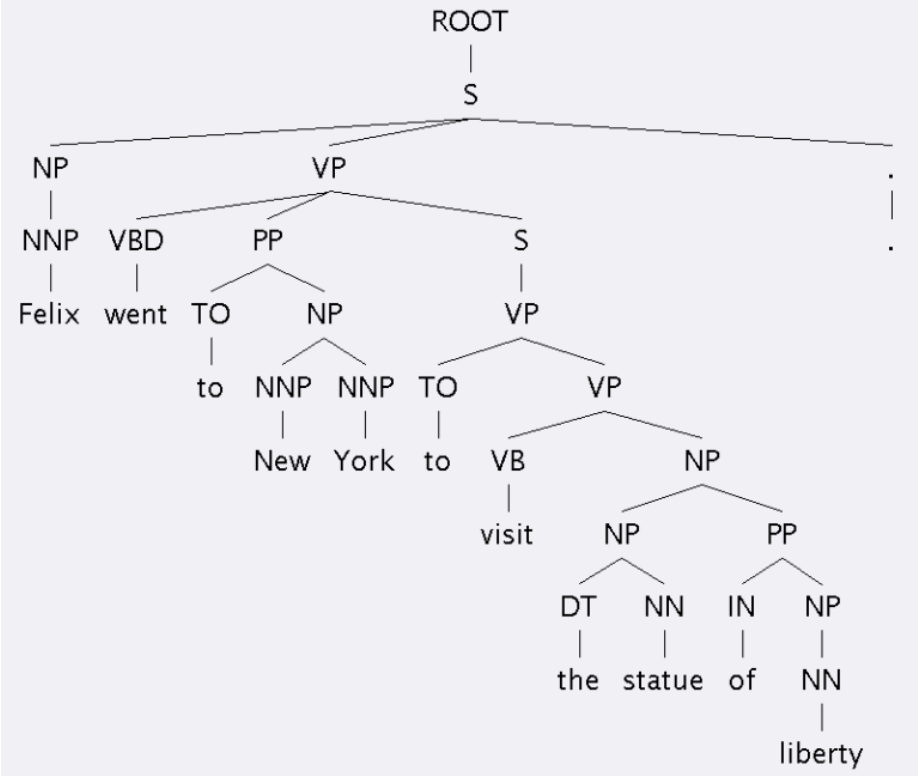


Figure 2.1: The constituent parse tree of the sentence ”Felix went to New York to visit the statue of liberty.”

cated way. Although parsing *SQL* statements is more complicated than parsing *XML*, it is much easier than parsing natural languages because the vocabulary of *SQL* is small in contrast to the vocabulary of a natural language.

String Representation of Trees Trees can always be represented as strings. Following the bracketing approach of the Penn treebank [Marcus et al., 1993] (see Section 4.1) each subtree of a tree is encapsulated in brackets – “(” for the beginning of the subtree and “)” for the end. This is done for every subtree available in the original tree.

The string representation of the tree shown in Figure 2.1 is:
(ROOT (S (NP (NNP Felix)) (VP (VBD went) (PP (TO to) (NP (NNP New) (NNP York))) (S (VP (TO to) (VP (VB visit) (NP (NP (DT the) (NN statue)) (PP (IN of) (NP (NN liberty)))))))) (. .)))

Figure 2.2: The dependency parse tree of the sentence ”Felix went to New York to visit the statue of liberty.”

2.2 Data preparation

The original data which is converted into an example set can contain different pieces of information which are impractical for the modeling process. The task of data preparation [Pyle, 1999] focuses on all the efforts needed to convert the original data into examples to be sufficiently handled by a modeling process. Data preparation does not only reconvert the data which could be of different formats as it is extracted from sensor networks or databases, for instance. Many reasons are crucial for data preparation. We present the three major reasons:

- The feature set contains defective features like noise, outliers or null-values.
- The feature set is too huge and its processing is computationally too expensive.
- The feature set contains features which are not applicable by certain models.

The feature set has to be analyzed in order to find defective features. Defective features can extremely affect the performance of certain models. If defective features are found they should be removed or replaced. The computational complexity of particular *models* relies on the dimension of the feature set. If the dimension of the feature set is

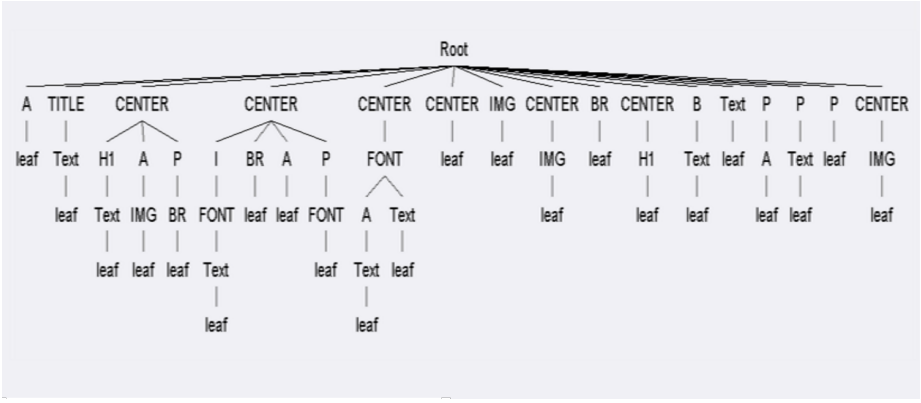


Figure 2.3: The tree representation of an *HTML* document

too great the calculation of the model becomes computationally complex. It is useful to reduce the number of features without losing (too much) information. In addition to removing all correlated features a feature selection can be applied to select a subset of features which delivers nearly as good results as the complete feature set. Some *models* can only handle specific types of features. *Support Vector Machines* (SVM), for instance, are only able to handle numerical features. If follows that the feature set has to be prepared for certain models in order to contain the correct types of features. Nominal features can be converted into numerical features as shown in Tables 2.1 and 2.2.

...	nominal attribute	...
...	red	...
...	blue	...
...	blue	...

Table 2.1: An example set containing a nominal attribute

...	numerical attribute red	numerical attribute blue	...
...	1	0	...
...	0	1	...
...	0	1	...

Table 2.2: The example set shown in Table 2.1 converted into a numerical representation

Although, the task of data preparation seems to be trivial 60% of the time needed for the complete *Data Mining* task is used for data preparation [Pyle, 1999]. We only present an overview of the possibilities of data preparation because it would go beyond the scope of this work to present this huge area completely.

2.3 Particular Specialties for *Information Extraction*

In contrast to traditional *Data Mining* performed on data which is available in databases or data warehouses, *Information Extraction* is facing an amount of unstructured information which first of all has to be converted into a example set comparable to example sets used in traditional *Data Mining* approaches. This means that, for instance, one document has to be split into a set of examples. In addition, these examples have to be enriched by a set of features. This extraction does not have to be performed in traditional *Data Mining* tasks because the example set and the according features are inherently given by the relations and the attributes in the database.

Definition 12. *The splitting of unstructured information in* Information Extraction *is called* ***tokenizing****. The resulting parts are called* ***tokens****.*

Although these tokens can be seen as an example set like it is used in traditional *Data Mining*, they contain inherent structures which contain additional information. Two relations of a database normally do not relate to each other, unless they share several attribute values. Two tokens extracted from one document or from one sentence may relate to each other in a stronger way. This behavior is described in the following section.

2.3.1 Interdependent Features and Examples

It is very important for *Information Extraction* to respect the original structure of the tokens. The preceding and following tokens for a certain token are indicating particular knowledge for that token. On the one hand the information contained in the surrounding tokens can be used to enrich the feature set for the current token. On the other hand the structure of the tokens itself can be used by the inference mechanisms like it is done by structured models (as presented in Section 2.4.2). It is indispensable that the structure of the tokens is preserved during the complete *Information Extraction* process. If this is respected the set of tokens can be used like an example set by traditional *Data Mining* approaches – for the comparison of *Information Extraction* with *Data Mining* methods, for instance.

Tokens are not restricted to single words. In contrast, an interesting information to be extracted is not restricted to a single token. If, for instance, each word of a document is used as a token the interesting information sometimes will cover more than one token. A person's name consisting of name and surname mostly is covering two or three tokens. It follows that the evaluation of a machine learning approach for *Information Extraction* is not as trivial as for traditional approaches (see Section 2.4.4). A dataset containing many informational units covering multiple tokens sometimes is tagged using the so called *IOB*-tagging [Ramshaw and Marcus, 1995]. The beginning token of an informational unit is prefixed by a "B-" (which means that a new informational unit *begins*). The following tokens of the the same informational unit are prefixed by "I-" (which means that these tokens are *inside* of an informational unit). Every token which does not belong to an informational unit is tagged by an "O" (meaning that this is a token *outside* of the informational units). This approach ensures that two neighboring

tokens can be tagged as two or one informational units. If for instance the two tokens should be tagged as two person names, the tags should be *B-PER B-PER*. If the tokens should be tagged as one person name the tags should be *B-PER I-PER*.

2.4 Models

In this Section we will show what kind of models can be used for *Information Extraction*. Rule-based models are historical methods which are too inflexible to be chosen for domain-independent *Information Extraction*. A short paragraph about rule-based models is written in Section 3.2.2. Especially statistical and structured models have to be preferred. Although structured models deliver the better results in cases of precision, recall and accuracy, statistical models should be chosen to achieve shorter runtime.

2.4.1 Statistical Models

Statistical models are based on the statistics of the training set. Some of these models are exhaustively optimized in a training phase whereas lazy models are memorizing the training data and the calculation which is relevant for the decision exclusively happens during the prediction phase. We split the statistical models into the group of Lazy Models, Optimized Models and models which cannot be put into one of both groups.

Lazy Models

[Aha, 1997] defines lazy learning methods by three characteristics:

- The inputs are memorized for future use. They will only be processed if a request for information is received.
- Responses for requests are given by combining the stored data (inputs).
- The response and any calculated results are discarded.

[Aha, 1997] in contrast mentions eager learning algorithms which "greedily compile their inputs into an intensional concept description". Both approaches have advantages and disadvantages. Although lazy models have less computational costs during training, they often require more storage and do have more computational costs during prediction than eager models. In this Section we present k-Nearest Neighbors which are a typical lazy learning approach.

k-Nearest Neighbors The k-Nearest Neighbors approach (kNN) [Hastie et al., 2003] stores all the examples from the training set for future use. If an example $\mathbf{x}$ should be predicted by kNN k examples from the training set which are most similar to $\mathbf{x}$ will be used for the prediction. Therefore, the k most similar examples have to be found. After that the class which is most frequent for these examples is chosen for the prediction of $\mathbf{x}$. An odd number which does not equal the number of classes mostly is chosen for k to assure that one class gets the majority of the k examples. The kNN approach is computationally complex during the prediction phase as the new example $\mathbf{x}$ has to be

compared with every example $\mathbf{x}'$ from the training set. The distance measure which is used to select the k most similar examples compared to $\mathbf{x}$ often is the Euclidean distance which uses every feature of the examples for the calculation of the distance measure. If the training set contains n examples having m features the complexity of each prediction is $\mathcal{O}(mn)$. In addition to the complexity of the prediction phase the storage complexity which is needed to keep the complete training set is significant.

Optimized Models

Although [Aha, 1997] already mentioned eager models in contrast to lazy models we would like to focus especially on the types of models which build an optimized decision model. In this context we assume that an optimized decision model only contains a subset of the training set which in addition is weighed according to emphasize the important parameters. In this Section we present Support Vector Machines which are a heavily optimized approach.

Support Vector Machine A Support Vector Machine (SVM) [Vapnik, 1995] is a machine learning approach that creates a separating hyperplane in the feature space of the examples to split the set of the positive examples from the set of negative examples. It becomes obvious that an SVM is a model which only can handle binary datasets. Using a certain strategy (see Section 2.4.3) allows to apply SVMs to multi-class datasets. The example set shown in Table 2.3 contains two features (weight and height) and a label (gender) containing two different values (m and w). The examples of that example set are located in the feature space $\mathbb{R}^2$. Figure 2.4 visualizes the example set in $\mathbb{R}^2$.

A linear model [Hastie et al., 2003] is calculating a prediction $\hat{y}$ given an example

ID	WEIGHT	HEIGHT	GENDER
1	90	190	M
2	60	170	W

Table 2.3: Example set containing two examples and two features

$\mathbf{x}$ by equation 2.1.

$$\hat{y} = \hat{\beta}_0 + \sum_{j=1}^{p} x_j \hat{\beta}_j \tag{2.1}$$

$$\hat{y} = \hat{\beta}_0 + \mathbf{x}^T \hat{\beta} \tag{2.2}$$

The optimal $\hat{\beta}$ is the one resulting in the minimum $RSS(\hat{\beta})$ (see equation (2.3)).

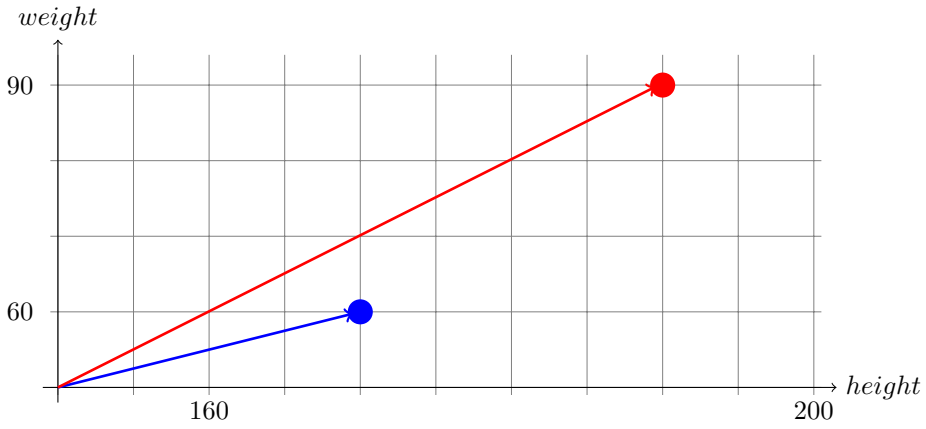


Figure 2.4: The example set shown in Table 2.3 visualized in $\mathbb{R}^2$

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - \mathbf{x}_i^T \beta)^2 \tag{2.3}$$

A linear model can be seen as an optimal separating hyperplane in the feature space. Figure 2.5 contains such hyperplane. An intuitively optimal hyperplane separates the positive from the negative examples and has the greatest distance to the nearest positive and negative examples.

Let the positive examples get the label-value 1 and let the negative examples get the label-value -1. Following this assumption equation 2.1 can be rewritten like in equation (2.4) and (2.5) for the two label-values.

$$1 \quad \leq \hat{\beta}_0 + \mathbf{x}_i^T \hat{\beta}, \text{ for } y_i = 1 \tag{2.4}$$

$$-1 \quad \geq \hat{\beta}_0 + \mathbf{x}_i^T \hat{\beta}, \text{ for } y_i = -1 \tag{2.5}$$

Three parallel hyperplanes are available, now. Two of them are crossing the positive and negative examples which are nearest to each other and the third one is between the other two ones having the same distance to both of them. The hyperplanes are shown in Figure 2.6.

The distance between the two hyperplanes located at the outer sides is $\frac{2}{\|\beta\|}$ and it is called margin. For the achievement of the optimal separating hyperplane the margin

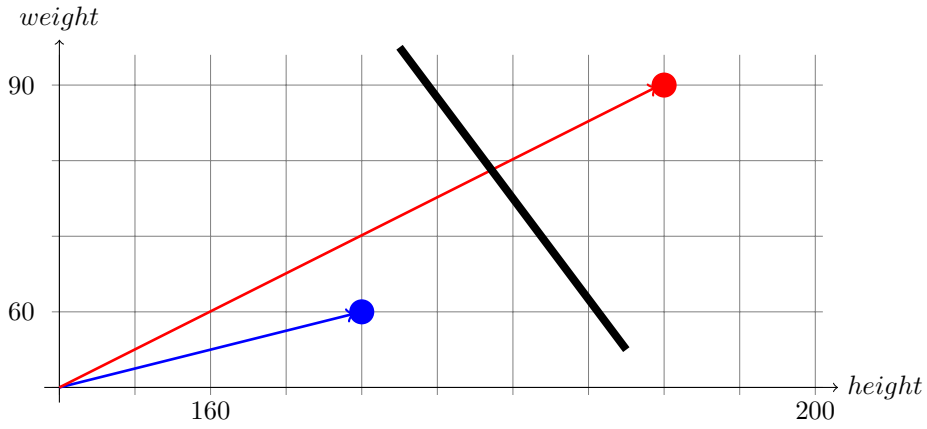


Figure 2.5: An intuitively optimal hyperplane separating the example set shown in Table 2.3

should be maximized. To maximize the margin $\frac{2}{\|\beta\|}$ it is possible to minimize $\| \beta \|$. For mathematical reasons $\frac{1}{2} \| \beta^2 \|$ will be minimized. Equations (2.4) and (2.5) generally can be written like shown in equation (2.6).

$$1 \quad \leq y_i \left(\hat{\beta}_0 + \mathbf{x}_i^T \hat{\beta} \right) \tag{2.6}$$

Given equation (2.6) as a side condition for the optimization problem $\min \frac{1}{2} \| \beta^2 \|$ makes the optimization to be solved by Lagrange multipliers [Hazewinkel, 2002]. The Lagrange optimization problem (in its primal form) is presented in equation (2.7)

$$L_P = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{N} \alpha_i [y_i(\mathbf{x}_i^T \beta + \beta_0) - 1] \tag{2.7}$$

$\alpha_i \geq 0$ are the Lagrange multipliers. To find the optimum the primal optimization problem is differentiated with respect to β_0 and β.

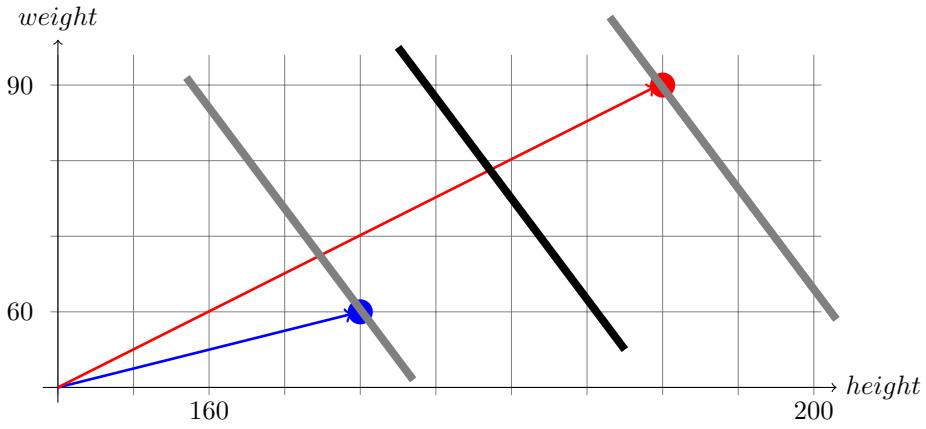


Figure 2.6: The separating hyperplane and the two parallel hyperplanes crossing the negative and positive examples of the set shown in Table 2.3

$$\frac{\partial L_P}{\partial \beta} = 0 \tag{2.8}$$

$$\Longrightarrow 0 = \beta - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{2.9}$$

$$\Longrightarrow \beta = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{2.10}$$

$$\frac{\partial L_P}{\partial \beta_0} = 0 \tag{2.11}$$

$$\Longrightarrow 0 = \sum_{i=1}^{N} \alpha_i y_i \tag{2.12}$$

If we insert equations (2.10) and (2.12) into equation (2.7) we will receive the dual optimization problem L_D which is shown in equation (2.13).

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \mathbf{x}_i^T \mathbf{x}_{i'} \tag{2.13}$$

In the prediction phase the optimized values for β and β_0 are used to predict new examples like it is shown in equation (2.2). Equations (2.10) and (2.12) show that only

examples $\mathbf{x}_i$ getting an $\alpha_i > 0$ during optimization are used for calculating β and β_0. These examples are called support vectors because only these examples are taken into account for the model which is used for later prediction. The remaining examples can be neglected.

It could happen that not all examples are to be classified correctly. Some might be located on the wrong side of the separating hyperplane and some might lie on the correct side but inside of the margin. Figure 2.7 shows three examples which are not located correctly. Those examples should be punished during optimization to achieve a most optimal separating hyperplane.

Every example $\mathbf{x}_j$ that is not positioned correctly according to the hyperplane will

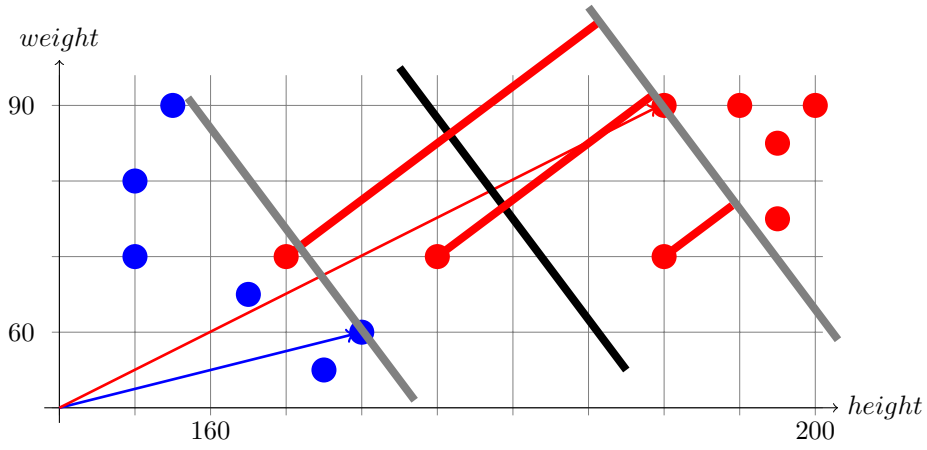


Figure 2.7: A *sub-optimal* hyperplane and several examples violating the definition of an optimal hyperplane

get an error-term $\epsilon_j > 0$. Examples $\mathbf{x}_j$ which are located correctly will get a value $\epsilon_j = 0$. The optimization problem is changed to $\min \frac{1}{2} \parallel \beta^2 \parallel + C \sum_{j=1}^{N} \epsilon_j$. C is a parameter which is used to leverage the impact of erroneous examples. It follows that $0 \leq \alpha_j \leq C \, \forall \, 0 \leq j \leq N$.

The optimization of an SVM is done by finding the most optimal values for $\alpha_j \forall 0 \leq j \leq N$. [Platt, 1999] presented an optimization approach which is called *sequential minimal optimization* (SMO). In contrast to optimizing all α-values together his approach is used to sequentially optimize the smallest number of α-values. Respecting equation (2.12) it follows that the smallest number of α-values to be optimized together is 2. Optimizing just one α_j would change the value for that α_j and equation (2.12) would not be valid anymore. We will not present the optimization process in detail

because it is out of the scope of this work.

Kernels By definition an *original* SVM is only creating models which can separate examples by a hyperplane. This is only possible if the examples are linearly separable. Unfortunately, some example sets are not linearly separable. Figure 2.8, for instance, shows a typical example set not being linearly separable. A possible approach to make this example set separable by a SVM is to convert the example set into another feature space. Every example $\mathbf{x}_j$ is converted using a mapping function $\phi(\cdot) : \mathcal{S} \rightarrow \mathcal{F}$. Equation (2.13) has to be changed in order to handle the converted examples $\phi(\mathbf{x}_j)$. The new dual Lagrangian is shown in equation (2.14).

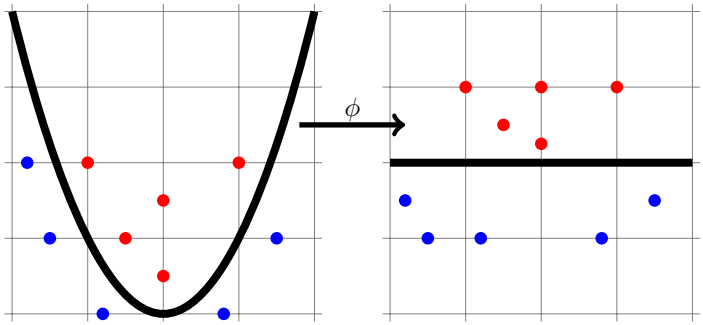


Figure 2.8: An example set not being linearly separable which is transformed to become linearly separable

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_{i'}) \tag{2.14}$$

The conversion and the calculation of the dot product which can be seen as a calculation of a similarity measure of the converted examples can be done in one step by using a kernel function [Hofmann et al., 2008].

$$k(\mathbf{x}_i, \mathbf{x}_{i'}) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_{i'}) \tag{2.15}$$

Equation (2.15) shows how kernel functions implicitly work. Although it seems as if the examples will be transformed into a feature space $\mathcal{F}$ the transformation does not really have to be performed. This so called *kernel trick* becomes obvious by having a look four typical kernel functions used for nonlinear problems in Table 2.4 presented by [Müller et al., 2001].

Gaussian RBF	$k(\mathbf{x}, \mathbf{x}') = \exp \frac{-\|\mathbf{x}-\mathbf{x}'\|^2}{c}$
Polynomial	$k(\mathbf{x}, \mathbf{x}') = ((\mathbf{x} \cdot \mathbf{x}') + \theta)^d$
Sigmoidal	$k(\mathbf{x}, \mathbf{x}') = \tanh(\kappa(\mathbf{x} \cdot \mathbf{x}') + \theta)$
inverse multiquadratic	$k(\mathbf{x}, \mathbf{x}') = \frac{1}{\sqrt{\|\mathbf{x}-\mathbf{x}'\|^2+c^2}}$

Table 2.4: Different kernel functions

Non-optimized, non-lazy Models

We presented optimized and lazy learning models in the previous Sections. Nevertheless, some kind of models are similar to both groups but they cannot be put exactly in one of them. In this Section we present Perceptrons and Naïve Bayes Classifiers. Although Perceptrons rely approximately on the same decision function like Support Vector Machines they are not really optimized because they are an iterative approach greedily collecting the relevant inputs for the decision function. Naïve Bayes Classifiers are no lazy learning approach following the definition of [Aha, 1997]. Anyway, they are comparable to lazy learning as they use the complete training set and only will calculate decisions if requests are coming in during the test phase.

Perceptron The perceptron classifier has been proposed by Rosenblatt [Rosenblatt, 1958] as a scheme for determining a separating hyperplane $\mathcal{H}$ within some feature space. It is an *error driven* learning algorithm, which calculates a hyperplane based on misclassified examples. The decision function f_w can be written as

$$f_w(\mathbf{x}) = \text{sign}\left(\langle \beta, \mathbf{x} \rangle + \beta_0\right)$$

The training of a simple perceptron is a collecting of misclassified examples to determine β and β_0. Algorithm 1 outlines the pseudo-code for the training of a perceptron where the parameter ρ determines a *learning rate*. Please note that the collection of misclassified examples as set M is not required in the original perceptron learner, but included here for later use. The result is a decision model by means of

$$\beta = \sum_{(\mathbf{x},y)\in M} \rho y \mathbf{x} \qquad \beta_0 = \sum_{(\mathbf{x},y)\in M} \rho y \tag{2.16}$$

The learning rate ρ can be chosen in different ways, where $\rho = 1$ reveals the perceptron as originally stated by Rosenberg and $\rho = \frac{1}{t}$ leads to a behavior comparable to a simple neural network (t being the number of training samples seen so far).

Naïve Bayes Classifier Naïve Bayes classifiers [Hastie et al., 2003] are well-known machine learning techniques. Based on the Bayes theorem the naïve Bayes classifiers deliver very good results for many machine learning tasks in practical use [Domingos and Pazzani, 1997].

Algorithm 1 Perceptron Algorithm

procedure TRAINPERCEPTRON
  Input: Example set $T \subset \mathcal{X} \times \mathcal{Y}$
  Output: weight vector β, bias β_0, example subset M
  Initialize $\beta := \mathbf{0}, \beta_0 := 0, M := \emptyset$
  for all $(\mathbf{x}, y) \in T$ **do**
    if $f_w(\mathbf{x}) \neq y$ **then**
      $\beta := \beta + \rho y \mathbf{x}$
      $\beta_0 := \beta_0 + \rho y$
      $M := M \cup \{\mathbf{x}\}$
    end if
  end for
  return β, β_0, M
end procedure

Many machine learning techniques like *Support Vector Machines* (SVMs), for instance, are suffering from a complex training phase. Naïve Bayes classifiers do not have this disadvantage because the training phase just consists of storing the training data efficiently. In this way a naïve Bayes classifier memorizes the training data. Although, naïve Bayes classifiers are not optimized, they deliver good results. Their ability to deliver good results while not needing exhaustive training makes naïve Bayes classifiers a good choice in resource-aware settings like mobile devices [Fricke et al., 2011, Morik et al., 2010]. The Naïve Bayes classifier (NBC) [Hastie et al., 2003] assigns labels $y \in Y$ to examples $\mathbf{x} \in X$. Each example is a vector of m attributes written as x_i, where $i \in \{1, ..., m\}$. The probability of a label given an example is, according to the Bayes Theorem:

$$p(y|x_1, x_2, ..., x_m) = \frac{p(y)p\left(x_1, x_2, ..., x_m|y\right)}{p\left(x_1, x_2, ..., x_m\right)} \tag{2.17}$$

By assuming that the features x_i are conditionally independent (naïve Bayes assumption) equation (2.17) is rewritten to create the *naïve Bayes classifier* as shown in equation 2.18. Domingos and Pazzani [Domingos and Pazzani, 1996] are calling it the *Simple Bayes Classifier* (SBC):

$$p(y|x_1, x_2, ..., x_m) = \frac{p(y)}{p\left(x_1, x_2, ..., x_m\right)} \prod_{j=1}^{m} p\left(x_j|y\right) \tag{2.18}$$

The classifier calculates the most probable class $y \in Y$ for a given example $\mathbf{x} = x_1, \ldots, x_m$:

$$\arg\max_y p(y|x_1, x_2, ..., x_m) = \frac{p(y)}{p\left(x_1, x_2, ..., x_m\right)} \prod_{j=1}^{m} p\left(x_j|y\right) \tag{2.19}$$

The term $p\left(x_1, x_2, ..., x_m\right)$ can be neglected in equation (2.19) because it is the same constant for every class $y \in Y$. The decision for the most probable class y for a given

example $\mathbf{x}$ only depends on $p(y)$ and $p\left(x_i|y\right)$ for $i \in \{1, \ldots, m\}$. The expected values for the probabilities are used which are calculated on the training data. The values can be calculated after one run on the training data. The training runtime is $\mathcal{O}(n)$, where n is the number of examples in the training set. The number of probabilities to be stored during training are $|Y| + \left(\sum_{j=1}^{m} |X_j||Y|\right)$ for nominal attributes, where $|Y|$ is the number of classes and $|X_j|$ is the number of different values of the jth attribute [Mitchell, 2010]. If the attributes are numerical, just two values for mean and standard deviation have to be stored resulting in $\mathcal{O}\left(m|Y|\right)$ probabilities.

Most implementations of naïve Bayes classifiers deliver the class y which results in the highest outcome for equation (2.19) for a given example $\mathbf{x}$. It is not stringently required to use values between 0 and 1 for y or during the calculation of the most probable y. This becomes important in Section 6.3.3. It has often been shown that SBC or NBC perform quite well for many *Data Mining* tasks [Domingos and Pazzani, 1996, Huang et al., 2003].

2.4.2 Structured Models

The kinds of models we presented in the former Sections all are independently creating decisions for examples. That means that the decision is only conditioned by the particular example. For some learning tasks the examples might depend each other and the labels of these examples might depend each other, too. Structured models like Conditional Random Fields [Lafferty et al., 2001] and Structured Support Vector Machines [Tsochantaridis et al., 2004] respect this dependencies and they do not only predict examples because of their feature assignments but also because of predictions for other examples already made or to be made. A current approach by [Fernandes and Brefeld, 2011] uses partially annotated sequences for the training of a transductive perceptrons. We will present Conditional Random Fields in detail in the following Paragraph because we are using these models in several experiments in this work.

Conditional Random Fields Conditional Random Fields (CRF) [Lafferty et al., 2001] are a special form of the Markov Random Fields (MRF) [Kindermann and Snell, 1980].

Definition 13. *A MRF is a set of random variables $\mathcal{Z}$ with respect to an undirected acyclic graph $G = (V, E)$ containing vertices $v \in V$ and edges $e \in E$, where $\mathcal{Z}$ is indexed by V ($Z_v \in \mathcal{Z} \; \forall \; v \in V$). The Markovian property has to be satisfied by $\mathcal{Z}$.*

The variable set $\mathcal{Z}$ internally is connected by an undirected graph. It follows that the calculation of conditional probabilities for certain variables Z_v cannot be calculated using an antecessor. In addition, not only one but several variables $Z_{v'}$ can be connected to Z_v. The Markovian property (see equation (2.20)) states that the probability of a certain variable Z_v only is conditioned by all its neighbors.

$$p(Z_v|Z_w \forall v \neq w) = p(Z_v|Z_w \forall w \in ne(v)) \tag{2.20}$$

where $ne(\cdot)$ is the set of neighbors of a certain node.

A *clique* (see Definition 14) is a subset of nodes where each node is connected to every other node of the subset.

Definition 14. *A **clique** is a set $Y \subseteq V$, where $(y, y') \in E \; \forall y, y' \in Y$.*

In the case of machine learning and *Data Mining* the random variables are dedicated to labels of particular examples. It follows that the prediction process has to work on the complete set of variables (and respectively labels) instead of predicting one label independently at a time. Let a certain assignment of the set of vertices V be z. Potential functions are used for the calculation of the probability of an assignment z.

Definition 15. *A **potential function** $\phi_{c_i}(\cdot) : A_{c_i} \to \mathbb{R}^+$ converts the assignments of a clique c_i into real-valued positive numbers. The set of all **potential functions** is ϕ.*

The probability of an assignment z now can be calculated by clique factorization like in equation (2.21).

$$p(z) = \frac{1}{Z} \prod_{\forall \phi_c \in \phi} \phi_c(z_{c_1}, \cdots, z_{c_{|c|}}) \tag{2.21}$$

$$Z = \prod_{\forall \phi_c \in \phi} \prod_{\forall z' \text{ for } \phi_c} \phi_c(z'_{c_1}, \cdots, z'_{c_{|c|}}) \tag{2.22}$$

where $z_{c_1}, \cdots, z_{c_{|c|}}$ is the particular clique-assignment of z for the potential function ϕ_c. Z is a normalization factor which is needed because the result of the potential functions (see definition (15)) is a real-valued number and has to be normalized to a probability. Z is the product of all potential functions using all possible assignments z' for each potential function ϕ_c.

The crucial part is the creation or selection of the potential functions to be used. The potential functions should model the observations made by visiting the *training set*. We will focus on this creation later on.

Definition 16. *A **Conditional Random Field** is a MRF that is conditioned by observations x.*

Conditional Random Fields (CRF) focus especially on the process of the construction of the potential functions using observations (examples) x.

Describing the handling of arbitrary graphs would go beyond the scope of this work. We will focus on very trivial graphs being used for NER (see Section 3.2). In NER the

examples and labels respectively are ordered sequentially. Additionally, sentences are building blocks. Each of these blocks can be seen as a CRF (MRF), where the labels $y_1, \ldots, y_n$ are the random variables and the examples $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are used to build the potential functions. Potential functions are defined on cliques. In a sequence every two neighboring nodes are building a clique. Neighboring nodes are edges contained in the set E. The potential functions on these edges are now denoted with $f(\cdot)$ as shown in equation (2.23). In addition, [Lafferty et al., 2001] are presenting the creation of potential functions not only for edges but for every single node, too. These functions are denoted with $g(\cdot)$, and they are shown in equation (2.24).

$$f(y, y'|x) \exists (y, y') \in E \tag{2.23}$$

$$g(y|x) \exists y \in V \tag{2.24}$$

We assume that for reasons of simplicity the two kinds of potential functions are in the set of potential functions ϕ. For the case of a sequence of the random variables the calculation of the conditional probability (equation (2.21)) can be rewritten like in equation (2.25).

$$p(z) = \frac{1}{Z} \prod_{\forall \phi_c \in \phi} \sum_{i=1}^{n} \phi_c(y_i, y_{i+1}) \tag{2.25}$$

Following the *Hammersley-Clifford-Theorem* [Hammersley and Clifford, 1971] equation (2.25) could be formed to equation (2.26) if the graph structuring the variables is a tree.

$$p(z) = \frac{1}{Z} \exp[\sum_{\forall \phi_c \in \phi} \sum_{i=1}^{n} \phi_c(y_i, y_{i+1})] \tag{2.26}$$

Afterwards we separate the particular potential functions into the *state features* $g(v \in V)$ and the *transition features* $f(e \in E)$ generating the following equation:

$$p(z) = \frac{1}{Z} \exp[\sum_{\forall f \in F} \sum_{i=1}^{n} f_c(y_i, y_{i+1}) + \sum_{\forall g \in G} \sum_{i=1}^{n} g_c(y_i)] \tag{2.27}$$

The potential functions $f(\cdot)$ and $g(\cdot)$ are extracted from the training set. F represents the set of potential functions $f(\cdot)$ and G represents the set of potential functions $g(\cdot)$. The definition for transition features $f(\cdot)$ is presented in Definition 17. The definition for state features $g(\cdot)$ is presented in Definition 18. The potential functions can be created out of the attributes of the training set. Each potential function gets a weighting-factor which is adjusted during training. These factors according to the potential functions are the CRF-model θ. The probability corresponding to θ for a particular assignment z is calculated by:

$$p_\theta(z) = \frac{1}{Z} \exp[\sum_{\forall f_c \in F} \sum_{i=1}^{n} \lambda_c f_c(y_i, y_{i+1}) + \sum_{\forall g_c \in G} \sum_{i=1}^{n} \mu_c g_c(y_i)] \tag{2.28}$$

Definition 17. *The transition features are defined by* $f(e \in E, x, i) = b(x, i)$ *if* $y \in e$ *and* $y' \in e$ *fulfill particular conditions. These conditions, for example, could be that* $y =$ *class A and* $y' =$ *class B.* $b(x, i) = 1$ *if an attribute of the* i*-th example* x_i *has a certain value.* $b(x, i) = 1$ *if, for instance, the* i*-th word of a sentence is* $Germany$*. Otherwise,* $b(x, i) = 0$*.*

Definition 18. *The state features are defined by* $g(v \in V, x, i) = b(x, i)$ *if* $y = v$ *fulfills a particular condition. This condition, for example, could be that* $y =$ *class A.* $b(x, i) = 1$ *if an attribute of the* i*-th example* x_i *has a certain value.* $b(x, i) = 1$ *if, for instance, the* i*-th word of a sentence is* $Berlin$*. Otherwise,* $b(x, i) = 0$*.*

During the training phase of a CRF the parameters λ_i and μ_i are adjusted to most optimally fit the potential features to the training set. This fitting is done by maximizing the log-likelihood function [Fisher, 1997]. The logarithms of the conditional probabilities of all subsets (for instance sentences) of the training set are summed up to build the log-likelihood (see equation (2.29)).

$$L(\theta) = \sum_{S \subseteq T} \log p(y_S | x_S) \tag{2.29}$$

The certain values for θ (λ_i and μ_i) have to be changed until the result of the log-likelihood function is maximal. This can be done by using multiple techniques. [Lafferty et al., 2001] present two approaches based on *iterative scaling* [Della Pietra et al., 1997]. [Wallach, 2002] has shown that numerical optimization techniques do find the optimal setting for θ faster than iterative methods. An often used numerical optimization technique is *L-BFGS* [Nocedal, 1980]. *L-BFGS* is a quasi-newton optimization technique approximating the optimization step by a Taylor series of second order. This technique uses the first and second derivative of the function to be optimized. The second derivative – the Hessian – is approximated by a Taylor approximation avoiding the computational complex calculation of the Hessian. [Vishwanathan et al., 2006] have shown that stochastic gradient methods in general and particularly *stochastic meta descent* (SMD) are more efficient for CRF training than *L-BFGS*. State of the art implementations for CRFs are based on the usage of general-purpose computation on graphical processing units for a parallel and therefore very efficient computation [Piatkowski, 2011, Piatkowski and Morik, 2011]. We used such implementation for the experiments we present in Section 3.3.

2.4.3 Binary Models for Multiple Classes

Binary models only can be used to classify binary problems which are datasets containing only two types of label-values. It is not straightforward to apply such models on multi-class datasets which contain more than two classes. To apply binary models

to such datasets a certain strategy has to be chosen. Some strategies are possible for this task. All of the strategies are remapping the original dataset to several datasets containing only two classes. Multiple binary models are applied on these datasets, afterwards. The decisions of these models are finally combined to a global decision containing multiple classes. During the combination of the models there can be several predictions for only one example. In this case only the most confident prediction is chosen.

In this work we will only focus on the so called one-against-all (1-vs-all) strategy. This strategy creates a binary learner for each class. The learners are trained by using the training set containing the relevant class (the class the learner is used for) as positive class and all other classes as negative class. "The one-vs-all scheme is conceptually simple, and has been independently discovered numerous times by different researchers.", [Rifkin and Klautau, 2004] stated.

Another strategy which exemplarily should be mentioned here is the one-against-one strategy. Using this strategy will create a model for each combination of pairs of classes from the original dataset. All examples having a class not belonging to that pair will be neglected for training the particular model.

2.4.4 Model evaluation

The created *models* should be precise in cases of prediction or grouping performance. In the case of *supervised* learning the precision has to be achieved for the training set and the test set. This is remarkable because the test set usually does not contain labels, yet. The predictions made by the learned model cannot be evaluated because they would have to be compared with the true *labels*. The distribution of the training set is supposed to be the same as for the test set. That means that a model created on the training set delivering good results on that set should also create good results on the test set. The calculated (evaluated) performance for a model delivered for the training set is nearly the same for the test set. If the model delivers bad results the parameters of the model or the feature set should be changed. Nevertheless, creating a model of the training set and determining its performance is not trivial.

Validating models

If we would use the same set for creating the model and for the determination of the performance it would be easy to create a perfect model by just memorizing the set used for training. As the set used for determining the performance does only contain examples already seen during training the model would be able to predict all the labels correctly. This is not the case for practical purposes. In practical supervised approaches, some examples are labeled and they can be used for training a model. Most of the examples are not labeled and should be predicted by the created model. The set used for training the model and the set which is used for determining the performance have to be different. To evaluate the performance which probably will be achieved by the created models on the unlabeled examples the parts of the training set can be used

to compare made predictions with correct labels. We present two often used methods for creating different sets out of the training set to be used for training and evaluating a model.

Split validation A split validation splits the training set into two parts. One part is used for training a model and the second part is used to evaluate the trained model. The labels of the second set are compared with the predictions made by the created model by using particular evaluation metrics which are presented in the following paragraphs. The disadvantage of this approach is the possibility of creating a biased split. The two sets could result in models which deliver too optimistic results on the second set.

Cross validation A cross validation splits the training set into n parts. During n iterations over these n parts $n - 1$ parts are used for training a model and 1 part is used to evaluate the model. In every iteration another part of the n parts is used for evaluating the model. Finally, the mean and standard deviation of the n performance values are calculated. This approach avoids the possibility that a split could be created resulting in too optimistic performance values. Typically, a value of $n = 10$ is chosen. If a value of $n = m$ is chosen, where m is the number of examples in the complete training set, the validation is called *leave-one-out*.

Overfitting

Overfitting (see [Hastie et al., 2003]) describes the behavior of a model which is extremely fitted to a given training set leading to a low prediction error on the training set, but, unfortunately, the prediction error on the test set is high. The model is not general enough to achieve good results on the test set. Not fitting the model to the training set too extremely would lead to lower prediction error on the test set. Validating models already during the training phase as it is done by split and cross validation can help to avoid overfitting.

Evaluation metrics

Different evaluation metrics can be used to evaluate the performance of the created models. As we are focusing mostly on classification tasks we therefore only are presenting evaluation metrics which are used for classification in this section. An example could be correctly predicted – the prediction equals the label – or the example could be falsely predicted.

The evaluation metrics presented in the following are meant to be used for the evaluation of the correctness of single tokens. In concrete *Information Extraction* tasks it is remarkable that the informational units to be extracted sometimes contain more than one token. Person's names, for instance, mostly contain two or three tokens. [De Sitter et al., 2004] define a framework for two different evaluations for *Information Extraction*. The *All Occurrences* approach is used to evaluate if all informational units are located and classified correctly in the document. This approach can be compared to and used for *Named Entity Recognition* (see Section 3.2). The *One Best per Document*

approach is used to evaluate slot-filling approaches like for *Information Extraction* presented at the beginning of Section 3. Following this approach it is not important to find all informational units but to find the best unit requested in the particular task.

In this work we only use the following evaluation metrics. If we analyze datasets containing informational units containing more than one token, we will either assume that each of the tokens is one informational unit or we will use the approach presented for the CoNLL 2002 shared task by [Tjong Kim Sang, 2002]. The latter approach is comparable to the *All Occurrences* approach, but it is very strict because an informational unit only is correctly predicted if all its tokens are correctly predicted.

Let the example set used for evaluation be S. S contains the predictions for every example made by a certain model. Let n_+ be the number of correctly predicted examples. Particular evaluation metrics are focusing on the correct and wrong predictions for certain classes. Let n_+^c be the number of correctly predicted examples containing label c. Furthermore, let $n_{\bar{c}}^c$ be the number of examples containing label c which are wrongly predicted, and let $n_c^{\bar{c}}$ be the number of examples not containing the label c but containing the prediction c.

Accuracy *Accuracy* is the fraction of correctly predicted examples and the number of all examples like stated in equation (2.30).

$$\text{accuracy}(S) = \frac{n_+}{|S|} \tag{2.30}$$

Precision *Precision* is always calculated for a particular class c. It is the fraction of correctly predicted examples of class c and the number of all examples of class c like stated in equation (2.31).

$$\text{precision}_c(S) = \frac{n_+^c}{n_+^c + n_{\bar{c}}^c} \tag{2.31}$$

Recall *Recall* is always calculated for a particular class c. It is the fraction of correctly predicted examples of class c and the number of all examples of class c like stated in equation (2.32).

$$\text{recall}_c(S) = \frac{n_+^c}{n_+^c + n_c^{\bar{c}}} \tag{2.32}$$

F-Measure *F-Measure* is the harmonic mean of *precision* and *recall* as it is stated in equation (2.33). To give *precision* and *recall* different impacts *f-measure*$_\beta$ as shown in equation (2.34) is used.

$$\text{f-measure}_c(S) = 2\frac{precision_c(S)recall_c(S)}{precision_c(S)+recall_c(S)} \tag{2.33}$$

$$\text{f-measure}_{c\beta}(S) = (1+\beta^2)\frac{precision_c(S)recall_c(S)}{(\beta^2 precision_c(S))+recall_c(S)} \tag{2.34}$$

ANOVA test

After evaluating the model performance it has to be shown if the achieved results are significant compared to performances of other models. The analysis of variance (ANOVA) [Cooper, 2003] is a well-known statistical test which can be used to proof if an evaluated performance is significantly better than another performance. The evaluated performance is given by its mean value μ, its variance ω and by the number of evaluations n (in a ten-fold cross-validation n is 10, for instance).

Let two exemplary performances be given by the values presented in Table 2.5.

performance	μ	ω	n
1	0.9	0.01	100
2	0.8	0.001	100

Table 2.5: Two performances and its values for mean, variance and number of iterations n

The sum of squares between (S_B) is calculated by

$$S_B = n * \left(\frac{\mu_1 * 2}{\mu_1 + \mu_2} \right) + n * \left(\frac{\mu_2 * 2}{\mu_1 + \mu_2} \right)$$

The degree of freedom between the groups (f_b) is calculated by

$$f_b = m - 1$$

where m is the number of groups, which is 2 in this case. The sum of squares within (S_W) is calculated by

$$S_W = (n-1) * (\omega_1 + \omega_2)$$

The degree of freedom within the groups (f_w) is calculated by

$$f_w = m(n-1)$$

where m is the number of groups, which is 2 in this case. The mean squares value between and within are calculated by

$$MS_B = \frac{S_B}{f_b}$$

$$MS_W = \frac{S_W}{f_w}$$

The $F - ratio$, finally, is calculated by

$$F = \frac{MS_B}{MS_W}$$

ratio	value
S_B	0.5
f_b	1
S_W	1.089
f_w	198
MS_B	0.5
MS_W	0.006
$F - ratio$	90.909
$p - value$	≈ 0.0

Table 2.6: The resulting values after performing an ANOVA test on the two measurements shown in Table 2.5

Using this $F - ratio$ allows the calculation of a $p - value$ which will state if a performance value is significantly better or worse than another one. The resulting values are shown in Table 2.6.
In this example the first performance achieved is significantly better than the second one if we assume a significance level of 5%. All results leading to a $p - value$ which is smaller than the significance level can be seen as statistically significant.

2.5 Summary

In this section we gave a sufficient overview on machine learning methods. We presented the basic techniques which will be consistently used in this work. First of all, we defined the formal notations of elements used in the domain of machine learning.

A very crucial point is the feature set which is needed for the learning process. The feature types are split into four groups: nominal, numerical, date and (tree-)structured. Especially the tree-structured features will be used very often in the later sections. In addition to the description of tree-structured features, we gave certain examples of tree structures which occur in *Information Extraction* or *Data Mining* tasks.

We gave an overview about data preparation which is done for several reasons. These reasons might be: defective feature set, huge feature set, or a feature set which is not applicable by the currently used models.

Additionally, we have presented particular specialties which should be respected by *Information Extraction* methods, and therefore these specialties also have to be respected by the feature set and by the methods which create this feature set.

The concrete task of learning from datasets is called modeling. We presented the techniques used for modeling datasets. In most of this work we focus on supervised learning techniques because these are the methods mostly needed for *Information Ex-*

traction. We split the family of supervised learning methods into rule-based, statistical and structured models. We particularly split the group of statistical methods into lazy learning and optimized methods to visualize the differences between both subgroups. We additionally present methods which are not described by one of the two groups.

We are especially focusing on three certain techniques which are support vector machines, perceptrons and naïve Bayes classifiers. Perceptrons and naïve Bayes classifiers are not optimized during the training phase which makes them not delivering any guarantees for the performance achieved by these approaches in case of accuracy. Although a good performance in case of accuracy is not guaranteed for perceptrons and naïve Bayes classifiers, the application of each of these methods delivers good results in practical. If the examples are separable either linearly or by the use of particular kernel functions, a support vector machine will find an optimal solution for that example set. Finding an optimal solution is computationally costly during the training phase which leads to a longer training runtime of support vector machines compared to perceptrons and naïve Bayes classifiers. We will empirically analyze these methods in the following sections.

Finally, we have shown how to validate and evaluate the created models. We defined the evaluation measures which will be used in most of the experiments we will present in the later sections, and we presented possibilities to validate models in order to ensure the significance of the achieved results.

Chapter 3

Evolution of Information Extraction

In this chapter we will focus on the historical development of *Information Extraction*. The *Message Understanding Conferences* (MUC) [Grishman and Sundheim, 1996] were the first conferences in which the topic of *Information Extraction* became apparent. Participants of the shared task of each conference had to extract predefined types of information out of documents from a particular domain. The MUCs were initiated by the *Defense Advanced Research Projects Agency* (DARPA) which is a military organization, for this reason the domains, the documents were extracted from, mostly were military ones. Although, beside *Fleet operations* and *Terrorism* another domain of the documents to be analyzed was *Joint Ventures*. Since the second MUC the task became a template-filling one. A template containing the most interesting informational types concerning the specific domain were predetermined and should be extracted by the systems/methods developed by the participants. Figure 3.1 shows an exemplary document of the third MUC [Sundheim, 1991].

A template containing 18 *slots* concerning particular types of information was given. The participants had to fill these *slots* in a most precise way. The certain template of the third MUC and the correctly extracted informational units for the document shown in Figure 3.1 is shown in Figure 3.2. We annotated the parts to be extracted out of the document in Figure 3.3.

Most of the subtasks it has been worked on during or after the MUCs have been processed by "handcrafted systems" that were based on many handcrafted rules. The systems became very domain-dependent. Additionally, heavy effort of linguists was needed to analyze the domains and to create the rules. These systems of course are very inflexible and therefore were replaced by statistical and machine learning methods. Machine learning methods are more domain independent. They are not as good but nearly as good as handcrafted systems. The achievements in flexibility make these methods more applicable. Although *Information Extraction* became apparent in the

TST-1-MUC3-0080
BOGOTA, 3 APR 90 (INRAVISION TELEVISION CADENA 1) - [REPORT] [JORGE ALONSO SIERRA VALENCIA] [TEXT] LIBERAL SENATOR FEDERICO ESTRADA VELEZ WAS KIDNAPPED ON 3 APRIL AT THE CORNER OF 60TH AND 48TH STREETS IN WESTERN MEDELLIN, ONLY 100 METERS FROM A METROPOLITAN POLICE CAI [IMMEDIATE ATTENTION CENTER]. THE ANTIOQUIA DEPARTMENT LIBERAL PARTY LEADER HAD LEFT HIS HOUSE WITHOUT ANY BODYGUARDS ONLY MINUTES EARLIER. AS WE WAITED FOR THE TRAFFIC LIGHT TO CHANGE, THREE HEAVILY ARMED MEN FORCED HIM TO GET OUT OF HIS CAR AND INTO A BLUE RENAULT.
HOURS LATER, THROUGH ANONYMOUS TELEPHONE CALLS TO THE METROPOLITAN POLICE AND TO THE MEDIA, THE EXTRADITABLES CLAIMED RESPONSIBILITY FOR THE KIDNAPPING. IN THE CALLS, THEY ANNOUNCED THAT THEY WILL RELEASE THE SENATOR WITH A NEW MESSAGE FOR THE NATIONAL GOVERNMENT. LAST WEEK, FEDERICO ESTRADA HAD REJECTED TALKS BETWEEN THE GOVERNMENT AND THE DRUG TRAFFICKERS.

Figure 3.1: An exemplary document of the shared task of the third *Message Understanding Conference* ([Sundheim, 1991], p. 8)

late 1980s, [Daelemans et al., 1997] still stated in 1997 that insufficient efforts are being made to use empirical (machine) learning in *Information Extraction* tasks.

To help message-understanding systems by providing special entities, which hopefully will be needed in every MU-task, the Named Entity Recognition (NER) were provided as a task in the 6th MUC in 1995. The task was to discover general entities in texts like persons, locations and organizations (see Section 3.2). In contrast to NER which analyzes the particular parts of a document, text mining (see Section 3.1) is a marginally related task. In text mining the whole document is classified. Although no informational units are extracted out of the documents during the text mining process, it can be used to determine the relevant documents out of a greater amount of documents to be used for *Information Extraction* purposes. Named Entity Recognition is a compellent step for *Relation Extraction* (see Chapter 5). *Relation Extraction* focuses on pairs of Named entities to decide if the two entities in such pairs are related. Therefore, it is very crucial to precisely extract the named entities. Wrongly extracted entities will result in mistakes during the *Relation Extraction* process.

3.1 Text Categorization

The term *Text Categorization* (also: *Text Classification* or *Text mining*) describes the automatic classification of complete documents. Although *text categorization* is not used for the extraction of informational units from documents, *text categorization* can be used to extract relevant documents for *Information Extraction* in a preprocessing step. Statistical machine learning methods like Support Vector Machines are the most popular tools for the classification of text documents [Joachims, 2002]. Following the notation of [Hastie et al., 2003] each document is described by observations x_i. For all

0	MESSAGE ID	TST1-MUC3-0080
1	TEMPLATE ID	1
2	DATE OF INCIDENT	03 APR 90
3	TYPE OF INCIDENT	KIDNAPPING
4	CATEGORY OF INCIDENT	TERRORIST ACT
5	PERPETRATOR: ID OF INDIV(S)	"THREE HEAVILY ARMED MEN"
6	PERPETRATOR: ID OF ORG(S)	"THE EXTRADITABLES"
7	PERPETRATOR: CONFIDENCE	CLAIMED OR ADMITTED: "THE EXTRADITABLES"
8	PHYSICAL TARGET: ID(S)	*
9	PHYSICAL TARGET: TOTAL NUM	*
10	PHYSICAL TARGET: TYPE(S)	*
11	HUMAN TARGET: ID(S)	"FEDERICO ESTRADA VELEZ" ("LIBERAL SENATOR")
12	HUMAN TARGET: TOTAL NUM	1
13	HUMAN TARGET: TYPE(S)	GOVERNMENT OFFICIAL: "FEDERICO ESTRADA VELEZ"
14	TARGET: FOREIGN NATION(S)	–
15	INSTRUMENT: TYPE(S)	*
16	LOCATION OF INCIDENT	COLOMBIA: MEDELLIN (CITY)
17	EFFECT ON PHYSICAL TARGETS	*
18	EFFECT ON HUMAN TARGETS	*

Figure 3.2: The template to be filled by the participants of the third *Message Understanding Conference* and the corresponding informational units for the document shown in Figure 3.1 ([Sundheim, 1991], p. 8)

documents the indexes of each vector have to contain the same type of information. If for instance the second entry of a vector of observations for a particular document contains the number of verbs in the document, the second entries of each other document have to contain the number of verbs of those documents, too. To represent documents by a well-suited vector of observations is a crucial task. Two particular subtasks have to be respected: the first one is to extract the types of observations which are used for representing the text (representation). The second subtask is to set the values for the certain types of observations (weighting). The vector containing the corresponding values for the represented text is called feature vector.

The most popular representation of documents for *text categorization* is the Bag of Words (BOW) representation. This representation originally is from the Information Retrieval community [Salton et al., 1975]. Each unique word in a document-collection (index term) is becoming an entry in the vector of observations. Finally, a document represented in the BOW representation has a value of 1 for every index term it contains. If an index term is not contained in a particular document, the document has a value of 0 for that term. Using a binary weighting by storing just 1 or 0 for each attribute is the most simple case of the BOW representation.

In addition to just using unique words for the feature vector, much research has been done on extracting multi-word sequences (n-grams) to be used in the feature vector, too (see [Fürnkranz, 1998]). [Bekkerman and Allan, 2004] furthermore defined n-grams as n unigrams which appear as a permutation in the document. The unigrams are defined as stems of words in the permutation which additionally can contain stop words making their approach more flexible.

TST-1-MUC3-0080
BOGOTA, 3 APR 90 (INRAVISION TELEVISION CADENA 1) - [REPORT] [JORGE ALONSO SIERRA VALENCIA] [TEXT] LIBERAL SENATOR FEDERICO ESTRADA VELEZ WAS KIDNAPPED ON 3 APRIL AT THE CORNER OF 60TH AND 48TH STREETS IN WESTERN MEDELLIN, ONLY 100 METERS FROM A METROPOLITAN POLICE CAI [IMMEDIATE ATTENTION CENTER]. THE ANTIOQUIA DEPARTMENT LIBERAL PARTY LEADER HAD LEFT HIS HOUSE WITHOUT ANY BODYGUARDS ONLY MINUTES EARLIER. AS WE WAITED FOR THE TRAFFIC LIGHT TO CHANGE, THREE HEAVILY ARMED MEN FORCED HIM TO GET OUT OF HIS CAR AND INTO A BLUE RENAULT.
HOURS LATER, THROUGH ANONYMOUS TELEPHONE CALLS TO THE METROPOLITAN POLICE AND TO THE MEDIA, THE EXTRADITABLES CLAIMED RESPONSIBILITY FOR THE KIDNAPPING. IN THE CALLS, THEY ANNOUNCED THAT THEY WILL RELEASE THE SENATOR WITH A NEW MESSAGE FOR THE NATIONAL GOVERNMENT. LAST WEEK, FEDERICO ESTRADA HAD REJECTED TALKS BETWEEN THE GOVERNMENT AND THE DRUG TRAFFICKERS.

Figure 3.3: Important information marked in the text shown in Figure 3.1

Table 3.1 shows three exemplary sentences which are used as documents, here. Table 3.2 shows the feature vectors for the three documents represented as a binary bag-of-words model. Using two-grams will result in a representation as shown in Table 3.3. The approach presented by [Bekkerman and Allan, 2004] using only stemmed bigrams will result in the representation shown in Table 3.4.

1	Felix goes to work.
2	Felix goes to IBM.
3	Felix will go to IBM.

Table 3.1: Three different documents which shall be represented for *text categorization*.

	Felix	goes	to	work	.	IBM	will	go
1	1	1	1	1	1	0	0	0
2	1	1	1	0	1	1	0	0
2	1	0	1	0	1	1	1	1

Table 3.2: The three documents shown in Table 3.1 represented as bag-of-words with binary weights.

It is remarkable that for mostly every document collection the number of entries in the feature vector become very huge. [Joachims, 1998] has shown that a support vector machine can handle many features sufficiently. Nevertheless, [Caropreso et al., 2001] state that feature selection should be performed because of two reasons. Firstly, the runtime of most approaches for *text categorization* depend on the number of features,

	_ Felix	Felix_ goes	goes_ to	to_ work	work_ .	._	to_ IBM	IBM_ .	Felix_ will	will_ go	go_ to
1	1	1	1	1	1	1	0	0	0	0	0
2	1	1	1	0	0	1	1	1	0	0	0
3	1	0	0	0	0	1	1	1	1	1	1

Table 3.3: The three documents shown in Table 3.1 represented as bag-of-bigrams with binary weights.

	Felix_go	go_work	go_IBM
1	1	1	0
2	1	0	1
3	1	0	1

Table 3.4: The three documents shown in Table 3.1 represented as bag-of-stemmed-bigrams with binary weights.

therefore, the lesser the number of features the better. The second reason to perform feature selection is for the avoidance of overfitting. Only the most relevant features should be selected. To calculate the relevance of each feature of the feature vector [Caropreso et al., 2001, Sebastiani, 2005] repeat the feature selection methods most often used in literature. These feature selection methods are *Information Gain*, *Chi Square*, *Gain Ratio*, *Odds Ratio* and *Document Frequency*.

[Joachims, 1998] is following the suggestion of [Yang and Pedersen, 1997] who recommend to use *Information Gain*. After selecting the most relevant features by selecting the ones having the highest *Information Gain* [Joachims, 1998] is using the *Term Frequency - Inverse Document Frequency* (TFIDF [Salton and Buckley, 1988]) weighting to weigh the particular features.

3.2 Named Entity Recognition

Named Entity Recognition (NER) is the task to detect and classify mentions of formerly defined entity types out of texts or documents. An entity is a unique token describing a particular element of the given entity type. There might be multiple occurrences of entities in a document or in a text. These references of particular entities are called entity mentions. In the shared task of the *MUC-7* [Chinchor, 1998] the NER task is split into three subtasks. The first task is to extract named entities (organizations, persons, locations), the second task is to extract temporal expressions (dates, times), and the third task is to extract number expressions (monetary values, percentages). In the shared task of the ConLL 2002 Named Entity Recognition is defined as follows: ”Named entities are phrases that contain the names of persons, organizations, locations, times and quantities” [Tjong Kim Sang, 2002]. Following these two definitions, it is getting clear that the most typical entities are organizations, persons and locations. Many Named Entity Recognition systems are trained to extract these typical

entities. Unfortunately, for different domains different named entities are representative and therefore they are interesting. In a biomedical domain for instance, it is more relevant to extract genes and amino-acids than to extract organizations and locations. It is necessary to train the named entity recognition systems for certain domains containing special entities of interest. For sufficient training it is important to have enough (manually) annotated data.

3.2.1 Data generation

We assume that a document which should be used for learning or training a machine learning technique for *Information Extraction* purposes is given. After having done a morphological analysis (see Chapter 4) to split the documents into sequences of tokens, the tokens have to be labeled somehow. The task of labeling tokens is called annotation. The tools for annotation are called annotators. Exemplary annotators are PALinkA[1] and GATE[2] (see Section 7.3.5). We present an annotation tool for our plugin, too (see Section 7.2.2).

If no training data, which should be used by machine learning approaches, is available for a particular domain, a sufficient amount of data has to be annotated for the creation of a training set. If, in addition, the task to create the annotations is done by several annotators, the resulting annotations might differ from annotator to annotator. Each annotation has to be done by multiple annotators and a so called inter-annotator agreement is calculated [Cohen, 1960]. The inter-annotator agreement is meant to be a measure indicating if the different annotators are annotating in a similar way. If the annotations are trustworthy they can be used for labeling the tokens. And finally, the tokens can be used as a training set for machine learning approaches.

3.2.2 Methods

The methods to be used for NER are manifold. The usage of particular learning methods need certain preprocessing of the texts. Although, for the extraction of tokens representing entity mentions it is mandatory to split the documents or texts into tokens. These tokens after that are represented in a certain feature space. For the named entity extraction each of these tokens is classified. Methods to be used for NER can be split into two categories:

- structured and
- non-structured methods

Non-structured methods do not respect the inherent structural information given by documents or texts. This means that every token is handled independently without respecting the contextual tokens. These methods may use contextual information about a particular token in form of contextual features contained in the feature space. Although, during training and prediction the predictions and labels of contextual tokens

[1] `http://clg.wlv.ac.uk/trac/palinka`
[2] `http://gate.ac.uk`

do not condition each other. Structured methods are respecting structural information during the training and prediction phase. The learnt model and the made predictions are conditioned by other labels or predictions. Chapter 2 defines structured and non-structured methods in machine-learning.

Rule-based Models

The first systems for *Information Extraction* were based on handcrafted rules. The creation of these rules has the disadvantage that experts are needed and that the resulting rules are domain-dependent. One of the first systems for NER, for instance, is *SPARSER* [McDonald, 1996]. *SPARSER* uses a three-step-model:

1. **delimit**
   identify the boundaries of the possible named entities
2. **classify**
   categorize the resulting constituents
3. **record**
   create a discourse model containing the classified constituents

Those systems were based on finite automata or context free grammars. The heavy efforts which have to be made for these approaches in addition to their domain-dependence make them not preferable. We will focus on more flexible approaches in this work.

3.2.3 Feature Space

A useful feature space is needed for the automatic extraction of NEs. [McDonald, 1996] stated that using internal and external evidence together is important for good results. Internal evidence is given by all information we can extract out of the token itself. External evidence is given by contextual information surrounding the currently focused token. [Daelemans, 1999] also mentioned to use contextual (external) information in addition to the currently focused word (token). Figure 3.4 shows an exemplary sentence containing three entity mentions of the entities *Felix*, *Hamburg* and *Germany*. We focus on the first mention *Felix* to explain internal and external evidence.

Information units being extracted out of the mention itself are the *word*, the *stem*,

Felix goes to Hamburg which is in Germany.

Figure 3.4: A sentence containing three entity mentions.

the *regular expressions* that match this entity mention, letter *n-grams* of the mention, *prefixes* or *suffixes* of the mention, other *generalizations* of the mention, and so on. Information units concerning external evidence are extracted of the context of the entity mention. The context is given by the tokens which are connected to the mention token by the internal structure. In many NER tasks the sequential word order is used

as the internal structure. In those cases the tokens before and after the entity mention are used for the extraction of external evidence. For the extraction of features the contextual tokens can be handled exactly like the token itself but with the addition that the information extracted out of them is stored in the feature space for the current token.

Table 3.5 shows information units extracted from the sentence shown in Figure 3.4.

Label	Word	Prefix	Word_-1	Word_+1
PER	Felix	Fel	?	goes
O	goes	goe	Felix	to
O	to	to	goes	Hamburg
LOC	Hamburg	Ham	to	which
O	which	whi	Hamburg	is
O	is	is	which	in
O	in	in	is	Germany
LOC	Germany	Ger	in	.
O	.	.	Germany	?

Table 3.5: Exemplary information units extracted out of the sentence shown in Figure 3.4

The information units contain internal and external evidence. Each line of the table represents one particular token, whereas each column represents one particular information or attribute describing this token. The *Label* attribute contains information about the classification of the tokens. *Word* and *Prefix* are extracted from certain tokens. And *Word_-1* and *Word_+1* contain information extracted from contextual tokens – in this case the *Word* attribute information units of the preceding and following tokens are extracted.

It is remarkable, that internal and external evidence as it is described here is only useful in plain text documents. Nevertheless, documents are not only containing plain text. HTML-documents, for instance, are offering a structure – given by the HTML-tags – which respectively contains plain text again. Additionally, meta information describing the complete document might be interesting for the classification of entities. Finally, the plain text itself can be seen as a structure by applying parsing methods which extract internal structures like constituency or dependency parse trees.

Particular Features for Named Entity Recognition

[Settles, 2004] defines two types of features to be used for *Named Entity Recognition*:

- orthographic features
- semantic features

He presents orthographic features as features which are extracted from the tokens or the context of the tokens itself. This extraction can be done without further knowledge of semantical or morphological characteristics of the certain token. Examples for

orthographic features are the tokens itself, contextual tokens, prefixes, n-grams, suffixes, generalized tokens, and so on. Semantic features are those containing semantical information concerning the particular token. These features are based on the use of semantical resources like gazetteers, lexicons and ontologies.

Marc Roessler who wrote his PhD-thesis [Roessler, 2006] about *Named Entity Recognition* developed a huge set of orthographic features. The *Wortoberflächenmerkmale* (word-surface-features) can be seen as regular expressions which do or do not match on particular words. Examples for such features are *exactly 4 numbers*, *capital and lower case letters inside of the word* or *only capital letters without a vowel*. These features indicate several types of word types. The first features probably indicates a year, the second features indicates a special form of abbreviation and the third also indicates an abbreviation. Additionally, Roessler presented features which are based on the substring representation of the particular words. For each word a maximum amount of eight of such features are being created. The features consisting of the last letter, the two last letters and a letter window of size three which is shifted from the beginning and from the end of the word creating a maximum of six additional features. Table 3.6 shows the resulting features for the word "Jungermann".

Jungermann	Jun	ung	nge	rma	man	ann	nn	n

Table 3.6: Substring representation features of [Roessler, 2006] for word "Jungermann"

We would like to introduce a third class of features for *Named Entity Recognition*:

- morphologic features

These features heavily rely on complex morphologic analysis of the particular token, and in addition to morphologic analysis they include part-of-speech tagging and parsing, too. The most of these features are presented in Chapter 4.

3.3 Experiments on Domain- and Language-independent NER

As we already mentioned the successful extraction of named entities is depending on the domain the documents are belonging to. Domain-experts are helpful to gain insights and for the construction of domain-relevant patterns. For the extraction of named entities in the domain of bio-medicine, for instance, [Settles, 2004] used features indicating the presence or absence of Greek letters. Such letters are often contained in gene-names. Although, without knowing about that fact no one would use the presence of Greek letters as a feature for named entity recognition. Unfortunately, the analysis of the domain the particular documents are from is very time-consuming. In [Jungermann, 2006] we started analyzing feature sets which only rely on domain-independent

features. In [Jungermann, 2007, Jungermann and Roessler, 2007] we embedded the semantic resource *wikipedia* into our feature set.

In this section we want to analyze the performance of an entity recognition system without having domain- and language-knowledge of the corresponding documents. Instead of analyzing the particular domains of the documents we focused on general feature spaces containing features representing internal and external evidence. For all the experiments we used the same feature set. This is remarkable because we will assume that such feature set can be used for any domain or language if the results achieved by using this feature set are good.

Datasets

We processed three datasets to analyze the possible insignificance of domain-relevant features. All of the datasets are well-known in the *Information Extraction*-community. Additionally, the datasets come with a training and a test set which directly lets us compare our achieved results with the results other researchers have achieved on those datasets.

The first dataset is the one presented at the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA) 2004 [Kim et al., 2004]. This dataset contains biomedical entities which is a good argument for not only to extract the typical well-known entities like person-names, locations and so on. The dataset contains 492.551 words which are tagged using six classes (*O*, *protein*, *DNA*, *cell_type*, *cell_line*, *RNA*). Table 3.7 shows the distribution of the certain classes. This dataset only provides the words and the labels for these words.

The second dataset is the one presented at the *Conference on Computational Natural Language Learning* 2000 (CoNLL) [Tjong Kim Sang and Buchholz, 2000]. The shared task offering the dataset was about chunking which is used for separating sentences into smaller parts containing words which belong to the same grammatical unit. Table 3.8 shows the particular classes and the distribution of the certain classes which have to be extracted. The dataset provides three attributes: a label, the words and the PoS-tags for the particular words.

Class	O	protein	DNA	cell_type	cell_line	RNA
Distribution	77.75 %	11.19 %	5.14 %	3.14 %	2.28 %	0.50 %

Table 3.7: The class-distribution of the JNLPBA dataset

The third dataset is the one presented at the shared task of the *Evaluation of NLP and Speech Tools for Italian* (EVALITA) workshop 2007 [Speranza, 2007]. It contains the typical named entities *Person*, *Location* and *Organization* (plus *Geopolitical entity (GPE)*). The language of the documents contained in the dataset is Italian. The dataset has three attributes. One of them is the label and the other two are the word and the

Class	NP	VP	PP	ADVP	SBAR	ADJP	PRT	CONJP	INTJ	LST	UCP
Dstr.	51 %	20 %	20 %	4 %	2 %	2 %	1 %	0 %	0 %	0 %	0 %

Table 3.8: The class-distribution of the CoNLL dataset (on the training set)

PoS-tag for the certain word.

Feature sets

We used *Conditional Random Fields* (CRF) for the analysis of the datasets. CRFs are the state of the art methods for NER and they are presented in Section 2.4.2. We used the following feature set for the experiments on the three datasets. The features completely were orthographic ones (see Section 3.2.3) leading to a feature set which can easily be adapted for several tasks of different domains and languages. The feature set was a combination of word-, generalize-, prefix- and suffix-features. These features all have been extracted from the token currently processed. In addition to the features of the current token the same features of the two preceding and the two following tokens also have been taken into account. The CoNLL and the EVALITA dataset additionally contain part-of-speech tags. We also use these tags for the feature set, and we embed the PoS-tags of the two preceding and the two following tokens into the feature set, too. It follows that every token has 20 (JNLPBA) or 25 (EVALITA and CoNLL) features, which are the words before and after the token, the pre- and suffixes of the words before and after the token (both of length 3), the generalized form of the words before and after the token, the PoS-tags of the tokens before and after the token. The particular features (word, prefix, ...) of the current word are used, too. Table 3.9 shows the feature set processed on an exemplary example set.

Results

Our results are promising. Especially the fact that a feature set only based on orthographic features has been used is remarkable. Table 3.10 shows the best and the worst results achieved on the particular datasets in combination with our results. All of the methods which achieve the best results make use of dictionaries or gazetteers. Our experiments show that the absence of gazetteers and dictionaries and no performing any feature selection result in good results . For the CoNLL dataset we created a domain-tailored feature set especially containing features created by a noun phrase chunker. We wanted to show that our experiment setting achieves the same results if the feature set contains domain-tailored features. The results, which are presented in row **Jungermann***, are equal to those achieved by the best performing system presented in [Zhang et al., 2001].

3.4 Summary

We have presented the historical development of *Information Extraction* in this chapter. We have shown that the origin of *Information Extraction* is considered to be the Message Understanding Conferences. The task to extract formerly defined informational

Label	Word_-2	Word_-1	Word	Word_+1	Word_+2	...
PER	?	?	Felix	goes	to	...
O	?	Felix	goes	to	Hamburg	...
O	Felix	goes	to	Hamburg	which	...
LOC	goes	to	Hamburg	which	is	...
O	to	Hamburg	which	is	in	...
O	Hamburg	which	is	in	Germany	...
O	which	is	in	Germany	.	...
LOC	is	in	Germany	.	?	...
O	in	Germany	.	?	?	...

...	Generalize_0	Generalize_+1	Generalize_+2
...	Aaaaa	aaaa	aa
...	aaaa	aa	Aaaaaaa
...	aa	Aaaaaaa	aaaaa
...	Aaaaaaa	aaaaa	aa
...	aaaaa	aa	aa
...	aa	aa	Aaaaaaa
...	aa	Aaaaaaa	x
...	Aaaaaaa	x	?
...	x	?	?

Table 3.9: Feature set used for the experiments extracted from the sentence shown in Figure 3.4

	CoNLL 2000	**JNLPBA 2004**	**EVALITA 2007**
Best result	94.13 % [Zhang et al., 2001]	72.6 % [Zhou and Su, 2004]	82.14 % [Pianta and Zanoli, 2007]
Jungermann	92.97 %	70.82 %	74.49 %
Jungermann*	94.13 %		
Worst result	85.76 % [Vilain and Day, 2000]	49.1 % [Lee et al., 2004]	50.88 % [Walker, 2007]

Table 3.10: Various results on NER datasets

units out of documents from a certain domain was generalized over the years.

One general subtask which emerged from *Information Extraction* is Named Entity Recognition. We have shown that the feature space for NER can be split into three types of features. The orthographic features extracted from the token itself represent internal evidence whereas features extracted from the contextual tokens represent the external evidence. Only the combination of internal and external evidence delivers adequate features for the extraction of correct classes for particular tokens. Semantic features are gathering information from semantical sources like lexicons and gazetteers, and finally, morphological features – based on complex morphological analysis – are offering rich semantic information. Orthographic features are most easily to process whereas semantic features need certain semantic sources and therefore become domain-dependent, and morphologic features are based on former or ad-hoc analysis which are very complex.

The choice of the certainly used machine learning approach is significant. We present the use of structured and non-structured methods. Structured methods deliver the most accurate results for NER.

We have shown that the creation of domain-dependent and computationally complex feature sets can be avoided. We made experiments on three datasets by using the same types of features for all of these datasets. We used a feature set which is only based on orthographic features without any domain-dependent features and especially without any domain-dependent feature tuning. The results are remarkable: we achieved very good performance for all of the datasets. Particularly, we achieved a performance nearly as good as the best published results for two of the datasets. We have additionally shown that if the feature set is domain-dependent the achieved results will be equal to the best results achieved on the particular datasets.

Although text classification is not really meant to be part of *Information Extraction* we present it anyway in Section 3.1 due to the fact that it might become relevant during preprocessing of *Information Extraction* tasks.

Chapter 4

Linguistic Resources for Information Extraction Systems

In this chapter we will present the possibilities to use linguistic knowledge for *Information Extraction* purposes. A sufficient feature space for these purposes heavily relies on linguistic features. In Section 3.2.3 we called those features *morphological*. Those features can be extracted for several types of tokens of the particular documents. Sentences, for instance, might be analyzed by a syntactic parser which creates a parse tree for the certain sentence. More specific, for word-level analysis, tokens containing single words can be enriched by linguistic knowledge like morphologic analysis, part-of-speech (PoS) tags, lemmas, stems or basic linguistic forms.

Linguistic systems or knowledge bases have to be used for the creation of such linguistic features. These bases can on the one hand be annotated corpora containing documents which are annotated for linguistic purposes or on the other hand these bases might be lexicons containing or delivering specific linguistic knowledge for certain types of single tokens.
Annotated corpora contain sets of sentences in which syntactical correlations of tokens are annotated if there are any.
Lexicons in this context are to be requested in order to deliver the linguistic knowledge stored for a particular token. Such linguistic knowledge mostly refers to the token itself and it is independent of the possible context of the requested token. Anyhow, certain lexicons may also deliver knowledge concerning the possible contexts the requested tokens might occur in.

It follows that lexicons directly can be used for enriching tokenized documents by additional attributes. If, for instance, a lexicon containing the basic forms of English words is employed to enrich an example set like the one presented in Table 3.5, it will be easy to develop an operator which makes use of the lexicon processing each token of the example set independently.

Annotated corpora of course also can be converted into a kind of lexicon containing linguistic information concerning particular tokens. Usually, those corpora are used as training sets for the adjustment of machine learning methods. [Sha and Pereira, 2003], for instance, presented the use of CRFs (see Section 2.4.2) for shallow parsing. That approach which is a supervised learning method needs a particular training set containing the linguistic information for each token which can be provided by such an annotated corpus.
Nevertheless, annotated corpora can contain linguistic information spreading over several tokens, like a dependency relation or an information concerning a phrase spreading several tokens. Such more complex information can be used for the training of linguistic parsers being able to later on create dependency or constituent parse trees, for example.

4.1 Penn Treebank – An exemplary annotated corpus

The Penn treebank [Marcus et al., 1993] is an annotated corpus for the English language. It contains text of several types of documents. The documents are listed in the following and they are sorted in descending order by the number of tokens they are containing:

- Stories of the Dow Jones Newswire
- the Brown Corpus
- Abstracts of the Department of Energy
- Messages from the 3rd MUC
- Texts from the Library of America
- Sentences from IBM manuals
- Bulletins of the Department of Agriculture
- Sentences from ATIS
- Transcripts from the radio WBUR

The treebank contains about 4.9 million tokens which are annotated with part-of-speech (PoS) tags. In contrast to other corpora which contain a more extensive tag set, the Penn treebank only offers 36 PoS tags and 12 additional tags used for punctuation and currency, for example. Exemplary PoS tags are *NN* for nouns or *NNS* for nouns in its plural form.

In addition to the given PoS tags, 2.9 million tokens contain syntactical tags given by a so called *skeletal parsing*. This parsing brings together phrases of tokens belonging to one syntactical group. Such group might be a noun phrase (*NP*) or a verb phrase (*VP*), for instance.

Figure 4.1 shows an exemplary sentence of the Penn treebank annotated by a skeletal parsing. The figure shows the string representation of the parse tree using the the bracketing approach already mentioned in Section 2.1.4.

```
( (S
   (NP Battle-tested industrial managers
       here)
    always
   (VP buck
       up
       (NP nervous newcomers)
       (PP with
           (NP the tale
               (PP of
                   (NP (NP the
                           (ADJP first
                                 (PP of
                                     (NP their countrymen)))
                           (S (NP *)
                              to
                              (VP visit
                                  (NP Mexico))))
                       ,
                       (NP (NP a boatload
                               (PP of
                                   (NP (NP warriors)
                                       (VP-1 blown
                                           ashore
                                           (ADVP (NP 375 years)
                                                 ago)))))
                           (VP-1 *pseudo-attach*)))))))
   .)
```

Figure 4.1: An annotated sentence (skeletal analysis) of the Penn treebank ([Marcus et al., 1993], p. 325)

4.1.1 Other Treebanks

Treebanks like the Penn treebank can be used to train parsers which afterwards are able to deliver syntactical parse trees for sentences. In our implementation (see Chapter 7) we use the *Stanford parser* which comes with some already trained models. These models are based on the Penn treebank (for English), the Chinese Penn treebank (for

Chinese) and the Negra corpus (for German).

Treebanks are available for many languages. A famous treebank for the German language is the mentioned Negra corpus [Skut et al., 1997]. The remarkable point is that the German language has a free word order which has to be respected by the annotation scheme. Due to the free word order a parse tree cannot trivially be mapped on the sentence in a direct way. Figure 4.2 shows the parse tree of a German sentence. Parse trees for German sentences sometimes have to respect crossing edges which is not really supported by a common tree structure.

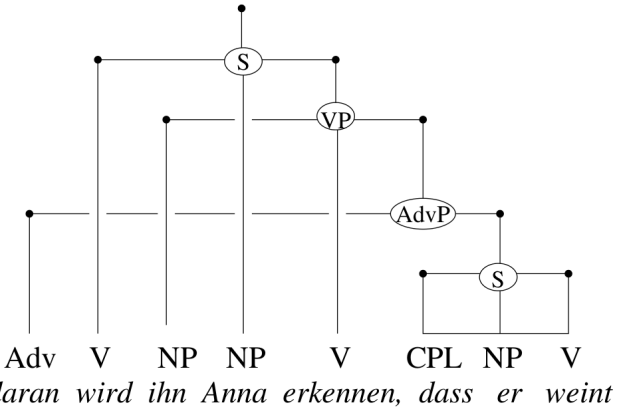


Figure 4.2: A parse tree mapped on a German sentence ([Skut et al., 1997], p. 2)

[Dima and Hinrichs, 2011] presented a treebank for the aid of a question answering systems for the CLARIN project (see Section 4.2.5). The development of such treebank becomes necessary because parsers trained on, for instance, newspaper corpora are not performing well on particular tasks like question answering.

Another corpus for German language is the *Tübinger Baumbank des Deutschen* (TüBa-D/Z) consisting of nearly 56.000 sentences containing about $1.000.000$ tokens. The features which are annotated in this corpus are morphological ones, word forms, lemmas, syntactical features, named entities and so on. In contrast to the TüBa-D/Z corpus which is extracted from newspaper articles, the TüBa-D/S corpus is extracted from dialogues which have been manually transliterated. Finally, the TüBa-D/D corpus is based on historical documents. The TüBa-D/S corpus contains 38.000 sentences and 360.000 words, and the TüBa-D/D corpus contains about $260.000.000$ words from $12.000.000$ sentences.

4.1.2 Graphical Access of Treebanks

[Lezius, 2002] present a system for accessing treebanks. The system called *TiGerSearch* can be seen as a search engine for treebanks which is able to load treebanks and to

retrieve results for particular requests for the given treebank. The graphical user interface of *TiGerSearch* easily lets novice users formulate requests by "drawing" linguistic patterns they are interested in. The formulated linguistic request is used to crawl the complete treebank delivering the patterns that match the request. These patterns are manifold: a user, for instance, might be interested in particular grammatical patterns which are represented as concrete subtrees of the treebank. These subtrees can – of course – be represented in a graph view. Another request might be to find all collocated nouns and verbs. The results can be represented by all particular verb-noun-pairs found in the treebank.

4.2 DWDS – An exemplary lexicon for German

DWDS is the acronym for *Digitales Wörterbuch der Deutschen Sprache* (digital lexicon of German language). The DWDS is a scientific project which has the goal to give an exhaustive overview of the German language of the present and the past. On the one hand the lexicon shall deliver a scientific definition for every requested word, and on the other hand exhaustive contextual information concerning the requested word shall be presented. The project is based on a set of lexicons and annotated corpora. The used corpora are annotated following the *Guidelines for Electronic Text Encoding and Interchange* [Burnard and Bauman, 2007]. The corpora are linguistically annotated which means that the sentences are extracted, the tokens are extracted from the sentences and the tokens are morphologically tagged.

4.2.1 Morphological Analysis

The morphologic analysis are done by the approach of [Geyken and Hanneforth, 2006] which is based on finite state transducers. For the morphological analysis two approaches are usable. The first one is based on full-form lexicons which directly deliver the word form for a requested word. The other approach uses lexicons of stems together with rules for decomposition and derivation. It is not reasonable to use full-form lexicons for the German language because of the productive compounding of that language. Most morphological analyzers for German therefore are based on the second approach. In contrast to the usually used two-level-morphologies like in Gertwol [Haapalainen and Majorin, 1994] the method presented by [Geyken and Hanneforth, 2006] uses concatenation mechanisms. The presented concatenation mechanism is based on a strategy which uses different weights for different segmentation boundaries choosing the solution with lowest weight as the most probable one. If, for instance, a word is found directly in the lexicon, it will get weight 0. If it is a composition of two stems found in the lexicons it will get higher weights if the boundary of the two stems is a strong composition boundary in contrast to being only a weak composition boundary. This morphologic analysis approach has a recognition rate of 99.1% on the archive of the German newspaper "Die Zeit" and of 98.2% on the central corpus of DWDS.

4.2.2 Part-of-speech tagging

The PoS tagging of the tokens in DWDS is done using the method of [Jurish, 2003] which is a two-fold approach. The approach combines a morphological analysis like the one of [Geyken and Hanneforth, 2006] with a hidden Markov model (HMM [Rabiner, 1989]). The hidden Markov model takes the formerly extracted morphological tags into account for the inference mechanism to predict the most probable PoS tags.

4.2.3 Dependency parsing

The method for syntactic tagging used in the DWDS system is based on the work of [Didakowski, 2007] combining chunking and syntactic tagging. The combination is actually done by incorporating 5 cascaded weighted finite state transducers: the first step is a basic morphological analysis. The second step performs the chunking, and the third step creates local dependency structures. The two last steps finally disambiguate the head functions of embedded and main clauses.

4.2.4 The DWDS-website

DWDS is accessible for free use via `http://www.DWDS.de`. If a request for a particular word is performed the GUI shown in Figure 4.3 will be presented. The definition of the basic form of the word is presented. Additionally, a thesaurus is presented which delivers alternatives for the requested word. A so called profile of the requested word presents the statistics for the word in the meaning of the morphological forms it occurs in. An etymological dictionary is requested as well as the newspaper archive of "Die Zeit" and "Die Zeit online". Especially requesting the newspaper archives delivers contextual information concerning the requested word in its currently used form.

4.2.5 German Research Groups on *Information Extraction*

Beside the DWDS project there are many other research groups focusing also on linguistic research. We will exemplarily announce some of the most famous other German groups which are working on linguistic analysis. These analysis more or less directly can be used for *Information Extraction* purposes:

- CLARIN-D
  CLARIN-D[1] is a research project being funded by the German government. It is a collaboration of different research groups on linguistic analysis which aims on developing an integrated, interoperable and scalable framework combining the resources and analysis of all partners.

- Center for Digital Humanities
  The "Trierer Kompetenzzentrum"[2] is an excellence initiative of the university

[1] `http://clarin-d.net`
[2] `http://kompetenzzentrum.uni-trier.de`

Figure 4.3: The result presented for a requested word on the DWDS website

of Trier which has the goal to combine the research interests of computer scientists and humanists. Particular fields of research are digitization, analysis and indexing of writings which needs *Information Extraction* methods.

- Jena University Language & Information Engineering (Julie) Lab
  The Julie Lab in Jena[3] is a research center working on nearly all areas of *Information Extraction*.

- FIRST[4]
  The "FIRST - Large scale information extraction and integration infrastructure for supporting financial decision making"-project is an EU funded project at the

[3] http://www.julielab.de
[4] http://www.uni-goettingen.de/de/209829.html

University of Göttingen which aims at supporting decision makers in the financial domain.

- Stratosphere[5]
  "Stratosphere - Information Management on the Cloud" is a group of researchers funded by the German science association (DFG). The project aims at the management of information in cloud-computing environments and it also contains *Information Extraction* relevant sub-projects.

- UKP Darmstadt[6]
  The Ubiquitous Knowledge Processing (UKP) Lab mostly focuses on research on natural language processing, semantic information management and knowledge discovery in the web.

4.2.6 Lexical nets

[Hamp and Feldweg, 1997] are presenting GermaNet which is a lexical-semantic net for the German language. GermaNet is comparable to WordNet [Miller et al., 1990] which is a lexical-semantic net for English. These lexical-semantic nets are comparable to ontologies by the presentation of particular relations between the words of the certain language. A relation between words in GermaNet is defining semantical information concerning the words. Exemplary relations are *synonymy*, *antonym*, *metonymy*, *hyponymy*, and so on. These lexicons can directly be helpful for the analysis of tokens in case of *Named Entity Recognition*, for instance. If a token which has to be classified is part of a *hyponomy*-relation with a country, the token will probably be a city or a territory. This makes WordNets a good resource for helpful features in *Information Extraction*.

DWDS also internally uses a lexical net called LexikoNet [Geyken and Schrader, 2006]. LexikoNet only contains nouns. 75.000 nouns are put into 1.200 concepts containing, for instance, *instruments*, *sports teams*, *human groups*, and so on.

[Henrich and Hinrichs, 2010] developed an editing framework for GermaNet called GernEdiT which provides possibilities for the easy and intuitive manipulation of GermaNet data. The framework works on a relational dataset containing the GermaNet data. Unfortunately, it cannot be directly used for other lexical nets because the data of those nets has to be transferred to a database first and afterwards the framework would have to be adjusted to handle the language specific structures of that particular lexcial net.

4.3 Summary

We presented the usage of linguistic resources for *Information Extraction*. In contrast to *orthographic* or *semantic* features the use of *morphological* features require exhaus-

[5] http://www.stratosphere.eu/
[6] http://www.ukp.tu-darmstadt.de/

tive preparatory work in order to generate the lexicons or to construct the analyzing approach. If that preparatory work finally is done, the morphological features will deliver useful and rich semantic knowledge.

We presented annotated corpora and lexicons based on linguistic knowledge in particular. Annotated corpora can be used as a gold-standard for supervised learning methods for the creation of parsers, PoS-taggers or NER-systems. These corpora are called treebanks, and one well-known treebank is the Penn Treebank we presented in detail in this chapter. Linguistic lexicons contain linguistic knowledge for a huge amount of words which are either accessible via an index or based on ad-hoc morphological analysis. We present the German online lexicon DWDS as an example. The specialty of this lexicon is that due to the grammar rules of German it is more complex than a lexicon for English. For some analysis it is not useful to store the information in the lexicon but to do the analysis ad-hoc during runtime.

We have shown in addition that certain languages like German require particular analyzing systems which sometimes are not available for those languages.

Chapter 5

Relation Extraction

In this chapter we will define the task of *Relation Extraction* as it is used in this work. After a formal and textual definition of *Relation Extraction* in Section 5.1 we will show the typical feature space which is used for *Relation Extraction* in Section 5.2. Two application domains in which *Relation Extraction* is applied are shown in Section 5.3. We present the broad area of kernels which are used for *Relation Extraction* in Section 5.4. The specialized tree kernels which are used to access tree-structured attribute values are presented in Section 5.5. In contrast to using different tree kernels it is possible to convert the tree-structures used by them. Methods based on such manipulation of trees are presented in Section 5.6. The datasets we use for *Relation Extraction* are presented in Section 8.2.1. An intuitive approach for the visualization of (extracted) relations is presented in Section 8.2.3. Section 8.2.2 contains the descriptions and the results of the experiments we made for *Relation Extraction*. Finally, we subsume this chapter in Section 5.7.

5.1 Definition

The task of *Relation Extraction* is a subsequent step of named entity recognition. If the task of finding the entities is sufficiently done, one can look for relations between these entities. The scientific field of finding relations between entities has become popular since the ACE-tasks. Especially the relation detection and classification (RDC) task in 2004 [Doddington et al., 2004] has gained much attraction in the scientific community.

Following the definition presented in the ACE RDC task [LDC, 2004a] we specify a relation as a valid combination of two entities being mentioned in the same sentence and being related to each other. Relations can be symmetric or asymmetric. In [Had et al., 2009] we defined the schema of i relations in a sentence s as follows:

Definition 19. Relation candidates R_i *in a sentence* s*:*

$$R_i(Sentence\ s) =< Type_m \in relationtypes,$$
$$(Argument_1 \in entities_s, Argument_2 \in entities_s) >$$

where $entities_s$ is the set of entities contained in the current sentence, and $relationtypes$ is the set of possible relations in the corpus.

To build pairs of entities the entities in a sentence or document first of all have to be detected. Making errors in this first step will cause errors in the following steps, too. During the creation of entity pairs one has to take care of the possible approaches to create these pairs. A trivial approach would be to create all pairs which will result in a sparse amount of correct relations in comparison to the incorrect (*negative*) ones. Therefore, one should create only those pairs which are defined. A relation-type that is only defined to contain arguments of type *Person*, for instance, should not be used in a creation process of a relation candidate containing other entities than *Person*. This proceeding would create a smaller number of *negative* relation candidates. In addition, although some combinations are defined by the relation-types, they never appear in the training set. It is possible to create only those relation-types which are present in the training set to still lower the number of *negative* relation-candidates.

After creating all possible pairs of entities for *Relation Extraction*, machine learning techniques can be used to create models for distinguishing between correctly and incorrectly related entities. Multiple machine learning techniques could be used for this task, but the techniques used in most publications are SVMs.

5.2 Feature space

The selection of the feature space for *Relation Extraction* is crucial because the usage of a great amount of features is possible. In Section 5.1 it became clear that first-of-all the original text – containing the marked entity mentions – has to be converted into examples. These examples have to be enriched by several features which are slightly different from the features we presented for NER in Section 3.2. The examples are extracted from a sentence containing several marked entity mentions. Each example contains information about the two entities it was created for. The other entity mentions contained in the sentence can be used as features, too. In addition to the information offered by the entity mentions, the sentence itself offers many pieces of information which can and should be used as features for future processing. The relative location of the used informational units – words or entities – is very useful. An entity mention which is located between the two entities building the relation, for instance, might offer different informational contents than an entity mention located outside of the part of the sentence bounded by the two relation-entities. Most publications concerning *Relation Extraction* are using similar feature sets. Nevertheless, some features are only used in particular publications. We will present a list of features which are used in different publications.

[Zhao and Grishman, 2005], for instance, used the features presented in Table 5.1. A $token$ in this context consists of three features concerning a particular word: $word$, pos and $base$. $word$ is the particular word itself. pos is the part of speech tag of the information available via $word$, and $base$ is the base form of $word$. Additionally,

Name	Description
arg_1	information about the first entity
arg_2	information about the second entity
seq	token vector of the sequence of words between first and second entity
$link$	token vector extracted from seq by utilizing the parse tree of the sentence
$path$	the dependency path connecting first and second entity of the dependency parse tree

Table 5.1: Features used for *Relation Extraction* by [Zhao and Grishman, 2005]

dependencies, which are extracted of the dependency tree (see Section 2.1.4), can be represented in $token$ features by storing the vector of all dependency arcs (arc) in the token for a particular word. An arc contains the word w, the word dw which is connected by a dependency arc of a dependency parse tree, the role label $label$ and the direction e of the dependency arc. The pieces of information about the entities contained in arg_1 and arg_2 are the (dependency-)$token$, $type$, $subtype$ and $mtype$ of the entity. Although [Zhao and Grishman, 2005] use the dependency tree structure of the corresponding sentence for the construction of the feature set, all the features still are *flat* ones.

[Bunescu and Mooney, 2005] also used the dependency of the particular sentences to extract *flat* features. They took the shortest dependency path between the two particular entities of the relation candidate and built a feature-vector out of it. The nodes of the dependency path which are particular words of the sentence are represented as a small feature vector containing the word itself and additional available information concerning the word like the part-of-speech tag and the named entity class. The dependencies (edges of the dependency tree) are represented as a binary feature indicating if a particular word x_i is depending from word x_{i+1} or if word x_{i+1} is depending from word x_i.

[Bunescu and Mooney, 2006] presented an approach completely waiving the structural information from trees. Using a generalized version of [Lodhi et al., 2002] they used the sequences of words before, between and after the relevant entities of the relation candidate for constructing the feature vector.

[Zhou et al., 2005] published a great set of *flat* features. They extracted several features concerning the words of the entity mentions building the relation candidate. As an example they extracted the first word before the first entity mention, they extracted the first word after the second entity mention, and so on. In sum they extracted 14 features concerning that word level. Additionally, they extracted the information already encoded in the entity mentions (called entity features). These are the entity types, the combination of the entity types and the mention levels and the combination of the mention levels. Some overlap features are extracted which contain the number of words and

entity mentions between the two entity mentions building the relation candidate. The overlap features also contain features which indicate if the first entity mention contains the second entity mention or vice-versa. In addition, combinations of these overlap features and the entity features are built to generate more effective features. Some features were generated out of base phrase chunking. The dependency and the constituent parse trees were used to generate path features for the relation candidate. Finally, semantic resources like WordNet (see Section 4.2.6) were used to generate features indicating if the entities contain known locations or if the entities indicate a social relationship.

[Zhang et al., 2006] were one of the first who used techniques directly processing features containing tree structures. In contrast to the formerly presented approaches which only converted the *structured* features into *flat* features for future processing, the approach of [Zhang et al., 2006] directly handles the *structured* features. Additionally, they show how to combine *flat* features with *structured* features in order to profit from both types of features.

5.3 Applications

In this section we will show two exemplary applications which benefit from *Relation Extraction*. The first example is text understanding which aims at the automatic extraction of relevant facts from documents in order to semantically understand the document without having to completely read it. The second example is biological knowledge acquisition. Biological knowledge acquisition aims at extracting new pieces of information out of biological documents. These pieces of information especially are interrelations – between genes, for instance – which are available in text documents but not yet published in databases for much easier access. The *Learning Language in Logic* challenge 2005 (LLL05) [Nédellec, 2005] covered exactly that topic.

5.3.1 Opinion Mining

Opinion mining is a field of research which gained much interest over the last years. The task is defined as the automatic extraction of opinions for particular entities. The task is very interesting for many organizations because it can be used to extract opinions or sentiments for certain products sold or produced by companies out of the WWW. The WWW and especially social networks or blogging platforms represent an actual view on the mood and the opinions of the community. The usage of opinion mining techniques therefore can be helpful to gain a current feedback concerning particular products. This feedback can be helpful on the one hand to gain insights of the needs of customers and on the other hand to get information about the drawbacks of a new product which has been just rolled out.

[Jiang et al., 2010] used *Relation Extraction* as a technique for opinion mining. First of all, they extracted two types of entities: *target features* and *sentiment words*. The *target features* are entities for which the opinions or sentiments should be extracted. *Target features* could be products, for example. The *sentiment words* are words or tokens

which are expressing some kind of sentiment. After the extraction of the particular entities the authors are constructing pairs of entities which only contain one *target feature* and one *sentiment word*. Finally, machine learning approaches can be used to classify the pairs in order to get to know if a relation is present between the particular entities or if it is not. In addition, if there is a relation it will be classified. Figure 5.1 shows an exemplary sentence containing two *target features* and two *sentiment words*. Each *target feature* is used in a relation candidate with each *sentiment word*. It becomes obvious that some of the pairs are not related. Especially the tree-structured features which are used by the methods presented in this chapter are helpful in order to extract the correct relations.

The computer is great but the hard drive is disgusting.

Figure 5.1: A sentence containing two *target features* (blue) and two *sentiment words* (red).

5.3.2 Biological Knowledge Acquisition

A vast amount of biological documents are available in the World Wide Web. The publication platform MEDLINE, for instance, contains abstracts and complete journal articles concerning medical and biological scientific research. This platform is accessible via Pubmed[1] which is an information retrieval system enabling users to access MEDLINE. Although Pubmed enables the access of MEDLINE, this access depends the query which is defined by the user. The really interesting task is to extract informational pieces which are still unknown and therefore cannot be formulated in a user query. One of these informational pieces is the extraction of relations between genes or amino-acids, for instance. The extraction of genes and amino-acids equals the task of NER, and after that *Relation Extraction* techniques can be used to extract – former unknown – interrelations between the found entities.

5.4 Kernel Usage

Machine learning methods for *Relation Extraction* are similar to methods used for other machine learning tasks like *Data Mining*, for instance. One major difference is the data preparation which is more complex for *Relation Extraction*. In Section 5.1 we have shown that a document containing entity mentions first of all has to be converted into a set of relation candidates. Additionally, some of the features used for the description of the candidates depend on both of the relevant entities of the relation candidate. Finally, the usage of *structured* features requires either a special preparation to extract a set of *flat* features from the *structured* ones, or specific machine learning methods are needed for the treatment of the *structured* features. State-of-the-art machine learning methods used for various problems are SVMs. In Section 2.4.1 we have shown that SVMs originally only have the ability to handle linearly separable data. The question

[1] `http://www.ncbi.nlm.nih.gov/pubmed`

arises for *structured* data how to make these data linearly separable. The solution is to use special kernel functions.

In this section we will focus on possible kernel functions used for *Relation Extraction*. In Section 5.4.1 we present a linear kernel which can be used to handle nominal features. The subsequence kernel of [Bunescu and Mooney, 2006] is presented in Section 5.4.2. The shortest path dependency kernel of [Bunescu and Mooney, 2005] is presented in Section 5.4.3. These kernel do not directly process *structured* features. The convolution kernel [Haussler, 1999] can handle discrete structures. This kernel and the extensions for this kernel are presented in Section 5.4.4. The tree kernels which have been explicitly developed for the handling of *tree-structured* features are presented in the following section (Section 5.5). In order to combine the *structured* and *flat* features the composite kernels can be used. These kernels combine the outputs of different kernels for the generation of one kernel result and they are presented in Section 5.4.5.

5.4.1 Linear Kernels

At the beginning of automatic *Relation Extraction* only flat features have been used for classifying relation candidates. A huge set of flat features has been created containing features according to the particular relation candidate. [Zhou et al., 2005] presented a set of flat features which is used as a kind of benchmark feature set on *Relation Extraction*data. Many of the features contained in this set are nominal features. The use of SVMs which are well-suited for the handling of large feature vectors is not trivial for nominal features because the SVM is used to find the optimal separating hyperplane in the (feature-)vector space of the example set (see Section 2.4.1). Internally the dot-product is used on the particular attribute values. Nominal values usually are presented as numerical values representing an index entry. This will lead to erroneous results because nominal values represented by index numbers which are nearby will be seen as more similar than values represented by index numbers which are more distant. A kernel-function (see Section 2.4.1) is needed to handle the nominal feature values. Given examples $\mathbf{x}$ and $\mathbf{x}'$ containing nominal feature values, the following linear kernel can be used for SVMs to handle the values:

Definition 20. *A* linear kernel*:*

$$k(\mathbf{x}, \mathbf{x}') = \sum_i c(x_i, x'_i) \tag{5.1}$$

where $c(x_i, x'_i) = 1$ *if the* i*-th feature of* $\mathbf{x}$ *has the same value as the* i*-th feature of* $\mathbf{x}'$*, otherwise* $c(x_i, x'_i) = 0$*.*

This kernel can be used to process nominal features using an SVM. The features can be used directly and do not have to be converted into numerical values.

5.4.2 Subsequence Kernel

[Bunescu and Mooney, 2006] define a relation candidate constructed out of two entity mentions by using the sequences of words before, between and after the entity mentions. In addition, another sequence covers the entity mentions and the words between them. Figure 5.2 shows a relation candidate of a sentence s. Not only the sequence of words but also sequences of other attributes – like POS tags – can be used for calculating the kernel function.

The more tokens the particular sequences of two compared examples have in common

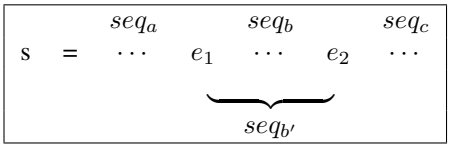


Figure 5.2: A relation candidate as used by the subsequence kernel by [Bunescu and Mooney, 2006]

the greater is the resulting kernel value. This kernel is not comparable to the following kernels which all of them access structural information for *Relation Extraction*, and it is not described in detail because of this reason.

5.4.3 Shortest Path Dependency Kernel

[Bunescu and Mooney, 2005] are using the informational pieces given by the dependency tree of a sentence and convert them into *flat* features. Figure 5.3 shows the sentence already presented by the dependency tree in Figure 2.2. The entities contained in this sentence are printed bold. Additionally, the dependencies between the entities are needed for the kernel calculation. These are indicated by the arrows. To extract relation candidates out of this sentence the pairs of entities are created. These pairs are enriched by the shortest dependency path between the particular entities. **Felix** and **New York**, for instance, are connected by the dependency relations from **Felix** to *went* and from **New York** to *went*. The direction of a dependency relation is crucial because it indicates which token is depending from which other token. The three resulting relation

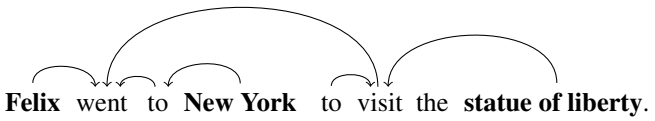


Figure 5.3: A sentence containing dependency relations

candidates and the corresponding shortest dependency paths are presented in Table 5.2.

The kernel function on two such sequences $\mathbf{x}$ and $\mathbf{x}'$ is calculated as presented in

Relation candidate	Shortest dependency path
<**Felix** , **New York**>	**Felix** $\rightarrow$ went $\leftarrow$ to $\leftarrow$ **New York**
<**Felix** , **statue of liberty**>	**Felix** $\rightarrow$ went $\leftarrow$ visit $\leftarrow$ **statue of liberty**
<**New York** , **statue of liberty**>	**New York** $\leftarrow$ to $\leftarrow$ went $\leftarrow$ visit $\leftarrow$ **statue of liberty**

Table 5.2: Relation candidates and the corresponding shortest dependency paths

equation (5.2).

$$k(\mathbf{x}, \mathbf{x}') = \begin{cases} \prod_{i=1}^{n} c(x_i, x_i') & m = n \\ 0 & m \neq n \end{cases} \tag{5.2}$$

where m and n are the length of the token-sequences $\mathbf{x}$ and $\mathbf{x}'$. It follows that the calculated kernel value for non-equal token-sequences is 0. If the sequences are of equal length the kernel value will be calculated by the product of function $c(\cdot, \cdot)$ applied on each index of the token-sequences. $c(\cdot, \cdot)$ calculates the entries the sequences have in common at the particular index. To give the reader an example we use the three dependency paths shown in Table 5.2 as a dataset to be processed by $k(\cdot, \cdot)$. This dataset is shown in Table 5.3.

Only the first and the second sequences are of equal length (length = 7). The ker-

ID	Shortest dependency path sequence
x_1	[**Felix**, $\rightarrow$, went, $\leftarrow$, to, $\leftarrow$, **New York**]
x_2	[**Felix**, $\rightarrow$, went, $\leftarrow$, visit, $\leftarrow$, **statue of liberty**]
x_3	[**New York**, $\leftarrow$, to, $\leftarrow$, went, $\leftarrow$, visit, $\leftarrow$, **statue of liberty**]

Table 5.3: Relation candidates and the corresponding shortest dependency paths stored as an example set

nel values for the first and the third example and the second and the third example both are 0 ($k(\mathbf{x}_1, \mathbf{x}_3)$ = $k(\mathbf{x}_2, \mathbf{x}_3) = 0$) because of the reason that the particular sequences are of different length (7 and 9). The kernel value $k(\mathbf{x}_1, \mathbf{x}_2)$ is calculated by multiplying the numbers of equal parts of the sequences.

$$k(\mathbf{x}_1, \mathbf{x}_2) = 1 \cdot 1 \cdot 1 \cdot 1 \cdot 0 \cdot 1 \cdot 0 = 0 \tag{5.3}$$

Equation (5.3) shows the calculation of the kernel value for $\mathbf{x}_1$ and $\mathbf{x}_2$. Although both sequences seem to be similar, the kernel value still is 0 like for the other two kernel calculations. The reason for this is the pretty simple feature set we are using. The entries of the sequences normally are enriched by additional features like the POS-tags and so on. Nevertheless, just one entry x_i in the sequence having nothing in common with x_i' will result in a kernel value of 0. This fact in combination with the fact that a kernel value is 0 if the length of both sequences are different, makes this kernel not very general.

5.4.4 Convolution Kernels

Although the approaches presented by [Zhao and Grishman, 2005] and [Zhou et al., 2005] already used syntactic structures, those structures were shattered and have been used as flat features. Such processing is tedious and will be avoided if the syntactic structure is used in a direct way. [Haussler, 1999] presented a kernel function processing discrete structures. Instead of shattering the syntactic structures explicitly, the kernel function implicitly converts the structures and calculates the scalar product.

Definition 21. *Suppose $x \in X$ is a composite structure and $\vec{x} = x_1, ..., x_p$ are its parts, where each $x_i \in X_i$ for $i = 1, ..., p$ and all X_i are countable sets. The relation $R(\vec{x}, x)$ is true, iff $x_1, ..., x_p$ are all parts of x. As a special case, X being the set of all p-degree ordered, rooted trees and $X_1 = ... = X_p = X$, the relation R can be used iteratively to define more complex structures in X.*
Given $x, z \in X$ and $\vec{x} = x_1, ..., x_p$, $\vec{z} = z_1, ..., z_p$ and a kernel k_i for X_i measuring the similarity $k_i(x_i, z_i)$, then the similarity $k(x, z)$ is defined as the following generalized convolution

$$k(x, z) = \sum_{\{\vec{x}|R(\vec{x},x)\}} \sum_{\{\vec{z}|R(\vec{z},z)\}} \prod_{i=1}^{p} k_i(x_i, z_i) \tag{5.4}$$

[Haussler, 1999]p.5f

Particular kernels which are based on the kernel defined by Haussler are presented in Section 5.5.

5.4.5 Composite Kernels

A recent extension is the combination of convolution kernels on the one hand with linear kernels on the other hand, resulting in a so called composite kernel [Zhang et al., 2006]. The kernel presented by [Haussler, 1999] is closed under product and addition implying that a new kernel can be created by combining this kernel with another valid kernel function. The *composite kernel* is defined as follows:

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') \circ k_2(\mathbf{x}, \mathbf{x}') \tag{5.5}$$

[Zhang et al., 2006] present two possible ways to combine the convolution and the linear kernel. Eq. (5.6) shows the linear combination whereas eq. (5.7) presents the

polynomial expansion.

$$k(\mathbf{x}, \mathbf{x}') = \alpha \cdot k_L(\mathbf{x}, \mathbf{x}') + (1 - \alpha) \cdot k_T(\mathbf{x}, \mathbf{x}') \tag{5.6}$$

$$k(\mathbf{x}, \mathbf{x}') = \alpha \cdot k_L^P(\mathbf{x}, \mathbf{x}') + (1 - \alpha) \cdot k_T(\mathbf{x}, \mathbf{x}') \tag{5.7}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are *relation candidates* consisting of flat features on the one hand and structured features on the other hand. The linear kernel k_L just processes the flat features, and the convolution kernel k_T processes the structured information. $k_L^P(\mathbf{x}, \mathbf{x}')$ is the polynomial expansion of $k_L(\mathbf{x}, \mathbf{x}')$ with degree d resulting in $k_L^P(\mathbf{x}, \mathbf{x}') = (k_L(\mathbf{x}, \mathbf{x}') + 1)^d$. The α value can be used to adjust the influence of each kernel. In addition to the combination of both kernels each kernel is normalized beforehand. A normalized kernel value $k'(\mathbf{x}, \mathbf{x}')$ of a kernel $k(\mathbf{x}, \mathbf{x}')$ is defined by:

$$k'(\mathbf{x}, \mathbf{x}') = \frac{k(\mathbf{x}, \mathbf{x}')}{\sqrt{k(\mathbf{x}, \mathbf{x})k(\mathbf{x}', \mathbf{x}')}} \tag{5.8}$$

5.5 Tree Kernels

In this section we will present tree kernels which are extensions of the approach presented by [Haussler, 1999] (see Section 5.4.4).

5.5.1 Tree Kernel by Collins and Duffy

[Collins and Duffy, 2001] were the first using parse tree information for the classification of relations. Their work is based on the convolution kernel presented by [Haussler, 1999] for discrete structures. To make the structural information of the parse tree applicable by a machine learning technique a kernel for the comparison of two parse trees is used. This kernel compares two parse trees and delivers a real-valued number which can be used by machine learning techniques. During kernel calculation, each subtree of the first tree is compared with each other subtree of the second tree, therefore the trees are implicitly represented as a vector of subtrees. The transformation Φ of the tree structure into a vector space of subtrees is given by:

$$\Phi(T) = (subtree_1(T), ..., subtree_m(T)) \tag{5.9}$$

where $subtree_i(T)$ is the number the i-th subtree occurs in tree T.
For this representation the m available subtrees of all trees (of the training set) have to be extracted. The number of subtrees (from $subtree_1, ..., subtree_m$) both trees have in common is summed up during kernel calculation. If the trees are present in the style of a vector like in equation (5.9) the calculation of the sum of subtrees is very simple. Especially for bigger trees the transformation of the trees into the vector representation is infeasible and should be avoided.

[Collins and Duffy, 2001] define a tree kernel working directly on the tree structures. The transformation into the vector space of all subtrees and the calculation of the sum of equal subtrees the two trees have in common is calculated implicitly as presented in Definition 22.

Definition 22. *The* tree kernel *defined by [Collins and Duffy, 2001] computes the following:*

$$k(T_1, T_2) =< \Phi(T_1), \Phi(T_2) > \tag{5.10}$$

$$\Phi_i(T_1) = \sum_{n_1 \in N_1} I_{subtree_i}(n_1) \tag{5.11}$$

where N_j is the set of nodes of tree T_j. $I_{subtree_i}$ is a function defined as to deliver 1 if the i-th subtree (in the vector of all subtrees of the training set) is rooted in node n_k, and 0 otherwise.

Eq. (5.11) equals eq. (5.9). It follows that

$$k(T_1, T_2) = \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \sum_i I_{subtree_i}(n_1) I_{subtree_i}(n_2) \tag{5.12}$$

$$= \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} C(n_1, n_2) \tag{5.13}$$

$C(n_1, n_2)$ represents the number of common subtrees for the two subtrees rooted at node n_1 and n_2. The number of common subtrees finally represents a syntactic similarity measure and can be calculated recursively starting at the leaf nodes in $\mathcal{O}(|N_1| \cdot |N_2|)$. In the following we will call this kernel the *Quadratic Tree Kernel* (QTK).

During the recursive calculation three cases are being respected:

1. If the productions at n_1 and n_2 are different, $C(n_1, n_2) = 0$
2. If the productions at n_1 and n_2 are the same and if n_1 and n_2 are preterminals, $C(n_1, n_2) = 1$
3. Else if the productions at n_1 and n_2 are the same and if n_1 and n_2 are not preterminals, $C(n_1, n_2) = \prod_j (1 + C(n_{1_j}, n_{2_j}))$, where n_{1_j} is the j-th children of n_1 (in a uniform manner for n_2).

Especially trees containing many nodes will result in large kernel-outputs which makes further processing by machine learning techniques complicated. To overcome that problem, [Collins and Duffy, 2001] present two possibilities:

- Normalization
- Scaling

Every kernel output can be normalized by using following equation:

$$k'(T_1, T_2) = \frac{k(T_1, T_2)}{\sqrt[2]{k(T_1, T_1) * k(T_2, T_2)}} \tag{5.14}$$

Unfortunately, calculating the kernel-outcome is computationally expensive, therefore it should be avoided to calculate this outcome multiple times like this is done during

the normalization. [Collins and Duffy, 2001] established a scaling factor $0 < \lambda <= 1$ which is used in two of the three cases for the recursive kernel calculation:

1. If the productions at n_1 and n_2 are different, $C(n_1, n_2) = 0$
2. If the productions at n_1 and n_2 are the same and if n_1 and n_2 are preterminals, $C(n_1, n_2) = \lambda$
3. Else if the productions at n_1 and n_2 are the same and if n_1 and n_2 are not preterminals, $C(n_1, n_2) = \lambda * \prod_j (1 + C(n_{1_j}, n_{2_j}))$.

For the calculation of $k(T_1, T_2)$ a total runtime of $\mathcal{O}((|N_1| \cdot |N_2|)^2)$ is needed, as for each node $n_1 \in N_1$ a combination with every node $n_2 \in N_2$ is performed to calculate $C(n_1, n_2)$ which has a runtime of $\mathcal{O}(|N_1| \cdot |N_2|)$.

5.5.2 Fast Tree Kernels

Moschitti ([Moschitti, 2006a, Moschitti, 2006b]) is presenting an efficient calculation of equation (5.13). Instead of visiting every node of both trees T_1 and T_2 he is building the pairs of nodes n_1 and n_2 for which the result of $C(n_1, n_2)$ is not 0. This node pair set NP is defined as:

$$NP = < n_1, n_2 > \in N_1 \times N_2 : p(n_1) = p(n_2) \tag{5.15}$$

If the productions of two nodes are not equal, $C(n_1, n_2) = 0$. The node pair set NP contains all node pairs which are relevant for the kernel calculation because these pairs might deliver results which are not 0. The Algorithm 2 shows the collecting mechanism for the relevant pairs needed to calculate the tree kernel outcome.

The trees are represented as ordered production lists like it is described in Section 6.2.1. The productions of each tree are stored in a list and the lists are ordered. Two lists are processed at the time. For each list an index is available to access particular productions. These indexes are initially set to the beginning of the lists. If the contained productions are equal a new node pair will be created and stored for returning it later. Otherwise, if one production is of higher order, according to the particular ordering, the other list's index is incremented.

Although this approach may require only $\mathcal{O}(|N_1| + |N_2|)$, it needs $|N_1||N_2|$ cycles in the worst case, which will happen if each of both trees just consists of one production type. The sorting of the production-lists of each tree requires $\mathcal{O}(|N_1|log(|N_1|))$. But this sorting can be done once during preprocessing. This results in a total runtime of $\mathcal{O}((|N_1| + |N_2|)|N_1||N_2|)$ because $C(n_1, n_2)$ has to be calculated for each of the created node pairs $< n_1, n_2 > \in NP$.

Because of the faster runtime in practical – compared to the tree kernel presented by [Collins and Duffy, 2001] – this tree kernel is called *Fast Tree Kernel* (FTK).

Algorithm 2 Collecting the relevant node pairs. ([Moschitti, 2006b], p. 3)

```
1:  Input: List L_1, L_2
2:  procedure EVALUATE NODE PAIR SET NP
3:      L_1 = T_1.ordered_list;
4:      L_2 = T_2.ordered_list;
5:      n_1 =extract(L_1);
6:      n_2 =extract(L_2);
7:      while n_1 and n_2 are not null do
8:          if p(n_1) > p(n_2) then
9:              n_2 = extract(L_2);
10:         else if p(n_1) < p(n_2) then
11:             n_2 = extract(L_2);
12:         else
13:             while p(n_1) = p(n_2) do
14:                 while p(n_1) = p(n_2) do
15:                     NP.add(¡n_1, n_2¿);
16:                     n_2 = next(L_2);
17:                 end while
18:                 n_1 = extract(L_1);
19:                 reset(L_2);
20:             end while
21:         end if
22:     end while
23:     return NP;
24: end procedure
```

5.5.3 Approximate Tree Kernels

[Rieck et al., 2010] present the approximate tree kernel which is taking just a subset of nodes into account during the tree kernel calculation. The tree kernels of [Collins and Duffy, 2001] and [Moschitti, 2006a] are quadratic in the sense that the calculation of $C(n_1, n_2)$ has a runtime of $\mathcal{O}(|N_1||N_2|)$. [Rieck et al., 2010] present a selection function $\omega : S \to \{0, 1\}$ on the set of all possible node symbols S which takes the particular nodes into account ($\omega(s) = 1$) for the calculation of the kernel function or not ($\omega(s) = 0$).

Definition 23. *Given a selection function $\omega : S \to \{0, 1\}$, the approximate tree kernel is defined as*

$$k_\omega(T_1, T_2) = \sum_{s \in S} \omega(s) \sum_{\substack{n_1 \in N_1 \\ l(n_1)=s}} \sum_{\substack{n_2 \in N_2 \\ l(n_2)=s}} C(n_1, n_2)$$

$l(n_i)$ is the extraction of the symbol which is assigned to node n_i. Although $C(n_1, n_2)$ is equal to the function already known from the former tree kernels, it contains the special case $C(n_1, n_2) = 0$, if $\omega(l(n_1)) = 0$ or $\omega(l(n_2)) = 0$.

The crucial part of this approach is to define the symbols $s \in S$ which are resulting in $\omega(s) = 1$ and the ones which result in $\omega(s) = 0$. The more symbols s result in $\omega(s) = 0$ the less functions $C(n_1, n_2)$ have to be calculated which leads to a shorter runtime. Additionally, the classification performance has to stay as good as without selecting only a subset of symbols for calculation. To find the optimal setting for ω, [Rieck et al., 2010] present the optimization shown in Optimization 1.

Optimization 1.

$$\omega^* = \underset{\omega \in [0,1]^{|S|}}{argmax} \sum_{\substack{i,j=1 \\ i \neq j}}^{n} y_i y_j \sum_{s \in S} \omega(s) \sum_{\substack{n_1 \in N_1 \\ l(n_1)=s}} \sum_{\substack{n_2 \in N_2 \\ l(n_2)=s}} C(n_1, n_2) \tag{5.16}$$

subject to $\sum_{s \in S} \omega(s) \leq N$.

N is a parameter which is the upper bound of the number of symbols which should only be used for the kernel calculation.

The predictive performance of the approximate tree kernel is comparable to the performance achieved by the tree kernels working on the complete set of symbols. The approximate tree kernel in contrast is significantly faster than the other mentioned tree kernels. This fact especially makes approximate tree kernels useful in domains in which the tree-structured values contain many nodes like *HTML*-documents, for instance.

5.5.4 Context-sensitive Tree Kernels

[Zhang et al., 2006] presented pruning methods for syntactic parse trees resulting in better classification performance (see Section 5.6). Unfortunately, these pruning methods cannot tackle certain relation types. Those particular relation types are indicated by a related verb. Figure 8.4 shows such relation for the German language. Especially the German language contains more of these relations than the English language. Nevertheless, also the ACE corpus containing English documents holds such relations. [Zhou et al., 2007] used syntactic features and embedded those directly into the parse tree.

[Zhou and Zhu, 2011] are presenting a context-sensitive tree kernel which is working in an approximate way comparable to the approximate tree kernel presented by [Rieck et al., 2010].

5.5.5 Related Tree Kernels

[Zhou et al., 2007] extended the FTK kernel to become context sensitive by looking back at the path above the ancestors of the root node of each subtree. The left side of the production rule is taking into account $m - 1$ steps towards the root. The kernel calculation itself sums up the calculations for each set of production rules created for

$1 \ldots m$. In the special case $m = 1$ the kernel result is the same as with the non context-sensitive kernel. Figure 5.4 shows the resulting production rules for different m values.

The kernel calculation is presented in equation (5.17).

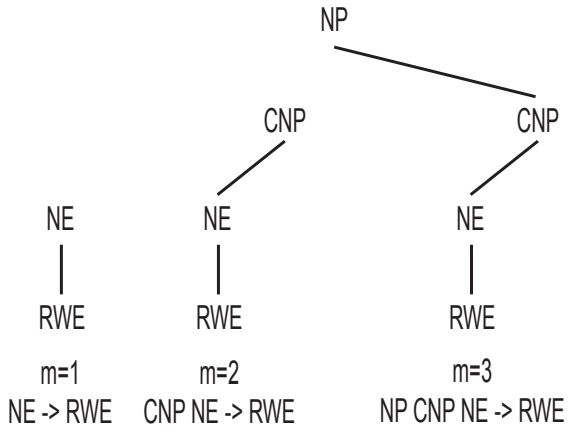


Figure 5.4: The production rules of a particular subtree root node *(*NE) for different m values ([Had et al., 2009], p. 5)

$$k_C(T_1, T_2) = \sum_{i=1}^{m} \sum_{n_1^i \in N_1^i} \sum_{n_2^i \in N_2^i} C(n_1^i, n_2^i), \tag{5.17}$$

where m is the number of ancestor nodes to consider. n_1^i is a node of a tree with a production rule over i ancestors. n_1 is the root node of the context free subtree. N_1^i is the set of all nodes with their production rules over i ancestors.

In addition to just using the composite kernel on the full parse tree of a sentence, [Zhang et al., 2006] examined several ways of pruning the parse tree in order to get differently shaped subtrees on which the tree kernel performs as well or better as on the full tree. They showed that the shortest path-enclosed tree (*PT*) which is the minimal subtree containing the two entities of a *relation candidate* performs best for the ACE 2003 and 2004 RDC corpus (see Section 5.6).

[Zelenko et al., 2003] present a slightly different kernel for *Relation Extraction* compared to the kernel by [Collins and Duffy, 2001]. Their kernel is processing shallow parse representations of text. Due to the fact that representations contain nodes which consist of several attributes, the kernel has to respect these attributes.

[Reichartz et al., 2010] present another special form of tree kernels by generalizing the original idea of tree kernels for handling typed dependency trees. In typed dependency trees the edges between particular nodes are typed (named). The original tree

kernel presented by [Collins and Duffy, 2001] would not be able to handle these typed edges sufficiently.

[Vishwanathan and Smola, 2003] are presenting a linear-time approach of string kernels which can be used to process the task of tree kernel calculation. Following their approach the trees have to be represented as strings where each opening bracket, the strings in between and the closing bracket represent a subtree. An exemplary sentence given in this form is shown in Figure 7.5. This kernel only is based on the extraction of complete common subtrees. It follows that only the existence of completely common subtrees affect the kernel calculation. The recursive computation of $C(n_1, n_2)$ as it is presented by [Collins and Duffy, 2001] defines $C(n_1, n_2) = \lambda \prod_j (1 + C(n_{1_j}, n_{2_j}))$ for the recursive step. If only the complete subtrees are affecting the kernel value this calculation should be rewritten to $C(n_1, n_2) = \lambda \prod_j (0 + C(n_{1_j}, n_{2_j}))$. This representation leads to a value which will only be not equal 0 if the subtrees rooted in n_1 and n_2 are completely equal. The kernel value of the calculation presented by [Collins and Duffy, 2001] will already be not equal 0 if the productions of n_1 and n_2 are equal but the productions of the children nodes are unequal. [Moschitti, 2006a] calls the kernel presented by [Collins and Duffy, 2001] a *subset-tree* kernel and the one by [Vishwanathan and Smola, 2003] a *subtree* kernel. In order to allow the calculation of both approaches the third step of the recursive calculation of the tree kernel function should be replaced by

$$C(n_1, n_2) = \lambda \prod_j (\sigma + C(n_{1_j}, n_{2_j})), \text{ where } \sigma \in \{0, 1\}.$$

5.6 Preparation of Trees

The greater the amount of nodes in a tree the more complex is the calculation of the kernel function. In addition, it is important that the relation candidate, the parse tree is used for, just covers a small part of the complete sentence, mostly. This allows to prune the used parse tree without loosing information about the embedded relation candidate but having the advantage of smaller complexity. Pruning is the most simple way to create small and informative subtrees as features for *Relation Extraction*. We present the possible known pruning methods in Section 5.6. Although pruned trees already contain many informational units helpful for *Relation Extraction*, these subtrees additionally can be enriched by semantic and syntactic information. The syntactic informational units are containing contextual information extracted from the context outside of the subtree. This latter approach is presented in Section 5.6.2.

5.6.1 Pruning trees

[Zhang et al., 2006] inspected five types of pruning methods:

- Minimum Complete Tree (MCT)
- Path-enclosed Tree (PT)

- Context-Sensitive Path Tree (CPT)
- Flattened Path-enclosed Tree (FPT)
- Flattened CPT Tree (FCPT)

Four of these pruning methods are shown in Figure 5.5.

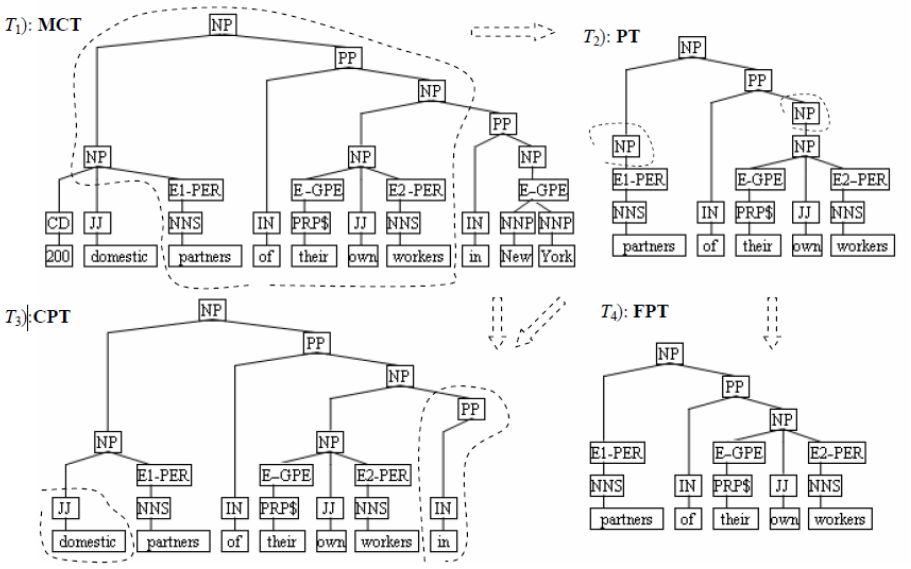


Figure 5.5: Pruning methods as presented by [Zhang et al., 2006]

The *MCT* is the the smallest complete subtree containing both entities. Cutting off every node and production except the path between the entities and the nodes in between will lead to the *PT*. The *CPT* is the same as the *PT* containing one word beside each entity, additionally. Non-terminal-nodes which just have one in- and out-arc are removed to create the *FPT* out of the *PT* and the *CFPT* out of the *CPT*.

During the evaluation of the five pruned trees on the ACE 2003 and 2004 corpora (see Section 6.5.2) it became clear that *MCT* performs worse compared to the other pruned trees. [Zhang et al., 2006] suggest that the reason for this might be the great context which is apparent in this trees compared to the other trees. The best performance was achieved by using the *PT*.

5.6.2 Enriching trees by syntactic and semantic information

[Zhou et al., 2010] present methods to enrich the *PT* by additional syntactic and semantic information. Three steps are performed: first, contextual information as already presented in [Zhou et al., 2007] are embedded in the tree. Second, structural refinements on the tree are performed. Certain parts of the tree are compressed or deleted as they

are not needed for classifying a relation. Figure 5.6 shows one of three structural refinements performed by [Zhou et al., 2010]: the noun-phrase conjunctions are compressed to one noun-phrase which contains the relevant entity E_2. In addition, contextual information is found outside the *PT* which has not been detected during the first step. Third, semantical information is embedded into the tree using the entity information. Various ways to embed these informational units are presented.

[Zhang et al., 2007] generalized the production rules of the parse tree in order to

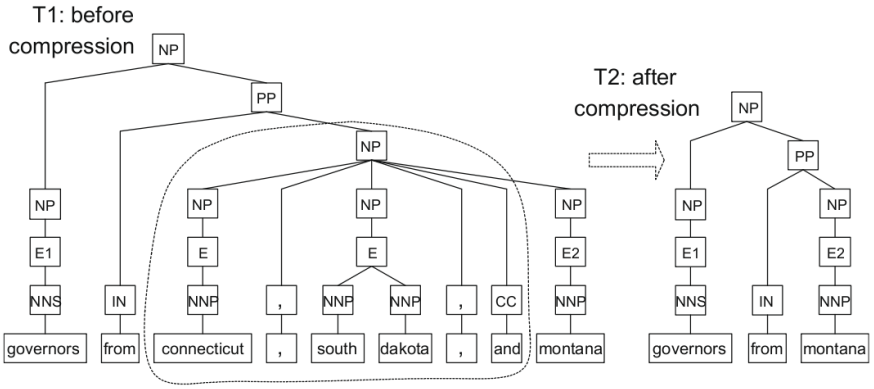


Figure 5.6: Pruning methods as presented by [Zhou et al., 2010]

achieve better performance. The strict decisions of a convolution tree kernel make the kernel returning 'unequal' confronted with two production rules "$NP \rightarrow Det\ Adj\ N$" and "$NP \rightarrow Det\ N$" although they might contain similar terminals ("$NP \rightarrow$ a red car" and "$NP \rightarrow$ a car"). To avoid such behavior they proposed the insertion of optional nodes into production rules to generalize them. Additionally similar part of speech tags in the parse tree can be processed in an equal way – multiplied with a penalty term.
This is a step into the right direction. However, only syntactic variance is handled. Since words carry most of the semantic information, moving them into the tree kernel could well help to generalize in a more semantic way.

Enriching the tree by word stems

In [Had et al., 2009] the usage of state-of-the-art tree kernels has been changed in order to

- First, enrich the feature set for the linear kernel.
- Second, add semantic information into the parse tree.

Informational units indicating a particular relation are spread all over a sentence. The main verb, for example, might occur at different positions of the sequence of words (see Figure 8.4 as an example). [Zhou et al., 2005] presented the usage of contextual flat features for *Relation Extraction*. We generalized that approach by taking the complete bag of words representation (see Section 3.1) into account as additional flat features for the relation candidates. This approach of course is very promising if the sentence only contains one relation candidate.

Our second enhancement concerns the parse tree directly. A particular parse tree of a sentence contained in the merger corpus (see Section 8.2.1) is shown in Figure 8.4. The parse tree contains two entities which are solidly underlined. In addition, the verb which indicates the relation is underlined by a dashed line. Pruning the parse tree and using the path-enclosed tree (PT), for instance, will not work well for this case. Nevertheless, using the complete parse tree without pruning will result in using equal parse trees for every relation candidate of the particular sentence.

The technique presented by [Zhou et al., 2007] to use context-sensitive parse trees is only useful for German datasets if good parsers are available. Our idea was to generalize the information contained in the parse trees by embedding syntactical information extracted from the tokens into the parse trees at different levels. We extracted the word stems for every entity involved in the relation candidates and put these stems into the certain parse trees.

In Figure 5.7 we show three possible ways to embed the word stem information into

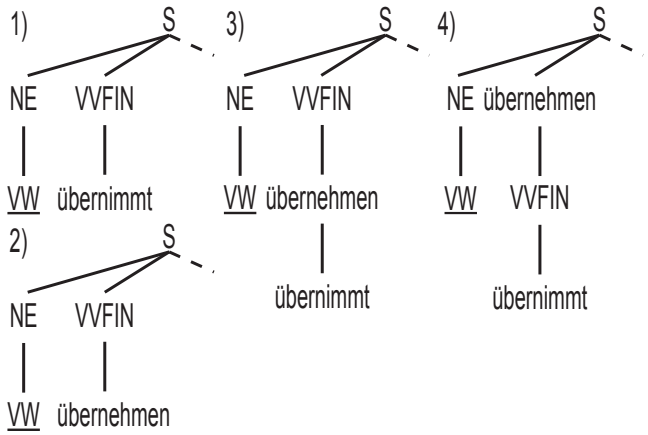


Figure 5.7: Word stems provided at different depth-levels in the parse tree ([Had et al., 2009],p.9)

the parse trees. The first way (Fig. 5.7 2)) is to replace the original tokens at the leaf nodes. The second way (Fig. 5.7 3)) is to generate new preterminal nodes containing the stems, and the third way (Fig. 5.7 4)) is to create an inner node above the pretermi-

nal nodes containing the word stem. Figure 5.7 1) shows the original parse tree. The token 'VW' can not be stemmed and therefore nothing is inserted.

5.7 Summary

In this chapter we have presented the task of *Relation Extraction* in the context of *Information Extraction*.

We defined *Relation Extraction* as a follow-up analysis of named entity recognition. Only already extracted entities are taken into account for further processing. This processing creates relation candidates out of pairs of entities. The original sequence of tokens therefore is destroyed. The relation candidates are enriched by a feature set containing *flat* and *structured* features. We presented possible features, and we showed which real world tasks are to be augmented by *Relation Extraction*.

Relation Extraction often is done by using kernel machines. In Section 5.4 we present possible kernels which are working on *flat* features being extracted from the entities itself, from the contextual tokens of the particular entities or from structural information available for the entity pair. These kernels lack the possibility to access inherently available information of structured features. This drawback is avoided by tree kernels. These kernels explicitly are designed to directly handle tree structured features.

We have shown the development of tree kernels which started with the convolution kernel of [Haussler, 1999]. [Collins and Duffy, 2001] used that kernel for tree-structured features. A more efficient extension has been developed by [Moschitti, 2006a, Moschitti, 2006b]. Finally, we presented tree kernel approaches embedding additional information inside of the tree structure [Had et al., 2009, Zhang et al., 2007].

An important point for the processing of trees is the complexity which depends on the number of nodes. In order to avoid complex calculations and for the creation of more meaningful trees, we present multiple pruning methods. These methods are using smaller subtrees of the original tree structures to achieve significant representatives for the certain entity pair.

The calculation of tree kernel values remains computationally complex. Although recent extensions of tree kernels have been developed, the recursive manner of the tree kernel calculation stays a remarkable factor in the computation routine. Especially SVMs which are exhaustively optimized during the training phase are performing many kernel calculations during this training phase. This results in longer runtime compared to using non-recursive kernels which unfortunately cannot process tree structures. Minimizing the number of kernel calculations should result in shorter runtime. In the next chapter we analyze the usage of tree kernels in machine learning approaches which are not optimized during the training phase which leads to a better runtime.

Chapter 6

Efficient Tree Kernel Usage

State-of-the-art techniques for relational learning are taking tree-structured attributes into account. Current machine learning approaches which respect tree-structured attribute types are mostly based on kernel machines such as *Support Vector Machines* (*SVM*). It hast been shown in Chapter 5 that many kernel calculations have to be performed during training and prediction phase. In many environments it becomes more and more important to use techniques which are as efficient as possible. In resource-aware settings like mobile devices, for instance, it should be avoided to perform complex calculations in order not to heavily stress the device. Every dispensable calculation can save power for the mobile device resulting in longer operating time. It follows that complex training is to be avoided on mobile devices.

Two possible solutions to avoid complex training arise: training should be done more efficiently, or training should be completely avoided. In contrast to making training more efficient it is much more interesting to entirely avoid complex training. The question arises if a machine learning method can achieve high performance by not relying on optimized training like for SVMs.

Machine learning techniques which memorize the training data instead of creating an optimized decision model are called lazy learners [Aha, 1997]. Unfortunately, it is not recommendable to memorize the training data completely. In fact, the training data has to be efficiently stored in a condensed way without losing too much relevant information. In contrast to lazy learners like *k-NN*, for instance, naïve Bayes classifiers do not memorize the complete training data. They create a condensed set of the training data which is not optimized in the sense of selecting just a set of representative examples to be used during prediction (see Section). In contrast to memorizing the complete training set, naïve Bayes classifiers are storing conditional probabilities for the certain attributes. These values are just expected values which calculated using the training data. In case of tree-structured attribute values, in particular, it is not trivial to calculate such probabilities (see Section 6). Although naïve Bayes classifiers are not optimized, they deliver good results. Their ability to deliver good results while not needing exhaustive training makes naïve Bayes classifiers to be preferred in resource-

aware settings like mobile devices [Fricke et al., 2011, Morik et al., 2010].

In this chapter we will show how to overcome the burden of too many kernel calculations during the usage of machine learning methods. We present two approaches of compressing forests of tree structures and we show how these approaches help to enhance machine learning from tree-structured attribute values. In addition, we show two particular machine learning techniques which profit from such compression.

Compression of tree structure forests and more efficient access on such forests is helpful in mostly every setting in which machine learning approaches have to access and iterate over multiple trees. In the prediction phase of *Support Vector Machines*, for instance, an example is classified by using the amount of support vectors extracted during training phase. This prediction is a sum of kernel calculations shown in equation (6.1) which is performed for each example $\mathbf{x}$ to be predicted.

$$\hat{y} = f(\mathbf{x}) = \sum_{i=1}^{n} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) \tag{6.1}$$

We suggest for simplicity that the examples $\mathbf{x}$ and $\mathbf{x}_i$ with $i \in \{1, \ldots, n\}$ are containing just one attribute which is tree-structured. Using the approach presented in equation (6.1) has two drawbacks: at first multiple kernel functions have to be evaluated. A kernel calculation has to be performed for every support vector which is every example $\mathbf{x}_i$ contained in the training set which got an $\alpha \neq 0$ during training. That results in a number of n kernel calculations for the worst case. Furthermore, the trees ($\mathbf{x}_i$ with $i \in \{1, \ldots, n\}$) might share several subtrees. These shared subtrees are representing redundant information. For the worst case, all trees $\mathbf{x}_i$ with $i \in \{1, \ldots, n\}$ are equal resulting in a sum of n equal kernel values being calculated. If the trees contained in the training set could be stored in one data structure, the number of kernel calculations will be reduced to just one calculation. In addition, multiple access on redundant subtrees will be avoided, if the right data structure is chosen.

$$\hat{y} = f(\mathbf{x}) = k(\mathcal{F}, \mathbf{x}) \tag{6.2}$$

Equation (6.2) shows the predictive function after storing the support vector trees in a merged manner. $\mathcal{F}$ represents the merged forest of support vector trees. The particular y_i and α_i values have to be stored in the tree structure forest, too. Instead of performing a sum of n kernel calculations just one kernel value is calculated on $\mathcal{F}$ and $\mathbf{x}$. We will use the term $\mathcal{F}$ in the following as a forest of tree-structures containing the informational units of multiple trees.

At first we will present related work existing in this field of research in Section 6.1. We will show how forests of tree-structures can be stored efficiently in a compressed manner in Section 6.2. In Section 2.4.1 we have shown how a naïve Bayes classifier is working, and in Section 6.3 we will show how to efficiently embed tree kernels in a naïve Bayes classifier. In Section 6.4 we present the usage of tree kernels in perceptron learning. In Section 6.5 we will demonstrate the experiments we made on three datasets

containing tree-structured attribute values. In Section 6.6 we present a summary of this chapter.

6.1 Related Work

[Pighin and Moschitti, 2009a, Pighin and Moschitti, 2009b, Pighin and Moschitti, 2010] presented an approach to speed up the handling of tree-structured values using *Support Vector Machines*. They analyzed the tree kernel feature-space of the fragments of the given trees. By utilizing *Support Vector Machines* on subsets of the training set they use the resulting support vectors to extract the most relevant fragments. These fragments are selected and they are exclusively used for later tree kernel processing. [Rieck et al., 2010] developed approximate tree kernels which are only working on a subset of subtrees of the original parse trees lowering the amount of kernel calculations to be evaluated.

SVMs are not the only machine learning technique tree kernels were used in. [Aiolli et al., 2007] showed that tree kernels can be embedded in perceptrons which may be used for online-learning. They are using directed acyclic graphs (DAG) to store forests of tree structures. If a tree should be stored in the DAG its nodes will be represented as vertices in the DAG. Redundant subtrees are avoided by introducing a frequency for each vertex. If a node – and the subtree it is the root of – is already stored in the DAG the frequency count of the corresponding vertex will be increased. This results in the avoidance of redundant storage of equal trees or subtrees. For binary classification problems one DAG is sufficient to perform the prediction. This is possible by using negative and positive frequencies for the negative and positive classes. The approach of [Aiolli et al., 2007] is precisely presented in Section 6.4.

To the best of our knowledge tree kernels never have been embedded in naïve Bayes classifiers before. We will present the possibilities to handle trees by naïve Bayes classifiers in Section 6.3.

6.2 Compression of Tree Forests

In this Section we will show how a set of trees can be merged into one data structure $\mathcal{F}$ resulting in more efficient access. We present two data structures which have different advantages and disadvantages. As an exemplary forest of tree-structures to be efficiently combined in one data structure we will use the 3 trees shown in Figure 6.1.

6.2.1 Merged Lists

As we use the formalism of [Moschitti, 2006b], the tree x_τ for an example $\mathbf{x}$ must be given in ordered production list form – like stated in Algorithm 2. The production of every node of a tree x_τ except the leaf-nodes is contained in the list. The productions are the roots of certain subtrees. If the productions are stored in the list they will be

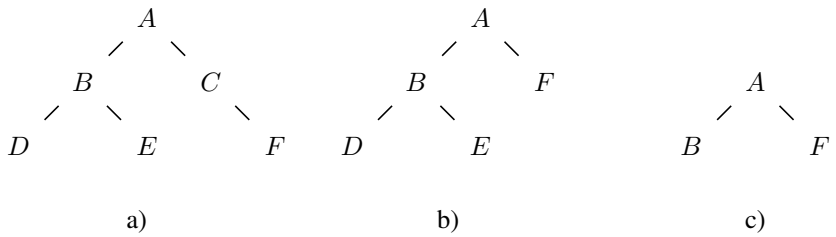


Figure 6.1: Three tree-structures

still linked to those subtrees. Figure 6.1 a) contains three such productions, namely *(A (B) (C)), (B (D) (E)), (C (F))*. The list of the productions linked to the corresponding trees is shown in Table 6.1. Figure 6.1 b) contains the productions *(A (B) (F)), (B (D) (E))*. The list of the productions linked to the corresponding trees is shown in Table 6.2. Figure 6.1 c) contains *(A (B) (F))*, and the corresponding list exposition is shown in Table 6.3.

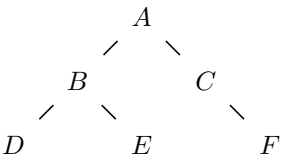
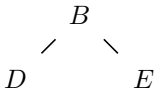
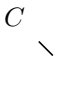
Production	Tree
(A (B) (C))	
(B (D) (E))	
(C (F))	

Table 6.1: List exposition of the tree shown in Figure 6.1 a)

Given these sorted lists of productions – which are still linked to the corresponding (sub-) trees – allows to combine them by storing them together in one sorted list. The resulting list is: *(A (B) (C)), (A (B) (F)), (A (B) (F)), (B (D) (E)), (B (D) (E)), (C (F))* and it is shown in Table 6.4. It is important to note that duplicate production entries cannot be easily merged. Having a look on the production *(A (B) (F))*, for instance, shows that equal productions can be linked to different trees. Those lists can be handled like any other tree x_τ given by sorted production lists. Instead of calculating a sum of kernel

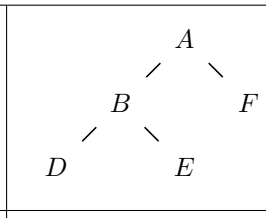
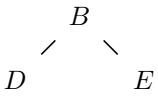
Production	Tree
(A (B) (F))	
(B (D) (E))	

Table 6.2: List exposition of the tree shown in Figure 6.1 b)

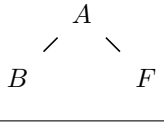
Production	Tree
(A (B) (F))	

Table 6.3: List exposition of the tree shown in Figure 6.1 c)

calculations like in equation (6.1), the tree-structured values can be stored in one list $\mathcal{F}$ which can be processed by using just one kernel calculation for each class which is shown in equation (6.2).

Runtime analysis

In this Section we will show that in practical use the runtime of the kernel calculations on multiple lists is worse than the runtime of one kernel calculation on one list containing all the individual lists together.

We assume that ψ_i is representing unique productions here. This means that production ψ_i does not equal production ψ_j, where $i \neq j$. The production list shown in Table 6.4 is encoded to $\{\psi_1, \psi_2, \psi_2, \psi_3, \psi_3, \psi_4\}$, so that the production lists shown in Tables 6.1, 6.2 and 6.3 are encoded to $\{\psi_1, \psi_3, \psi_4\}$; $\{\psi_2, \psi_3\}$; $\{\psi_2\}$.

The crucial part of the processing is the construction of the node pairs (see Algorithm 2). Following the declaration of [Moschitti, 2006b] building such node pairs can be done in $\mathcal{O}(|N_1||N_2|)$ for the worst case. This case occurs if two trees are containing only one particular type of production. The list of such tree contains e.g. ψ_i multiple times. In that case, the pairs are built by iterating one time over the first list of productions for each element of the second list of productions.

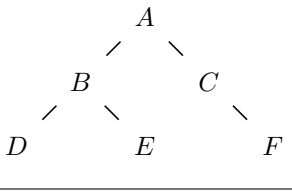
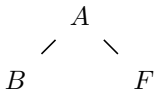
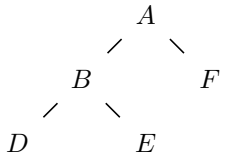
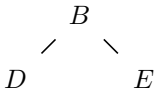
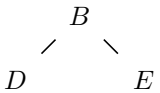
Production	Tree
(A (B) (C))	
(A (B) (F))	
(A (B) (F))	
(B (D) (E))	
(B (D) (E))	
(C (F))	

Table 6.4: List exposition of the merged trees

In our example the kernel and respectively the node pair list is calculated $|M| = 3$ times, where M is the amount of trees contained in $\mathcal{F}$. That leads to a runtime of $\mathcal{O}(|M||\bar{N}_1||N_2|)$ in the worst case, where $\bar{N}_1$ is the amount of nodes of the tree in M containing the most productions. Using only one list results in calculating only one node pair list, but one of the trees consists of all productions of the merged lists which leads to $\sum_{i=1}^{|M|} |N_i|$ productions. This also leads to a runtime of $\mathcal{O}(|M||\bar{N}_1||N_2|)$ for the worst case, where $\bar{N}_1$ is the amount of nodes of the tree in M containing the most productions.

[Moschitti, 2006b] also states that the runtime of the node pair construction can need $\mathcal{O}(|N_1|+|N_2|)$ for special cases (see Section 5.5.2). As an example the tree consisting of the production ψ_2 like shown in Figure 6.1c) is going to be classified by the particular tree-structure forest $\mathcal{F}$ presented in Table 6.4. A pointer on the merged production list is shifted until the production ψ_2 – being the first and only production of the tree to be classified – is found. As the list contains the production ψ_2 two times, two nodepairs (ψ_2, ψ_2) are constructed. The production-list of the example to be classified is shifted and the algorithm ends because the example to be classified does not contain any more productions. The productions contained in the nodepairs of course are still linked to the corresponding trees. This processing was done in $\mathcal{O}(|N_1| + |N_2|)$. Especially the shifting of the productions of the example to be classified has to be repeated for every new tree in a list of trees. Using just one structure $\mathcal{F}$ avoids such computational overhead.

To act on this assumption, the runtime of a method using a tree kernel on multiple lists is $\mathcal{O}(|M|(|\bar{N}_1| + |N_2|))$ and the runtime of a method using just one list $\mathcal{F}$ is $\mathcal{O}(|M||\bar{N}_1| + |N_2|)$, where $\bar{N}_1$ is the amount of nodes of the tree in M containing the most productions.

6.2.2 Directed Acyclic Graphs

Another way to store multiple tree-structured values is to use a directed acyclic graph as presented by [Aiolli et al., 2007]. A minimal directed acyclic graph (DAG) containing a minimal number of vertices can be used to store all the tree structures which have been seen in the training set. The DAG $G = (V, E)$ contains a set of vertices V and each vertex $v \in V$ consists of a *label* (denoted by $l(v)$) on the one hand and a *frequency* value (denoted by $f(v)$) on the other hand. The DAG contains a set of directed edges E connecting some of the vertices. Nodes which are connected in the original tree-structures also are connected in the corresponding DAG.

A DAG containing all information of the trees apparent in Figure 6.1 is shown in Figure 6.2. The algorithm to create a minimal DAG representing multiple trees is given in Algorithm 3. This algorithm converts every tree into its inverse topological ordered list of vertices (using the method *invTopOrder*$(\cdot)$. The first elements of the list are vertices with zero outdegree. After that vertices containing at most children with zero outdegree are contained in the list, and so on. The vertices are sorted in ascending order by the length of the longest path from each vertex to a leaf. The tree shown in

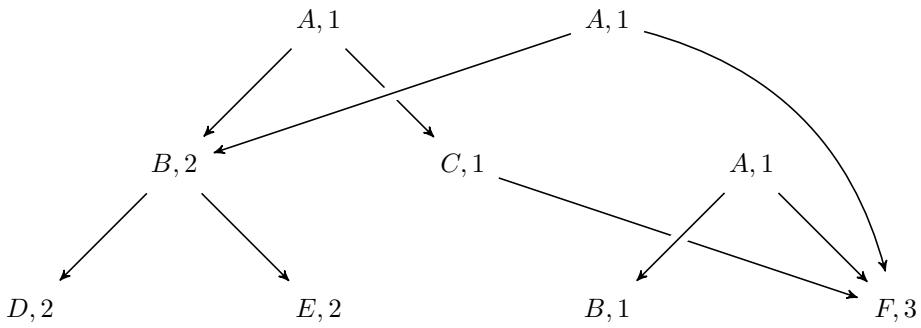


Figure 6.2: A DAG containing the information given by the tree-structures shown in Figure 6.1

Figure 6.1 a) becomes list $\{D, E, F, B, C, A\}$, the tree shown in Figure 6.1 b) becomes $\{D, E, B, F, A\}$, and finally, the tree shown in Figure 6.1 c) becomes $\{B, F, A\}$.

The lists are processed and for every vertex it is checked if it already exists in the DAG or if it is not. The formalism $dag(u) \equiv dag(v)$ means that the DAG rooted at vertex u is equivalent to the DAG rooted at vertex v. If a particular vertex is already available in the DAG the frequency of the corresponding node in the DAG is incremented by the frequency of the vertex. If a tree is added to a DAG, the frequencies of the nodes in the tree are (usually) 1. Otherwise, a node containing *label* and *frequency* of the vertex is created in the DAG. After that all the corresponding children (denoted by $ch[\cdot]$) of the new node are connected by edges. It is very important that the vertices of the trees are sorted because it is guaranteed that the children of a newly created node are already present in a DAG (if the created node has any).

6.3 The Tree Kernel naïve Bayes Approach

Unfortunately, memorizing specially shaped attribute values like trees is not as trivial as it is for numerical or nominal attribute values. In addition, it is not useful to memorize all trees seen in the training set because the storage needs would be too high. Like for nominal attributes it is not obvious how to calculate a similarity or distance of two trees which are not totally equal.

Tree kernels (see Section 5.5) allow a flexible calculation of similarities for trees because it will deliver greater values if the two trees used by the kernel function are more similar. Current machine learning approaches which respect tree-structured attribute types by using tree kernels are based on kernel machines. Suffering from relatively complex training, these techniques are not useful in resource-aware settings like mobile devices, for instance. Our contribution in this Section is threefold: we show how to embed tree kernels into naïve Bayes classifiers. We present a lot of options to handle

Algorithm 3 Creating a minimal DAG for a given tree-structure forest ([Aiolli et al., 2007], p. 3)

procedure CREATE MINIMAL DAG(A TREE-STRUCTURE FOREST $\mathcal{F} = x_{i_1}, ..., x_{i_n}$)
  Initialize an empty DAG D
  for $int\ j = 1; j <= n; j++$ **do**
    vertex_list $\leftarrow$ invTopOrder(x_{i_j})
    for all $v \in$vertex_list **do**
      if $\exists u \in D | dag(u) \equiv dag(v)$ **then**
        $f(u) += f(v)$
      else
        add node w to D with $l(w) = l(v)$ and $f(w) = f(v)$
        for all $ch_i[v]$ **do**
          add arc (w, c_i) to D where $c_i \in$ Nodes(D)
and $dag(c_i) \equiv dag(ch_i[v])$
        end for
      end if
    end for
  end for
  Return D
end procedure

the tree kernel values in a naïve Bayes classifier, and we evaluate our tree kernel naïve Bayes approach on three real-world datasets.

6.3.1 Using Tree Kernels as Distance Measure in naïve Bayes Classifiers

In Section 2.4.1 it became clear that the important parameters for a naïve Bayes classifier are $p(y)$ and $p(x_i|y)$. $p(y)$ is not affected by attribute-values and therefore is not affected by tree-structured attributes, too. $p(x_i|y)$ is the crucial term which should be regarded in the following. We distinguish three types of attribute value types: nominal, numerical and tree-structured value types.

For nominal attribute value types x_i, $p(x_i|y)$ is calculated by simply counting the occurrences of the particular values of x_i for each class $y \in Y$. For numerical attribute value types x_i, the mean (μ_i) and the standard-deviation (σ_i) of the attribute are used to calculate the probability density function for every class $y \in Y$ as seen in equation (6.3). A Gaussian normal distribution is expected, here.

$$p(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_i}} e^{-\frac{1}{2}\left(\frac{x_i - \mu_i}{\sigma_i}\right)^2} \tag{6.3}$$

The calculation of the probability for tree-structured attribute values is not that trivial. A simple approach would be just to store each unique tree-structured value and count

its occurrences like for nominal attributes. This approach is not promising as it is not general enough. It is not very probable that many examples are containing exactly the same tree-structured value. We are presenting a more flexible approach which uses tree kernels for the calculation of the conditional probabilities $p(x_i|y)$ for tree-structured attribute values. Algorithm 4 shows the calculation of the probabilities for naïve Bayes classifiers in pseudo-code. The input parameters are the training set S and the set to be predicted is set T.

Algorithm 4 Calculating probabilities for Naïve Bayes classifiers

```
1:  procedure CALCULATEPROBABILITIES
2:      Input: Example sets ( T ⊂ X × Y, S ⊂ X × Y )
3:      for all x ∈ T do
4:          for all x_i ∈ x do
5:              for all y ∈ Y do
6:                  if x_i is numerical or nominal then
7:                      calculate p(x_i|y)
8:                  else
9:                      if x_i is tree-structured then
10:                         v_y = f_y(x_i) = Σ_{x_j ∈ x'|(x',y')∈S,y'=y} k(x_i, x_j)
11:                         calculate p(x_i|y) using v_y
12:                     end if
13:                 end if
14:             end for
15:         end for
16:     end for
17: end procedure
```

If an attribute x_i is tree-structured the tree kernel value v is calculated by applying a tree kernel on x_i and on every tree structure x_j which has been seen in the training-set S for this attribute. To calculate this sum of kernel calculations all the tree-structures which have been seen in the training set have to be taken into account during the test phase. Storing every tree-structure that has ever been seen in the training-phase on its own in a list, for instance, has two major drawbacks:

1. the storage requirements are high
2. the number of kernel calculations correspond to the quantity of the training set

To reduce the storage needs and the number of kernel-calculations we can use both approaches we already presented in Section 6.2 to store all tree-structured values in a compressed manner.

6.3.2 Calculation of probabilities using kernel-values

The tree kernel values can be processed like every other numerical attribute value in naïve Bayes classifiers. Equation (6.3) shows the calculation of the probability for an

attribute value x_i given a particular class $y \in Y$ by using the mean and the standard deviation of the attribute in the training dataset. Unfortunately, to calculate the mean and standard deviation of the tree kernel values during training, it is necessary to calculate the kernel values for each example in the training set using the global data structure $\mathcal{F}$ (merged list or DAG) which is used during the prediction phase, too. To calculate the conditional probability, first of all a data structure $\mathcal{F}$ for every particular label class is needed. This means that our approach has to perform one former run over the training set to create all $\mathcal{F}$s. After that, another run over the training set is needed to calculate the kernel values. This step cannot be performed during the first run over the training set as it needs the complete set $\mathcal{F}$s. The kernel values finally are needed to calculate mean and standard deviation. Algorithm 5 and 6 present our approach for training and predicting in pseudo-code.

Algorithm 5 Tree Kernel Naïve Bayes classifier training

procedure TREE KERNEL NAÏVE BAYES TRAINING + MEAN CALCULATION
  Input: Example set $T \subset \mathcal{X} \times \mathcal{Y}$
  Output: Probabilities P, Kernel-values K, Mean-values μ, Standard deviations σ
  Initialize $P, K, \mu, \sigma :=$ empty sets
  for all $(\mathbf{x}, y) \in T$ **do**
    for all $x_i \in \mathbf{x}$ **do**
      if x_i is numerical or nominal **then**
        calculate $p(x_i|y)$
        add $p(x_i|y)$ to P
      else
        if x_i is tree-structured **then**
          add x_i to $\mathcal{F}_y$
        end if
      end if
    end for
  end for
  for all $(\mathbf{x}, y) \in T$ **do**
    calculate $k(x_i, \mathcal{F}_y)$, where x_i is tree-structured
    add $k(x_i, \mathcal{F}_y)$ to K
    calculate μ_i and σ_i
    add μ_i to μ
    add σ_i to σ
  end for
  return P, K, μ and σ
end procedure

Algorithm 6 Tree Kernel Naïve Bayes classifier prediction

```
1:  procedure TREE KERNEL NAÏVE BAYES PREDICTION
2:      Input: Example ( x )
3:      for all x_i ∈ x do
4:          for all possible y ∈ Y do
5:              if x_i is numerical or nominal then
6:                  calculate probabilities p(x_i|y)
7:              else
8:                  if x_i is tree-structured then
9:                      calculate tree-kernel-value v_y = f_y(x_i) = k(x_i, F_y)
10:                     calculate the probability p(x_i|y) using v_y
11:                 end if
12:             end if
13:         end for
14:     end for
15: end procedure
```

6.3.3 Pseudo-probabilities using kernel-values

The calculation of the mean and standard deviation of the tree kernel-values is very time consuming because the tree kernel has to be applied on every example of the training set with each data structure $\mathcal{F}$. This results in $n|Y|$ kernel calculations, where n is the number of examples. We try to overcome this computational complexity by not calculating the mean and standard deviation of the tree-structured values. We are using various normalization methods to create *pseudo-probabilities*, instead. Although, these *pseudo-probabilities* are used like every other probability in equation 2.19, they are real-valued and not necessarily bounded by 0 and 1.We are replacing the calculation of the probability in line 10 of Algorithm 6 by various approximation methods – some of them are normalizations. We present 6 methods which are finally evaluated on three real-world datasets:

- **none**

  The calculated tree kernel value v_y is directly used as a probability

  $$p(x_i|y) = v_y$$

- **normalize by frequency**

  The calculated tree kernel value v_y is normalized by the number of all subtrees in $\mathcal{F}_y$ for class y (this is the length of the merged list or the sum of frequencies in the DAG)

  $$p(x_i|y) = \frac{v_y}{freq_{\mathcal{F}_y}}$$

- **normalize by tree number**

The calculated tree kernel value v_y is normalized by the number of trees contained in $\mathcal{F}_y$ for class y

$$p(x_i|y) = \frac{v_y}{trees_{\mathcal{F}_y}}$$

- **normalize by maximum**

  The calculated tree kernel value v_y is normalized by the maximum value calculated on the training set by $\mathcal{F}_y$ for class y

$$p(x_i|y) = \frac{v_y}{max\{v_y^j | j \in \{1, \ldots, |S|\}\}}$$

- **normalize by all**

  The calculated tree kernel value v_y is normalized by the sum of the values calculated on the training set by $\mathcal{F}_y$ for class y

$$p(x_i|y) = \frac{v_y}{\sum_{j=1}^{|S|} v_y^j}$$

- **normalize by tree size**

  The calculated tree kernel value v_y is normalized by the fraction of frequency and tree number for class y

$$p(x_i|y) = \frac{v_y}{\left(\frac{freq_{\mathcal{F}_y}}{trees_{\mathcal{F}_y}}\right)}$$

- **normalize by example set size**

  The calculated tree kernel value v_y is normalized by the number of examples given for the particular class

$$p(x_i|y) = \frac{v_y}{|\{(\mathbf{x}', y') \in S | y' = y\}|}$$

6.4 Perceptrons with Tree Kernels

In this section we will show how to use perceptron models 2.4.1 in combination with tree kernels for the processing of tree-structured data.

6.4.1 Kernel-based Perceptron

Extending the perceptron to incorporate kernels has been proposed by several researchers [Crammer et al., 2003, Weston et al., 2005, Dekel et al., 2008]. The basic observation is that the hyperplane parameters β and β_0 are given by linear representations of misclassified training examples in equation (2.16). For *linear* kernels, it is possible to compute $\beta = \sum_i y_i \mathbf{x}_i$ allowing for a compact representation of the decision function due to

$$\sum_{(\mathbf{x}_i, y_i) \in M} y_i k(\mathbf{x}_i, \mathbf{x}) = k(\beta, \mathbf{x}) = \langle \Phi(\beta), \Phi(\mathbf{x}) \rangle.$$

However, as the mapping Φ implicit within the kernel $k(\cdot, \cdot)$ may be non-linear, we are forced to explicitly use the misclassified examples M, yielding

$$f_{M,k}(\mathbf{x}) = \text{sign}\left(\sum_{(\mathbf{x}_i, y_i) \in M} y_i k(\mathbf{x}_i, \mathbf{x}) + \beta_0\right). \tag{6.4}$$

Obviously this impacts the classification/training time. As shown in equation (6.4) it involves multiple kernel computations for each new example opposed to the original perceptron.

6.4.2 Tree List Perceptron

As follows from (6.4) the composite nature of the decision function is a major drawback of non-linear kernels such as tree kernels. Approaches to overcome this generally aim at restricting the number of support vectors, i.e. limiting $|M|$ by some constant [Dekel et al., 2008]. Unfortunately, the difference between kernel-perceptrons which calculate a sum of kernel-calculated values (equation (6.4)) and traditional perceptrons which only update their vector β (equation (2.16)) remains present. For the latter case the vector β is just updated iteratively and the prediction is simply performed. For the kernel-perceptron the set M grows with every misclassified example and for every prediction a sum of kernel-calculations has to be performed.

Our approach is to divide M into disjoint sets for negative and positive classes. Thus, we end up with M^+ being the misclassified examples $(\mathbf{x}, y)$ with $y = +1$ and M^- containing the ones with $y = -1$. To store M^+ and M^- we can use any of both data structures presented in Section 6.2. Instead of M^+ and M^-, we write $\mathcal{F}_+$ and $\mathcal{F}_-$, now.

To reflect this separation, the decision function $f_\tau(\mathbf{x})$ is now defined as

$$f_\tau(\mathbf{x}) = \text{sign}(k(x_\tau, \mathcal{F}_+) - k(x_\tau, \mathcal{F}_-)) \tag{6.5}$$

where $k(\cdot, \cdot)$ is a tree kernel function and x_τ represents the tree to be used. Thus, instead of calculating the sum of kernel-function values of every misclassified example our approach only calculates two kernel-function values.

We already mentioned in Section 6.2.1 and 6.2.2 that the presented data structures can very elegantly be used for binary classification problems by using only one data structure for two classes. Perceptrons which are binary classification methods therefore can benefit from this fact by only maintaining one data structure $\mathcal{F}$ instead of one for each class ($\mathcal{F}_+$ and $\mathcal{F}_-$). The decision function $f_\tau(\mathbf{x})$ is then defined as

$$f_\tau(\mathbf{x}) = \text{sign}(k(x_\tau, \mathcal{F})). \tag{6.6}$$

Algorithm 7 has to be changed, respectively, by using only one data structure $\mathcal{F}$ instead of two data structures. Instead of using a list for storing the tree structures forest the approach also can benefit from using a DAG as a data structure. The comparison of different internal data structure is shown in Section 6.5.4.

Algorithm 7 Tree List Perceptron Algorithm

procedure TRAINTREELISTPERCEPTRON
Input: Training set $S \subset \mathcal{X} \times \mathcal{Y}$
Output: Positive misclassified examples $\mathcal{F}_+$ and negative misclassified examples $\mathcal{F}_-$
Initialize $\mathcal{F}_+, \mathcal{F}_- :=$ empty data structures
for all $(\mathbf{x}, y) \in S$ **do**
if $f_\tau(\mathbf{x}) \neq y$ **then**
if $y = 1$ **then**
$\mathcal{F}_+ \cup x_\tau$
else
$\mathcal{F}_- \cup x_\tau$
end if
end if
end for
return $\mathcal{F}_+, \mathcal{F}_-$
end procedure

6.5 Experiments

In the following Section we evaluate our method on three real-world datasets containing tree-structured values. We implemented the presented naïve Bayes tree kernel approach within the *Information Extraction Plugin* [Jungermann, 2010, Jungermann, 2011a] for the open source *Data Mining* toolbox *RapidMiner* [Mierswa et al., 2006]. The state-of-the-art and the most often used implementation of tree kernels in SVMs is the tree kernel implementation of Moschitti[1] which is using the SVM^{light}-implementation of Joachims [Joachims, 1999]. To show the competitiveness of our approach, we compared the runtime of our approach with the runtime of the tree kernel implementation of Moschitti. The running times presented in Table 6.6 are measured for 100 ten-fold cross-validation over the complete dataset. Table 6.7 contains the values for a five-fold cross-validation on a 10% sample of the dataset. Although the tree kernel naïve Bayes approach is faster the true gain in case of runtime can not be evaluated because our approach is implemented in *Java* while SVM^{light} is implemented in C.

6.5.1 Datasets

Syskill and Webert Web Page Ratings

The first dataset *Syskill and Webert Web Page Ratings* (SW) is available at the UCI machine learning repository [Frank and Asuncion, 2011]. The dataset originally was used to learn user preferences on websites. It contains websites of four domains. Additionally, user ratings for each website are given. We will only focus on the classification of the four domains by neglecting the user ratings. That leads to a four-class classification

[1] `http://disi.unitn.it/moschitti/Tree-Kernel.htm`

task. We parsed the websites for the construction of a tree-structured attribute value for each website by using a *HTML*-parser[2]. Unfortunately, some leafs of the resulting *HTML*-tree contain huge text fragments (textual nodes). These fragments would be disadvantageous for the calculation of tree kernels because few trees would contain completely the same text fragments. This will lead to smaller tree kernel values. To create more comparable trees we converted every leaf which contains text into a leaf containing just the word *'leaf'*. Unfortunately, the trees still were too huge to be processed by the tree kernel implementation of Moschitti which is restricted to smaller trees in the original implementation. An analysis of the trees showed that many equally shaped subtrees are contained many times in the trees. We pruned the trees by a very trivial heuristic: we deleted equally shaped subtrees in each tree. A tree which is processed in this way just contains unique subtrees. Using this heuristic creates smaller trees making them applicable by the tree kernel implementation of Moschitti.

To compare our approach to traditional naïve Bayes classifiers we converted the string representation (see Section 2.1.4) of the trees into its BOW representation containing the corresponding *TF-IDF* values. We split the string representation by using spaces and brackets for splitting. In addition, we used the two-grams of this split representation as features. This preprocessing finally resulted in 445 attributes. That attribute set already contains the tree attribute. The dataset contains 341 examples of four classes. 136 examples belong to class *BioMedical*, 61 to class *Bands*, 64 to class *Goats* and finally, 70 examples belong to class *Sheep*.

SQL requests dataset

[Bockermann et al., 2009] presented a dataset containing SQL-requests which are crawled from a database-server web log. Some of the requests are attempts to attack the database for accessing crucial data. The analysis of SQL-request to detect attacks is called intrusion detection. The requests in the dataset are classified as normal request and attack. The relevant class in this binary dataset is relatively rare because the dataset is containing only 15 attacks on 1000 requests. [Bockermann et al., 2009] used an SVM with a tree kernel to detect the attacks.

We made several experiments on that SQL dataset. Each example has been converted into a representation containing different types of attributes. One attribute is the tree-structured value of the SQL statement parsed by an SQL-parser. The other attributes are extracted out of this tree-structured values. We converted the tree structure into a string representation (see Figure 7.5) and converted it into a kind of BOW representation. We therefore split the strings at each parenthesis and used the resulting tokens and the bigrams of tokens for the creation of the feature set. This leads to a sum of 1695 *flat* and 1 *structured* attribute.

[2] `http://htmlparser.sourceforge.net`

ACE 2004

In addition to the experiments we made on the SQL and the Syskill and Webert dataset, we made experiments on the well-known ACE 2004 dataset.

The shared task of the Automatic Content Extraction conference 2004 offered one of the first tasks for relational learning [LDC, 2004a, LDC, 2004b] which meanwhile became a benchmark dataset for *Relation Extraction* methods. Since the publication of the paper of [Zhao and Grishman, 2005] most publications only use a part of the dataset for a better comparison of the approaches. This part contains 348 documents and 125.000 words. Generating all relation candidates leads to 50.924 relation candidates containing 4.356 relations. The dataset contains seven types of relations, namely: *ART, DISC, EMP-ORG, GPE-AFF, OTHER-AFF, PER-SOC, PHYS*. These relations are split into several sub-relations. Although the sub-relations were part of the evaluation of the RDC task at the ACE conference, we will only focus on the seven main relation types in this work. Because of the long runtime shown in Table 6.7 we extracted a sample of 10% of the dataset to evaluate the best parameter setting.

6.5.2 Evaluation of the Tree Kernel naïve Bayes Approach

Syskill and Webert Web Page Ratings

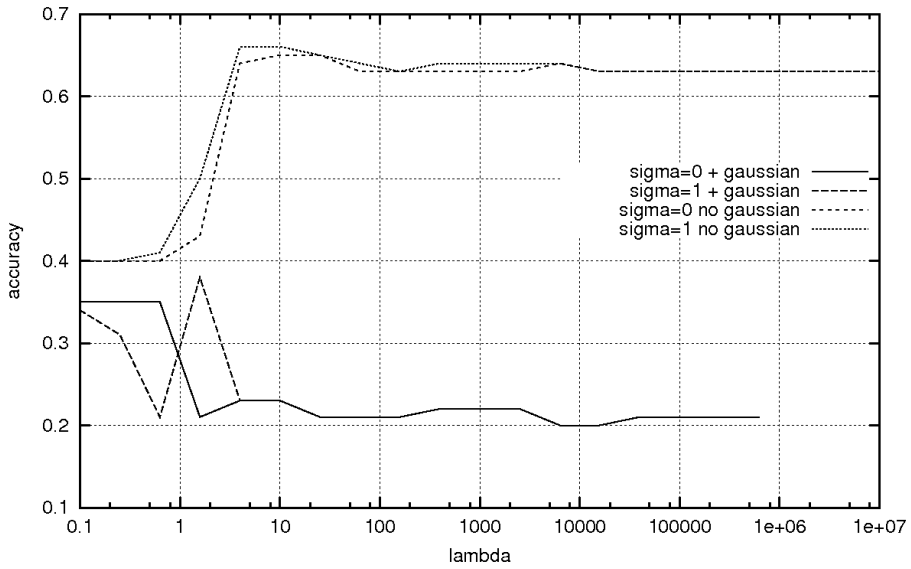
We made a parameter-optimization by only using the tree-containing attribute to analyze the best parameter-setting for the tree kernel naïve Bayes approach. We used all seven possible approximation methods. We used σ-values of 0 and 1. We used 21 different λ-values between 0.1 and 10^7, and in half of the settings we handled the kernel values like numerical values expecting a Gaussian normal distribution. On the other half of the settings we expected no Gaussian normal distribution and used the values directly. Table 6.5 shows the parameter-settings we used for evaluation. This

parameter	range
normalization by	none, DAG frequency, tree number, maximum, all, tree size, example number
λ	$\left[0.1 \ldots 10^7\right]$
σ	$0, 1$
Gaussian normal distribution	true,false

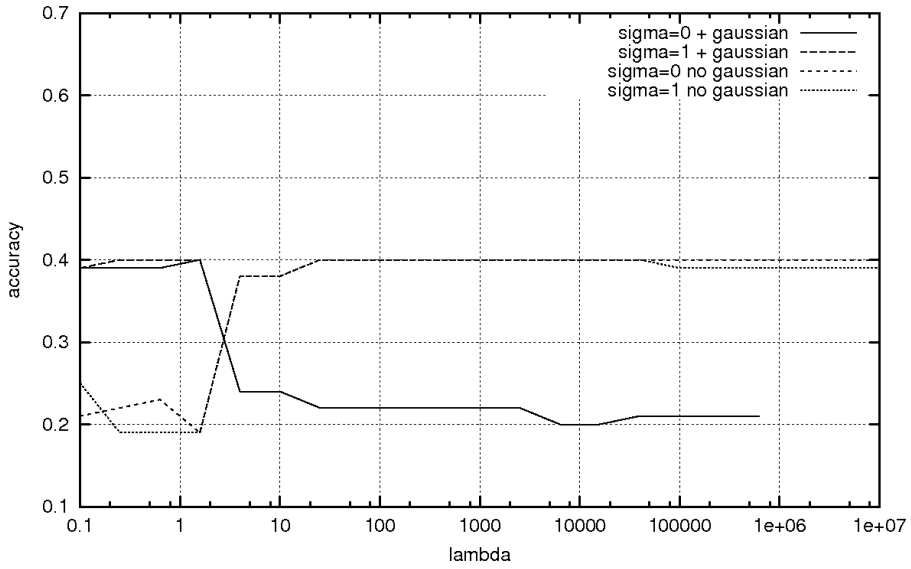
Table 6.5: Comparison of results of the naïve Bayes experiments on the SW dataset

setup results in 588 individual experiment-settings. Each of this setting is evaluated by a ten-fold cross validation. Figure 6.3 is the visualization of the results of the seven normalization methods.

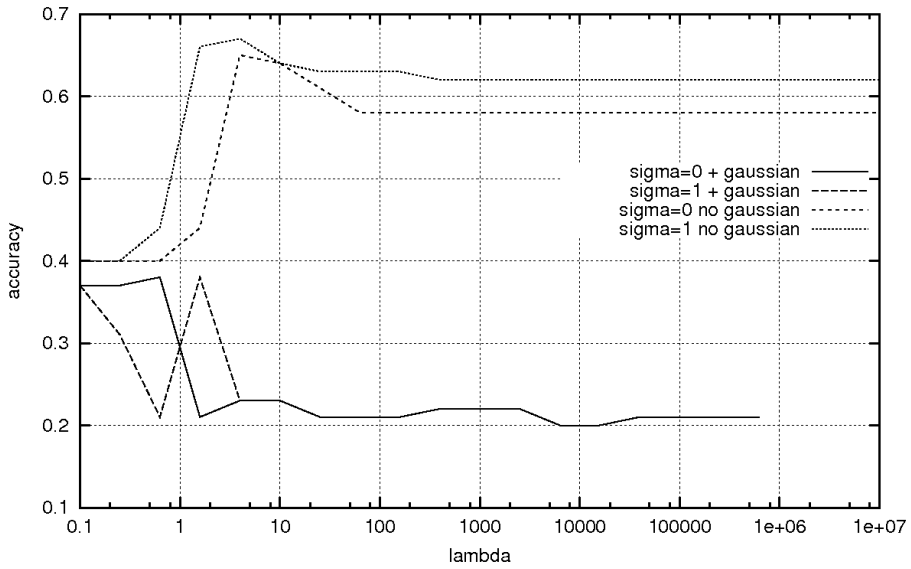
The original tree kernel is restricted to λ-values of $0 < \lambda \leq 1$. For the tree kernel naïve Bayes approach λ-values greater than 1 are delivering the best results. Another interesting fact is that using the kernel values directly as probabilities – without expecting



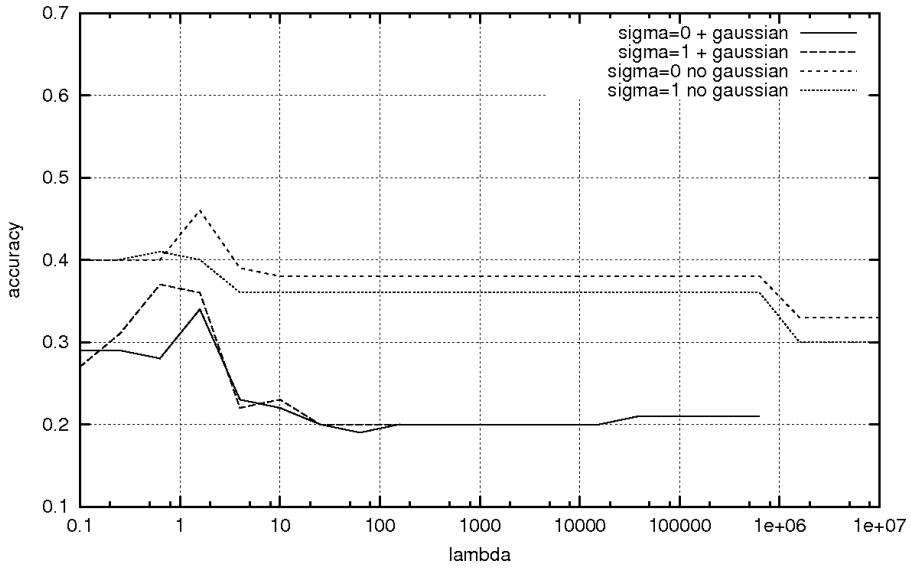
a) Different parameter settings and calculating the probabilities by using ”No normalization”



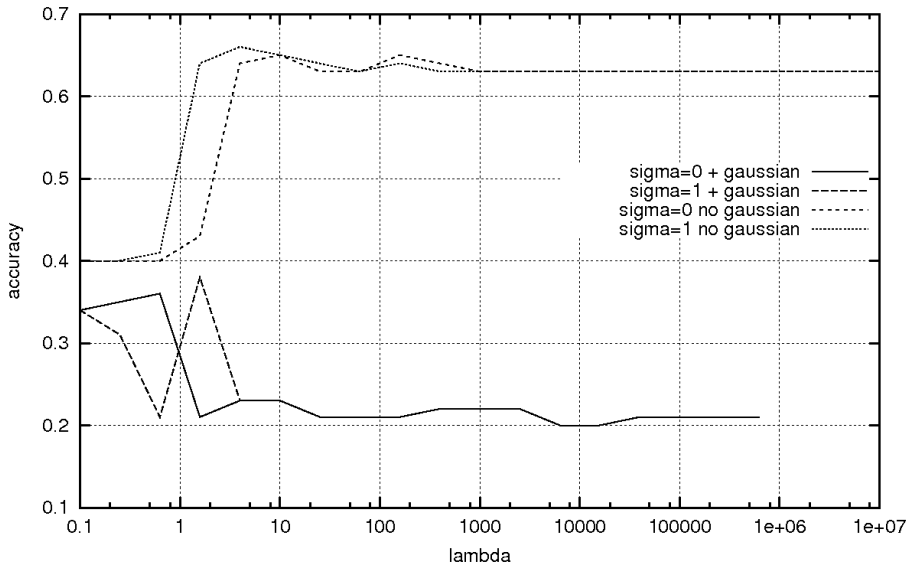
b) Different parameter settings and calculating the probabilities by using ”Normalize by maximum”



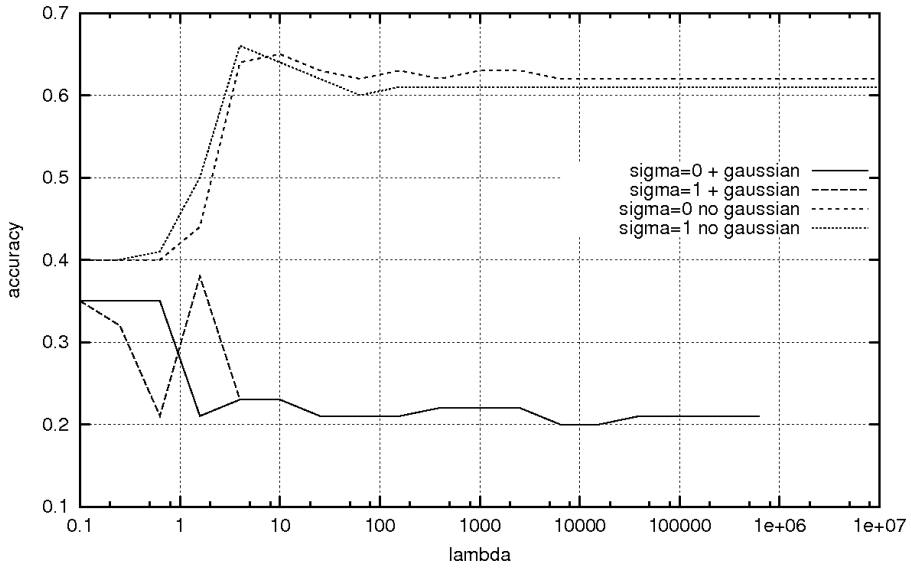
c) Different parameter settings and calculating the probabilities by using ”Normalize by DAG frequency”



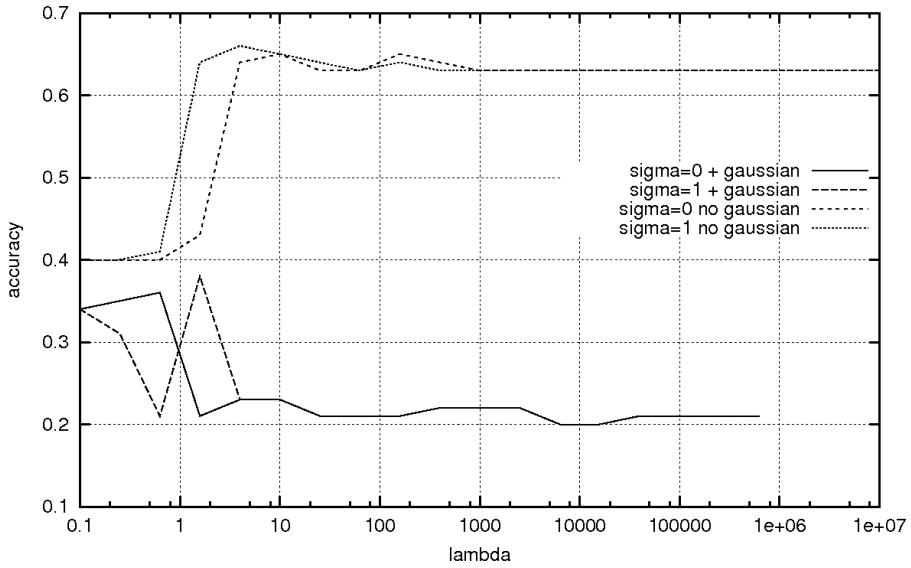
d) Different parameter settings and calculating the probabilities by using ”Normalize by all”



e) Different parameter settings and calculating the probabilities by using ”Normalize by example number”



f) Different parameter settings and calculating the probabilities by using ”Normalize by tree size”



g) Different parameter settings and calculating the probabilities by using ”Normalize by tree number”

Figure 6.3: Optimization experiments for SW

a normal Gaussian distribution and without normalization – delivers almost the best results. This is remarkable because expecting a normal Gaussian distribution means to calculate the mean and standard deviation after the construction of the DAGs. Not calculating the mean and standard deviation for the training set results in a more ordinary calculation and finally a better runtime (see Tables tab:ResourceAware:SQLresults and tab:ACEresults).

If we are calculating the tree kernel values for all trees in the training data for achieving numerical values we will be able to use them as we usually use numerical values. We calculate mean and standard deviation of those numerical values. Assuming a Gaussian normal distribution we are able to calculate probabilities given new and not yet seen numerical values and the mean and standard deviation of the training data. It is remarkable that this way of calculating the probabilities achieves the worst performance. A reason for this could be outliers contained in the training data. If a tree-structured value contained in the training data delivers an exceptional huge tree kernel output the mean and standard deviation will be useless for the calculation of correct conditional probabilities. Unfortunately, finding outliers in tree-structured attributes is not trivial. The extraction of outliers only can be done by some kind of similarity measure. A possible solution to manage this is to use tree kernels, too. This means that first of all, the DAGs have to be constructed. After that the kernel-values for all tree-structured values of the training data have to be calculated. Comparing the resulting numerical

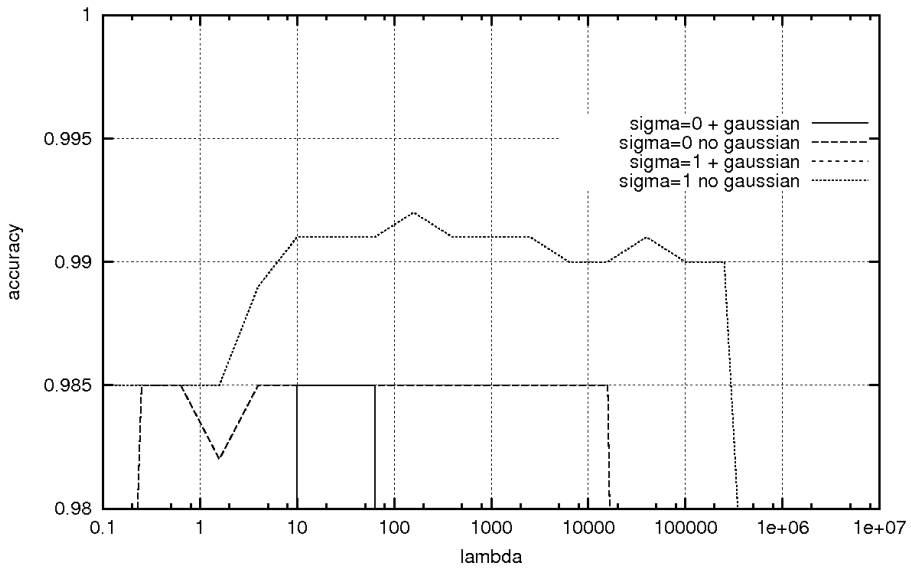
values will probably state out the outliers. If we now want to remove such outliers, the formerly created DAGs will have to be reconstructed because they still contain the defect tree-structured value.
Another argument for the assumption that outliers are contained in the training data is visible in Figure 6.3 b) and d). Normalizing the tree kernel outcome will deliver bad results if the maximum and the sum of the kernel outcomes of the training phase are used for normalization. The reason for this again could be outliers creating exceptional huge kernel outcomes which finally result in inaccurate values for the approximations during prediction.
Using one of the other normalization methods deliver better results, again, which are more or less similar to.

We extracted the best parameter setting for the tree kernel naïve Bayes approach using the tree and the BOW features together, too. After that we used the best settings to evaluate the performance of our approach. Table 6.6 shows the performances concerning accuracy, recall and precision. Those values have been evaluated using a loop of 100 ten-fold cross-validations for the naïve Bayes and tree kernel naïve Bayes approaches. The tree kernel SVM only was also evaluated on 100 ten-fold cross-validations. The results show that our naïve Bayes tree kernel approach delivers comparable performance, at least.

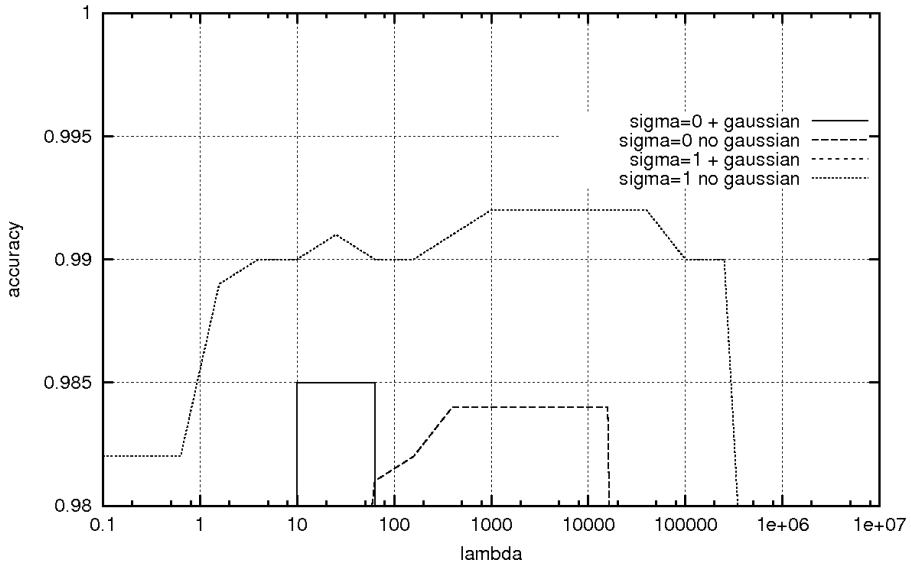
Using an analysis of variance test (ANOVA, see Section 2.4.4) it becomes apparent that using the tree kernel naïve Bayes approach on tree-structured values delivers significantly better results in cases of accuracy and precision than a naïve Bayes approach using no tree-structured values. Using the tree kernel naïve Bayes approach on tree-structured and BOW features delivers slightly better results in cases of accuracy.

SQL requests dataset

To optimize the parameters for our tree kernel naïve Bayes approach on the SQL dataset, we used the same setting as already described for the SW dataset (see Table 6.5). Figure 6.4 shows the results using the optimal parameters. The best performance is achieved by using either no approximation method or by DAG frequency approximation which both achieve 99.2% accuracy. For both (and all the other) normalization methods the results heavily depend on the chosen value for the λ-parameter. A default learner which only predicts the majority class would achieve 98.5% accuracy because 985 of 1000 requests are no attacks. Like for the *Syskill and Webert Web Page Ratings* dataset the normalization methods using the sum or the maximum of the outcome been seen during training are not flexible enough to achieve good results. Those approximations rely on values calculated on the training set. They never perform better than the default learner which predicts the majority class. The difference between training and prediction set seems to be too grave for the achievement of good results. It is very likely that the model is *overfitted* to the training set. The usage of the remaining normalization methods does not deliver better results than using no normalization or DAG frequency normalization. Additionally, the results are poor if we are expecting a Gaussian normal distribution on the values calculated on the training set and using this



a) Different parameter settings and calculating the probabilities by using ”No normalization”



b)Different parameter settings and calculating the probabilities by using ”Normalize by DAG frequency”

Figure 6.4: Optimization experiments for SQL

for calculation of the probabilities during prediction.

Table 6.6 shows that using the tree kernel naïve Bayes approach on tree-structured values compared to a naïve Bayes approach using no tree-structured values delivers significantly better results in cases of precision and accuracy. All experiments were evaluated using a loop of 100 ten-fold cross-validations and calculating the average. An interesting fact is that using a tree kernel SVM on the tree-structured and BOW values on the SQL dataset ends up in shorter runtime compared to the application of a tree kernel SVM alone. This might be due to the faster convergence of SVM processing the kernel on the *structured* in combination with the *flat* features.

Combining the tree-structured values with the BOW features using our tree kernel naïve Bayes approach delivers results which are worse than using the BOW or the tree-structured features exclusively. To develop a smoother combination of both probability distributions inside of the naïve Bayes classifier might overcome this problem.

Method	**Accuracy**	**Recall**	**Precision**	**Time** (in s)
SW				
Baseline (majority vote)	39.9%	25.0%	10.0%	
Naïve Bayes on BOW	$60.9 \pm 1.5\%$	$63.3 \pm 1.0\%$	$64.2 \pm 0.9\%$	0.2
Tree Kernel Naïve Bayes	$66.6 \pm 0.7\%$	$63.1 \pm 0.8\%$	$70.5 \pm 2.2\%$	1.90
(+ Gaussian)				2.73
Tree Kernel Naïve Bayes & BOW	$66.0 \pm 0.9\%$	$62.8 \pm 0.9\%$	$65.5 \pm 2.3\%$	3.04
Tree Kernel SVM	$66.6 \pm 7.0\%$	$60.8 \pm 8.1\%$	$71.8 \pm 9.3\%$	55.0
Tree Kernel SVM & BOW	$\mathbf{72.3 \pm 6.0\%}$	$\mathbf{68.0 \pm 6.8\%}$	$\mathbf{75.8 \pm 8.9\%}$	98.5
SQL				
Baseline (majority vote)	98.5%	50.0%	49.3%	
Naïve Bayes on BOW	$96.6 \pm 0.2\%$	$\mathbf{81.0 \pm 2.2\%}$	$62.3 \pm 1.0\%$	3.12
Tree Kernel Naïve Bayes	$\mathbf{99.3 \pm 0.1\%}$	$79.7 \pm 2.4\%$	$\mathbf{84.0 \pm 4.0\%}$	60.6
(+ Gaussian)				612
Tree Kernel Naïve Bayes & BOW	$96.6 \pm 0.2\%$	$81.2 \pm 2.7\%$	$64.3 \pm 2.5\%$	66.5
Tree Kernel SVM	$98.5 \pm 0.5\%$	$50.0 \pm 0.0\%$	$49.3 \pm 0.3\%$	64.5
Tree Kernel SVM & BOW	$98.9 \pm 0.6\%$	$62.5 \pm 18.3\%$	$67.3 \pm 24.2\%$	56.8

Table 6.6: Results of various machine learning approaches on the SW and SQL datasets (the values are arithmetic means and the corresponding standard deviations)

ACE 2004

We evaluated the best experiment setting and the performance of the tested methods on that sample, too. To calculate the values for precision and recall, we calculated the average values of the certain values for each of the seven relation classes. These seven classes are the relevant ones. The values for precision and recall for the negative examples are good just because of the amount of negative examples. A default learner only predicting no relation class will result in more or less good performance as the number of relations which are not classified is very huge compared to the classified relations.

The performance values of the naïve Bayes approaches are evaluated using a loop of 100 five-fold cross-validations.

We used the features presented by [Zhou et al., 2005] (see Section 5.2) in several settings shown in Table 6.7. Again, the tree kernel SVM is a little bit faster if it is applied on a tree-structured attribute together with non-tree-structured attributes. If the tree kernel SVM is applied on tree-structured attributes alone, it is on the other hand a little bit slower than our tree kernel naïve Bayes approach. This relies on the faster convergence of the SVM processing *flat* and *structured* features together. Although the results for the tree kernel SVM are not significant, the results achieved by the tree kernel naïve Bayes approach are comparable. Using the tree-structured attributes in contrast to just using the features presented by [Zhou et al., 2005] (Zhou-features) results in significantly better results in case of precision and accuracy. In addition to the other two datasets we evaluated the f-score (f-measure, see Section 2.4.4), too. The best result is achieved by the tree kernel naïve Bayes approach in combination with the Zhou-features.

Method	**Accuracy**	**Recall**	**Precision**	**F-measure**	**Time** (in s)
NB on Zhou-features	$79.0 \pm 0.2\%$	$\mathbf{56.2 \pm 2.0}\%$	$18.1 \pm 0.6\%$	27.4%	0.07
TKNB	$91.0 \pm 0.2\%$	$15.3 \pm 1.4\%$	$\mathbf{29.7 \pm 2.6}\%$	20.2%	320
(+ Gaussian)					1400
TKNB & Zhou-features	$86.8 \pm 0.3\%$	$48.3 \pm 1.9\%$	$25.7 \pm 1.0\%$	$\mathbf{33.5}\%$	320
TKSVM	$92.5 \pm 0.3\%$	$4.0 \pm 3.8\%$	$13.5 \pm 10.2\%$	6.2%	620
TKSVM & Zhou-features	$\mathbf{93.3 \pm 0.3}\%$	$12.6 \pm 6.8\%$	$27.7 \pm 16.9\%$	17.3%	585

Table 6.7: Results of various machine learning approaches on the ACE dataset (the runtime values are rounded)

Discussion

Using the tree kernel naïve Bayes approach in most of the cases results in short runtime compared to the SVM tree kernel approach. In addition the tree kernel naïve Bayes approach delivers comparable results in case of accuracy, precision, recall and f-measure compared to the results achieved by the SVM.

We want to analyze the different runtimes of the tree kernel naïve Bayes approach on the SQL and the SW dataset. In Table 6.8 we show a comparison of the runtime of the tree kernel naïve Bayes approach on the datasets SW and SQL in detail. It is getting obvious that the runtime depends on the characteristics of the particular dataset. For the SW dataset, for instance, it lasts longer to create the DAGs than to use the DAGs for predicting the test dataset. For SQL it is the other way round: creating the DAGs does not need as much time as predicting the examples during the prediction phase. In addition, the runtime of a complete cross-validation on the SQL dataset is generally longer than it is for SW.

Dataset	DAGs creation time (in s)	Examples in training	Prediction time (in s)	Examples to predict	Nodes in DAGs
SW	1.57	3069	0.22	341	227, 1188, 316, 291
SQL	13.7	8100	60.7	1000	242, 7360

Table 6.8: Runtime of ten-fold cross-validations on various datasets

Reasons for this are on the one hand the greater amount of examples to be processed in the SQL dataset. On the other hand the tree-structures in the SQL dataset are more complex than in the SW dataset. For the binary SQL dataset just two DAGs are constructed but one of them is very huge because it contains more than 7300 vertices. The greatest DAG of the four DAGs created for the SW dataset just contains 1188 vertices. Calculating the tree kernel values is costly. These calculations are just performed during the prediction – like for every lazy learning approach.

If great and complex trees have to be evaluated like for the SQL and especially for the ACE dataset the fraction of prediction and training time becomes bigger.

6.5.3 Evaluation of the Tree Kernel Perceptron Approach

In this Section we analyze the performance with respect to runtime and kernel calculations of the proposed Tree List Perceptron-Algorithm (Algorithm 7) compared to SVMs or Kernel-Perceptrons.

ACE 2004 dataset

We used the ACE 2004 dataset for Relation Detection and Characterization (RDC) (see Section 6.5.2), again. In contrast to our setting in Section 6.5.2 we used the complete dataset.

As we use binary classifiers we cannot train one model for all relation types. We used a one-against-all strategy (see Section 2.4.3) and evaluated every experiment by a five-fold cross-validation. This setting is used by various experiments on the ACE 2004 dataset like [Zhao and Grishman, 2005, Zhang et al., 2006, Zhou et al., 2007, Qian et al., 2008].

In a first experiment-setting we compared the results of the SVM-LIGHT-TK of [Moschitti, 2006b] with a perceptron using a *fast tree kernel* and the tree list perceptron approach. For the perceptron- and tree list perceptron-training we only made one pass over the data and used the resulting perceptron/tree list perceptron for prediction. This setting is used to analyze the performance which can be achieved by approximated training. Our suggestion is that the runtime is shorter and the resulting performance is not as good as for the SVM which is optimized during several runs on the training set.

	Perceptron		
	Precision	Recall	F-Measure
ART	27.6	28.8	27.95
DISC	39.2	32.2	34.33
EMP-ORG	56.2	55.4	55.39
GPE-AFF	25.4	42.2	30.36
OTHER-AFF	24.2	25.4	24.3
PER-SOC	50.4	53.6	50.81
PHYS	46.4	34.4	39.24
	Tree List Perceptron		
	Precision	Recall	F-Measure
ART	24.6	28.6	25.7 (28)
DISC	38.8	34.6	36.2 (35.2)
EMP-ORG	59.2	53.8	55.7 (54)
GPE-AFF	20.2	36.6	24.0 (32.8)
OTHER-AFF	22.6	23.4	22.6 (29.8)
PER-SOC	55.8	55.2	54.8 (48)
PHYS	46.8	35.6	40.3 (39)
	SVM^{light}		
	Precision	Recall	F-Measure
ART	84.7	17.0	28.3
DISC	84.8	3.8	45.3
EMP-ORG	84.1	55.0	66.5
GPE-AFF	72.9	22.5	34.4
OTHER-AFF	72.1	17.0	27.4
PER-SOC	87.1	46.3	60.5
PHYS	75.8	30.6	43.6

Table 6.9: Classification results on the ACE dataset using different machine learning techniques

Accuracy	**Precision**	**Recall**	**Time (in s)**
Perceptron			
$90.5 \pm 1.5\%$	$32.5 \pm 3.9\%$	$24.3 \pm 2.6\%$	5904
Tree List Perceptron			
$90.6 \pm 1.4\%$	$32.1 \pm 3.7\%$	$23.9 \pm 2.8\%$	1698
DAG Perceptron			
$90.7 \pm 1.1\%$	$32.9 \pm 3.7\%$	$24.1 \pm 2.6\%$	115

Table 6.10: Classification results on the ACE dataset using different perceptron approaches

The results are shown in Table 6.9. Although the results for precision are better for the SVM, the results for f-measure achieved by the perceptron and the tree list perceptron approach are comparable to those achieved by the SVM. Especially, the fact that the SVM iterates over the training set several times relativizes the results. Another observation is that the performance achieved by the perceptron and the tree list perceptron is nearly the same. Perceptrons might achieve results which are overestimated because of a positive order of the examples. We made five different randomized runs on the same dataset to avoid such behavior and presented the average results in brackets for the tree list perceptron. Additionally, we made 100 ten-fold cross-validations using the three different perceptron approaches on the data sample. The average results are presented in Table 6.10. It is remarkable that the results are comparable but the runtime is totally different. The perceptron based on the DAG is much faster than the two other perceptrons.

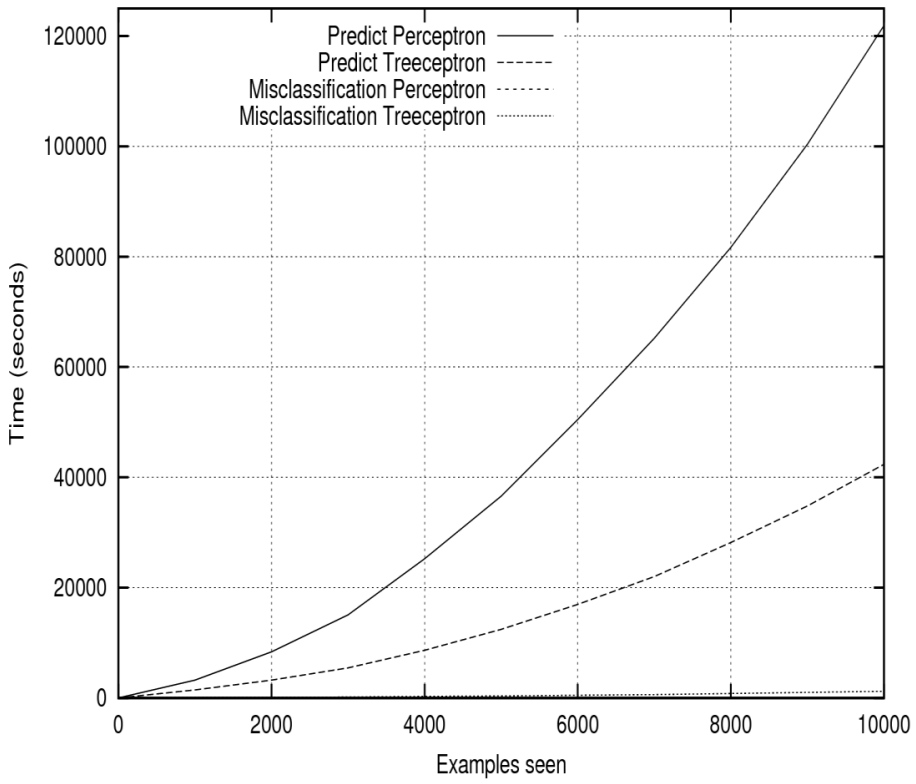


Figure 6.5: Runtime of the perceptron efficiently storing the trees in a list in contrast to a perceptron calculating the tree kernel values for each tree of the set.

In a second experiment-setting, we explored the number of kernel executions done dur-

ing the training-phase of the perceptron and the tree list perceptron. We differentiated

	Perceptron		Tree List Perceptron	
	direct	rec.	direct	rec.
	(* 10^6)			
ART	11.0	45.3	0.1	45.7
DISC	14.0	125.4	0.1	123.9
EMP-ORG	43.5	295.3	0.1	295.0
GPE-AFF	22.0	144.1	0.1	144.8
OTHER-AFF	9.0	33.7	0.1	34.0
PER-SOC	13.6	72.5	0.1	72.6
PHYS	42.5	447.6	0.1	451.0

Table 6.11: Number of kernel calculations executed by the perceptron and the tree list perceptron

between the number of direct and recursive kernel-calculations. The results are shown in Table 6.11. The number of direct kernel-executions is the number of executions on the root of the particular trees, whereas the number of recursive kernel-executions is the number of all the recursive kernel executions on the subtrees. Although the number of direct kernel executions differ strongly, the numbers of recursive executions are nearly equal comparing perceptron and tree list perceptron. The difference in direct executions is obvious: the tree list perceptron just performs two kernel executions for every prediction, whereas the perceptron performs an execution for every misclassified example per prediction. As the two kernel executions of the tree list perceptron are performed on production lists which contain the same information which is also contained int the multiple trees the prediction of the perceptron is performed on, the numbers of recursive executions are nearly the same.

In a third experiment-setting, the runtime of the perceptron and the tree list perceptron is compared empirically. We made experiments on two equally sized datasets and calculated the mean of the runtime. The results are shown in Figure 6.5. It is obvious that the formally presented runtime (Section 6.2.1) has a great empirical argument. The runtime of the perceptron always is more than two times slower than the runtime of the tree list perceptron. The two lines at the bottom of the plot show the time needed for the required computations after an example was misclassified. Whereas the perceptron just stores the example, the tree list perceptron needs to merge the list of productions with the positive or negative productions-list, respecting the order of the productions.

Syskill and Webert Web Page Ratings

We also evaluated the perceptron using tree kernels on the Syskill and Webert dataset we already presented in Section 6.5.1. We always made 100 ten-fold cross-validations to evaluate the experiments. Table 6.12 shows the results. The time we are mentioning in the last column is the mean value for one cross-validation. The feature sets we used for the experiments were manifold: the first and the second experiment have been

done by using a perceptron which only used the flat features extracted from the tree structures of the *HTML* trees. For the first experiment we used one- and two-gram-features of the flat features extracted from the trees and for the second experiment we only used the one-grams. The following four experiments all were done by using tree kernels embedded in a perceptron. The third experiment has been done by using a DAG handling positive and negative classes. It is important that a perceptron is a binary decision model which has to be used in a *1-vs-all* strategy (see Section 2.4.3) for the handling of multiple classes like it is needed for the SW dataset. The fourth experiment was using the tree list perceptron approach, and finally the fifth experiment only used the FTK (see Section 5.5.2) without any efficient storing of the tree set.

Method	**Accuracy**	**Recall**	**Precision**	**Time** (in s)
Using a perceptron on one- and two-gram features	$66.4 \pm 1.5\%$	$64.1 \pm 1.7\%$	$66.0 \pm 2.1\%$	118.7
Using a perceptron on one-gram features	$62.7 \pm 1.6\%$	$61.3 \pm 1.8\%$	$62.1 \pm 2.3\%$	18.9
Using a DAG	$67.9 \pm 1.5\%$	$64.0 \pm 1.6\%$	$68.4 \pm 2.5\%$	1.78
Using the tree list perceptron approach	$68.3 \pm 1.6\%$	$64.5 \pm 1.7\%$	$69.1 \pm 2.4\%$	55.0
Using the FTK approach	$68.3 \pm 1.5\%$	$64.6 \pm 1.7\%$	$69.2 \pm 2.4\%$	78.9

Table 6.12: Results of the Perceptron approaches on SW

6.5.4 Comparison of different internal data structures

In another experiment we tried to analyze the use of different data structures. We used a perceptron on the SQL dataset (see Section 6.5.1) and we used different data structures for storing the set of already misclassified examples containing tree structures. We present the trivial approach which just stores each tree and calculates the sum on the set of these trees. This approach is two-fold as the calculation is possible using the approach by [Collins and Duffy, 2001] or a faster approach by [Moschitti, 2006a, Moschitti, 2006b]. The approach to store the set of trees in one list to use the approach by [Moschitti, 2006a, Moschitti, 2006b] later on is presented in Section 6.4.2 and it is the third method to be used here. The fourth approach to store the set of trees is a DAG.

Table 6.13 shows the results of the perceptron experiments on the SQL dataset. It

Method	**Accuracy**	**Recall**	**Precision**	**Time** (in s)
One DAG	$98.9 \pm 0.6\%$	$87.3 \pm 3.1\%$	$82.4 \pm 6.8\%$	3.58
Tree List Perceptron	$99.0 \pm 0.6\%$	$87.3 \pm 3.6\%$	$83.2 \pm 6.4\%$	18.6
FTK	$99.0 \pm 0.6\%$	$87.3 \pm 3.6\%$	$83.2 \pm 6.5\%$	23.89
QTK	$98.9 \pm 0.5\%$	$87.6 \pm 3.3\%$	$82.3 \pm 6.6\%$	31.34

Table 6.13: Results of the Perceptron approaches on SQL

becomes obvious that quality of the results concerning recall, precision and accuracy do not depend on the internally used data structure. Figure 6.6 shows the runtimes for one 10-fold cross-validation using the five different data structures internally by a perceptron on the SQL dataset.

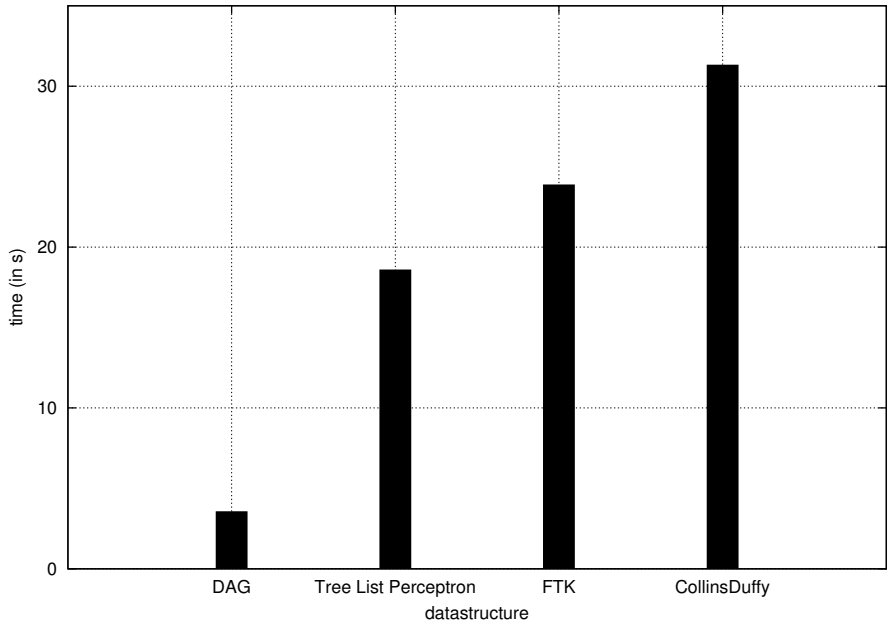


Figure 6.6: Runtime for a cross-validation using different data structures using a perceptron on SQL

6.6 Summary

In this chapter we have shown that a crucial point to be respected for efficient tree kernel usage is the data structure storing the set of trees.

We presented two data structures which both can handle sets of trees. One data structure is based on our idea to store the set of trees in a list of productions. This idea is affected by the FTK by [Moschitti, 2006b] which presents the usage of lists of productions to store trees. The other data structure is a directed acyclic graph (DAG) which – during tree kernel calculation – acts like a tree but contains a whole set of trees. The advantage of using a DAG is that the nodes contain frequencies in addition to the labels. This behavior results in a smaller amount of nodes which leads to a less complex calculation of the tree kernel values. The usage of both data structures leads to a more efficient calculation for tree kernels in contrast to not storing the certain set of trees in one data structure.

Our presented approach aims at enhancing the calculation over sets of trees in order to make the access more efficient. The internal calculation of $C(n_i, n_j)$ is not affected by our approach. The approximate tree kernel approach presented by [Rieck et al., 2010] aims at speeding up the calculation of $C(n_i, n_j)$. The combination of our approach and the approximate tree kernel could, again, lead to a more efficient tree kernel calculation.

Additionally, we presented two machine learning approaches which by definition have a shorter runtime than, for instance, support vector machines (SVMs). The first approach is a perceptron which is evaluated with all presented data structures. The second approach is our main contribution in this chapter which is the development of a tree kernel naïve Bayes classifier. This classifier is on the one hand significantly better than naïve Bayes classifiers applied on *flattened* structured features with respect to precision and accuracy. On the other hand our approach is significantly faster on particular datasets than comparable tree kernel methods based on optimized models like SVMs.

We presented experiments on three real-world datasets which show that our tree kernel naïve Bayes approach is a fast and efficient alternative to tree kernel methods based on kernel machines.

Chapter 7

The Information Extraction Plugin for RapidMiner

In this chapter we will present the plugin we developed for the open source framework *RapidMiner* [Mierswa et al., 2006]. Relevant publications concerning the extension are [Jungermann, 2009, Jungermann, 2010, Jungermann, 2011b, Jungermann, 2011c, Jungermann, 2011a].

Our plugin allows the combination of *Information Extraction* and *Data Mining* methods. In addition, it is possible to use all techniques which are already available in *RapidMiner*. These techniques do not only include models like decision trees or support vector machines. A great benefit of *RapidMiner* is the possibility to easily validate machine learning tasks. By the application of our plugin the validation process can also be used for *Information Extraction* processes making the results more significant. Additionally, we are able to evaluate and validate several different parameter settings for multiple techniques. This makes our plugin a toolbox for the comparison and development of new machine learning approaches for *Information Extraction*. The plugin is open source and easily to extend.

RapidMiner, which is shortly presented in Section 7.1, supports a certain data structure for storing datasets. This data structure has to be respected by extensions and operators of those extensions. This circumstances lead to particular requirements which have to be fulfilled by our extension. These requirements in addition to the data structure used in *RapidMiner* are presented in Section 7.1.1.

The process of a particular *Data Mining* task can be separated in four distinct parts in *RapidMiner*. The first part is the retrieval of the data. We present the possible ways to retrieve data for *Information Extraction* purposes in *RapidMiner* in Section 7.2.1. After the retrieval the data has to be prepared for future use. This preparation is often called preprocessing. Although the task of preprocessing sometimes contains the process of data preparation (see Section 2.2), we just focus on the enrichment of the

data by features to allow more precise analyses in this work. The preprocessing of the datasets is presented in Section 7.2.2. The preprocessed datasets finally can be used to create models which in turn can be used to analyze formerly unknown datasets. The process of creating models is called modeling and it is presented in Section 7.2.3. After – and sometimes also during – the process of modeling the models have to be evaluated to get the optimal model for a given datasets. The task of evaluation is presented in Section 7.2.4.

In Section 7.3 we present frameworks which are comparable to our plugin. We will show that state-of-the-art frameworks for *Information Extraction* are based on a comparable architecture like our plugin. It will become obvious that our plugin is superior compared to those frameworks because it enables the close collaboration of *Information Extraction* and *Data Mining*.

Section 7.4 summarizes this chapter. The particular reference for each operator is presented in Appendix A.

7.1 RapidMiner

RapidMiner is an open source framework for *Data Mining* purposes. It offers many *Data Mining* methods which can be plugged together to form a *Data Mining* analysis. The functional objects in *RapidMiner* are called operators, and the set of operators being plugged together are defined to be a so called process. The major function of a process is the analysis of the data which is retrieved at the beginning of the process. The framework offers a graphical user interface (GUI) that offers the possibility to connect operators with each other in the process. The particular panel visualizing the process is called process view. Operators have interfaces for achieving and presenting data. These interfaces are called ports. Input ports are receiving the data which will be presented to the operator and the output port is delivering the data which has been processed by the operator. Most operators have at least one input and one output port. Data that is passed to an input port of an operator is processed internally and it is presented at the output port, finally. The data which is processed is passed to any operator which is connected to the certain output port. Other types of objects may be created during the process. These objects are also presented at certain output ports. The data can be used to create models (see Section 2.4 for more information concerning models), for instance. These models can be evaluated, and performance results can be generated out of this evaluation process, and so on. These kinds of objects all can be passed as data objects from operator to operator by connecting the operators. The complete process has global output ports. Data, results or models which are passed to these ports are represented in the result view panel after finishing the process.

The GUI of *RapidMiner* is shown in Figure 7.1. Six main areas of the GUI, which can be rearranged by the user, are to be distinguished:

1. **Overview**
   The *Overview* tab is representing an overview of the complete process window.

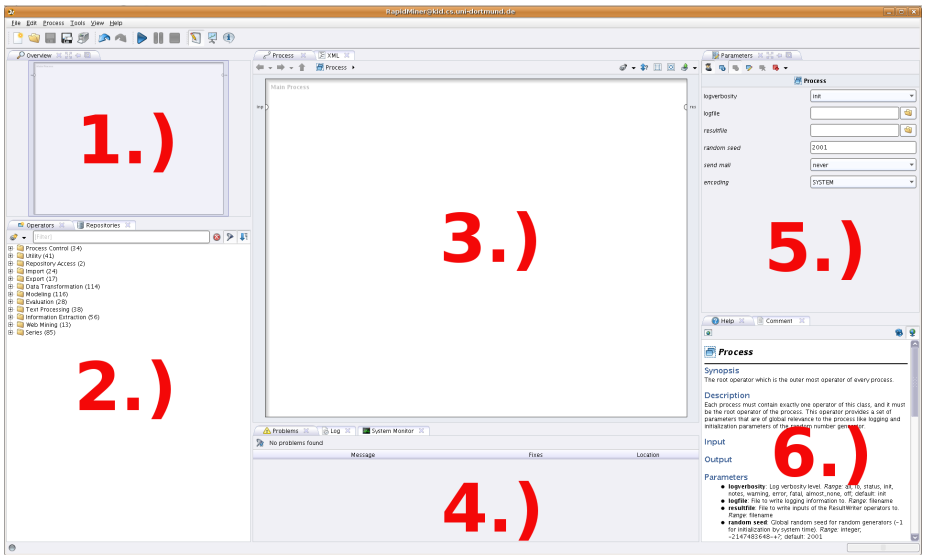


Figure 7.1: *RapidMiner* graphical user interface

If the process is too large to be displayed in the process window, the overview window will help to navigate to certain positions in the process window.

2. **Operators and Repositories**
   These tabs allow accessing operators or repositories of *RapidMiner*. Operators are the basic elements for building a process. Repositories store datasets to avoid the loading and converting process of files for each run of a process. This behavior leads to a faster access on datasets.

3. **Process**
   The *Process* window makes the whole process of connected operators accessible. An overview of this window which could become very large is available in the *Overview* tab.

4. **Problems, Log and System Monitor**
   This tab contains possible log messages, information about problems and about the system load.

5. **Parameters**
   The *Parameters* tab shows the parameters of the operator which is currently focused. Parameters are very important because the results of *Data Mining* tasks often depend on the right choice of the particular parameters.

6. **Help**
   The *Help*-tab contains information about operators which are focused.

Each *RapidMiner*-process can be split into four distinct phases. These phases are shown in Figure 7.2:

1. **Retrieve**
   The leftmost operator in Figure 7.2 is a *Retrieve*-operator. During the *Retrieve* phase the data which is processed later on is loaded from specific data sources.

2. **Preprocessing**
   The retrieved data has to be prepared or enriched in the *Preprocessing* phase. The second operator shown in Figure 7.2 (the purple one) is a particular preprocessing operator which is converting nominal values to numerical ones.

3. **Modeling**
   The prepared data is used in the *Modeling* phase to extract or create models which can be used for the analysis of unlabeled data. The third operator shown in Figure 7.2 is creating an SVM model.

4. **Evaluation**
   The two rightmost operators shown in Figure 7.2 are used to apply the learned model to a dataset and to evaluate the performance achieved by the applied model. The expected or real performance of the created models is evaluated during the *Evaluation* phase.

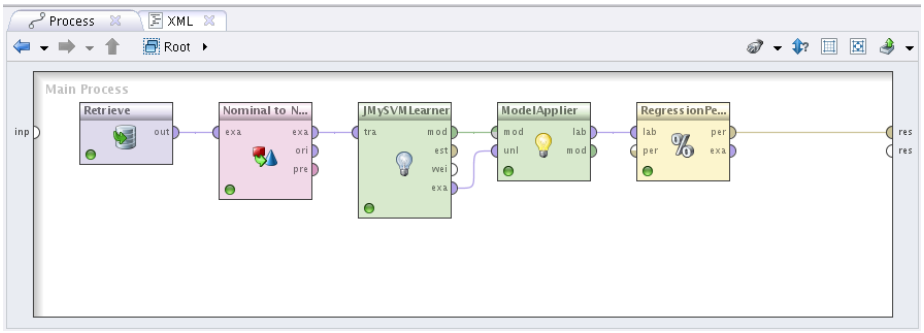


Figure 7.2: Exemplary process in *RapidMiner*

These phases are similar for every *RapidMiner* process. Therefore, we will define the particular phases and the corresponding specialties using the *Information Extraction Plugin* in Section 7.2.

7.1.1 Data Structure

The data structure in *RapidMiner* for handling example sets is comparable to a spreadsheet as it is used in many spreadsheet programs. The lines of the spreadsheet represent examples and the columns represent attributes. Table 7.1 shows an exemplary dataset containing n examples and d attributes.

It is remarkable that for many *Data Mining* tasks the examples are independently processed. It follows that the analysis of a particular example i only depends on the

ID	Label	Att_1	Att_2	...	Att_d
1	true	2	'Felix'	...	5.4
2	false	3	'goes'	...	7.1
...	...	...	...	...	...
n	true	1	'Detroit'	...	2.3

Table 7.1: Spreadsheet-like data structure internally used by *RapidMiner*

attributes of example i instead of taking, for instance, other examples into account. We already mentioned in Section 2.3.1 that the structure of documents and texts should be respected for *Information Extraction* tasks. CRFs, for instance, process all the tokens of a particular sentence at the same time, and the tokens of the certain sentence condition each other. To be precise: each token is conditioning the following one. It follows that the ordering of the tokens of sentences have to be respected. In addition, the set of tokens of a sentence has to be stored together in a grouped fashion. Another example is the creation of relation candidates which takes all the pairs of entities of one sentence into account. We used the data structure of *RapidMiner* and we developed mechanisms to respect the particular circumstances of *Information Extraction* tasks.

7.2 Information Extraction Plugin

The process for *Information Extraction* tasks also can be split in the four phases which are already presented for usual *Data Mining* tasks. These phases and the according operators to be used in each phase are described in this section.

7.2.1 Retrieve

The process to retrieve datasets into *RapidMiner* is crucial for *Information Extraction* purposes because of the circumstances we presented in Section 7.1.1. We present two particular approaches to retrieve data. The first of these approaches should be used if the data is available in form of document files, for example PDF files. The second approach is a loading mechanism based on well-known data formats like the one of the CoNLL 2003 shared task on NER[1]. The resulting dataset contains an additional special attribute (*batch*-attribute) which groups the examples. In addition to the grouping the sequential ordering of the examples is respected by the operators of the plugin. Table 7.2 shows the same dataset as presented in Table 7.1 after being converted into m groups (probably sentences). We will present two possible ways to convert documents into example sets in order to use them in *RapidMiner*.

[1] `http://www.cnts.ua.ac.be/conll2003/ner/`

ID	batch	Label	Att_1	Att_2	...	Att_d
1	0	true	2	'Felix'	...	5.4
2	0	false	3	'goes'	...	7.1
...	...	...	...	...	...	...
n	m	true	1	'Detroit'	...	2.3

Table 7.2: Spreadsheet data structure internally used by the *Information Extraction Plugin*

Retrieve via Document

It is important to respect the structural characteristics of datasets consisting of textual data as presented in Section 7.1.1. The *Text Mining Extension* of *RapidMiner* already offers the possibility to retrieve data which is available as a document. Although the structure of these documents in later steps is shattered by the *Text Mining Extension*, the mechanism for retrieving documents can be used to load documents into *RapidMiner*. The *Text Mining Extension* uses a special class for handling documents: the *Document*-class. This class stores the whole document in combination with additional meta-information. In the case of text mining the document is split into unique tokens which are finally used to classify the complete document. For *Information Extraction* purposes we would like to tokenize the document and to preserve the order of such tokens, therefore, we implemented tokenizers which are able to process example sets extracted from the *Document* classes. The application of these tokenizers result in a spreadsheet containing the tokens in the particular order as they have been found in the document. Each token contains a certain number indicating from which general unit it has been created. Each word-token of a particular sentence, for instance, contains the number of the sentence, whereas each sentence-token of a document contains the number of that document. The *Tokenizer*-class can be easily extended to create own tokenizers. Figure 7.3 shows a process containing two operators of the *Text Mining Extension* (the two leftmost operators) and two operators (the two rightmost ones) of the *Information Extraction Plugin*. The process loads a document, converts it into an example set containing an example which holds the complete document-text and the two tokenizers are splitting the text into multiple tokens (examples). The third operator splits the text into sentences, and the fourth operator splits the sentences into words. After having finished the process the resulting dataset consists of examples holding one word each. Additionally, the words are containing sentence numbers allowing to access all the words of a particular sentence.

Retrieve via File

Although *RapidMiner* already offers many operators (e.g., the *Read CSV*-operator) to retrieve datasets contained in spreadsheet-like files, the specific structure of certain datasets containing documents necessitates a particular operator. Datasets like the one for the CoNLL 2003 shared task on NER which is well-known in the NER community

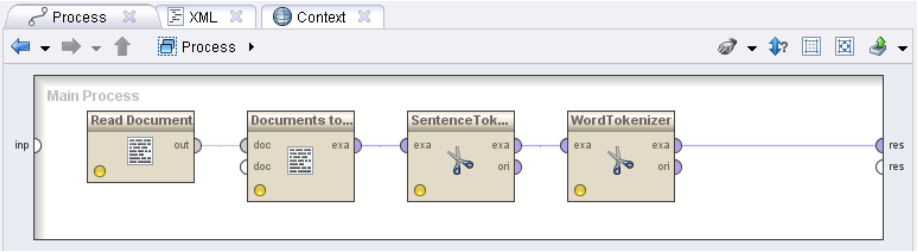


Figure 7.3: Retrieving a document for *Information Extraction* in *RapidMiner*

are already available as tokenized documents. Comparable to csv-files the datasets contain a token each line whereas the tokens additionally contain features which are also stored in the same line. The main difference to ordinary csv-files is the fact that sentences are split by empty lines. It follows that after the tokens for one sentence an empty line is located. Although the *Read CSV*-operator of *RapidMiner* can be adjusted to neglect such lines, we developed an own retrieve operator which respects the empty lines to distinguish between distinct sentences. We therefore use a special attribute containing the sentence numbers. These special attributes later can be used by other operators which work on complete sentences (like CRFs, for instance).

7.2.2 Preprocessing

Preprocessing is crucial for *Information Extraction*. In contrast to traditional *Data Mining* tasks, the examples/tokens in *Information Extraction* tasks are initially not providing any information except the tokens itself. The tokens have to be enriched by attributes to get a more general representation. We will distinguish two different types of preprocessing in the following. The first type of processing is used to enrich tokens for NER by additional attributes. The other type of processing is used for enriching relation candidates for *Relation Extraction*. Both techniques are presented in the following subsections.

Named Entity Recognition

As already mentioned in Section 3.2 it is necessary to enrich tokens for NER by internal and external information. We developed an abstract class for preprocessing operators that can focus on tokens before or after the current token and on the current token itself. The sentence

Felix Jungermann studied computer sciences at the University of Dortmund from 1999 until 2006.

is converted into a spreadsheet as shown in Table 7.3.

Using this abstract class we can access contextual tokens in a relative way. Each token is processed and the abstract class accesses a number of tokens before and after

ID	batch	Label	Att_1
1	0	PER	'Felix'
2	0	PER	'Jungermann'
3	0	O	'studied'
4	0	O	'computer'
5	0	O	'sciences'
6	0	O	'at'
7	0	O	'the'
8	0	O	'university'
9	0	O	'of'
10	0	LOC	'Dortmund'
11	0	O	'from'
12	0	O	'1999'
13	0	O	'until'
14	0	O	'2006'
15	0	O	'.'

Table 7.3: Spreadsheet representation of a sentence.

the current token. The number of tokens to access before and after the current token are parameters that can be adjusted by the user. Let the number of tokens surrounding the current token be 2 and the current token shall be example number 10: 'Dortmund'. In that case the preprocessing method would also access the 8th, 9th, 11th and 12th token in addition to the 10th token. The most simple way of preprocessing would be to enrich the current token by the surrounding tokens. In this way we can bring the context of particular tokens into the tokens itself. Non-structured models will achieve better performance working on such models because the context is not internally respected by the model but by the feature space the model is using.

Table 7.4 shows the dataset presented in Table 7.3 enriched by two surrounding tokens before and after each token. If a token is at the beginning or at the end of a sentence some of the contextual attributes will contain *null*-values because the tokens of one particular sentence only contain informational units from that sentence.

In addition to the relative contextual tokens to be taken into account another interesting parameter to set for preprocessing operators is the parameter *length*. Some attributes like *prefixes* or *suffixes*, for instance, have a specific length which has to be adjusted by this parameter.

Annotations Another step to be performed during the analysis of NER datasets is the labeling of documents and texts. The number of labeled tokens compared to tokens which are not labeled is sparse for most NER datasets. Additionally, the labels are sometimes spread over multiple tokens. Because of these two reasons it should easily

ID	batch	Label	Att_1	$token_{-2}$	$token_{-1}$	$token_{+1}$	$token_{+2}$
1	0	PER	'Felix'	*null*	*null*	'Jungermann'	'studied'
2	0	PER	'Jungermann'	*null*	'Felix'	'studied'	'computer'
3	0	O	'studied'	'Felix'	'Jungermann'	'computer'	'sciences'
4	0	O	'computer'	'Jungermann'	'studied'	'sciences'	'at'
5	0	O	'sciences'	'studied'	'computer'	'at'	'the'
6	0	O	'at'	'computer'	'sciences'	'the'	'university'
7	0	O	'the'	'sciences'	'at'	'university'	'of'
8	0	O	'university'	'at'	'the'	'of'	'Dortmund'
9	0	O	'of'	'the'	'university'	'Dortmund'	'from'
10	0	LOC	'Dortmund'	'university'	'of'	'from'	'1999'
11	0	O	'from'	'of'	'Dortmund'	'1999'	'until'
12	0	O	'1999'	'Dortmund'	'from'	'until'	'2006'
13	0	O	'until'	'from'	'1999'	'2006'	'.'
14	0	O	'2006'	'1999'	'until'	'.'	*null*
15	0	O	'.'	'until'	'2006'	*null*	*null*

Table 7.4: Dataset of Table 7.3 enriched by contextual tokens

be possible to mark specific tokens somewhere in a document directly and to assign a label to the marked tokens. We implemented an annotator operator which visualizes the dataset as a textual document allowing the user on the one hand to create new labels and on the other hand to use those labels for annotating the tokens. After having used that operator the dataset contains a label attribute carrying the annotations and a formerly defined default value if no annotation is given for a particular token. Figure 7.4 shows a screenshot of the annotator screen containing the text presented already in Table 7.3 and Table 7.4.

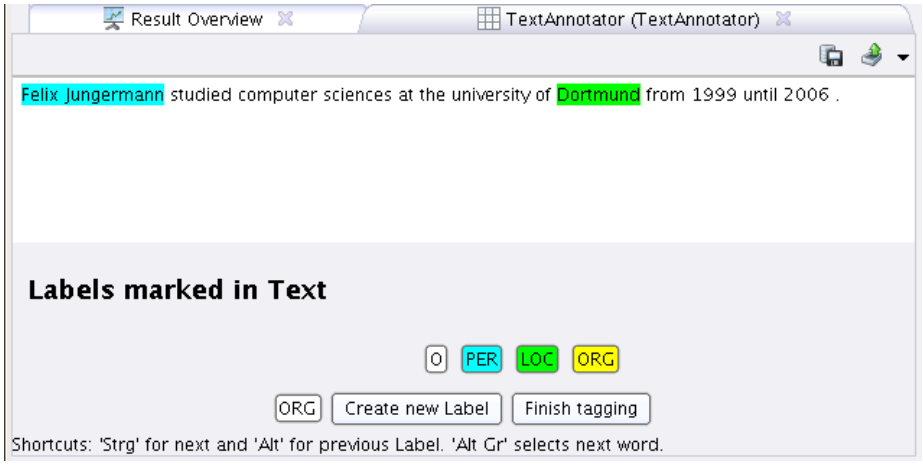


Figure 7.4: Annotating operator for *RapidMiner*

Relation Extraction

We developed particular preprocessing operators working especially for relational learning purposes. Although the flat features developed by [Zhou et al., 2005] can be seen as contextual information, they are only relative to a pair of tokens. Respecting these two tokens is much more complex than as it is for the single tokens for NER. In addition to the flat features *Relation Extraction* heavily relies on structural information like parse trees, for instance. We developed parsing operators to first of all create tree-structured attributes. Additionally, we implemented pruning methods for the creation of more condensed tree-structures.

Tree structures can be represented as nominal values as it is shown in Figure 7.5. It would be a computational overhead to parse these nominal values into tree objects for every time they are needed. We developed a generic form of attribute which allows the storage of every type of *Java* object. This generic object-attribute can be used to work with tree structures in *RapidMiner*. Like for nominal values, the object-attribute is storing a mapping which is an index mapping numerical values to particular unique objects.

(ROOT (S (NP (NNP Felix)) (VP (VBD went) (PP (TO to) (NP (NNP New) (NNP York))) (S (VP (TO to) (VP (VB visit) (NP (NP (DT the) (NN statue)) (PP (IN of) (NP (NN liberty)))))))) (. .)))

Figure 7.5: String representation of the constituent parse tree shown in Figure 2.1

7.2.3 Modeling

We implemented or embedded most of the techniques we presented in this work in *RapidMiner*. The particular models are CRFs (see Section 2.4.2), Tree Kernel SVMs (see Section 5.5), Tree Kernel Perceptrons (see Section 6.4) and Tree Kernel Naïve Bayes Classifier (see Section 6). Most of the publications concerning tree kernel learning are evaluated using the SVM_{light}[2] which is implemented in C. We used the JNI interface of the SVM_{light} of Martin Theobald[3] and embedded tree kernel calculation into it. This $Java\ SVM_{light}$ can be used for the purpose of comparison.

7.2.4 Evaluation and Validation

Some *Information Extraction* tasks need specific evaluation. NER datasets, for instance, sometimes are labeled using the IOB-tagging (see Section 2.3.1). During the evaluation of predictions made on a test set the predicted tokens in some evaluation schemes only are considered to be correct if all the single tokens of an entity consisting of multiple tokens are predicted correctly. *George Walker Bush*, for instance, is an entity of type *PERSON* consisting of three tokens. If only one of these three tokens

[2] `http://svmlight.joachims.org/`
[3] `http://www.mpi-inf.mpg.de/~mtb/svmlight/JNI_SVM-light-6.01.zip`

is not predicted to be of type *PERSON*, the complete entity is incorrectly tagged. Although the tokens are represented as single examples, the evaluation has to be done on the entity-mentions which sometimes consist of multiple tokens. We implemented particular operators respecting this circumstance.

7.3 Comparable Frameworks

Information Extraction is a very popular topic in data analysis. Many software tools for *Information Extraction* have been developed during the last years. The functionalities of most of these tools are similar and they are comparable to the functionalities of the *RapidMiner* extension we presented in Section 7.2.

In this section we want to present two frameworks which have been developed for *Information Extraction* purposes. We will present the differences and similarities of these tools in comparison to the software we implemented for *Information Extraction*. Although many frameworks for *Information Extraction* purposes are available, we only present two of them which are well-known in the *Information Extraction* community. Both tools are open source and they are offering a graphical user interface making them easily applicable to new tasks.

The particular frameworks we are presenting in this chapter are Apache UIMA (Section 7.3.4) and GATE (Section 7.3.5). Both frameworks are based on a general annotation concept which is presented in Section 7.3.2. The idea of such an annotation concept is the maintenance of analysis results by annotations layers. The results of NER, for instance, are represented by an annotation layer which points to the found named entities in a certain document.

Apache UIMA is the implementation of the Oasis UIMA standard [Ferrucci et al., 2009]. This standard, which is presented in Section 7.3.3, "defines platform-independent data representations and interfaces for software components or services called analytics, which analyze unstructured information and assign semantics to regions of that unstructured information." ([Ferrucci et al., 2009], p. 1f).

7.3.1 Historical Frameworks

The two frameworks we want to present at a glance are historical ones. Especially the framework of Grishman which is one of the first *Information Extraction* frameworks already provides a pipelined analysis structure which is apparent in many current *Information Extraction* frameworks, too.

Ralph Grishman is one of the pioneers of *Information Extraction* in his work [Grishman, 1997] he presented the basic principles which should be respected by *Information Extraction* systems. The *Information Extraction* system *Proteus* [Yangarber and Grishman, 1998] was one of the first systems which already was based on a general pipelined

approach. This approach was used in order to get rid of the domain-dependent analysis. *Proteus* is based on the following 7 particular analyzing steps:

1. Lexical Analysis
2. Name Recognition
3. Partial Syntax
4. Scenario Patterns
5. Reference Resolution
6. Discourse Analysis
7. Output Generation

These steps all can be used for every *Information Extraction* task. The certain steps are based on knowledge bases which are exchangeable in order to stay flexible. Nevertheless, for particular domains the knowledge bases have to be set up or generated.

Another pioneer of *Information Extraction* is Walter Daelemans who is one of the developers of *TiMBL* [Daelemans et al., 2003] which was developed at the university of Tilburg. Since 1997 the research group on *Induction in Linguistic Knowledge* is active at that university. The principles of *TiMBL* are based on memory-based learning.

7.3.2 Annotation concept

The enriching of textual data by annotations should not affect the original data. This is, for example, stated in the UIMA standard (see Section 7.3.3). A trivial way of annotating documents is to create tags which are directly embedded into the documents. This would lead to a manipulation of those documents which should be avoided. The manipulated documents could become intractable in different ways which is described later on. In contrast to the manipulation of the original documents, the annotations are contained in an additional view referencing the documents. These types of annotations are called stand-off annotations or stand-off markup. The paradigm for these annotations is based in the markup language community and it is based on the principle not to change the original document but to create additional content. [Thompson and McKelvie, 1997] give three reasons for using stand-off annotations in contrast to manipulating the original documents:

- Stand-off annotations avoid copying the source document which would create redundant content.
- Stand-off annotations easily allow the creation of overlapping levels of annotations.
- Stand-off annotations still can be used if the propagation of the original document is restricted in some way. The annotations, at least, can be published.

[Thomas and Brailsford, 2005] state that the possibility to simultaneously create many annotation levels is another advantage in addition to the creation of overlapping annotation levels.

If, for instance, a document is to be tokenized into single words, a new type of annotation will be created containing a set of annotations referencing the original document. The new annotation types can be seen as layers which are put on the original data. Each annotation contained in the set of one particular layer references a part in the original document. Following this concept it is possible either to work on already created annotations or to process the original document. It is also possible to work on both types of information. Figure 7.6 shows the paradigm of stand-off annotations: on the left-hand side the original document is shown which is referenced by several annotation sets on the right-hand side. The annotations and the document are separated.

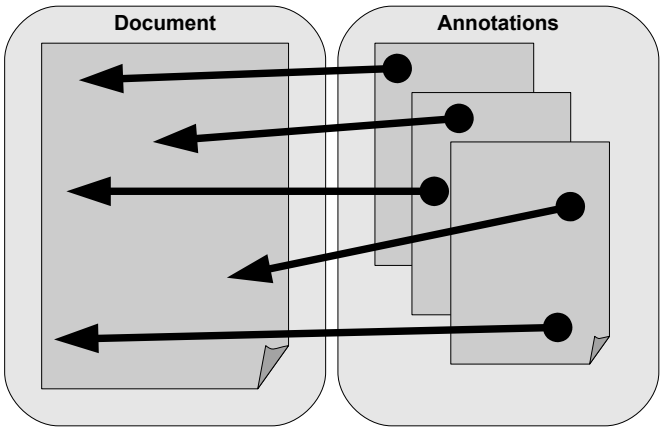


Figure 7.6: Stand-off annotations: the original document is separated from the annotations which only reference it.

Our plugin also follows this annotation concept because an annotation layer in our plugin just can be seen as a certain feature. A process which is extracting named entities, for instance, labels the tokens. These labels can afterwards be used as features for *Relation Extraction* tasks.

7.3.3 The UIMA standard

UIMA is the abbreviation of *Unstructured Information Management Architecture*. The UIMA standard [Ferrucci et al., 2009] was developed for the definition of methodical processing of unstructured information. Unstructured information in this case is any information which is not structured by any semantic. Examples for unstructured in-

formation are an audio file or a document containing free text. A document could of course contain a structured content like a table or a list, but this content is more easily to process and does not need complex methods like *Information Extraction*.

The UIMA standard was developed because popular tools for analyzing unstructured information such as GATE[4], Mallet[5], Open-NLP[6], LingPipe[7], and so on have not clearly defined a standard "enabling the interoperability of analytics across platforms, frameworks and modalities (text, audio, video)" ([Ferrucci et al., 2009], p. 7).

The standard first of all defines the elements which are needed for the *Information Extraction* analysis process:

- artifact
  An *artifact* is a segment of unstructured content. This can be a textual document, an audio or a video segment.
- analysis
  The *analysis* is the process which assigns semantics to the *artifact*.
- analytic
  The concept *analytic* is used for the piece of software performing the *analysis*.
- artifact metadata
  The *artifact metadata* is the resulting semantics created by the *analytic* during the *analysis*.

The goal of the standard is to support the reusability of *analytics* in a platform-independent way.

The *Common Analysis Structure* (CAS) is the joint representation of the *artifact* together with the extracted *artifact metadata* in an XML schema. Two fundamental types of objects are represented by the CAS: the *subject of analysis* (Sofa) holds the artifact, and the set of *annotations* are types of *artifact metadata* referencing regions in the Sofa. A region mostly is defined by a start and end point in the Sofa. The annotations are following the concept presented in Section 7.3.2.

7.3.4 The Apache UIMA Framework

Apache UIMA[8] is an open source framework respecting the UIMA specification (see Section 7.3.3). The framework in its original form is used to plug together analysis engines for the annotation of unstructured content. The framework itself has not been developed to deliver analysis engines. It has been developed to create an architecture allowing to easily construct and deploy analysis engines.

[4] http://gate.ac.uk/
[5] http://mallet.cs.umass.edu/
[6] http://incubator.apache.org/opennlp/
[7] http://alias-i.com/lingpipe/
[8] http://uima.apache.org

Apache UIMA offers several tools [UIMA Community, 2010] which help distributing new UIMA components following the UIMA standard. Plugins for the well-known open source platform Eclipse[9] enable the usage of many of these tools directly in Eclipse.

Plugins of Apache UIMA

We will present some of the plugins which are delivered together with Apache UIMA in this section.

One plugin is the *Component Descriptor Editor*. Users can easily use this plugin for the creation of new descriptors specifying different components of UIMA.

The most important components which need such descriptors are:

- Analysis Engines
- Type Systems
- Cas Consumers
- Cas Initializers
- Collection Readers
- ...

Another plugin delivered with Apache UIMA is the *Collection Processing Engine Configurator*. This configurator allows to process a collection of unstructured information by (several) analysis engines. The resulting CASes can be handled by CAS consumers. These consumers are located at the end of the analysis process and they "consume" the incoming CASes and perform final processes on these CASes – like to store them in the file system.
For the essential analysis of documents the *Document Analyzer* can be used. This program allows to run analysis engines on text files. The results will be written to a given directory. Additionally, the results, which are the original text files containing annotations produced by the analysis engines, are displayed in a visualization component.

The *CAS Visual Debugger* is used to run analysis engines. After finishing the analysis the results are presented in a panel containing three parts (see Figure 7.7). The first part – located in the top left corner of the panel – contains the indexes described in the analysis engine. The second part – located in the bottom left corner – contains further information about the indexes. On the right-hand side of the panel the analyzed text is presented. Users get aware of corresponding annotations by selecting parts of this text. Selecting indexes in the former two containers will highlight the corresponding annotation in the text window.

[9] `http://www.eclipse.org`

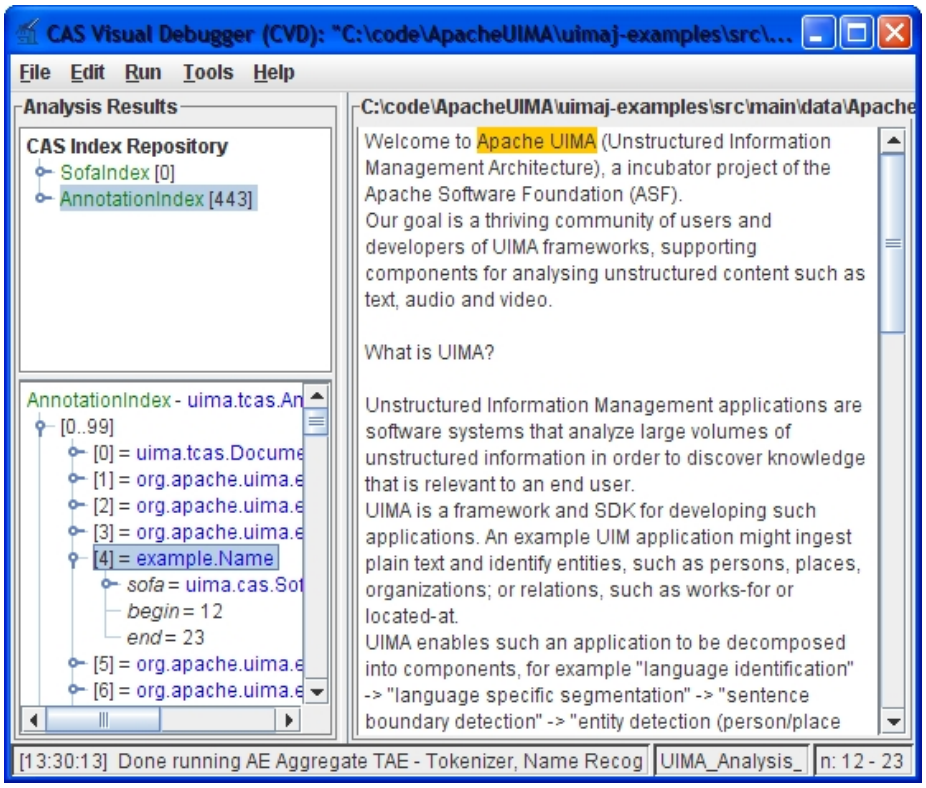


Figure 7.7: CAS Visual Debugger output ([UIMA Community, 2010], p. 53)

The *CAS Editor* is used to visualize and edit CASes. A CAS can be annotated manually or automatically. The automatic annotation is done by using a particular UIMA annotator.

7.3.5 The GATE Framework

GATE[10] is the abbreviation of *general architecture for text engineering*. In contrast to Apache UIMA which has been developed for the analysis of general unstructured contents the GATE framework only focuses on text analysis. Another difference is that GATE does not deliver or respect a standard like it is done by Apache UIMA. Nevertheless, the processing and especially the workflow is comparable to Apache UIMA and to the *Information Extraction Plugin* presented in Section 7.2.

The concept of GATE [Cunningham et al., 2011] is to split the process of natural language processing into more specific parts. The particular resources which are used in the certain parts are split into three different types, namely *LanguageResources*, *Pro-*

[10] `http://gate.ac.uk/`

cessingResources and *VisualResources*.

- *LanguageResources* represent textual corpora like documents, lexicons, and so on.
- *ProcessingResources* represent the methods used for processing textual data. These methods can be manipulating ones like tokenizers or analytical ones like part of speech taggers.
- *VisualResources* are resources that are used for visualization or annotation based on graphical user interfaces.

The set of resources in GATE is called CREOLE which is an abbreviation for Collection of REusable Objects for Language Engineering. Figure 7.8 shows the GUI of GATE.

On the lefthand-side the utilities for the creation of a GATE process are available:

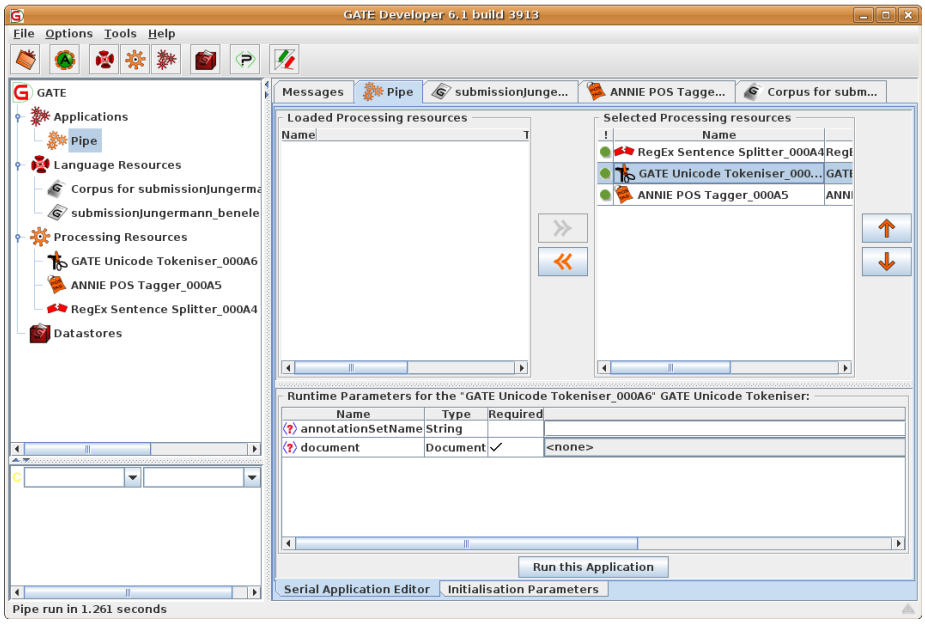


Figure 7.8: The graphical user interface of GATE

Applications, *Language Resources*, *Processing Resources* and *Datastores*. *Applications* are used to define the workflow of the process. In the example shown in Figure 7.8 a simple pipe is used to create a sequence of analyzing operators. *Language Resources* are representing documents or textual corpora which should be processed later on. *Processing Resources* contains all analyzing operators which can be arranged in the particular workflow. *Datastores* contains resources used for storing the created results.

On the righthand-side of the GUI the particular settings for the analyses process can be made. The tabs can be used to navigate to certain objects. In this example a pipe of analyzing operators is shown.

At the bottom of the GUI the parameters for an operator which is focused in the workflow can be set.

Like for Apache UIMA the created results are available as annotations together with the original document. The annotation concept is presented in the following paragraph.

Annotation concept in GATE

The first annotation concept for GATE was based on the concept of TIPSTER [Grishman, 1996]. The data structure based on this concept stores the textual data and the corresponding annotations in a data structure shown in Table 7.5. The text is contained in the first line, and the corresponding annotations are stored given their ID, $Type$, $Span\ Start$, $Span\ End$ and $Attributes$. The ID has to be unique, and it can be referenced by other annotations – the annotations of $Type\ parse$ with IDs $8-11$ are, for instance, referencing formerly defined annotations of $Type\ token$ $(8-10)$ or of $Type\ parse$ $(10-11)$. In the current version the annotations are stored in a graph. The

Text **Cyndi savored the soup.** |0...|5...|10..|15..|20				
Annotations				
ID	Type	Span Start	Span End	Attributes
1	token	0	5	pos=NP
2	token	6	13	pos=VBD
3	token	14	17	pos=DT
4	token	18	22	pos=NN
5	token	22	23	
6	name	0	5	name_type=person
7	sentence	0	23	constituents=[1],[2],[3],[4],[5]
8	parse	0	5	symbol="NP",constituents=[1]
9	parse	14	22	symbol="NP",constituents=[3],[4]
10	parse	6	22	symbol="VP",constituents=[2],[9]
11	parse	0	22	symbol="S",constituents=[8],[10]

Table 7.5: Exemplary data structure used by TIPSTER ([Grishman, 1996], p. 20)

annotations can be seen as the arcs between the beginning and the end of the annotation referenced in the document.

7.4 Summary

We presented the *Information Extraction Plugin* in this chapter. The plugin is an extension to the open source framework *RapidMiner*.

After having presented the graphical user interface of *RapidMiner* we showed that most of the *RapidMiner* processes can be split into four particular phases. The first phase is the Retrieve-phase in which datasets are loaded into the *RapidMiner* process. The second phase is the Preprocess-phase in which the feature set of the dataset is cleaned, enriched or manipulated in order to later on receive better learning results. The Modeling-phase is the third phase. The datasets are used for learning in this phase. The learning results are put into so called models to make them usable in the further processing. The fourth and final phase is the Evaluation-phase. The learned models are evaluated to find out how well the models will perform on other datasets than the set used for training these models.

Compared to traditional *Data Mining* tasks we have shown that those phases are also apparent for *Information Extraction* tasks. Due to the spreadsheet data structure internally used by *RapidMiner* we developed a representation of datasets for *Information Extraction* based on that data structure. In addition, we implemented several operators helpful and needed for *Information Extraction* purposes.

We presented *Information Extraction* systems which are comparable to the *Information Extraction Plugin*. These systems, in particular GATE and Apache UIMA, are based on a similar annotation concept. This concept has been developed not to manipulate the document itself. It can be seen as a layer which is put on the original document containing additional information. Although GATE and UIMA are comparable to our plugin in cases of *Information Extraction* analysis, they are not similar.

Our plugin directly allows to rapidly use *Information Extraction*-results in *Data Mining* tasks because of the close collaboration between *Information Extraction* and *Data Mining* techniques. In the next chapter we will present particular real-world tasks which are a good argument for such collaboration. The plugin is easy to extend which makes it to a powerful toolbox for the significant evaluation and validation of (new) *Information Extraction* methods.

Chapter 8

Applications

In this chapter we present three particular applications which are collaborations of *Information Extraction* and *Data Mining* techniques. The first application presented in Section 8.1 is an information management system for the German parliament's domain. We show that after having extracted particular entities out of the documents, it is possible to gain further knowledge of events the entities form. These events furthermore can be analyzed by *Data Mining* methods to achieve more insights concerning these events.

The second application is another information management system which is presented in Section 8.2. Informational units concerning companies have been extracted from the WWW for the creation of a company network. Additionally, the names of the particular companies are used to crawl a dataset from the WWW. This dataset consists of sentences which contain at least two company names. We used this dataset for *Relation Extraction*. The resulting relations are put in the company network. An intuitive graphical visualization of such network allows to rapidly gain additional information concerning the entities which are present in the network.

The third application which is presented in Section 8.3 a network of related entities can be analyzed only by focusing on the relations between particular entities, instead of analyzing the entities itself. The network of related entities can be given or it can be created as a post processing step after *Relation Extraction*.

8.1 The German Parliament Application

Several information systems make available large collections of documents through the Internet. Their documents can not only be retrieved by a search engine, but also by a built-in retrieval service based on the structuring of the content. To the user, the making of the structures is hidden. The structure is presented in terms of categories among which the user might choose in order to navigate to the desired document. Although search for documents becomes more focused and user-driven, the user needs to understand the categories. Having a particular question in mind, the user is guessing under

which heading she might find a relevant document. Moreover, the user needs to read the document in order to determine the answer to her question.

Some systems offer full-text search so that snippets of text are returned which include the keyword of the query. This eases already the burden of reading, but still the user needs to compose the answer out of some text excerpts. Again, the keyword needs to be chosen carefully in order to receive the right parts of relevant documents.

In contrast to the retrieval of documents, the Message Understanding Conferences (MUC) focused on the extraction of structured information from natural language texts. Event extraction means to fill in the slots of a frame with named entities (NE) of the appropriate type, e.g., person, location, organization or temporal and numeric expressions (cf. the more recent work [Aone and Ramos-Santacruz, 2000]). Hence, NER became a subtask in its own right within MUC-6 and MUC-7 [MUC, 1995, MUC, 1998]. Methods ranged from linguistic rules over pattern-based approaches to machine learning techniques. Linguistic knowledge is not only exploited by the hand-written rule or pattern-based extractions, but also when applying learning algorithms. For instance, part of speech (POS) tagging delivers class, case and number features of words, dictionaries classify known instances of NE types, and tagged texts allow to retrieve the context of NEs which becomes additional features to the word in focus. A knowledge-poor approach has applied machine learning to construct the required linguistic resources (e.g., name lists, gazetteers) in a bootstrap manner when learning NER [Roessler and Morik, 2005]. Tagging documents with NER already offers some services to the user, namely highlighting words or phrases in the text. This might help the user to selectively read only the relevant parts of a long document.

Currently, the restriction to NER is being dropped and approaches towards event extraction are undertaken, anew. *Relation Extraction* (see Chapter 5) aims at recognizing semantic relations between NEs, e.g., interactions of proteins [Blaschke and Valencia, 2001]. Again, the hand-written rules were followed by learning approaches, first by learning the extraction rules [Bunescu et al., 2004]. Syntactic knowledge was used by patterns for the extraction of relations [Hahn and Romacker, 2000] and syntactic dependency trees were used as features for probabilistic learning [Katrenko and Adriaans, 2007]. Learning approaches outperformed the hand-written ones. Relation learning removes irrelevant occurrences of NEs, selecting only the ones in the relation of interest. Hence, readers are confronted with a smaller number of text excerpts.

Event extraction is similar to *Relation Extraction*, but usually events contain more slots than relations have arguments. Several definitions are possible. Here, we simply define relations as parts of events and the relation learning in the way of [Katrenko and Adriaans, 2007].

Definition 24. *Given a set of documents D and an n-ary relation schema R with arguments $A_1, A_2, ..., A_n$, find instances $r(x_1, x_2, .., x_n)$ with $x_1 \in dom(A_1), x_2 \in$*

$dom(A_2), ..., x_n \in dom(A_n)$ *in* D.

Typically, the relation r is represented in natural language by a verb, and the domains of arguments can be constrained by the case of a noun and a NE type. Typically, the arity is small, $n \leq 4$. We could write the form of events similarly, just allowing more complex arguments. Where a relation instance has just a (possibly composite) word as argument, an event may have a NE, a phrase, a relation, or a reference to another event as argument. In order not to confuse the reader, we use another notation for events.

Definition 25. *Given a set of documents* D *and a schema* E *defining slots* $S_1, ..., S_n$ *as elements, find instances* $< e >< S_1 > ... < /S_1 >, < S_2 > ... < /S_2 >, ..., < S_n > ... < /S_n >< /e >$ *in* D.

Typically, the schema corresponds to an XML schema, the slots correspond to XML elements, and instances are tagged text. Since XML elements can be structured themselves, relations can become slots of an event. The name of the schema corresponds typically to a noun, indicating the type of the event. Event extraction allows to build up a database. This, in turn, allows to do analyses ranging from simple queries over statistics to knowledge discovery (*Data Mining*). Although not being comparable to a natural language dialog system, more service is offered to the user.

In this Section, we propose to combine IR, NER, *Relation Extraction*, event extraction, and *Data Mining* in order to offer more services to users. We illustrate our framework by the application of the German Parliament document collection. The services we are aiming at are introduced in Section 8.1.1. The IR techniques and how we are using them for answering simples questions and as a basis for following steps are presented.

The website of the German parliament (`http://www.bundestag.de`) is an excellent example of a web-based information system. It is structured according to the categories: parliament, members, committees, documents, knowledge, European Union, international, and visitors. To each category, a number of documents with links to other content is stored.

In particular, all plenary sessions are documented, from the 8th period until today (16th period). Also the printed papers which form the basis of discussions and decisions in plenary sessions, the recommendations of committees, small and large interpellations ("kleine Anfrage", "Anfrage"), legal proposals, are available as well as information about the members of parliament. We are focusing on the document collection, here. There is an information system, DIP21 (`http://dip21.bundestag.de`), which already offers some services to process the documents.

These services seem to merely use an index over the given documents. The documents are available in PDF format (mostly), the pages of the members of parliament are written in HTML. The identifier numbers of printed papers and plenary sessions need to

be explained. For each plenary session there is an agenda, where to each of its points printed papers form the basis. In the plenary session, a topic is called by the numerical identifier of the respective printed paper. For instance, a committee might have decided to recommend the rejection of a request. This means, that there is a printed paper with, let's say, ID=16/5540 which proposes something, i.e., is a *request*. The committee's *recommendation* to reject the proposal has another number, e.g., ID=16/5561. In the plenary session, the parliament decides about the recommendation 16/5561. If the recommendation is accepted, this means, that the request 16/5540 is rejected. If the recommendation is rejected, the request must again be discussed (and changed) in the committee yielding another paper of type *changed request* with a new number, e.g., ID=16/6102. For users who just want to know, for instance, whether the old age pension increases, or not, it is cumbersome to follow all these references through all the documents. Hence, for users it is not easy to actually receive answers to their information needs, although all information is publicly available.

Such a situation is quite common for web-based information systems. We use the one of the German parliament, because it is publicly available and we, as citizens, are allowed to analyze the documents. The goal is to enhance the services for users by moving more into the documents.

While the German Parliament web site is typical with respect to the services it offers, it is rather exceptional with respect to the language. Analyzing word frequencies, we receive the typical power law distribution. However, the word length is even for German extraordinary. There are words like
"Konsensfindungserleichterungsmassnahme",
"Fernstrassenbauprivatfinanzierungsgesetzesänderungsgesetz", or
"Grundstücksverkehrsgenehmigungszuständigkeitsübertragungsverordnung".
Our corpus of the periods 13 - 16 contains 50,363 documents with about 470,000 different words. This shows that the political language is extremely challenging for extraction purposes.

8.1.1 Services

Services of the parliament's web-site are currently restricted. We would like to offer more to users. There is a variety of questions which people like to ask assuming different answers. For instance, the following questions were raised by a group of our students:

1. How many members of parliament have children?

2. How many of the female members of parliament have children?

3. Which requests (which plenary sessions) dealt with the relation between Germany and Turkey?

4. Which decisions were related to students from foreign countries?

5. Under which aspects have the relationship between Germany and Turkey been discussed?
6. Which party has signed the most requests?
7. Which party has the most requests rejected?
8. How many changes were necessary before the law for unemployed (Hartz law I, II, III, IV) has been decided?
9. Which parties or members of parliament were against the student registration fees?
10. Given the three levels of additional income, is there a correlation between the party and members with the highest level of additional income?
11. Which events were used as argument in favor of restricting civil rights in favor of enhancing the state's security?

The questions demand for different types of results.

- Some questions just ask for excerpts of documents, questions 3 and 4 are examples of this type. The answer set can be determined by full-text search and some post-processing.
- Some questions ask for counts of entities which are easy to recognize, questions 1 and 2 are examples of this type.
- Some questions ask for statistical analysis of relations, question 9 is an example of this type.
- Some questions ask for counts of events, questions 7, 8, 9 are examples.
- Some questions ask for statistical analysis of events or machine learning, questions 7 and 10 are examples.
- Some questions demand text understanding, questions 5 and 11 are examples. We exclude these difficult questions, here.

The architecture of our system for targeted *Information Extraction* is shown in Figure 8.1. In contrast to only using the *Information Extraction Plugin* of *RapidMiner* which is presented in Chapter 7, we want to show the benefit of the combination of IR, IE and *Data Mining*, in this section.

The graphical user interface offers menus for simple questions, questions about relations or events, and questions requiring some statistical analysis or learning. Documents from the web-based information system are retrieved, transformed from PDF into ascii format, and stored in the document base, which is then indexed. The simple questions are answered on the basis of the document's index. Some annotations are quite easy and can be achieved on the basis of simple patterns. The annotated documents become stored in the document base, as well. Questions about easy to recognize

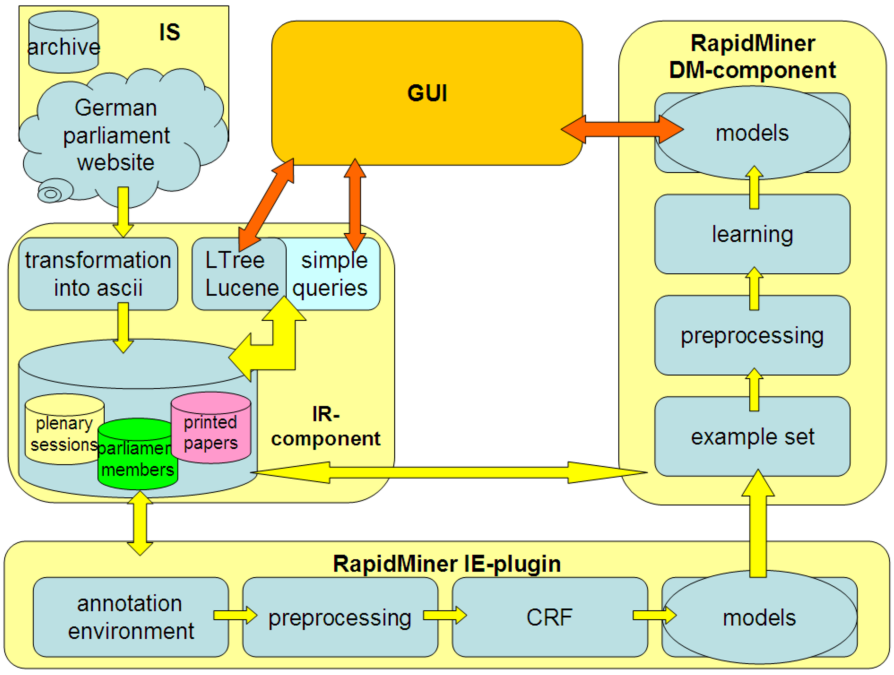


Figure 8.1: The system design for targeted *Information Extraction* using *RapidMiner*

entities are answered on the basis of these data. Other annotations according to NE require to learn the NER. This is performed by the IE-plugin. Again, annotated documents are stored for further use in the document base. In order to learn about relations and events, the regular *RapidMiner* with designed experiments is integrated.

This architecture is quite general and a blueprint for a class of applications, namely enhancing web-based information systems. Depending on the application is the particular set of pre-processing operators, the regular expressions or other patterns to be used, and the particular sets of relation and event schemata. Due to *RapidMiner*'s flexibility, it is easy to adapt the system to a new domain.

8.1.2 Targeted Information Retrieval

In this section we explain how we prepare for answering various questions (see Section 8.1.1) using our system or particular components of it.

Referring to Figure 8.1 one can see that our system consists of three components which are now presented in detail.

IR-Component

Many questions already are to be answered by IR techniques. To get these questions answered is on the one hand a nice benefit for users and on the other hand relatively easy to offer.

The system first of all extracts all the plenary session documents and printed papers and stores them for later use in ascii-format. Additionally the websites of the members of parliament are extracted in order to fill personal event templates which contain names, birthday, birthplace, family status, children, education and so on. In addition we build up an index of all the documents using the open-source indexing environment *lucene*[1].

Furthermore, we extract the information of every printed paper – similar to the approach using for the members of parliament – into an event-like template for accessing the information easily. This extraction by now is done using trigger-words like, e.g., "geboren in" (born in) for birthplaces.

The printed papers follow a special formatting which helps to extract the number of the printed paper, the members of parliament, the parties involved, the date, and an abstract of the printed paper. These are the slots of the printed-papers-template.

The member-of-parliament-templates, the printed-paper-templates, the original (complete) documents and the index over all documents form our repository which is ready to use for answering questions 1, 2, 3, 4 and 6. The data of the repository is used for further analysis in the IE-component and in the *Data Mining*(DM)-component in order to answer more difficult questions.

Our system contains all printed papers of the 14th, 15th and 16th period as templates currently. Requests like *'How many requests has the party SPD signed?'* or *'Show me all the printed papers of type "Gesetzentwurf"!'* can be easily processed.

Experiments using the IR-component

As an example we extracted all the requests and its corresponding recommendations of the committees either to reject, to accept or to depose the proposals. For this task, one has to look for all the printed papers of type *request*. Then, one has to extract all the *recommendations* considering these requests. Finally, one has to search snippets according to the request numbers in the recommendations to find the decision of the committees. Users now can easily look at relevant excerpts of large documents, focused on particular recommendations to see what happened to a request.

The recommendation is a relation embedded into the event *request* consisting of the outcome (i.e., accept, reject, depose), the number of the recommendation and the recommending committee (e.g. committee of justice). The following, for instance, is a

[1] `http://lucene.apache.org`

positive recommendation of the committee of justice (in German: 'Rechtsausschuss'):
$recommend(< printed\_paper > 14/358 < /printed\_paper >, < recommendation > accept < /recommendation >, < committee > Rechtsausschuss < /committee >)$

IE-Component

The IE-component is profitable for users when, for example, highlighting special objects (NEs) in the texts for achieving a better user-guidance. It is necessary for other processes for which it captures trigger-words or patterns. Additionally, the IE-plugin delivers the *Relation Extraction*.

As learning and *Information Extraction* environment we use the open-source software *RapidMiner*[2] [Mierswa et al., 2006] which is implemented in Java. *RapidMiner* offers several interfaces to other systems and an elegant mechanism to plugin specialized add-ons. We have developed such a plugin for information extraction (IE – see Chapter 7 for more information). Converting natural language to examples means that every word occurrence becomes one example which has a bunch of attributes. Preceding and following words become attributes of the current word. Corresponding to the learning task (in this case NER), a label attribute is added (the NE).

Using the IE-plugin for *RapidMiner* one must first of all define a dataset (text). Then, one can use various pre-processing-operators. Corresponding to McDonald's definition of internal and external evidence ([McDonald, 1996]) the categorization of words depends on internal features – extracted directly from the form of a word – and external features features – extracted from the context of a word. The pre-processing operators make use either of internal or of external evidence. Considering current work on sequence labeling ([Leaman and Gonzalez, 2008]) one sees that there is a set of features which delivers good results. These features consist of character n-grams, prefixes, suffixes, word-generalizations and so on. The external evidence is used by encoding the knowledge of surrounding contexts into attributes of words. So particular words also have attributes corresponding to words in front or behind the word. Most of the features presented in [Leaman and Gonzalez, 2008] also are implemented in our IE-plugin.

After the pre-processing steps, one can use multiple machine learning methods. In case of sequence labeling tasks, [Nguyen and Guo, 2007] showed that the structured support vector machine (SVMstruct [Tsochantaridis et al., 2005]) delivers the best results compared to other methods like hidden Markov models and especially conditional random fields (CRF) ([Lafferty et al., 2001]). But [Keerthi and Sundararajan, 2007] showed that the better performance of the SVMstruct compared to CRFs is only due to different internal features used for learning. Their comparison showed that the performance of both methods are nearly the same. Hence, both methods can well be used. Since the SVMstruct is not yet implemented in Java, we integrated CRFs into *RapidMiner* – we use MALLET [McCallum, 2002] as basis for our CRF-operator.

[2] formerly known as YALE

Relation Experiment using the IE-plugin

On the basis of the snippets according to the request numbers in the recommendations, we have extracted the relation $recommend$ with the arguments $reject$, $accept$, and $depose$. We present the results in Table 8.1.

Found requests	1.935
Requests without recommendation	680
Requests with recommendation	1.255
Recommendation 'reject'	794
Recommendation 'accept'	251
Recommendation 'depose'	44
Recommendation not extractable	166

Table 8.1: Recommendation extraction for all requests

NER Experiments using the IE-plugin

We made an exemplary NER-experiment using a manually annotated document of the plenary sessions. The document contained about 2.700 sentences containing nearly 56.000 words or in other words 'tokens' – points for example are tokens, too. Table 8.2 shows the numbers of NEs and the performance (measured with *f-measure*) which was achieved on the dataset using a ten-fold-cross-validation. The results show that some NEs are easy to spot like, for instance, name and party. In contrast, the recognition of a plenary session and of printed paper numbers is difficult for NER. However, the numbers and the dates do not need to be extracted by NER, but can be easily set by regular expressions during pre- or post-processing. Usually, NER tasks are recognizing institutions, locations, and organizations. Without background knowledge, organizations are hard to detect.

	name	**institution**	**party**	**person**	**location**
count	1.115	823	703	422	409
f-measure in %	90,3	87,9	96,1	68,4	68,2
	organization	**reaction**	**date**	**p.p. no.**	**p.s. no.**
count	238	120	78	45	18
f-measure in %	52,1	55,4	60,3	37,7	36,7

Table 8.2: NEs in the examined document (printed papers (p.p.), plenary session (p.s.))

DM-Component

The innovation of our system is the opportunity to use extracted events as input for *Data Mining* experiments. It is nice to get questions answered like *'How many re-*

quests are recommended to be rejected?', but *Data Mining* goes beyond that. It offers the opportunity to get to know why or under which circumstances a request has been rejected. We are using *RapidMiner* for *Data Mining*. According to the example of former paragraph we converted all found requests into examples as an input for a *Data Mining* task. The event *request* has the form as shown in Figure 8.2.

There are only 5 parties in the parliament. The supporters of a request can be a

```
<e type = "request">
  <printed_paper>14/138</printed_paper>
  <recommend>
    <printed_paper> 14/358 </printed_paper>
    <recommendation> accept </recommendation>
    <committee> Rechtsausschuss </committee>
  </recommend>
  <party 1> SPD </party 1>
  <party 2> BUENDNIS90/DIE GRUENEN </party 2>
  <party 3> null </party 3>
  <party 4> null </party 4>
  <party 5> null </party 5>
  <multiMOP> false </multiMOP>
  <justParty> true </justParty>
  <government> false </government>
</e>
```

Figure 8.2: An event of type *request* in XML-format

number of members of parliament (MOP), some parties, or the complete government. The slot $< party1 >$ indicates the party initiating the request. The shown example states that two parties together have signed the request 14/138 and the request got a positive recommendation by the committee of justice. This event is transformed into a data set for learning, where the slots become attributes. The recommend arguments for the decision are encoded by numbers: 0 for no recommendation, 1 for accept, 2 for reject, and 3 for depose. These arguments are the class labels for learning. A simple decision tree learner delivers the results presented in Figure 8.3.

The possible label allocations are given in brackets at the leaves of the learned tree. Using this little number of attributes one would not think of getting interesting results, but there are some: if the attribute *PARTY 2* is null (just one party is signing the request), one can see at the leaf with *PARTY 1 = PDS* that most of the requests of this party, the leftist party, are recommended to be rejected. This answers question 7 from the former paragraph. A ten-fold-cross-validation over the requests ended up in *67,1 % accuracy* to predict the label.

```
Tree

PARTY 2 = CDU/CSU
|   PARTY 1 = SPD:
0 {0=37, 2=0, 1=4, 3=0}
|   PARTY 1 = PDS:
2 {0=0, 2=2, 1=0, 3=0}
PARTY 2 = BUENDNIS 90/DIE GRUENEN
|   Just Parties = true:
0 {0=95, 2=1, 1=44, 3=3}
|   Just Parties = false:
1 {0=75, 2=1, 1=155, 3=3}
PARTY 2 = null
|   PARTY 1 = SPD:
2 {0=44, 2=164, 1=1, 3=9}
|   PARTY 1 = PDS:
2 {0=171, 2=413, 1=9, 3=20}
|   PARTY 1 = CDU/CSU:
2 {0=46, 2=108, 1=1, 3=3}
|   PARTY 1 = BUENDNIS 90/DIE GRUENEN:
0 {0=141, 2=102, 1=2, 3=6}
|   PARTY 1 = CDU:
2 {0=0, 2=1, 1=0, 3=0}
|   PARTY 1 = FDP:
0 {0=2, 2=0, 1=0, 3=0}
|   PARTY 1 = DIE LINKE.:
0 {0=1, 2=0, 1=0, 3=0}
PARTY 2 = SPD
|   Just Parties = true:
0 {0=22, 2=0, 1=3, 3=0}
|   Just Parties = false:
0 {0=31, 2=0, 1=23, 3=0}
PARTY 2 = DIE LINKE.:
2 {0=0, 2=1, 1=0, 3=0}
```

Figure 8.3: A decision tree which is trained on extracted *request* events of the German parliament

8.1.3 Related Research and Conclusion

The first trainable systems for event extraction were based on wrapper induction (WI) [Kushmerick et al., 1997]. WI-based systems are processing a huge amount of already structured data – typically labeled by HTML-tags. By learning which tags *wrap* the interesting data, WI-based systems are capable of filling event slots after having found special tags. These systems work well if and because the HTML-syntax is well-formed and thus offers kinds of slots, already.

A more specific problem is the analysis of semantic roles and its corresponding relations. [Yangarber and Grishman, 2000], for instance, tried to extract patterns for the relation 'position statement' automatically – given a little seed example set. Their results show that this automatic pattern extraction works as good as hand-crafted ones, being not as time-consuming. Actually 'relational search' is used to extract relations between entities automatically without using seed-examples [Cafarella et al., 2006] in an unsupervised and highly scalable manner. Therefore any object-string-pair which occurs in a neighboring context in a (huge) document collection is extracted with its corresponding relation – which just is the text between the two object-strings. An *extraction graph* – consisting out of objects and relations between these objects – is built up in order to answer queries which cannot be answered easily by traditional search-engines. [Popov et al., 2003] present a similar system which uses an ontology to annotate semantic categories (NEs) in texts and in turn uses the annotated texts to update and advance its ontology.

Related systems like the one from [Popov et al., 2003] or the one from [Cafarella et al., 2006] are powerful for *Relation Extraction*. It should be investigated if these techniques are applicable for the event- and relation-extraction in our system, and whether it would improve our extraction of events.

Our *Relation Extraction* is currently quite heuristic. Here, more sophisticated approaches could extract relations from the plenary sessions. It is extremely hard to extract the opinion (in favor, against) from speeches in parliament. The language used is most elaborated and full of subtle irony. Currently, we restrict ourselves to the ballot results where the language is more standardized.

Similarly, event extraction from the plenary sessions is currently referring to decision making events (recommendations, decisions, votes, passes of a law), interjections, and the extraction of the list of speakers in a plenary session dealing with a certain topic. This is already challenging, since the speeches can be nested, and the topics are called in various ways. Far more difficult is to recognize the position in a speech. For instance, does a speaker argue in favor or against nuclear plants? Extracting the opinion of a politician at several points in time would allow to register changes in the political belief of a member of parliament. However, the difficulty is the extraction of the opinion from text. Hence, we work on events which relate requests (standing for a political position) and politicians or parties. The summarizing text of a request represents the

position and is only interpreted by the reader.

The major focus of this section is the combination of IR, IE, and DM. Indexing services and *Information Extraction* transform a document collection into a set of events and relations which form the basis of *Data Mining*. The efforts of finding relevant paragraphs in the documents, of writing regular expressions for the extraction of simpler entities such as, e.g., document numbers or dates, are turned into operators of the IE plugin of *RapidMiner* and, hence, are available for other applications, as well. The IE-plugin offers these preprocessing operators. It interacts with the IR-tool *lucene*. It offers an annotation tool and runs the CRF NER in a loop of cross validation. Hence, we have not merely shown an application but the development of a system which eases to build an application. Such a principled approach to enhancing services of web-based information systems by event extraction and *Data Mining* is a novelty, as far as we know.

8.2 Company Information Extraction from the Web

In this section we present an application focusing on *Information Extraction* on company related information from the WWW. We extracted seed information from a semi-structured newspaper website containing information about the biggest German companies. These informational units already can be presented in a company network. This network is represented using a visualization which is based on a graph framework (see Section 8.2.3). In an additional step we used these information to crawl the WWW for the extraction of a dataset containing sentences which might contain two or more related companies. This dataset is described in Section 8.2.1. We made several *Relation Extraction* experiments on that dataset. The experiment setting and the results are presented in Section 8.2.2. The extracted relations finally can be embedded in the company network to reveal additional knowledge.

8.2.1 The Merger Relation Dataset

In this section we present dataset which can be used for *Relation Extraction*. It is a dataset a student created during his diploma thesis [Had, 2009]. The dataset contains relations indicating the so called merger-relation. This relation type defines the merging of two companies.

Information about companies and economics are valuable information for the stock market. Extracting these information automatically is of great use in order to gain knowledge about processes leading to the rising of shares, for instance.

An interesting relation in this domain is the merger relation indicating the merging of two companies. The merging of two firms mostly leads to great consequences at the stock market. The dataset is created in the following way.

The 30 German companies indexed in the DAX have been used to crawl documents concerning these firms from the WWW. These documents are used to create a pool of

relevant documents. The sentences containing at least two companies of a list of known company names are extracted for further processing. The sentences are tagged using the IOB-tagging (see Section 2.3.1). The resulting dataset only contains sentences in which company names are tagged. Pairs of companies of the sentences are converted into relation candidates. Especially sentences containing more than two companies are crucial because such sentences are offering more than one relation candidate. Figure 8.4 shows as an example the parse tree of a sentence containing a found merger-relation.

The final corpus contains 1698 sentences which in sum contain 3602 relation candi-

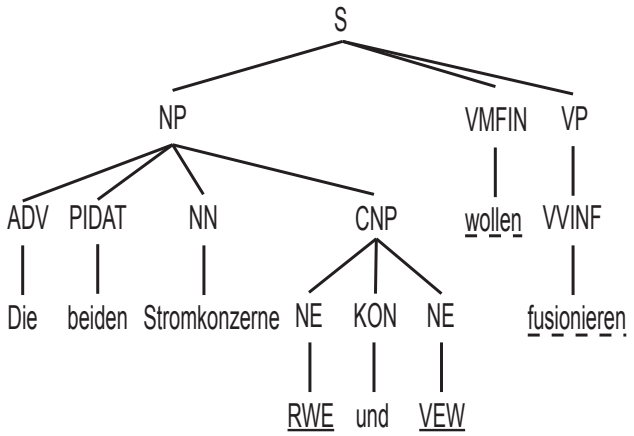


Figure 8.4: A parse tree for a German sentence containing a merger-relation ([Had et al., 2009] p. 2)

dates. Nearly 3000 of these candidates are no merger-relation. 672 relation candidates are tagged as real merger relations. Unfortunately, most of the sentences contain only relation candidates of one label. Only few sentences (98) contain relation candidates of both classes. This distribution is very skewed leading to the fact that the analysis of the sentence itself already delivers well-suited models. The ACE corpus (see Section 6.5.2), for instance, offers nearly 3000 sentences, and more than a half of them contain at least a negative candidate and a true relation.

8.2.2 Experiments

We made several experiments on the merger relation dataset. In the following we present the feature set we used for the particular experiments:

- *baseline-feature set*
  This feature set represents the features as presented in [Zhou et al., 2005]. To additionally use the tree kernel we took the parse tree of the certain sentences the

relation candidates are part of, and we used these parse trees in the feature set, too.

- *word-vector-feature set*
  This feature set only holds the bag of word representation of the sentence the certain relation candidate is extracted from.
- *big-word-vector-feature set*
  This feature set is a combination of the *baseline-feature set* and the *word-vector-feature set*.
- *stem-x-tree-feature sets*
  In this feature set we use the stems of the tokens and embedded them into the parse tree at level x. x is always relative to the preterminals. $x = 0$ leads to replacing the preterminals by stems. This feature set equals the *baseline-feature set* but only the parse tree is different.

All used parse trees are created by the Stanford parser [Klein and Manning, 2002]. We applied 10-fold cross validation using the composite kernel with a parameter setting of $C = 2.4$, $m = 3$ and $\alpha = 0.6$ (see Section 5.4.4).

Performance

The performance achieved by our experiments is shown in Table 8.3. The results are given by the mean on the particular cross validation results together with the standard deviations.

The best values for precision and recall are achieved in the feature set containing

Feature set	Precision	Recall	F-meas.
baseline	33.47 ± **3.9**%	52.27 ± 22.0%	38.64 ± 8.9%
word-vector	36.41 ± 12.2%	69.93 ± 11.6%	45.45 ± 5.1%
big-word-vector	36.83 ± 5.2%	74.86 ± 9.6%	**48.73** ± **3.1**%
stem-replace-tree	31.46 ± 4.6%	**76.03** ± **9.0**%	44.08 ± 4.3%
stem-0-tree	37,94 ± 6.3%	47.90 ± 14.6%	41.79 ± 9.1%
stem-1-tree	**44,33** ± 7.6%	53.42 ± 9.9%	47.51 ± 4.4%
stem-2-tree	36.28 ± 4.0%	62.91 ± 10.9%	45.64 ± 4.6%

Table 8.3: Performance of *Relation Extraction* experiments on the merger relation dataset using ten-fold cross validation.

the parse trees which are enriched by the stems of the tokens. Although the feature set containing the bag of words representation of the sentences results in the best f-measure, the results of the feature set containing the parse trees enriched by stems are promising. The good results using the bag of word representation might be based on the fact that the merger relation dataset contains many sentences which only contain one relation candidate. This makes the task of *Relation Extraction* to a kind of sentence classification for this dataset.

8.2.3 Visualization of Extracted Relations

In [Had et al., 2009] an economic network is presented for the visualization of relations between companies and persons. This economic network has been constructed out of informational units for companies and persons who are active in the board of directors of such companies. A newspaper website has been crawled to gather information concerning some of the most influential companies for the creation of the network. Such information is given in a semi-structured way which makes the extraction easy by using regular expressions. About 2.000 tuples containing information about companies have been extracted. Although some of these tuples only contain trivial information like industry type or address, the greater amount of these tuples contained information about the board of directorate, shareholding, some performance indicators and share owner information. Many of the extracted companies are related to other companies. We assume that a relation will be given if two companies *share* a person. This person, for instance, is member in the board of directorate in both companies.

The resulting graph is visualized using the JUNG-Framework [O'Madadhain et al.,

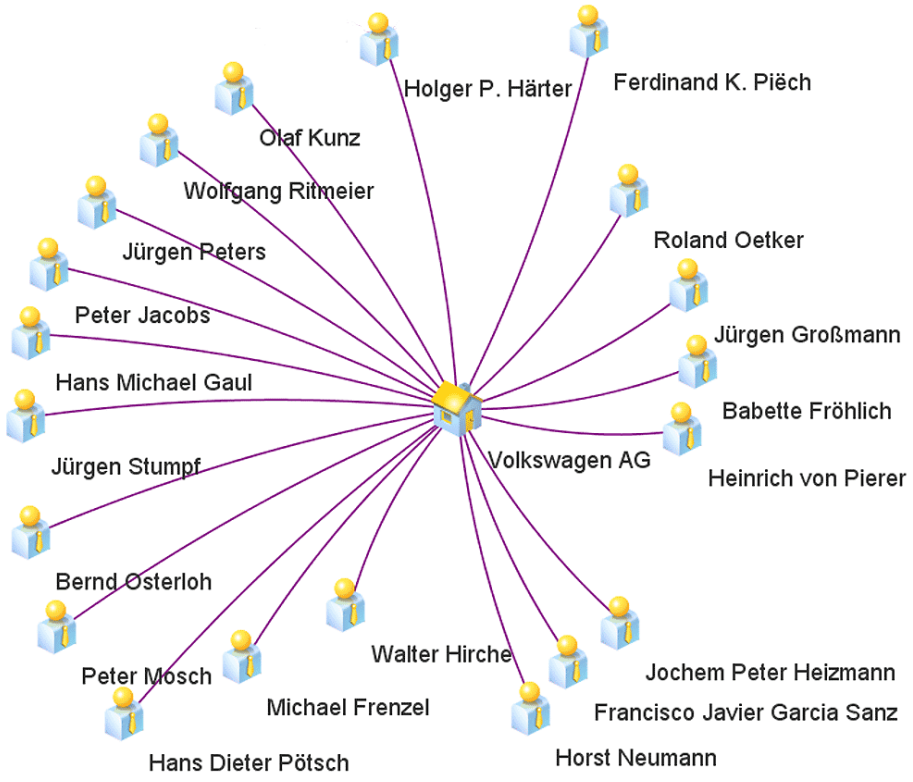


Figure 8.5: A company and all involved persons visualized. ([Had et al., 2009], p. 7)

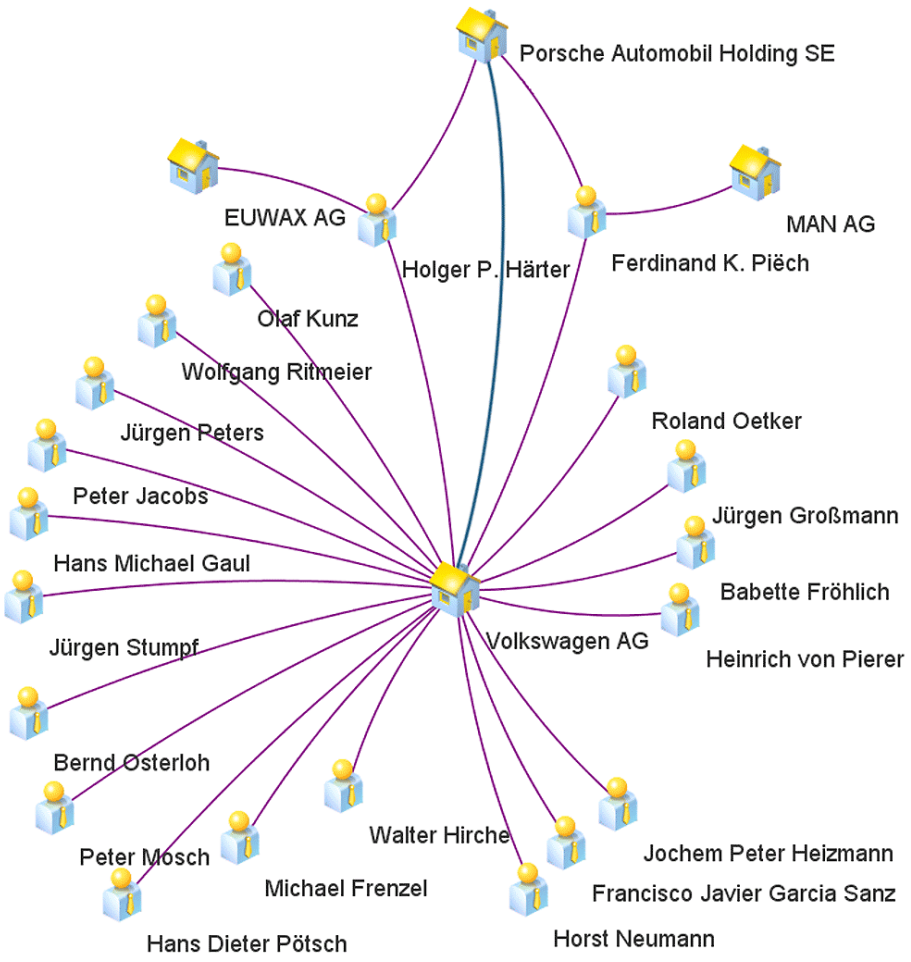


Figure 8.6: Two companies related by a merge. Former relations are revealed, too. ([Had et al., 2009], p. 7)

2003]. A graphical user interface allows the selection of particular companies in order to gather all involved persons. Figure 8.5 shows the extracted persons for the *Volkswagen AG* as an example. Figure 8.6 shows the same company, additionally, a found merger-relation between two companies is visualized using a thicker line between the two involved companies. Having a look on the graph now reveals the fact that two persons are involved in both firms. This fact might generate hypothesis or questions like "Have the merge of the two companies been launched by these two persons?". It becomes clear that the extracted relations will reveal additional knowledge if they are embedded in an already available relation graph.

Visualizing relations by a network makes the information intuitively better understandable for the users.

8.3 Relation Graph Analysis

In this section we will show how to analyze graphs consisting of related entities without using any information of the entities itself. Only the fact that entity i is connected to entity j makes them somehow similar.

Relations between entities do not necessarily have to be extracted to gain insights on the related entities. In modern web frameworks such relations already inherently are given. Those relations can be used to extract groups of similar entities. The similarity of these groups is only based on the relations between the entities in contrast to being based on characteristics of the entities itself. By focusing on the well-known social network *Twitter* we will show that relations inherently are given in a social network and that these relations can be used for clustering the entities of such network. In addition, we show that the permanent evolution of such network can be tackled by stream-based approaches.

8.3.1 Introduction

Social networks like *LinkedIn*, *facebook* and *Twitter* have gained lots of attention during the last years. The interest on social networks mostly is based on the success of *facebook* which inspires other companies like *google* to develop social networks (*google+*), too. The name already states it: social networks are connecting people. Those networks offer their users a platform to communicate and to share different types of information. Users can get connected by establishing friendship relations, for instance. Additionally, users can offer content by chatting and sharing information like videos, music clips, photos, links, and so on.

An interesting question concerning the analysis of social networks is if the network contains groups of similar users. Social networks do not only connect users. They also contain informational entities like messages, photos, links, and so on. This allows not only the finding of groups of similar users, it is also possible to find groups of similar entities.

For the purpose of analyzing such networks we have to have a more formal view on such networks. Every type of content contained in the network shall be seen as an entity. In practical the entities are users, messages, photos, videos, links, and so on. In addition, these entities can be related. A relation can be manifold: a user is a friend of another user, a user posts a photo, a user writes a message. The relations also can connect multiple entities: a user posts a message which is related to another user, or a user marks another user on a photo. We assume that every communication in the network creates a particular type of relation and a particular instance of such relation type.

In this work we certainly focus on the well-known network *Twitter* which offers its users the possibility to create short messages which are propagated to those users which are connected to the particular user. Users can "follow" other users which makes them possible to receive messages created by such users. The messages itself can contain additional content like *urls*, *tags* and other *users* which can be linked in a relation.

Figure 8.7 shows an exemplary message (called *tweet*) as it might occur in the *Twitter*-framework. This particular *tweet* contains information about the author, the message itself, a link and 4 tags. This *tweet* can be represented as 4 four-ary relations between *user*, *message*, *url* and *tag*. To not generate too complex relations the 4 tags are not represented in one relation but 4 relations containing one tag each are created.

Defining a social network as a collection of entities which are somehow related to

Figure 8.7: Example tweet of the *Twitter* platform

each other, results in the definition of a social network which is given in Definition 26.

Definition 26. *A social network is an undirected graph $G = (V, E)$, where V is the set of nodes being the set of entities apparent in the social network and E being the set of edges connecting entities $v \in V$. Entities v_i and v_j will be connected ($(v_i, v_j) \in E$) if v_i and v_j are part of a particular relation of one of the relation types occurring in the social network.*

The particularly possible relation types of the *Twitter* framework are presented in Section 8.3.5 and in Figure 8.8.

Interrelated entities normally are described using a triangular adjacency matrix $\mathbf{A}$ of dimension $d \times d$, where d is the number of entities. Matrix $\mathbf{A}$ normally is sparse

containing w at position $\mathbf{A}_{i,j}$ if $entity_i$ is related to $entity_j$ with weight w, and 0 otherwise. This matrix unfortunately cannot be used to describe n-ary relations with $n \geq 3$ as it only allows the interaction of two entities.

For the description of multi-dimensional relations, containing more than two inter-related entities, tensors can be used like it is presented in several publications [Acar et al., 2005, Acar et al., 2006, Banerjee et al., 2007, Shashua and Hazan, 2005, Kolda et al., 2005, Cai et al., 2006, Wang et al., 2006]. A tensor is a generalization of matrices which contains not only two but n dimensions. A 3-dimensional tensor $\boldsymbol{\mathcal{X}}$ for instance is a cube which can be used like an adjacency matrix to describe relations connecting three types of entities. If for social network analysis a relation connecting *users*, *tags* and *urls* is given, the corresponding tensor $\boldsymbol{\mathcal{X}}$ can be used in the following way:

$$\boldsymbol{\mathcal{X}}_{i,j,k} = \begin{cases} w & \text{if user } i\text{, tag } j \text{ and url } k \text{ are related} \\ 0 & \text{otherwise.} \end{cases}$$

Community Discovery using Tensors

Clustering or the detection of communities on multi-relational data given a tensor representation is done by decomposing the tensor $\boldsymbol{\mathcal{X}}$ into matrices $\boldsymbol{U}^{(i)}$. Each matrix $\boldsymbol{U}$ is representing one particular dimension of the tensor. The matrices are of dimension $\mathbb{R}^{m_i \times k}$, where m_i is the number of entities of dimension i of the original tensor and k is the number of groups or clusters to be created. The parameter k in this connection is a parameter which is set by the user. The created matrices should approximate the original tensor best given the following equation:

$$\boldsymbol{\mathcal{X}} \approx [z] \prod_i \times_{d_i} \boldsymbol{U}^{(i)}.$$

The matrices $\boldsymbol{U}^{(i)}$ can be seen as a mapping of the particular entities to the k clusters. $[z]$ is a super-diagonal tensor connecting all the matrices (see Section 8.3.3). $\times_{d_i}$ is the mode-d product which is presented in Section 8.3.3.

[Allen, 1984, Lathauwer et al., 2000, Harshman, 1970] have presented different types of decomposition techniques like *PARAFAC* (CP) and Tucker3. How well the original tensor has been approximated is measured by a divergence function. [Banerjee et al., 2007], for instance, used the Bregman divergence to evaluate the approximation of their clustering framework. [Bader et al., 2007] tried to find and track informative discussions in the Enron email dataset by applying a CP tensor decomposition approach on the relation (*term*,*author*,*time*). Although both presented approaches are clustering multi-relational data available in tensors, they are only working on one tensor presenting only one type of relation.

METAFAC which is presented by [Lin et al., 2009] is a factorization of multiple multi-relational datasets. Multiple types of relations are stored in multiple tensors which finally are analyzed all together. In detail, the particular relation types share types of entities. This leads to the fact that one particular matrix $\boldsymbol{U}^{(i)}$ is used to approximate

several dimensions of multiple tensors. Finally, a global clustering on all entity types is created respecting all of the relation types.

Stream-based Community Discovery

Most of the approaches for tensor decomposition are not able to process data which is presented as a stream. The approaches which are called *stream-based* more or less collect the data and create a new tensor which is presented to the decomposing method. In [Lin et al., 2009], for instance, the stream is split into blocks. The information contained in these blocks is converted into a tensor representation which is used to update the former created approximating decomposition. The update procedure is comparable to the original decomposition: the (update-)tensor is decomposed and the result is used to update the former result by using a trade-off factor to combine both decompositions.

Two other approaches which in some way incorporate streaming data are presented for comparison. Dynamic tensor analysis presented by [Sun et al., 2006] pretends to handle streaming data. If new data is coming in to be processed, their approach will create a new tensor containing the data. After that, that tensor is unfolded to each mode (dimension) and the resulting matrices are used to update their model in a stream-based way. In our approach we have to update not only one but multiple relations at a time.

GraphScope by [Sun et al., 2007] also collects new data and aggregates them to new tensors being used to update the model. In addition, their approach finds the optimal time point for updating. The two major drawbacks of these approaches is that first the data is aggregated to new tensors which is not really stream-based and second the dimensions of the tensors have to be known. A former created model which, for instance, has information about $users_1$ until $users_{100}$ is not able to be updated by information about another *user* which is unknown ($user_{101}$, for example).

Own Contributions

[Lin et al., 2009] state that the runtime of their approach is bound by $\mathcal{O}(N)$, where N is the number of entries available in the particular tensors. For our *Twitter* dataset we crawled about 200.000 messages containing about 590.000 relations. In order to achieve faster runtime, we will bound the size of tensors and allow a stream-based continuous updating of the decomposition model.

Our contribution in this chapter is an adaption of METAFAC by [Lin et al., 2009]. The adaption consists of four particular steps:

1. We bounded the dimension of the certain $\boldsymbol{U}^{(i)}$ matrices. In this way we only allow a particular number of entities to be part of our model. If the maximum number of entities is reached the most outdated entities are replaced by new ones.

2. The strategy for the replacement for outdated entities is presented based on a time-dependent weighting mechanism. This mechanism decreases the weight of each entity over time.

3. New relations are continuously integrated. Instead of collecting the relational data for the block-wise updating of the model, we directly integrate new relations, and the particular model is updated.

4. Our new approach is evaluated on real-world data crawled from the social network *Twitter*.

The rest of this chapter is structured as follows. In Section 8.3.2 we present how to use graphs for the representation of relations containing more than two involved entities. We call this relations n-ary in contrast to binary relations. In Section 8.3.3 we show how tensors can be used to store the particular n-ary relations. In addition, we present the possible calculations used for tensors. We show how tensors are decomposed and how this methods helps to cluster n-ary relational data. We also present the related approach which we enhanced in Section 8.3.4 in order to make it usable in a completely stream-based manner. Additionally, we present the three major contributions we developed for the stream-based approach in that section. Section 8.3.5 contains the presentation of the results we achieved by using our approach on data extracted from the blogging platform *Twitter*. Finally, Section 8.3.6 summarizes this chapter.

8.3.2 Graphs representing n-ary Relations

We did not distinguish the different types of entities which can be related to each other in a relational graph G up to now. Nevertheless, the different types of entities play an important role in the different relation types, because only specific entity types interrelate in a particular type of relation. We split the set of entities V into multiple sets V_i of entities containing only particular types of entities like *users* or *tags*, for instance.

$$V = \bigcup_i V_i$$

A relation is a particular mention of a relation type R. If in one relation, for instance, a *user*, a *message* and an *url* are interrelated, the relation containing that information will be (u_i, m_j, ur_k), where u_i is a *user*, m_j is a *message* and ur_k is an *url*. (u_i, m_j, ur_k) is of $V_{user} \times V_{message} \times V_{url}$. The relation type R of (u_i, m_j, ur_k) is $R := V_{user} \times V_{message} \times V_{url}$. The entries u_i, m_j and ur_k are only indices pointing on a list of nominal values to avoid redundant information. For accessing these lists we need a mapping function φ_i accessing the i-th set of entities V_i:

$$\varphi_i : V_i \to \{0, \ldots, |V_i| - 1\}.$$

This mapping function on the one hand maps particular values w to a list of nominal values and on the other hand the inverse of that mapping function delivers the particular entry given the index. $\varphi_i(w)$ is the index of entry w in the list of nominal values of entity-set V_i. $\varphi_i^{-1}(j)$ is the particular value stored at index j in the list of nominal values of entity-set V_i.

After defining the mapping function we can show how to store sets of relations using a tensor. As presented in the former sections a tensor is a multi-dimensional array.

An n-ary relation can be stored in a mode-n tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{|V_1| \times \ldots \times |V_n|}$. A particular entry in such a tensor

$$\boldsymbol{\mathcal{X}}_{i_1,\ldots,i_n} \in \mathbb{R}, \varphi_j^{-1}(i_j) \in V_j$$

denotes a relation between the indices

$$(i_1 \in V_1, \cdots, i_j \in V_j, \cdots, i_n \in V_n).$$

The indices are mapped to concrete values, and

$$\varphi_1^{-1}(i_1), \cdots, \varphi_j^{-1}(i_j), \cdots, \varphi_n^{-1}(i_n)$$

are the corresponding values. For a set of relations $X \subseteq V_{i_1} \times \ldots \times V_{i_{l(i)}}$ we will present the corresponding binary tensor in the following. The tensor contains a 1 if the particular entities are related and a 0 otherwise.

$$\boldsymbol{\mathcal{X}}_{\nu_1,\ldots,\nu_{l(i)}} = \begin{cases} 1 & \text{if } (\varphi_1^{-1}(\nu_1), \ldots, \varphi_{l(i)}^{-1}(\nu_{l(i)})) \in X \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

MetaGraph

By using a matrix sets of binary relations can be stored like it is done by using adjacency matrices. A network containing multiple entity types which might be related in different relation types only could be tackled by a set of adjacency matrices. Every relation type between two entity types is presented by an adjacency matrix containing the particular interrelated pairs of the entity types.

A tensor can be used to store a set of multi-dimensional relations. 3-dimensional relations, for instance, can be stored in mode-3 tensor. MetaGraph presented by [Lin et al., 2009] is based on tensors enabling the use of multi-dimensional relations. MetaGraph is a graph $G = (V, E)$, where V is the set of vertices representing particular types of entities, and E is the set of edges connecting the vertices. The edges in this presentation are hyperedges connecting two or more vertices. By using hyperedges it is getting possible to tackle multi-dimensional relations. Each relation type R is represented by a hyperedge in G. It follows that each hyperedge is represented by a tensor $\boldsymbol{\mathcal{X}}^{(i)}$ representing the particular relations.

Every type of relation apparent in the social network which should be used for the analyses has to be presented on the one hand as a hyperedge in the MetaGraph and on the other hand as a tensor. The choice for the types of relations used in the MetaGraph is to be made by the user. Only the most important relation types have to be chosen. Choosing too many relation types will lead to a greater amount of tensors and of course to a greater runtime. If we want to analyze all possible relations (connecting at least two entity types) between all entity types a number of $|\mathcal{P}(V)| - |V| - 1 = 2^{|V|} - |V| - 1$ tensors will be needed.

We presented a MetaGraph for the Twitter framework in [Bockermann and Jungermann, 2010b]. This graph is shown in Figure 8.8. The hyperedges are indicated by

R_i, and the four entity types used in this graph are *users*, *tags*, *url* and *tweets*. One can see that entity-types can be connected to itself. This means that for instance a *user* is mentioning another *user* in a *tweet* like it is done in relation type R_4.

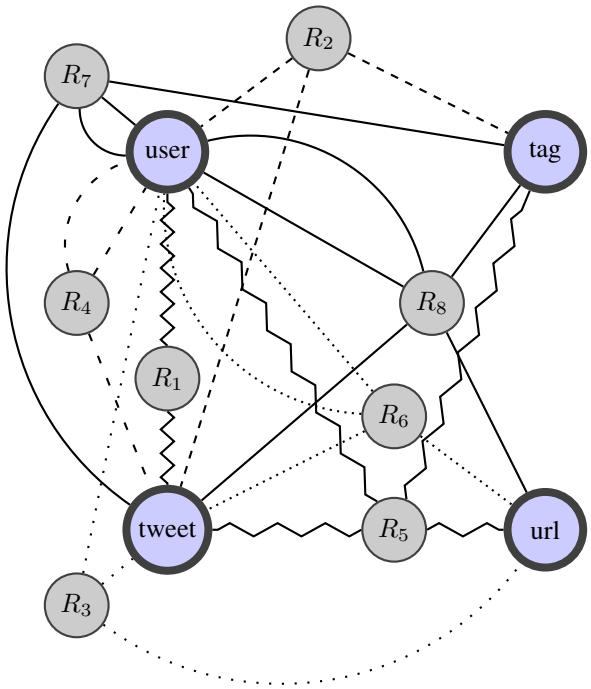


Figure 8.8: A MetaGraph for *Twitter* ([Bockermann and Jungermann, 2010a], p. 3)

Clustering Problem

By only choosing a subset of relations to be represented in the MetaGraph only an approximation of the social network is given. The relation types R are represented as tensors $\boldsymbol{\mathcal{X}}$. The MetaGraph G is described by those tensors:

$$G \mapsto \left\{ \boldsymbol{\mathcal{X}}^{(1)}, \ldots, \boldsymbol{\mathcal{X}}^{(n)} \right\}.$$

The task of clustering the entities presented by this description of the graph G is presented in the following. [Lin et al., 2009] are presenting a factorization approach working on all the tensors $(\boldsymbol{\mathcal{X}}^{(1)}, \ldots, \boldsymbol{\mathcal{X}}^{(n)})$. A certain tensor $\boldsymbol{\mathcal{X}}^{(i)}$ is factorized in the following way

$$\boldsymbol{\mathcal{X}}^{(i)} \approx [\mathbf{z}] \prod_{j=1}^{l(i)} \times_j \boldsymbol{U}^{(i_j)}. \tag{8.2}$$

$[z]$ is the global super-diagonal tensor which is used for factorizing all of the tensors. The matrices $\boldsymbol{U}^{(q)}$ which are shared by some relation types are of dimension $\mathbb{R}^{|V_q| \times k}$, where k is the user-defined number of clusters to be found and $|V_q|$ is the cardinality of entity-set V_q. Each entity is represented in the certain matrices. For each entity type only one matrix $\boldsymbol{U}$ is available. That means that relations sharing entity types t also share the resulting matrix $\boldsymbol{U}^{(t)}$.

The mathematical basics for tensor calculations are presented in Section 8.3.3.

[Lin et al., 2009] are using a normalization method leading to the fact that the entries in a matrix $\boldsymbol{U}^{(q)}$ are values of $[0, 1]$. The entries in the matrices can be seen as a sort of probability for each entity belonging to the particular clusters. To extract the most probable cluster $C(l)$ for a given entity $\varphi_q^{-1}(l)$ one just has to find the maximum entry in the corresponding matrix:

$$C(l) = \arg\max_m \boldsymbol{U}^{(q)}_{l,m}, m \in \{1, \ldots, k\} \tag{8.3}$$

To achieve a global clustering for all types of relations the factorization has to be simultaneously performed on all tensors. Like for the factorization of only one tensor, a distance measure $D : \mathbb{R}^{I_1 \times \ldots \times I_l} \times \mathbb{R}^{I_1 \times \ldots \times I_l} \rightarrow \mathbb{R}$ is needed to evaluate the approximating performance. In contrast to the factorization of a single tensor we now have to calculate the global distance. This distance has to be minimal leading to an optimization problem:

$$\arg\min_{[\mathbf{z}], \boldsymbol{U}^{(q)}} \sum_{i=1}^{n} D(\boldsymbol{\mathcal{X}}^{(i)}, [\mathbf{z}] \prod_{j=1}^{l(i)} \times_j \boldsymbol{U}^{(i_j)}) \tag{8.4}$$

Later on we will analyze the possibilities to do such clustering on data streams representing the behavior of an original social network. A huge amount of new data which can be represented as particular relations is created in a social network every second. The *twitter* platform already is offering a stream of the newly created content. It follows that only stream-based analysis will tackle this amount of data well.

[Lin et al., 2009] present a batch processing approach to handle a stream of data. The stream is converted into a stream of subsets each containing just a disjoint subset of the original stream. If t is the current subset the currently available model is presented by $\mathbf{z}_{t-1}, \{\boldsymbol{U}^{(q)}_{t-1}\}$. The creation of the new model is done by

$$\arg\min_{\boldsymbol{z}_t, \{\boldsymbol{U}^{(q)}_t\}} (1 - \alpha) \sum_{i=1}^{n} D(\boldsymbol{\mathcal{X}}^{(i)} \parallel [\boldsymbol{z}_t] \prod_{j=1}^{l(i)} \times_j \boldsymbol{U}^{(i)}_t) + \alpha L_{prior} \tag{8.5}$$

$$L_{prior} = D([\boldsymbol{z}_{t-1}] \parallel [\boldsymbol{z}_t]) + \sum_{i=1}^{n} D(\boldsymbol{U}^{(i)}_{t-1} \parallel \boldsymbol{U}^{(i)}_t). \tag{8.6}$$

The divergence between old and new model L_{prior} is used to handle the impact of how strongly the new created model is taken into account. α is used as a trade-off factor

in order to select whether the new model ($\alpha \to 0$) or the older model ($\alpha \to 1$) should stronger affect the new model.

8.3.3 Tensors

For the further processing of tensors we need to describe some basic methods for the handling of tensors. Most of these methods are needed for the clustering approach we are presenting for social networks. In this section we will formally introduce the needed formalisms for tensor analysis. We will start with the most important operations.

Operations on Tensors

Although a tensor always can be represented in its original form as a multi-dimensional matrix, it also can be represented in multiple other ways. We will present the operations *vec*, *unfold* and *fold* in the following. These methods are used to transform a tensor into different representations without changing its content.

Representations

Given a tensor $\boldsymbol{\mathcal{X}}$ it can be converted into a (long) vector representation using the *vec* operation

$$\text{vec} : \mathbb{R}^{I_1 \times \cdots \times I_M} \to \mathbb{R}^J \text{ where } J = \prod_{k=1}^{M} I_k.$$

By using this method, a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_M}$ is converted to a vector $\boldsymbol{x} \in \mathbb{R}^J$, by mapping its entries $a_{i_1,\cdots,i_M}$ to position $i_1 + (I_1 i_2) + \cdots + (\prod_{k=1}^{M-1} I_k) i_M$, i.e.

$$\boldsymbol{x}_{i_1+(I_1 i_2)+\cdots+(\prod_{k=1}^{M-1} I_k) i_M} = a_{i_1,\cdots,i_M}.$$

The operations *unfold* and *fold* are used to convert tensors into regular (two-dimensional) matrices and vice versa. *Unfolding* is an operation

$$\text{unfold} : \mathbb{R}^{I_1 \times \cdots \times I_M} \times N \to \mathbb{R}^{I_d \times J} \quad \text{with } N = \{1, \ldots, M\}$$

which transforms a tensor $\boldsymbol{\mathcal{X}}$ along a given mode $d \in \{1, \ldots, M\}$. This special mode is used as the first and the other modes are representing the other dimension of the resulting matrix $\boldsymbol{X} \in \mathbb{R}^{I_d \times J}$ with $J = \prod_{i=1, i \neq d}^{M} I_i$. The entries $a_{i_1,\ldots,i_M}$ of the tensor $\boldsymbol{\mathcal{X}}$ are mapped to position $a_{i_d,j}$

$$a_{i_1,\ldots,i_M} \mapsto a_{i_d,j} \text{ where } j = \prod_{k=1, k \neq d}^{M} i_k I_k.$$

Folding is the inverse operation of unfold, and it is converting a matrix $\boldsymbol{X}_d \in \mathbb{R}^{I_d \times \prod_{i=1, i \neq d}^{M} I_i}$ to a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_M}$.

The *accumulation* of tensors does not create another representation of a tensor. Similar to the *vectorize* method this method creates a vector using the tensor values by summing up the values along a given mode:

$$\text{acc} : \mathbb{R}^{I_1 \times \cdots \times I_M} \times N \to \mathbb{R}^{I_d} \quad \text{with } N = \{1, \ldots, M\},$$

where $d \in \{1, \ldots, M\}$. The resulting vector $\boldsymbol{x} \in \mathbb{R}^{I_d}$ is the original i-th mode of the tensor containing the values of all other modes summed up:

$$\boldsymbol{x}_{i_d} = \sum a_{i_1, i_2, \ldots, i_d, \ldots, i_M}.$$

The *accumulation* methods also can be performed along two modes resulting in a matrix.

$$\text{acc} : \mathbb{R}^{I_1 \times \cdots \times I_M} \times N \times N \to \mathbb{R}^{I_c \times I_d} \quad \text{with } N = \{1, \ldots, M\},$$

Given two modes c and d, the *accumulation* method will take all other modes except these two and sums up the values: $\boldsymbol{X} \in \mathbb{R}^{I_c \times I_d}$ with

$$\boldsymbol{x}_{i_c, i_d} = \sum a_{i_1, i_2, \ldots, i_c, \ldots, i_d, \ldots, i_M}.$$

where $c, d \in \{1, \ldots, M\}$.

Mode-d Product

The mode-d product is another necessary operation for the approach of tensor factorization. Given an m-way tensor $\boldsymbol{\mathcal{X}}$ and a matrix $\boldsymbol{U}$ the mode-d product of $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{U}$ is an operation

$$\times_d : \mathbb{R}^{I_1 \times \cdots \times I_M} \times \mathbb{R}^{I_d \times J} \to \mathbb{R}^{I_1 \times \cdots \times I_{d-1} \times J \times I_{d+1} \times \cdots \times I_M}$$

where

$$(\boldsymbol{\mathcal{X}} \times_d \boldsymbol{U})_{i_1, \ldots, i_{d-1}, j, i_{d+1}, \ldots, i_M} = \sum_{i_d} x_{i_1, \ldots, i_M} \cdot u_{i_d, j}.$$

Tensor Decomposition

Tensor decomposition is the approach to find the best matrices for approximating the original tensor. [Kiers, 2000] present PARAFAC (or *CP*) decomposition. Following the definition of CP, a third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ is decomposed to

$$\boldsymbol{\mathcal{X}} = \left[\sum_{r=1}^{R} \boldsymbol{a}_r \circ \boldsymbol{b}_r \circ \boldsymbol{c}_r \right] + \boldsymbol{\mathcal{E}} \tag{8.7}$$

where R is a positive integer and $\boldsymbol{a}_r \in \mathbb{R}^I$, $\boldsymbol{b}_r \in \mathbb{R}^J$, $\boldsymbol{c}_r \in \mathbb{R}^K$. See Figure 8.9 for a schematic illustration of this decomposition. The decomposition is just an approximation of $\boldsymbol{\mathcal{X}}$ which becomes visible by error tensor $\boldsymbol{\mathcal{E}}$. If we want to formally neglect $\boldsymbol{\mathcal{E}}$

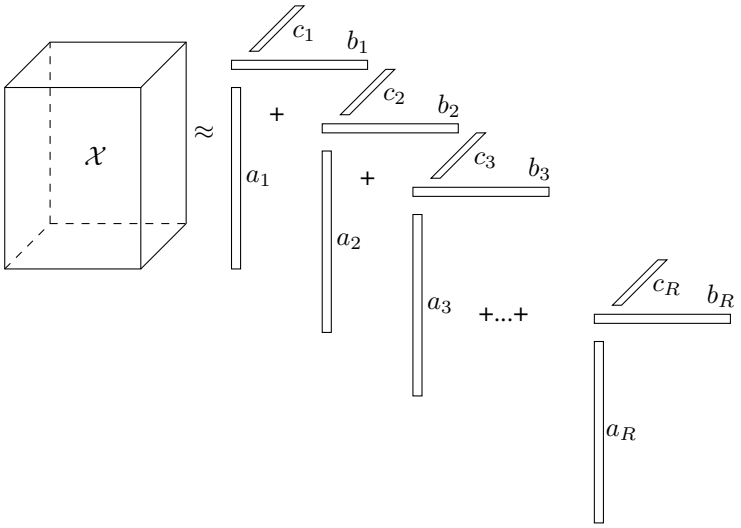


Figure 8.9: CP tensor decomposition ([Bockermann and Jungermann, 2010a], p. 4)

we can use the approximate notation of $\boldsymbol{\mathcal{X}}$:

$$\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \boldsymbol{a}_r \circ \boldsymbol{b}_r \circ \boldsymbol{c}_r. \tag{8.8}$$

The elementwise presentation of equation (8.8) is

$$x_{ijk} \approx \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr} \tag{8.9}$$

We will need probability values for later processing, therefore we have to normalize the rank-one tensors $\boldsymbol{a}_r$, $\boldsymbol{b}_r$ and $\boldsymbol{c}_r$ for $r = 1, ..., R$ to one. We will achieve the following form

$$x_{ijk} \approx \sum_{r=1}^{R} z_r a_{ir} b_{jr} c_{kr} \tag{8.10}$$

where $\boldsymbol{z} \in \mathbb{R}^R$ is a weight-vector. This decomposition sometimes is called higher-order singular value decomposition (HOSVD) ([Kilmer and Moravitz Martin, 2004]). The rank-one tensors $\boldsymbol{a}_r$, $\boldsymbol{b}_r$ and $\boldsymbol{c}_r$ for $r = 1, ..., R$ represent the singular values. These values later on can be utilized for the derivation of a clustering of the data.

$\boldsymbol{a}_r, \boldsymbol{b}_r$ and $\boldsymbol{c}_r$ for $r = 1, ..., R$ can be represented as matrices $\boldsymbol{U}^1 \in \mathbb{R}^{I \times R}$, $\boldsymbol{U}^2 \in \mathbb{R}^{J \times R}$ and $\boldsymbol{U}^3 \in \mathbb{R}^{K \times R}$. Constructing a superdiagonal tensor $[\boldsymbol{z}] \in \mathbb{R}^{R \times ... \times R}$ of $\boldsymbol{z}$ containing just zeros apart from positions $z_{r,...,r}$ where it contains value z_r of eq. (8.10).

Following this approach makes eq. (8.10) to be written as *mode-n product*:

$$\boldsymbol{\mathcal{X}} \approx ((([\boldsymbol{z}] \times_1 \boldsymbol{U}^1) \times_2 \boldsymbol{U}^2) \times_3 \boldsymbol{U}^3) = [\boldsymbol{z}] \prod_{i=1}^{3} \times_i \boldsymbol{U}^i \tag{8.11}$$

Figure 8.10 shows the *n-mode product* of equation (8.11) for a better understanding.

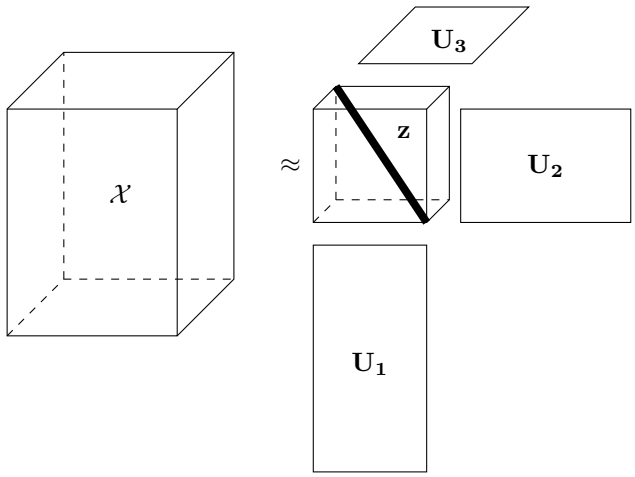


Figure 8.10: CP tensor decomposition incorporating weights ([Bockermann and Jungermann, 2010a], p. 4)

METAFAC - Metagraph Factorization

The METAFAC algorithm presented by [Lin et al., 2009] is used to optimally decompose a set of tensors $\boldsymbol{\mathcal{X}}^{(i)}$ with shared factors $[\boldsymbol{z}], \{\boldsymbol{U}^{(q)}\}$. They evaluate the approximation performance by using the Kullback Leibler divergence D_{KL} [Kullback and Leibler, 1951], introducing the following optimization problem:

$$\underset{[z],\{\boldsymbol{U}^{(q)}\}}{\arg\min} \sum_{i=1}^{n} D_{KL}(\boldsymbol{\mathcal{X}}^{(i)}, [\boldsymbol{z}] \prod_{i_1,\ldots,i_{l(i)}} \times_j \boldsymbol{U}^{(i_j)}) \tag{8.12}$$

To solve the problem of eq. (8.12) [Lin et al., 2009] developed an approximate solution which is based on the following variables

$$\boldsymbol{\mu}^{(i)} = \mathrm{vec}(\boldsymbol{\mathcal{X}}^{(i)} \oslash ([\boldsymbol{z}] \prod_{j=1}^{l(i)} \times_j \boldsymbol{U}^{(i_j)})) \tag{8.13}$$

$$\boldsymbol{\mathcal{S}}^{(i)} = \mathrm{fold}(\boldsymbol{\mu}^{(i)} * (\boldsymbol{z} * \boldsymbol{U}^{M_i} * \cdots * \boldsymbol{U}^{1_i})^T) \tag{8.14}$$

where $\oslash$ is the elementwise division of tensors, and $*$ is the Khatri-Rao product of matrices. $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\mathcal{S}}^{(i)}$ are afterwards used for iteratively updating $\boldsymbol{z}$ and $\{\boldsymbol{U}^{(q)}\}$ by

$$\boldsymbol{z} = \frac{1}{n}\sum_{i=1}^{n} \text{acc}(\boldsymbol{\mathcal{S}}^{(i)}, M_i + 1) \tag{8.15}$$

$$\boldsymbol{U}^q = \sum_{l:e_l \sim v_q} \text{acc}(\boldsymbol{\mathcal{S}}^{(i)}, q, M_e + 1) \tag{8.16}$$

where *acc* is the *accumulation*-function (see Section 8.3.3) and $M_i + 1$ is the last dimension of $\boldsymbol{\mathcal{S}}^{(i)}$. This step is repeated until convergence is reached. Convergence is reached if eq. (8.4) delivers sufficiently small values which means that the approximation is well suited. The optimization algorithm is presented as Algorithm 8.

Algorithm 8 MF algorithm ([Lin et al., 2009], p. 5)

Input: Meta-Graph $G = (V, E)$, data tensors $\{\boldsymbol{\mathcal{X}}^{(e)}\}$
Output: z and $\{\boldsymbol{U}^q\}$
procedure METAFAC(G, $\{\boldsymbol{\mathcal{X}}^{(e)}\}$)
    Initialize z, $\{\boldsymbol{U}^q\}$
    repeat
        for all $e \in E$ **do**
            compute $\{\boldsymbol{\mathcal{S}}^{(e)}\}$ by eq. (8.13), (8.14)
            compute z by eq. (8.15)
        end for
        for all $q \in V$ **do**
            update $\{\boldsymbol{U}^q\}$ by eq. (8.16)
        end for
    until *convergence*
end procedure

Timestamped Metagraph Factorization

As already mentioned, [Lin et al., 2009] propose to use a batch processing for handling streams of data. They are creating timestamped batches of the complete data, and these batches are used for the iterative updating of the model. Instead of creating one optimal model the model is iteratively updated which requires a slightly different objective:

$$min_{\boldsymbol{z},\{\boldsymbol{U}^q\}}(1-\alpha)\sum_{e\in E} D(\boldsymbol{\mathcal{X}}^e \parallel [\boldsymbol{z}] \prod_{i:v_i\sim e} \times_i \boldsymbol{U}^i) + \alpha l_{prior} \tag{8.17}$$

$$l_{prior} = D(\boldsymbol{z_{t-1}} \parallel \boldsymbol{z}) + \sum_i D(\boldsymbol{U}_t^i \parallel \boldsymbol{U}^i) \tag{8.18}$$

The functions to be updated are defined as follows:

$$\boldsymbol{z} = (1-\alpha)\sum_{e\in E} acc(\boldsymbol{\mathcal{S}}^e, M_e+1) + \alpha \boldsymbol{z}_{t-1} \tag{8.19}$$

$$\boldsymbol{U}^q = (1-\alpha)\sum_{l:e_l\sim v_q} acc(\boldsymbol{\mathcal{S}}^j, q, M_e+1) + \alpha \boldsymbol{U}^q_{t-1} \tag{8.20}$$

The adaption of the METAFAC algorithm 8 is done by substituting the computations in lines 7, 8 and 11 using the equations (8.18), (8.19) and (8.20) respectively.

8.3.4 Stream-based Clustering using Tensors

In this section we are presenting the adaption we made on the METAFAC framework. We are presenting a method to bound the amount of entities which are currently active in the model. This amount of active entities is determined by choosing only the most novel entities for processing. To always choose the most novel entities we present a weighting mechanism which automatically downgrades older entities. This approach should cause the model to incorporate only novel *Twitter* messages.

In the *Twitter* network the users are creating messages (*tweets*) which are read by other users. We assume, that this messages are given as a sequence M:

$$M := \langle m_0, m_1, \ldots \rangle$$

where each message m_i is consisting of a set of relation types $\mathcal{R}(m_i)$. Let $\tau(m_i) \geq 0$ be the timestamp the message m_i was created. The resulting sequence of relations is

$$S := \langle \mathcal{R}(m_0), \mathcal{R}(m_1), \ldots \rangle.$$

These relations are directly added to the social network graph G. The network graph G is permanently evolving leading to a sequence of graphs or graph-states

$$\langle G_{t_0}, G_{t_1}, \ldots \rangle$$

where each G_{t_i} contains all information about the relations of all messages up to time t_i.
Let t, t' be two ordered timestamps with $t < t'$. In the following $G_{[t,t']}$ denotes the graph containing only information concerning the messages in the timespan between t and t'. G_t, for instance, equals $G_{[0,t]}$. The graphs which are metagraphs are still presented by a set of tensors $\boldsymbol{\mathcal{X}}^{(1)}, \ldots, \boldsymbol{\mathcal{X}}^{(n)}$. The time bound graphs are also represented by a set of tensors:

$$G_{[t,t']} \mapsto \left\{ \boldsymbol{\mathcal{X}}^{(1)}, \ldots, \boldsymbol{\mathcal{X}}^{(n)} \right\}_{[t,t']}.$$

The MFSTREAM Algorithm

In contrast to the sliding window approach presented in the METAFAC algorithm in equation (8.17), we present a more continuous approach. METAFAC factorizes tensors $\{\boldsymbol{\mathcal{X}}^{(i)}\}_{[t_{j-1},t_j]}$, where the time slots t_{j-1} and t_j always cover the same amount of

messages. The new factorization results are updating the former model weighted by a trade-off factor.

Our more continuous approach directly adds new relations to the network graph. This graph is permanently used for factorizing the most current tensors. In order to only use the most novel relations for factorization we on the one hand use an upper bound to lower the maximum number of entities and on the other hand introduce a time-based weighting strategy to exclude old entities from our model. In the following we will present the weighting mechanism in addition to the bounding mechanism of the entity sets. Our algorithm is presented in Section 8.3.4.

Relation Weighting

Up to now a relation just was a number of entities which relate to each other. It is sufficient to use a binary property to represent if, for instance, entities i, j and k are related:

$$\boldsymbol{\mathcal{X}}_{i,j,k} = w,$$

with $w \in \{0, 1\}$.

As we are extracting time-stamped relations, now, we are willing to use the information concerning the particular timestamps. Our idea was to use the age of a particular relation during the factorization process. The process is accessing the entries of the certain tensors which allows the introduction of the timestamps as entries of the tensors. In addition, the entries should decrease over time to represent the fact that older messages contain older information that should not affect the model much.

Each relation $r \in R_i$ is associated with a timestamp $\tau(r)$. This timestamp is the time the relation was created. We denote S as the set of relations which have been extracted from messages. It follows that the corresponding tensor for relation type $R_i = V_{i_1} \times \ldots \times V_{i_{l(i)}}$ should contain the following values

$$\boldsymbol{\mathcal{X}}^{(i)}_{i_1,\ldots,i_{l(i)}} = \begin{cases} \tau(r) & \text{if } r = (\varphi_1^{-1}(\nu_1), \ldots, \varphi_{l(i)}^{-1}(\nu_{l(i)})) \in S \\ 0 & \text{otherwise} \end{cases} \tag{8.21}$$

The tensor entries which represent a relation between different entities is not binary anymore. The entries now are equal to the creation time of the corresponding message in the *Twitter* framework. Additionally, we introduce a *global clock*, denoted by $\tau_{\max}$, representing the maximum timestamp of all relations which have been seen so far:

$$\tau_{\max} := \max \{\, \tau(r) \mid r \in X \,\}.$$

$\tau_{\max}$ and the timestamps $\tau(r)$ for each entry r in the tensors now can be used to calculate a weighting function on-the-fly during accessing the tensor entries. A simple example for a parametrized weighting function is given as

$$\omega_{\alpha,\beta}(r) := \frac{\alpha}{\alpha + \frac{1}{\beta}(\tau_{\max} - \tau(r))}. \tag{8.22}$$

The advantage of calculating the weighting function on-the-fly is the fact that otherwise the tensors have to be updated at each time step. The global clock has to be maintained anyway for the allocation of timestamps to the relations which makes it an efficient approach to only calculate the weights if they are needed.

Restricting the amount of Entities

Streaming algorithms better should not create models which are growing proportional to the time the streaming data is analyzed. An important property of such algorithms therefore is to remove obsolete elements for the creation of condensed models. Using our presented time-based weighting approach allows us to identify the mostly out-dated relations which should not affect the model much. Nevertheless, these relations will still affect the factorization process if they are not deleted from the tensors. Another reason for storing less relations in the tensors is the fact that the runtime of each iteration of the factorization is manifested by N the number of non-zero entries in the tensors. For reducing the runtime for optimization we restrict the size of each tensor by introducing constants $C_q \in \mathbb{N}$ and providing new entity mappings φ_q by

$$\varphi_q : V_q \to \overline{V}_q \text{ with } \overline{V}_q = \{1, \ldots, C_q\}.$$

Two facts are following: These φ_q mappings will not be bijective anymore if $|V_q| > C_q$. Second, the size of the matrices $\boldsymbol{U}^{(q)}$ also will be limited to $C_q \times k$.

During the processing of the stream we will not know if a new unknown entity will occur. If a new entity occurs our approach will have to organize the mapping of the new entity in a dynamic way. We define dynamic entity mappings φ_q, which map a new entity e to the next free index of $\{1, \ldots, C_q\}$. If the maximum number of entities is reached, we will choose $f \in \{1, \ldots, C_q\}$ as the element that is mostly out-dated, i.e. contains the oldest timestamp.
In addition, all relations containing that particular entity f are removed from all tensors and from the current cluster model. This means that the matrices $\boldsymbol{U}$ have to be updated, too: $\boldsymbol{U}^{(q)}_{f,i} = \frac{1}{k} \; \forall \; i = 1, \ldots, k$.
The METAFAC approach must have to know the number of entities beforehand and especially for the batch processing of a new time window the entities must correspond to the entities which have already been used for creating the former model. Our approach is not restricted in this way as formerly unknown entities easily are mapped to new indices. It makes no difference if the maximum number of entities is reached or not.

Continuous Integration

We presented our contributions to a stream-based factorization of tensors for clustering social networks in Sections 8.3.4 and 8.3.4. Our stream-based adaption of METAFAC is called MFSTREAM, and it is presented in Algorithm 9. Our approach is totally dynamic in contrast to METAFAC as it is adding new relations to the corresponding tensor $\{\boldsymbol{\mathcal{X}}^{(q)}\}$ and it is fitting $[\mathbf{z}], \{\boldsymbol{U}^{(q)}\}$ after having seen T new messages. The parameter T is set by the user. After having seen T new messages, our approach performs just

one iteration of the optimization – using the time-based weighting method presented in Section 8.3.4. The time complexity per iteration of the MFSTREAM algorithm is the same as for the METAFAC algorithms (refer Section 8.3.3). Due to the fact that we fix the tensor dimensions, the maximum number of non-zero elements N in the tensors is constant, which implies a runtime of $O(1)$.

Algorithm 9 The MFSTREAM algorithm ([Bockermann and Jungermann, 2010b], p. 9)

Input: MetaGraph $G = (V, E)$, Stream $M = \langle m_i \rangle$, capacities C_q, constant $T \in \mathbb{N}$
procedure MFSTREAM
  Initialize z, $\{\boldsymbol{U}^{(q)}\}$, $c := 0$
  while $M \neq \emptyset$ **do**
    $m := m_c$, $c := c + 1$ ▷ *Pick the next message from the stream*
    for all $(r_{j_1}, \ldots, r_{j_{l(j)}}) \in \mathcal{R}(m)$ **do**
      for all $p = 1, \ldots, l(j)$ **do**
        if $\varphi_p(r_{j_p}) = nil$ **then** ▷ *Replacement needed?*
          if $|\varphi_p| = C_q + 1$ **then**
            $f^* := \arg\min_{f \in \varphi_p} \tau(f)$
            $\boldsymbol{U}^{(p)}_{f^*,s} := \frac{1}{k} \; \forall \; s = 1, \ldots, k$
          else
            $f^* := \min_{f \in \{1,\ldots,C_p\}} \varphi_p^{-1}(f) = nil$ ▷ *Pick next unmapped f^**
          end if
          $\varphi_p(r_{j_p}) := f^*$, $\tau(f^*) := \tau(m)$
        end if
      end for
    end for
    $\nu_i := \varphi_p(r_{j_i})$ for $i = 1, \ldots, l(j)$
    $\boldsymbol{\mathcal{X}}^{(p)}_{\nu_1,\ldots,\nu_{l(j)}} := \tau(m)$ ▷ *Update corresponding tensor*
    if $c \equiv 0 \mod T$ **then** ▷ *Single opt.-iteration every T steps*
      for all $i \in \{1, \ldots, n\}$ **do**
        compute $\{\boldsymbol{\mathcal{S}}^{(i)}\}$ by eq. (8.14) and (8.13)
        update z by eq. (8.15)
      end for
      for all $j \in \{1, \ldots, q\}$ **do**
        update $\{\boldsymbol{U}^{(j)}\}$ by eq. (8.16)
      end for
    end if
  end while
end procedure

8.3.5 Experiments

To evaluate our approach we created a real-world dataset crawled from the microblogging framework *Twitter*. *Twitter-users* can have *followers* and they can *follow* other *users*, too. If a *user* is writing a message it will be propagated to all the *followers*. The other way round, a *user* gets all the messages of the *users* she is following. The messages can contain a maximum of 140 characters. In spite of such limitations *users* are not only creating messages – called *tweets* – but also enriching their *tweets* by *tags*, *urls* or *mentions*, which allows users to address other *users*. We finally identified the entity types {*user*, *tweet*, *tag*, *url*}. The metagraph we created for the the *Twitter* platform, is shown in Figure 8.8. Although the maximum number of possible relation types is $\mathcal{P}(V) = 2^4$, some of them will never occur or are redundant. Each *tweet* is written by a user which means that every relation should contain an entity *user* – the relation (*tweet*,*tag*) which does not refer to a *user* would never occur. We extracted 8 relation types $\{R_1, \ldots, R_8\}$ for our metagraph:

- R_1: a *user* writing a *tweet*.
- R_2: a *user* writing a *tweet* containing a special *tag*.
- R_3: a *user* writing a *tweet* containing a special *url*.
- R_4: a *user* mentioning another *user* in a written *tweet*.
- R_5: a *user* writes a *tweet* containing a *tag* and an *url*.
- R_6: a *user* writing a *tweet* containing an *url* and mentioning another *user*.
- R_7: a *user* writes a *tweet* containing a *tag* and a mentioned *user*.
- R_8: a *user* mentioning another *user* in a *tweet* containing a *tag* and an *url*.

The particular creation of our dataset containing a large set of relations was done by extracting 1000 seed users and the *users* they *follow* and their *followers*. By using an English stopword filter we extracted users which are writing in English and we also processed all the *users* they *follow* and their *followers* of the seed users in the same way. This leads to about 478.000 *users* for whom we extracted all the messages written between the 19th and 23rd of February 2010. Out of these 2.274.000 *tweets* we used the *tweets* written at the 19th of February for our experiments, leaving about 389.000 *tweets* from 41.000 *users*.

Model Evaluation

To evaluate our approach we compared MFSTREAM with METAFAC. For the comparison of the resulting clusterings of both approaches we employed the "within cluster" point scatter [Hastie et al., 2003]. This is given as

$$W(C) := \sum_{k=1}^{K} N_k \sum_{C(i)=k} \| x_i - \overline{x}_k \|^2 \tag{8.23}$$

where K is the number of clusters, x_i is a member of a cluster $C(i)$ and $\overline{x}_k$ is the centroid of a cluster k. The larger the result of $W(C)$ the more examples of particular clusters are not similar to the corresponding centroid which makes this evaluation measure to be be seen as a sum of dissimilarities between elements in the particular clusters.

We used a stream of 200k messages to create the clusterings with MFSTREAM, and we restricted the dimensions of the tensors to $C_{user} = C_{tweet} = 5000$ and $C_{tag} = C_{url} = 1000$. We used several weighting functions such as $\omega_{1,1}, \omega_{10,1000}$ and $\omega_{100,1000}$ as well as a binary weighting which equals the unweighted model (i.e. $w \in \{0, 1\}$).

MFSTREAM and METAFAC are not trivially comparable. METAFAC only can process a static set of relations. We processed all messages until the first entity type V_i reached its limit. After that we stored the resulting clustering. We reset the φ mappings and started anew, revealing a new clustering every time an entity type V_i reached C_i, revealing a total of 93 clusterings for our complete dataset. We applied METAFAC on the messages of the corresponding 93 clusterings and computed their similarities using $W(C)$. Table 8.4 shows that MFSTREAM delivers results comparable to METAFAC for different weighting functions. The most similar result is achieved by using $\omega_{1,1000}$. Figure 8.11 shows that using timestamped values instead of binary values for calculation of the MFSTREAM delivers better results. The decrease of T, which leads to a larger number of optimization steps, intuitively increases the quality of MFSTREAM. This is presented in Table 8.5 and Figure 8.12. Additionally, we want to show the ef-

Weighting	$W(C)$ (mean)	std. deviation
METAFAC	$5.685 \cdot 10^7$	$1.32 \cdot 10^7$
binary	$7.511 \cdot 10^7$	$2.00 \cdot 10^7$
$\omega_{1,1}$	$6.142 \cdot 10^7$	$1.57 \cdot 10^7$
$\boldsymbol{\omega_{1,1000}}$	$\mathbf{6.002 \cdot 10^7}$	$\mathbf{1.38 \cdot 10^7}$
$\omega_{10,1000}$	$6.272 \cdot 10^7$	$1.43 \cdot 10^7$
$\omega_{100,1000}$	$6.724 \cdot 10^7$	$1.55 \cdot 10^7$

Table 8.4: Mean of $W(C)$ of different weights, comparing MFSTREAM and METAFAC

T	$W(C)$ (mean)	std. deviation
5	$\mathbf{6.043 \cdot 10^7}$	$\mathbf{1.35 \cdot 10^7}$
10	$6.133 \cdot 10^7$	$1.36 \cdot 10^7$
50	$6.058 \cdot 10^7$	$1.40 \cdot 10^7$
100	$6.465 \cdot 10^7$	$1.49 \cdot 10^7$
250	$10.882 \cdot 10^7$	$3.13 \cdot 10^7$
500	$43.419 \cdot 10^7$	$11.47 \cdot 10^7$

Table 8.5: Mean of $W(C)$ with different update steps T

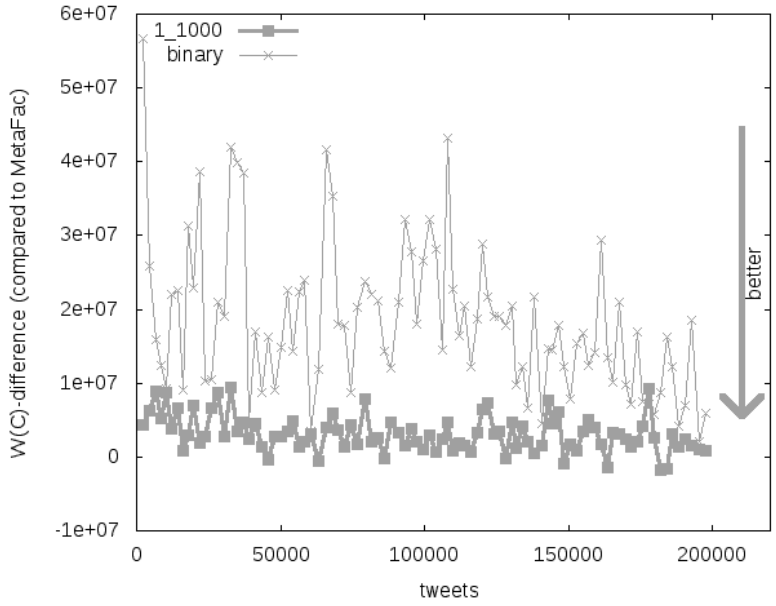


Figure 8.11: $W(C)$ for MFSTREAM compared to the METAFAC clusterings ([Bockermann and Jungermann, 2010b], p. 11)

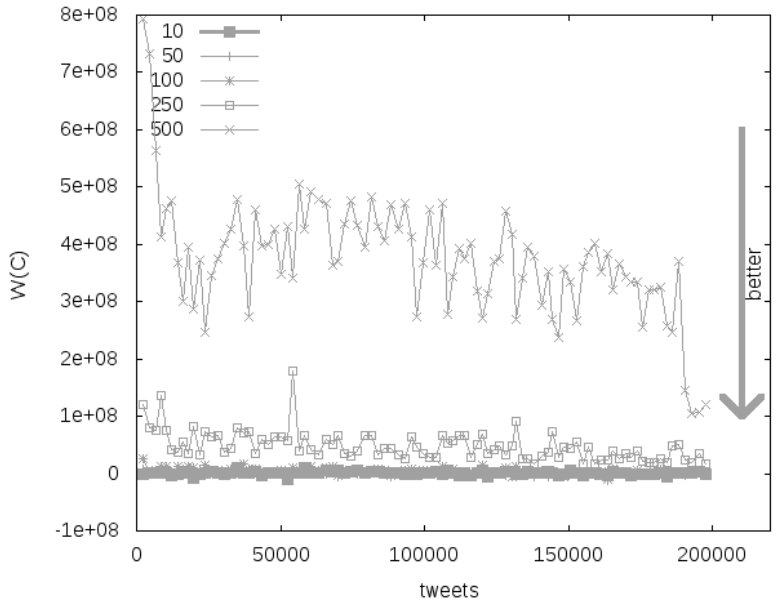


Figure 8.12: Relative $W(C)$ of MFSTREAM using different update step sizes T ([Bockermann and Jungermann, 2010b], p. 11)

fect of the update frequency T. Figure 8.13 shows the relative runtime of MFSTREAM where $T = 1$ corresponds to the baseline at 1.0. Raising T results in a shorter runtime, since the model is updated less frequently, which is the major time factor. The upper curve shows the runtime for updating after 5 relations ($T = 5$), the middle one shows $T = 10$, and the latter refers to $T = 50$.

Varying the sizes of entity types given by C_q results in clusterings of different numbers of entities. These clusterings cannot be directly compared by $W(C)$ anymore. Hence, we normalized $W(C)$ by the variance $\mathcal{V}$ of each clustering. Larger models of course incorporate more information, which results in more stable clusterings as can be seen in Figure 8.14.

8.3.6 Summary

In this chapter we presented MFSTREAM which is a flexible and dynamic algorithm for clustering multi-relational data from evolving networks. Our approach is derived from the METAFAC framework presented in [Lin et al., 2009]. We proposed four contributions in this work: Our first contribution is the possibility to continuously integrate new relations for an incremental update of the model. This makes our approach a "real" stream-based approach.

In order to respect especially the most novel relations during the creation of our model we developed a time-based weighting strategy. This strategy which is based on the timestamps of the relations and a global timestamp is creating time-based weights on-the-fly without updating the timestamps for each relation in every step.

The number of relations used for creating the model is affecting the runtime of the approach. We restricted the number of maximum tensor entries to a fixed value which leads to a constant runtime. If the maximum number of maximum entries for a particular entity type is reached the most out-dated relation is replaced by the newer one. For this replacement the time-based weighting also is utilized.

We crawled the *Twitter* network and created a dataset containing about 400.000 messages from 40.000 users. For the evaluation of our approach we implemented both approaches METAFAC and MFSTREAM in Java. We compared the clusterings achieved by our stream-based approach and METAFAC using the within-cluster point scatter metric.

Evaluation Problem

The evaluation with the cluster point scatter metric in addition to a manual revision of the clustering results from the MFSTREAM algorithm show reasonable results. Lowering the number of optimization steps for the MFSTREAM algorithm improves the clustering. The reason for this is that more iterations are applied on the same data. MFSTREAM is able to handle relations containing new, unseen entities which is not possible for METAFAC. By offering a replacement strategy especially respecting novel

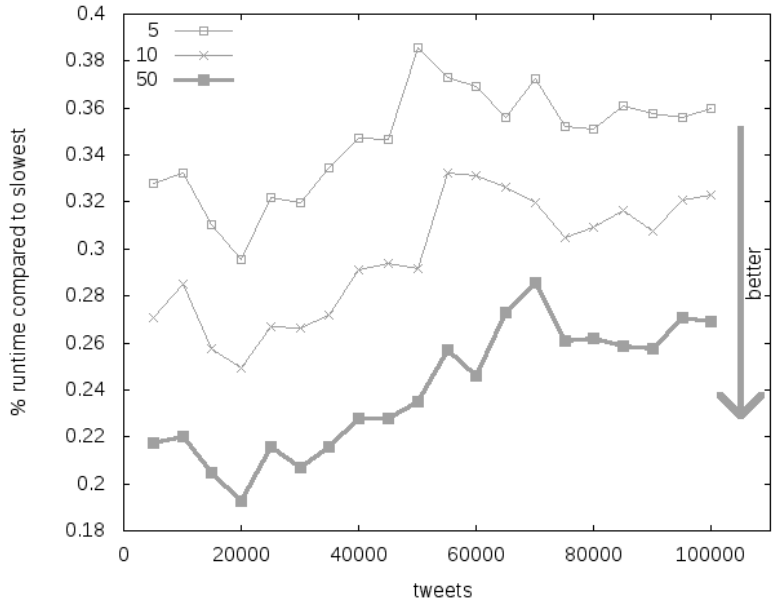


Figure 8.13: Relative runtime of MFSTREAM using different numbers of relations for update ([Bockermann and Jungermann, 2010b], p. 11)

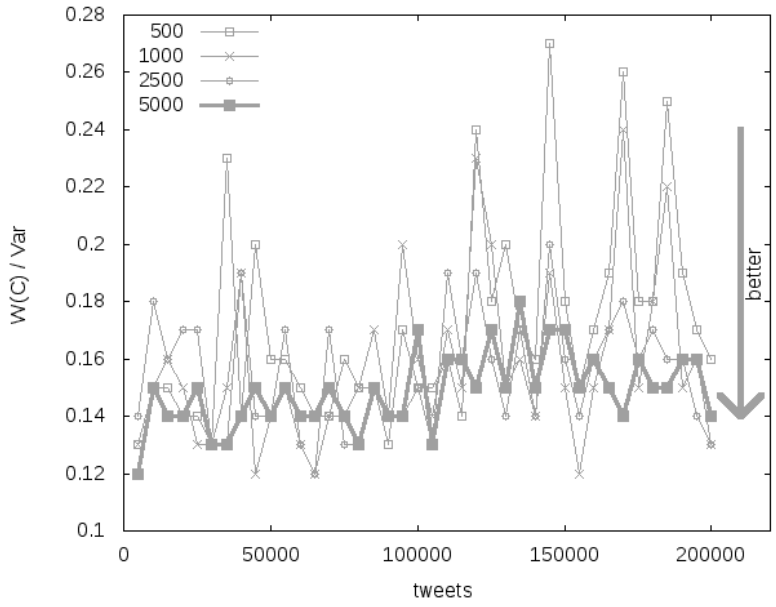


Figure 8.14: $W(C)/\mathcal{V}$ of MFSTREAM using different sizes of models ([Bockermann and Jungermann, 2010b], p. 11)

relations, the approach is suitable to continuously integrate new data from a stream.

The evaluation of the comparison between a batch- and a stream-algorithm remains difficult. The METAFAC approach optimizes a single, static problem derived from a fixed set of data. In contrast, its stream variant approximates solutions of a non-stationary optimization problem. As the objective function remains constant, the data changes continuously. The replacement strategy available within the MFSTREAM approach, results in different datasets the approach is working on than the ones used by METAFAC. The RandIndex, defined in [Rand, 1971], provides a similarity measure for the comparison of two clusterings. We applied that metric in a first evaluation. Although the results were very promising they seemed to be misleading. The index is comparing tuples which are occurring together in a cluster for both clusterings. The clusterings created by MFSTREAM do not contain all the entities which are contained in the clusterings of the METAFAC experiments, because they have been deleted by the replacement strategy. This makes the deleted entities (tuples) inaccessible by the evaluation metric.

Future Work

The deletion of out-dated relations from the tensors is very strict in the current version of our approach. Permanently deleting this information will delete the knowledge which possibly might get interesting in the future, again. One could analyze the usage of some kind of storage procedure or caching which allows a flexible re-entering of already seen relations. Especially the entities which are deleted from the $\boldsymbol{U}$ matrices have had weight-values which should be respected if these entities re-enter the model.

We analyzed the impact of the number of time steps after which the model should be updated and the weighting strategy. An important factor for stream-based approaches is runtime. One should analyze if runtime can be used as an optimization criterion. On the one hand the runtime can be used to adjust the parameters and on the other hand one could think about a multi-objective optimization approach using concurrent criteria like runtime and quality, for instance.

It is a drawback that the selection of the number of clusters is just done in a strict way. Recent work, like the one from [Lin et al., 2010], deliver first results on how to automatically achieve the best number of clusters. Embedding such methods into our approach would make it parameter-free and more flexible.

The adjustment of time steps after which our approach starts its optimization phase is crucial. Methods for finding these steps automatically and in a dynamic way could be embedded in our approach which will make it more flexible.

8.4 Summary

We presented three applications which show how nowadays applications can benefit from *Information Extraction*. Two of the applications especially benefit from the combination of *Information Extraction* and *Data Mining* which is a good argument for the implementation of a framework creating a strong collaboration between these two research areas. We formed that framework by developing the *Information Extraction Plugin*.

In the German parliament application we have shown that a vast number of tasks have to be fulfilled to offer an *Information Retrieval* / *Information Extraction* system. For a special task in this system we showed which steps have to be performed for the extraction of accepted / rejected requests of the parliament. We additionally have shown that the combination of *Information Extraction* techniques and *Data Mining* methods allow the later analysis of extracted informational units. Making this combination possible is one of the benefits of the *Information Extraction Plugin*.

The Company *Information Extraction* system delivers and intuitive and user-friendly graphical view on graphs containing related entities. A basic dataset consisting of informational units of particular companies has been used to crawl sentences from the WWW which contain at least two company names. This additional dataset was used to perform *Relation Extraction* for the generation of additional relations which finally are embedded in the company network. In particular we extracted relations that indicate the merge or fusion of two companies. If these relations finally are put in the graph they will reveal interesting facts which can furthermore be analyzed.

Additionally we presented an approach for relation graph analysis which can be used to cluster graphs containing related entities. This analysis is a follow-up analysis after having done a *Relation Extraction*. The extracted relations can be stored in a graph. Due to the fact that the relations are not only binary ones the information concerning the relations has to be stored using tensor representations. By using tensor factorization methods the contained entities can be grouped into particular clusters. The remarkable facts of the presented approach is that on the one hand no information beside the relation has to be given and on the other hand that the approach is stream-based. That makes our approach applicable to online-learning methods. This allows the handling of big data on the one hand and efficient and fast analysis on the other hand. Our approach is not restricted to graphs containing extracted relations. Arbitrary graphs containing any type of related entities like social networks can be clustered by our approach, too.

Chapter 9

Conclusion

We presented several approaches of machine learning techniques for *Information Extraction* purposes.

We have shown how *Information Extraction* developed over the years since its first appearance during the Message Understanding Conferences in Chapter 3. At the beginning *Information Extraction* has been performed in a very domain-dependent way which led to *Information Extraction*-systems which achieved good performance but only for the given tasks and domains. The task of *Information Extraction* therefore was split into several subtasks. Two of these subtasks are presented in this work: Named Entity Recognition and *Relation Extraction*. *Relation Extraction* relies on entities extracted by Named Entity Recognition.

We have presented how the feature sets for Named Entity Recognition should look like. The features can be semantically split into three groups of features: orthographic, semantic and morphologic features. The orthographic features are easily extracted from the tokens or from the surrounding tokens. Semantic features are based on given semantic lexicons or gazetteers and they are created by checking for matching tokens in these lexicons. Although this, of course, leads to a richer feature set, the feature set becomes more domain-dependent. A lexicon containing city names will help for a task extracting location concerning entities, but it will not help for a gene extraction task, for instance. Finally, morphological features are based on rich linguistic analysis. These analysis might be based on systems which deliver results for a requested token, ad hoc. These analysis also might be done to construct morphologic lexicons which later on can be used to achieve features for requested tokens. For the first approach the complex linguistic analysis is performed for every requested token, and for the other approach the complex analysis has to be performed beforehand to create the lexicon. These features deliver knowledge being very helpful for Named Entity Recognition tasks. Unfortunately, these features are created in a complex manner and for many languages no or not sufficient resources are available to create morphological features.

We made experiments on several different datasets. We always used a CRF on sim-

ilarly created feature sets. We have shown that a domain-independent feature set only consisting of orthographic features will deliver nearly as good results as approaches which are tailored to the certain domain. By using such feature set we avoid the time-consuming analysis of the domain and the creation of domain-dependent features.

In the following chapter (Chapter 4) we put the focus especially on the morphological features. The creation of such features is computationally complex. The generation of morphological features can be done by creating a lexicon or by creating a system performing the morphological analysis during runtime. Although morphological features can be helpful for *Information Extraction* tasks their creation should be avoided if the computational resources are not sufficient enough.

Relation Extraction is particularly described in Chapter 5. *Relation Extraction* heavily relies on sufficiently well performed Named Entity Recognition. Given such entities the task of *Relation Extraction* first of all is based on a reconstruction of the example- or token set. Each pair of tokens of each sentence is being converted into relation candidates. Out of these candidates the correct relations have to be extracted and classified. This task also benefits from the three types of features we already presented. We presented the state-of-the-art feature sets and we especially showed that the morphological information extracted from the parse trees for the particular sentences are used to extend the feature sets. At first these parse trees have been used as *flat* features by shattering the parse trees into smaller parts which were used like any other nominal feature. Following publications presented the application of certain methods to use the complete parse trees because the splitting of parse trees into smaller subtrees results in the loss of information which is inherently stored in the trees. Using these *structured* features requires certain methods being able to respect such structures. This is done by tree kernels. We presented different tree kernels which all are based on the kernel presented by [Collins and Duffy, 2001].

Although many extensions on tree kernels have been performed, the recursive calculation of the tree kernels we presented in this chapter remains computationally complex. We focused on particular solutions in the following chapter.

In Chapter 6 we presented the possible solutions for a more efficient use of tree kernels accessing tree sets or tree forests. The support vectors of a trained SVM containing tree structures can be seen as a tree forest. This tree forest has to be traversed during The training or test phase, for instance, and for each of the trees stored in the forest a kernel value has to be calculated.

We presented approaches to avoid such behavior and we additionally show that machine learning approaches exist which are not relying on exhaustive training. These approaches namely are perceptrons and naïve Bayes classifiers. These approaches will result in much faster runtime in comparison with SVM which rely on an exhaustive training phase. If these approaches additionally are combined with the efficient use of tree forests they will again speed up in case of runtime. Naïve Bayes classifiers are an approach relying on a minimum training phase. We presented analysis on naïve

Bayes classifiers in combination with tree kernel usage. The tree kernels are used as a distance- or similarity measure to make tree structures applicable by naïve Bayes classifiers.

In Chapter 7 we presented the *Information Extraction Plugin* which has been developed for the open source framework *RapidMiner*. *RapidMiner* originally is a *Data Mining* framework. Our plugin on the one hand allows the use of *Information Extraction* methods in *RapidMiner*, and on the other hand the users still can use all the methods and techniques originally developed for *Data Mining* purposes. The latter point allows users to compare their results achieved by *Information Extraction* models with results achieved by other approaches, and furthermore the extracted information gathered by the *Information Extraction Plugin* can directly be used to be analyzed by *Data Mining* methods.

The *Information Extraction Plugin* is the only *Information Extraction* framework which directly aims at the close collaboration of *Information Extraction* and *Data Mining*. We have shown that the architecture of state-of-the-art *Information Extraction* frameworks is comparable to the one of the *Information Extraction Plugin*. The *Information Extraction Plugin* profits by the machine learning environment of *RapidMiner* which delivers multiple evaluation and validation tools.

Chapter 8 contains three particular real-world applications which show that the collaboration of *Information Extraction* and (traditional) *Data Mining* techniques is beneficial for the analysis of certain resources of information. In the first application we have shown that Named Entity Extraction also can be used to extract particular entities needed for event extraction. We presented the combination of *Information Extraction* and *Data Mining* in order to gain insights of extracted entities in the politics domain. Additionally, we presented experiments on a dataset containing relations concerning the merge of organizations. We also presented a graph based approach for the visualization of extracted relations. This graph based visualization enables not only the graphical representation of relations, it also visualizes inherently available relations becoming interesting for certain purposes. A merge of two organizations which share multiple members of their leader-board becomes more interesting if the latter information is known. The third application is about the possible further use of relations which have been extracted from several sources. In that section we present an approach to find clusters in a graph consisting of entities which might be related to each other. In contrast to focus on the information concerning the entities the presented approach only takes into account that an entity takes part in several relations. In addition to finding clusters in such a graph we assume that the particular graph continuously changes over time. We therefore developed a stream-based approach to respect this behavior. As an example to demonstrate the presented approach on the well-known social network twitter. This platform is continuously changing and therefore a good argument for using such a stream-based approach like the one we presented.

Joint Work and other Work by the Author

Joint Work

Chapter 3

The work on the EVALITA dataset is based on a joint work together with Marc Roessler [Roessler et al., 2007, Jungermann and Roessler, 2007]. Marc contributed his highly acknowledged knowledge in the field of named entity recognition.

The analysis of the German parliament website is based on a joint work together with Katharina Morik [Jungermann and Morik, 2008].

Chapter 5

In this chapter the part concerning the economic network and the enhanced parse tree containing stems of tokens is based on a joint work together with Katharina Morik and Martin Had [Had et al., 2009]. Martin Had implemented a system which crawled and generated the economic network. In addition, he implemented a first version of the tree kernel SVM.

Chapter 8.3

This chapter mostly is based on a joint work together with Christian Bockermann [Bockermann and Jungermann, 2010b]. Christian contributed his highly acknowledged knowledge in the field of data streams and stream-based *Data Mining*.

Other Publications

Towards Adjusting Mobile Devices To User's Behaviour

[Fricke et al., 2011, Fricke et al., 2010] present possibilities in adjusting the scheduling of system calls depending on the user behavior on mobile devices. A dataset containing file access calls has been created. It is possible to partially or to fully access the

corresponding files. The authors analyzed the usage of machine learning techniques predicting the type of file access. An evaluation has shown that only for some reason the machine learning techniques achieved as good performance as the original operating system.
The author contributed his knowledge in Naïve Bayes classifiers to this work. The Naïve Bayes classifiers are one of the analyzed machine learning techniques.

Enhancing Ubiquitous Systems Through System Call Mining

In [Morik et al., 2010] the author contributed his knowledge in Naïve Bayes classifiers. Like in [Fricke et al., 2011, Fricke et al., 2010] the authors have analyzed system calls in this work. In contrast to the preceding publication this publication is rather focusing on the analysis of structured methods. The authors are analyzing if and how much the structure of system calls impact for system call scheduling performance.

Bibliography

[MUC, 1995] (1995). *Proceedings of the Sixth Message Understanding Conference*, Columbia, Maryland. Morgan Kaufmann.

[MUC, 1998] (1998). *Proceedings of the Seventh Message Understanding Conference*, Fairfax, Virginia. Morgan Kaufmann.

[LDC, 2004a] (2004a). *The ACE 2004 Evaluation Plan*. Linguistic Data Consortium.

[Acar et al., 2005] Acar, E., Çamtepe, S. A., Krishnamoorthy, M. S., and Yener, B. (2005). Modeling and multiway analysis of chatroom tensors. In *ISI*, pages 256–268.

[Acar et al., 2006] Acar, E., Çamtepe, S. A., and Yener, B. (2006). Collective sampling and analysis of high order tensors for chatroom communications. In *ISI*, pages 213–224.

[Aha, 1997] Aha, D., editor (1997). *Lazy Learning*. Kluwer Academic Publishers.

[Aiolli et al., 2007] Aiolli, F., Da San Martino, G., Sperduti, A., and Moschitti, A. (2007). Efficient kernel-based learning for trees. In *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*, pages 308 –315.

[Allen, 1984] Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154.

[Aone and Ramos-Santacruz, 2000] Aone, C. and Ramos-Santacruz, M. (2000). REES: a large-scale relation and event extraction system. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, pages 76–83, Seattle, Washington. Morgan Kaufmann Publishers Inc.

[Bader et al., 2007] Bader, S., Urfer, W., and Baumbach, J. I. (2007). Reduction of ion mobility spectrometry data by clustering characteristic peak structures. *Journal of Chemometrics*, 20:128–135.

[Banerjee et al., 2007] Banerjee, A., Basu, S., and Merugu, S. (2007). Multi-way clustering on relation graphs. In *Proceedings of the Seventh SIAM International Conference on Data Mining, 2007*, Minneapolis, Minnesota, USA. SIAM.

[Bekkerman and Allan, 2004] Bekkerman, R. and Allan, J. (2004). Using Bigrams in Text Categorization. Technical report, CIIR.

[Blaschke and Valencia, 2001] Blaschke, C. and Valencia, A. (2001). Can bibliographic pointers for known biological data be found automatically? protein interactions as a case study. *Comparative and Functional Genomics*, 2:196–206.

[Bockermann et al., 2009] Bockermann, C., Apel, M., and Meier, M. (2009). Learning sql for database intrusion detection using context-sensitive modelling. In *Proceedings of the 6th Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 196 – 205. Springer.

[Bockermann and Jungermann, 2010a] Bockermann, C. and Jungermann, F. (2010a). Stream-based community discovery via relational hypergraph factorization on evolving networks. In Atzmueller, M., Benz, D., Hotho, A., and Stumme, G., editors, *Lernen, Wissen & Adaptivität (LWA 2010) – Workshop Proceedings*.

[Bockermann and Jungermann, 2010b] Bockermann, C. and Jungermann, F. (2010b). Stream-based community discovery via relational hypergraph factorization on evolving networks. In *Proceedings of the Workshop on Dynamic Networks and Knowledge Discovery (DyNaK 2010)*.

[Bunescu et al., 2004] Bunescu, R., Ruifang, G., Kate, R. J., Marcotte, E. M., Mooney, R. J., Ramani, A. K., and Wong, Y. W. (2004). Comparative experiments on learning information extractors for proteins and their interactions. *Journal of Artificial Intelligence in Medicine*.

[Bunescu and Mooney, 2005] Bunescu, R. C. and Mooney, R. J. (2005). A shortest path dependency kernel for relation extraction. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 724–731, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

[Bunescu and Mooney, 2006] Bunescu, R. C. and Mooney, R. J. (2006). Subsequence kernels for relation extraction. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Advances in Neural Information Processing Systems 18*, pages 171–178. MIT Press.

[Burnard and Bauman, 2007] Burnard, L. and Bauman, S. (2007). TEI P5: Guidelines for electronic text encoding and interchange.

[Cafarella et al., 2006] Cafarella, M. J., Banko, M., and Etzioni, O. (2006). Relational web search. Technical report, University of Washington, CSE.

[Cai et al., 2006] Cai, D., He, X., and Han, J. (2006). Tensor space model for document analysis. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 625–626, New York, NY, USA. ACM.

[Cardie, 1997] Cardie, C. (1997). Empirical methods in information extraction. *AI Magazine*, 18.

[Caropreso et al., 2001] Caropreso, M. F., Matwin, S., and Sebastiani, F. (2001). *A Learner-Independent Evaluation of the Usefulness of Statistical Phrases for Automated Text Categorization*, pages 78–102. IGI Publishing, Hershey, PA, USA.

[Chinchor, 1998] Chinchor, N. (1998). Overview of MUC-7. In *Proceedings of the Message Understanding Conference, MUC*.

[Cohen, 1960] Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.

[Collins and Duffy, 2001] Collins, M. and Duffy, N. (2001). Convolution kernels for natural language. In *Proceedings of Neural Information Processing Systems, NIPS 2001*.

[Cooper, 2003] Cooper, C. (2003). *The A-Z of Social Research*, chapter Analysis of variance (ANOVA), pages 9–12.

[Crammer et al., 2003] Crammer, K., Kandola, J. S., and Singer, Y. (2003). Online classification on a budget. In Thrun, S., Saul, L. K., and Schölkopf, B., editors, *Proceedings of the Sixteenth Annual Conference on Neural Information Processing Systems (NIPS)*. MIT Press.

[Cunningham et al., 2011] Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Aswani, N., Roberts, I., Gorrell, G., Funk, A., Roberts, A., Damljanovic, D., Heitz, T., Greenwood, M. A., Saggion, H., Petrak, J., Li, Y., and Peters, W. (2011). *Text Processing with GATE (Version 6)*.

[Daelemans, 1999] Daelemans, W. (1999). *Machine learning approaches*, pages 285–304. Kluwer, Dordrecht.

[Daelemans et al., 1997] Daelemans, W., van den Bosch, A., and Weijters, T. (1997). *Empirical learning of natural language processing tasks*, pages 337–344. Springer, Berlin.

[Daelemans et al., 2003] Daelemans, W., Zavrel, J., van der Sloot, K., and van den Bosch, A. (2003). *TiMBL: Tilburg memory based learner, version 5.0: reference guide [= ILK Research Group technical report 03-10]*. Tilburg University, Tilburg.

[De Sitter et al., 2004] De Sitter, A., Calders, T., and Daelemans, W. (2004). A formal framework for evaluation of information extraction.

[Dekel et al., 2008] Dekel, O., Shalev-Shwartz, S., and Singer, Y. (2008). The forgetron: A kernel-based perceptron on a budget. *SIAM J. Comput.*, 37:1342–1372.

[Della Pietra et al., 1997] Della Pietra, S., Della Pietra, V., and Lafferty, J. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393.

[Didakowski, 2007] Didakowski, J. (2007). Syncop - combining syntactic tagging with chunking using weighted finite state transducers. In *Proceedings of the Sixth International Workshop on Finite-State Methods and Natural Language Processing (FSMNLP)*.

[Dima and Hinrichs, 2011] Dima, C. and Hinrichs, E. W. (2011). A semi-automatic, iterative method for creating a domain-specific treebank. In Angelova, G., Bontcheva, K., Mitkov, R., and Nicolov, N., editors, *Proceedings of Recent Advances in Natural Language Processing, RANLP 2011*, pages 413–419. RANLP 2011 Organising Committee.

[Doddington et al., 2004] Doddington, G., Mitchell, A., Przybocki, M., Ramshaw, L., Strassel, S., and Weischedel, R. (2004). The Automatic Content Extraction (ACE) Program–Tasks, Data, and Evaluation. *Proceedings of LREC 2004*, pages 837–840.

[Domingos and Pazzani, 1996] Domingos, P. and Pazzani, M. (1996). Beyond independence: Conditions for the optimality of the simple bayesian classifier. In *Machine Learning*, pages 105–112. Morgan Kaufmann.

[Domingos and Pazzani, 1997] Domingos, P. and Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130.

[Fernandes and Brefeld, 2011] Fernandes, E. R. and Brefeld, U. (2011). Learning from partially annotated sequences. In *Proceedings of the 2011 European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I*, ECML PKDD'11, pages 407–422, Berlin, Heidelberg. Springer-Verlag.

[Ferrucci et al., 2009] Ferrucci, D., Lally, A., Verspoor, K., and Nyberg, E. (2009). Unstructured Information Management Architecture (UIMA) Version 1.0, OASIS Standard. Technical report, OASIS.

[Fisher, 1997] Fisher, R. A. (1997). On an absolute criterion for fitting frequency curves. *Statistical Science*, 12(1):pp. 39–41.

[Frank and Asuncion, 2011] Frank, A. and Asuncion, A. (2011). UCI machine learning repository.

[Fricke et al., 2010] Fricke, P., Jungermann, F., Morik, K., Piatkowski, N., Spinczyk, O., and Stolpe, M. (2010). Towards adjusting mobile devices to user's behaviour. In *Proceedings of the International Workshop at ECML PKDD on Mining Ubiquitous and Social Environments (MUSE 2010)*, pages 7 – 22.

[Fricke et al., 2011] Fricke, P., Jungermann, F., Morik, K., Piatkowski, N., Spinczyk, O., Stolpe, M., and Streicher, J. (2011). *Towards Adjusting Mobile Devices To User's Behaviour*, pages 99–118. Springer-Verlag, Heidelberg.

[Fürnkranz, 1998] Fürnkranz, J. (1998). A study using n-gram features for text categorization.

[Geyken and Hanneforth, 2006] Geyken, A. and Hanneforth, T. (2006). Tagh: A complete morphology for german based on weighted finite state automata. In Yli-Jyrä, A., Karttunen, L., and Karhumäki, J., editors, *Finite-State Methods and Natural Language Processing*, volume 4002 of *Lecture Notes in Computer Science*, pages 55–66. Springer Berlin / Heidelberg. 10.1007/11780885_7.

[Geyken and Schrader, 2006] Geyken, A. and Schrader, N. (2006). LexikoNet, a lexical database based on role and type hierarchies. In *Proceedings of LREC*.

[Grishman, 1996] Grishman, R. (1996). Tipster text phase ii architecture design. In *Proceedings of a workshop on held at Vienna, Virginia: May 6-8, 1996*, TIPSTER '96, pages 249–305, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Grishman, 1997] Grishman, R. (1997). Information extraction: Techniques and challenges. In Pazienza, M., editor, *Information Extraction A Multidisciplinary Approach to an Emerging Information Technology*, volume 1299 of *Lecture Notes in Computer Science*, pages 10–27. Springer Berlin / Heidelberg.

[Grishman, 2003] Grishman, R. (2003). Information extraction. In *Handbook of Computational Linguistics Information Extraction*, chapter 30. Oxford University Press, USA.

[Grishman and Sundheim, 1996] Grishman, R. and Sundheim, B. (1996). Message understanding conference - 6: A brief history. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*.

[Haapalainen and Majorin, 1994] Haapalainen, M. and Majorin, A. (1994). GERTWOL: Ein System zur automatischen Wortformerkennung deutscher Wörter. Technical report, Lingsoft, Inc.

[Had, 2009] Had, M. (2009). Relation extraction zur ergänzung deutschsprachiger firmendossiers. Master's thesis, Technische Universität Dortmund.

[Had et al., 2009] Had, M., Jungermann, F., and Morik, K. (2009). Relation extraction for monitoring economic networks. In Horacek, H.; Metais, E. M. R. W. M., editor, *Proceedings of the 14th International Conference on Applications of Natural Language to Information Systems (NLDB)*, volume 5723 of *Lecture Notes in Computer Science*, pages 103–114. Springer Berlin / Heidelberg.

[Hahn and Romacker, 2000] Hahn, U. and Romacker, M. (2000). An integrated model of semantic and conceptual interpretation from dependency structures. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, volume 1, pages 271–277. Association for Computational Linguistics.

[Hammersley and Clifford, 1971] Hammersley, J. M. and Clifford, P. (1971). Markov fields on finite graphs and lattices.

[Hamp and Feldweg, 1997] Hamp, B. and Feldweg, H. (1997). GermaNet – a lexical-semantic net for German. In Vossen, P., Calzolari, N., Adriaens, G., Sanfilippo, A., and Wilks, Y., editors, *Proceedings of the ACL/EACL-97 Workshop on Automatic Information Extraction and Building Lexical Semantic Resources for NLP applications*, Madrid.

[Harshman, 1970] Harshman, R. (1970). Foundations of the parafac procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16.

[Hastie et al., 2003] Hastie, T., Tibshirani, R., and Friedman, J. H. (2003). *The Elements of Statistical Learning*. Springer, corrected edition.

[Haussler, 1999] Haussler, D. (1999). Convolution kernels on discrete structures. Technical report, University of California at Santa Cruz, Department of Computer Science, Santa Cruz, CA 95064, USA.

[Hazewinkel, 2002] Hazewinkel, M. (2002). *Encyclopaedia of Mathematics*. Springer-Verlag Berlin Heidelberg New York.

[Henrich and Hinrichs, 2010] Henrich, V. and Hinrichs, E. (2010). Gernedit - the germanet editing tool. In Chair), N. C. C., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., and Tapias, D., editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

[Hofmann et al., 2008] Hofmann, T., Schölkopf, B., and Smola, A. J. (2008). Kernel methods in machine learning. *Annals of Statistics*, 36(3):1171–1220.

[Huang et al., 2003] Huang, J., Lu, J., and Ling, L. C. X. (2003). Comparing naive bayes, decision trees, and svm with auc and accuracy. In *Third IEEE International Conference on Data Mining, ICDM 2003*, pages 553–556. IEEE Computer Society.

[Jiang et al., 2010] Jiang, P., Zhang, C., Fu, H., Niu, Z., and Yang, Q. (2010). An approach based on tree kernels for opinion mining of online product reviews. In *Proceedings of the IEEE International Conference on Data Mining*, pages 256 – 265. IEEE.

[Joachims, 1998] Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In Nédellec, C. and Rouveirol, C., editors, *Proceedings of the European Conference on Machine Learning*, pages 137 – 142, Berlin. Springer.

[Joachims, 1999] Joachims, T. (1999). Making Large-Scale SVM Learning Practical. In Schölkopf, B., Burges, C., and Smola, A., editors, *Advances in Kernel Methods – Support Vector Learning*, pages 169 – 184. MIT Press, Cambridge.

[Joachims, 2002] Joachims, T. (2002). *Learning to Classify Text using Support Vector Machines*, volume 668 of *Kluwer International Series in Engineering and Computer Science*. Kluwer.

[Jungermann, 2006] Jungermann, F. (2006). Named entity recognition mit conditional random fields. Master's thesis, Computer Science, University of Dortmund.

[Jungermann, 2007] Jungermann, F. (2007). Named entity recognition without domain-knowledge using conditional random fields. In *Workshop Notes of the Machine Learning for Natural Language Processing Workshop*, pages 16 – 17. van Someren, Marten and Katrenko, Sophia and Adriaans Pieter.

[Jungermann, 2009] Jungermann, F. (2009). Information extraction with RapidMiner. In Hoeppner, W., editor, *Proceedings of the GSCL Symposium 'Sprachtechnologie und eHumanities'*, pages 50–61. Universität Duisburg-Essen, Abteilung für Informatik und Angewandte Kognitionswissenschaft Fakultät für Ingenieurwissenschaften.

[Jungermann, 2010] Jungermann, F. (2010). An information extraction plugin for RapidMiner 5. In *Proceedings of the RapidMiner Community Meeting And Conference (RCOMM 2010)*, pages 67 – 72.

[Jungermann, 2011a] Jungermann, F. (2011a). *Documentation of the Information Extraction Plugin for RapidMiner*.

[Jungermann, 2011b] Jungermann, F. (2011b). Handling tree-structured values in rapidminer. In *Proceedings of the 2nd RapidMiner Community Meeting and Conference (RCOMM 2011)*, pages 151 – 162.

[Jungermann, 2011c] Jungermann, F. (2011c). Tree kernel usage in naive bayes classifiers. In *Proceedings of the LWA 2011*.

[Jungermann and Morik, 2008] Jungermann, F. and Morik, K. (2008). Enhanced services for targeted information retrieval by event extraction and data mining. In *Proceedings of the 13th International Conference on Applications of Natural Language to Information Systems, NLDB 2008*, volume 5039/2008 of *Lecture Notes in Computer Science*, pages 335 – 336. Springer Berlin / Heidelberg.

[Jungermann and Roessler, 2007] Jungermann, F. and Roessler, M. (2007). Ner in texts without domain- or language-knowledge using wikipedia. *intelligenza artificiale*, Anno IV(2):75,76.

[Jurish, 2003] Jurish, B. (2003). A hybrid approach to part-of-speech tagging. *Final Report at BerlinBrandenburgische Akademie der Wissenschaften Berlin*.

[Katrenko and Adriaans, 2007] Katrenko, S. and Adriaans, P. (2007). Learning relations from biomedical corpora using dependency trees. *Knowledge Discovery and Emergent Complexity in Bioinformatics*, Volume 4366/2007:61–80.

[Keerthi and Sundararajan, 2007] Keerthi, S. S. and Sundararajan, S. (2007). Crf versus svm-struct for sequence labeling. Technical report, Yahoo! Research.

[Kiers, 2000] Kiers, H. (2000). Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics*, 14(3):105–122.

[Kilmer and Moravitz Martin, 2004] Kilmer, E. and Moravitz Martin, C. D. (2004). Decomposing a tensor. *SIAM News*, 37(9).

[Kim et al., 2004] Kim, J.-D., Otha, T., Yoshimasa, T., Yuka, T., and Collier, N. (2004). Introduction to the bio-entity recognition task at JNLPBA. In *Proceedings of the Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA-2004)*, Geneva, Switzerland.

[Kindermann and Snell, 1980] Kindermann, R. and Snell, J. L. (1980). Markov Random Fields and Their Applications. *Contemporary mathmatics*, 1.

[Klein and Manning, 2002] Klein, D. and Manning, C. D. (2002). Fast extract inference with a factored model for natural language parsing. In *Proceedings of Advances in Neural Information Processing Systems*.

[Kolda et al., 2005] Kolda, T. G., Bader, B. W., and Kenny, J. P. (2005). Higher-order web link analysis using multilinear algebra. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 242–249, Washington, DC, USA. IEEE Computer Society.

[Kübler and Prokic, 2006] Kübler, S. and Prokic, J. (2006). Why is german dependency parsing more reliable than constituent parsing? In *Proceedings of the Fifth International Workshop on Treebanks and Linguistic Theories - 2006*, pages 7–18, Prague, Czech Republic.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86.

[Kushmerick et al., 1997] Kushmerick, N., Weld, D. S., and Doorenbos, R. (1997). Wrapper induction for information extraction. In *Proceedings of IJCAI-97*.

[Lafferty et al., 2001] Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA.

[Lancaster, 1968] Lancaster, F. W. (1968). *Information Retrieval Systems: Characteristics, Testing and Evaluation*. John Wiley & Sons, New York.

[Lathauwer et al., 2000] Lathauwer, L. D., Moor, B. D., and Vandewalle, J. (2000). A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21(4):1253–1278.

[LDC, 2004b] LDC (2004b). *Annotation Guidelines for Relation Detection and Characterization (RDC)*. LDC, version 4.3.2 edition.

[Leaman and Gonzalez, 2008] Leaman, R. and Gonzalez, G. (2008). Banner: An executable survey of advances in biomedical named entity recognition. In *Proceedings of the Pacific Symposium on Biocomputing 13*, pages 652–663.

[Lee et al., 2004] Lee, C., Hou, W.-J., and Chen, H.-H. (2004). Annotating multiple types of biomedical entities: A single word classification approach. In *Proceedings of the Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA-2004)*, pages 28–29.

[Lezius, 2002] Lezius, W. (2002). TIGERSearch – ein suchwerkzeug für baumbanken. In *Proceedings of the Konferenz zur Verarbeitung natürlicher Sprache (KONVENS 2002)*.

[Lin et al., 2010] Lin, Y.-R., Sun, J., Cao, N., and Liu, S. (2010). Contextour: Contextual contour visual analysis on dynamic multi-relational clustering. In *Proceedings of the SIAM Conference on Data Mining (SDM10)*, pages 418 – 429.

[Lin et al., 2009] Lin, Y.-R., Sun, J., Castro, P., Konuru, R., Sundaram, H., and Kelliher, A. (2009). Metafac: Community discovery via relational hypergraph factorization. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD 2009)*, pages 527–536, Paris, France. ACM.

[Liu, 2010] Liu, B. (2010). Sentiment analysis and subjectivity. *Handbook of Natural Language Processing*, (1):1–38.

[Liu and Nocedal, 1989] Liu, D. C. and Nocedal, J. (1989). On the limited memory method for large scale optimization. In *Mathematical Programming*, volume 45, pages 503–528. Springer Berlin / Heidelberg.

[Lodhi et al., 2002] Lodhi, H., Saunders, C., Shawe-Taylor, J., Christianini, N., and Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444.

[Marcus et al., 1993] Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: the penn treebank. *Comput. Linguist.*, 19:313–330.

[McCallum, 2002] McCallum, A. K. (2002). Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu.

[McDonald, 1996] McDonald, D. (1996). Internal and external evidence in the identification and semantic categorization of proper names. In Boguraev, B. and Pustejovsky, J., editors, *Corpus Processing for Lexical Acquisition*, pages 21–39. MIT Press, Cambridge, MA.

[Mierswa et al., 2006] Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T. (2006). YALE: Rapid Prototyping for Complex Data Mining Tasks. In Eliassi-Rad, T., Ungar, L. H., Craven, M., and Gunopulos, D., editors, *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006)*, pages 935–940, New York, USA. ACM Press.

[Miller et al., 1990] Miller, G., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1990). Five papers on WordNet. Technical Report CSL Report 43, Cognitive Science Laboratory. Princeton University.

[Mitchell, 2010] Mitchell, T. (2010). Generative and discriminative classifiers: Naive bayes and logistic regression.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill, New York.

[Morik et al., 2010] Morik, K., Jungermann, F., Piatkowski, N., and Engel, M. (2010). Enhancing ubiquitous systems through system call mining. In *Proceedings of the ICDM 2010 Workshop on Large-scale Analytics for Complex Instrumented Systems (LACIS 2010)*.

[Moschitti, 2006a] Moschitti, A. (2006a). Efficient convolution kernels for dependency and constituent syntactic trees. In Fuernkranz, J., Scheffer, T., and Spiliopoulou, M., editors, *Procs. ECML*, pages 318 – 329. Springer.

[Moschitti, 2006b] Moschitti, A. (2006b). Making tree kernels practical for natural language learning. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006)*.

[Moschitti et al., 2008] Moschitti, A., Pighin, D., and Basili, R. (2008). Tree kernels for semantic role labeling. *Computational Linguistics Journal*, Special Issue on Semantic Role Labeling.

[Müller et al., 2001] Müller, K.-R., Mika, S., Rätsch, G., Tsuda, K., and Schölkopf, B. (2001). An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201.

[Nédellec, 2005] Nédellec, C. (2005). Learning language in logic - genic interaction extraction challenge. In *Proceedings of the Learning Language in Logic 2005 Workshop at the International Conference on Machine Learning*.

[Nguyen and Guo, 2007] Nguyen, N. and Guo, Y. (2007). Comparisons of sequence labeling algorithms and extensions. In *In Proceedings of the International Conference on Machine Learning (ICML'07)*, pages 681–688.

[Nocedal, 1980] Nocedal, J. (1980). Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782.

[O'Madadhain et al., 2003] O'Madadhain, J., Fisher, D., White, S., and Boey, Y.-B. (2003). The JUNG (java universal network/graph) framework. Technical Report Technical Report UCI-ICS 03-17, School of Information and Computer Science University of California, Irvine, CA 92697-3425.

[Pianta and Zanoli, 2007] Pianta, E. and Zanoli, R. (2007). Exploiting svm for italian named entity recognition. *Intelligenza Artificiale*, 4(2):69–70.

[Piatkowski, 2011] Piatkowski, N. (2011). Parallel algorithms for gpu accelerated probabilistic inference. In *Workshop on Big Learning, NIPS2011*.

[Piatkowski and Morik, 2011] Piatkowski, N. and Morik, K. (2011). Parallel inference on structured data with crfs on gpus. In *International Workshop at ECML PKDD on Collective Learning and Inference on Structured Data (COLISD2011)*, Athens, Greece.

[Pighin and Moschitti, 2009a] Pighin, D. and Moschitti, A. (2009a). Efficient linearization of tree kernel functions. In *CoNLL'09: Thirteenth Conference on Computational Natural Language Learning*, Boulder, CO, USA.

[Pighin and Moschitti, 2009b] Pighin, D. and Moschitti, A. (2009b). Reverse engineering of tree kernel feature spaces. In *EMNLP'09: Empirical Methods of Natural Language Processing*, Singapore.

[Pighin and Moschitti, 2010] Pighin, D. and Moschitti, A. (2010). On reverse feature engineering of syntactic tree kernels. In *Conference on Natural Language Learning (CoNLL-2010)*, Uppsala, Sweden.

[Platt, 1999] Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. In Schölkopf, B., Burges, C., and Smola, A., editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 12. MIT-Press.

[Popov et al., 2003] Popov, B., Kiryakov, A., Manov, D., Kirilov, A., Ognyanoff, D., and Goranov, M. (2003). Towards semantic web information extraction. In *Proceedings of the ISWC'03 Workshop on Human Language Technology for the Semantic Web and Web Services*, pages 1–21.

[Pyle, 1999] Pyle, D. (1999). *Data Preparation for Data Mining*. Morgan Kaufmann Publishers.

[Qian et al., 2008] Qian, L., Zhou, G., Kong, F., Zhu, Q., and Qian, P. (2008). Exploiting constituent dependencies for tree kernel-based semantic relation extraction. In *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 697–704, Morristown, NJ, USA. Association for Computational Linguistics.

[Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

[Ramshaw and Marcus, 1995] Ramshaw, L. and Marcus, M. (1995). Text chunking using transformation-based learning. In Yarovsky, D. and Church, K., editors, *Proceedings of the Third Workshop on Very Large Corpora*, pages 82–94, Somerset, New Jersey. Association for Computational Linguistics.

[Rand, 1971] Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):pp. 846–850.

[Reichartz et al., 2010] Reichartz, F., Korte, H., and Paass, G. (2010). Semantic relation extraction with kernels over typed dependency trees. In *KDD*, pages 773–782.

[Rieck et al., 2010] Rieck, K., Krueger, T., Brefeld, U., and Müller, K.-R. (2010). Approximate tree kernels. *J. Mach. Learn. Res.*, 11:555–580.

[Rifkin and Klautau, 2004] Rifkin, R. and Klautau, A. (2004). In defense of one-vs-all classification. *J. Mach. Learn. Res.*, 5:101–141.

[Roessler, 2006] Roessler, M. (2006). *Korpus-adaptive Eigennamenerkennung*. PhD thesis, Universitaet Duisburg Essen.

[Roessler and Morik, 2005] Roessler, M. and Morik, K. (2005). Using unlabeled texts for named-entity recognition. In Scheffer, T. and Rüping, S., editors, *ICML Workshop on Multiple View Learning*.

[Roessler et al., 2007] Roessler, M., Wagner, A., Jungermann, F., and Hoeppner, W. (2007). Applying walu to annotate named entities in italian texts. *intelligenza artificiale*, Anno IV(2):77,78.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

[Rueping, 2000] Rueping, S. (2000). *mySVM Manual*. Universitaet Dortmund, Lehrstuhl Informatik VIII. http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/.

[Salton and Buckley, 1988] Salton, G. and Buckley, C. (1988). Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523.

[Salton et al., 1975] Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Commun. ACM*, 18:613–620.

[Sebastiani, 2005] Sebastiani, F. (2005). Text categorization. In Zanasi, A., editor, *Text Mining and its Applications to Intelligence, CRM and Knowledge Management*, pages 109–129. WIT Press, Southampton, UK.

[Settles, 2004] Settles, B. (2004). Biomedical named entity recognition using conditional random fields and rich feature sets. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, JNLPBA '04, pages 104–107, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Sha and Pereira, 2003] Sha, F. and Pereira, F. (2003). Shallow parsing with conditional random fields. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 134–141, Morristown, NJ, USA. Association for Computational Linguistics.

[Shashua and Hazan, 2005] Shashua, A. and Hazan, T. (2005). Non-negative tensor factorization with applications to statistics and computer vision. In *Proceedings of the 22nd International Conference on Machine Learning (ICML '05)*, pages 792–799, New York, NY, USA. ACM.

[Skut et al., 1997] Skut, W., Krenn, B., Brants, T., and Uszkoreit, H. (1997). An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP)-97*, Washington, DC.

[Speranza, 2007] Speranza, M. (2007). Evalita 2007 - named entity recognition task - guidelines for participants.

[Sun et al., 2007] Sun, J., Papadimitriou, S., Yu, P. S., and Faloutsos, C. (2007). Graphscope: Parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2007)*, pages 687–696, San Jose, California, USA. ACM New York, NY, USA.

[Sun et al., 2006] Sun, J., Tao, D., and Faloutsos, C. (2006). Beyond streams and graphs: Dynamic tensor analysis. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (KDD '06)*, pages 374–383, New York, NY, USA. ACM.

[Sundheim, 1991] Sundheim, B. M. (1991). Overview of the third message understanding evaluation and conference. In *Proceedings of the 3rd Conference on Message Understanding*, pages 3–16, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Sutton and McCallum, 2007] Sutton, C. and McCallum, A. (2007). An introduction to conditional random fields for relational learning. In Getoor, L. and Taskar, B., editors, *Introduction to Statistical Relational Learning*. MIT Press.

[Thomas and Brailsford, 2005] Thomas, P. L. and Brailsford, D. F. (2005). Enhancing composite digital documents using xml-based standoff markup. In *Proceedings of the 2005 ACM Symposium on Document Engineering (DocEng)*, pages 177–186, New York, NY, USA. ACM.

[Thompson and McKelvie, 1997] Thompson, H. S. and McKelvie, D. (1997). Hyperlink semantics for standoff markup of read-only documents. In *Proceedings of SGML Europe 1997: The next Decade – Pushing the Envelope*, pages 227 – 229.

[Tjong Kim Sang, 2002] Tjong Kim Sang, E. F. (2002). Introduction to the conll-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pages 155–158. Taipei, Taiwan.

[Tjong Kim Sang and Buchholz, 2000] Tjong Kim Sang, E. F. and Buchholz, S. (2000). Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7*, ConLL '00, pages 127–132, Morristown, NJ, USA. Association for Computational Linguistics.

[Tomanek, 2010] Tomanek, K. (2010). *Resource-Aware Annotation through Active Learning*. PhD thesis, University of Dortmund.

[Tsochantaridis et al., 2004] Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector learning for interdependent and structured output spaces. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML 2004)*.

[Tsochantaridis et al., 2005] Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large Margin Methods for Structured and Interdependent Output Variables. *Journal of Machine Learning Research*, 6:1453–1484.

[UIMA Community, 2010] UIMA Community, A. (2010). *UIMA Tools Guide and Reference – Written and maintained by the Apache UIMA Development Community*, 2.3.1 edition.

[Van-Rijsbergen, 1979] Van-Rijsbergen, C. J. (1979). *Information Retrieval*. Butterworths, London, 2 edition.

[Vapnik, 1995] Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer, New York.

[Vilain and Day, 2000] Vilain, M. and Day, D. (2000). Phrase parsing with rule sequence processors: an application to the shared conll task. In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7*, ConLL '00, pages 160–162, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Vishwanathan et al., 2006] Vishwanathan, S. V. N., Schraudolph, N. N., Schmidt, M. W., and Murphy, K. P. (2006). Accelerated training of conditional random fields with stochastic gradient methods. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 969–976, New York, NY, USA. ACM.

[Vishwanathan and Smola, 2003] Vishwanathan, S. V. N. and Smola, A. J. (2003). Fast kernels for string and tree matching. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15 — Proceedings of the 2002 Neural Information Processing Systems Conference (NIPS 2002)*, pages 569 – 576, Vancouver, British Columbia, Canada.

[Walker, 2007] Walker, C. (2007). Resource integration for named entity tagger development in italian. *Intelligenza Artificiale*, 4(2):71–72.

[Wallach, 2002] Wallach, H. (2002). Efficient training of conditional random fields. Master's thesis, Division of Informatics, University of Edinburgh.

[Wang et al., 2006] Wang, X., Sun, J.-T., Chen, Z., and Zhai, C. (2006). Latent semantic analysis for multiple-type interrelated data objects. In *Proceedings of the 29th Annual International ACM Conference on Research and Development in Information Retrieval (SIGIR 2006)*, pages 236–243, New York, NY, USA. ACM.

[Weston et al., 2005] Weston, J., Bordes, A., and Bottou, L. (2005). Online (and offline) on an even tighter budget. In Cowell, R. G. and Ghahramani, Z., editors, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, January 2005, Barbados*, pages 413–420. Society for Artificial Intelligence and Statistics.

[Yang and Pedersen, 1997] Yang, Y. and Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *Proceedings of 14th International Conference on Machine Learning*, pages 412–420. Morgan Kaufmann.

[Yangarber and Grishman, 1998] Yangarber, R. and Grishman, R. (1998). NYU: Description of the proteus/PET system as used for MUC-7. In *In Proceedings of the Seventh Message Understanding Conference (MUC-7)*.

[Yangarber and Grishman, 2000] Yangarber, R. and Grishman, R. (2000). Machine learning of extraction patterns from unannotated corpora. In *Proceedings of the Workshop on Machine Learning for Information Extraction, 14th European Conference on Artificial Intelligence*.

[Zelenko et al., 2003] Zelenko, D., Aone, C., and Richardella, A. (2003). Kernel methods for relation extraction. *Journal of Machine Learning Research*, pages 1083–1106.

[Zhang et al., 2007] Zhang, M., Che, W., Aw, A. T., Tan, C. L., Zhou, G., Liu, T., and Li, S. (2007). A grammar-driven convolution tree kernel for semantic role classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 200–207. Association for Computational Linguistics.

[Zhang et al., 2006] Zhang, M., Zhang, J., Su, J., and Zhou, G. (2006). A composite kernel to extract relations between entities with both flat and structured features. In *Proceedings 44th Annual Meeting of ACL*, pages 825–832.

[Zhang et al., 2001] Zhang, T., Damerau, F., and Johnson, D. (2001). Text chunking using regularized winnow. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ACL '01, pages 539–546, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Zhao and Grishman, 2005] Zhao, S. and Grishman, R. (2005). Extracting relations with integrated information using kernel methods. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 419–426, Morristown, NJ, USA. Association for Computational Linguistics.

[Zhou et al., 2010] Zhou, G., Qian, L., and Fan, J. (2010). Tree kernel-based semantic relation extraction with rich syntactic and semantic information. *Inf. Sci.*, 180:1313–1325.

[Zhou and Su, 2004] Zhou, G. and Su, J. (2004). Exploring deep knowledge resources in biomedical name recognition. In *Proceedings of the Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA-2004)*, Geneva, Switzerland.

[Zhou et al., 2005] Zhou, G., Su, J., Zhang, J., and Zhang, M. (2005). Exploring various knowledge in relation extraction. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 427–434, Ann Arbor. Association for Computational Linguistics.

[Zhou et al., 2007] Zhou, G., Zhang, M., Ji, D. H., and Zhu, Q. (2007). Tree kernel-based relation extraction with context-sensitive structured parse tree information. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics.

[Zhou and Zhu, 2011] Zhou, G.-D. and Zhu, Q.-M. (2011). Kernel-based semantic relation detection and classification via enriched parse tree structure. *J. Comput. Sci. Technol.*, 26:45–56.

Abbreviations

CAS Common Analysis Structure
CRF Conditional Random Fields
DM Datamining
FTK Fast Tree Kernel
IE Information Extraction
HMM Hidden Markov Model
KDD Knowledge Discovery in Databases
ML Machine Learning
MRF Markov Random Field
MUC Message Understanding Conference
NB Naïve Bayes
NE Named Entity
NER Named Entity Recognition
PoS Part of Speech
QTK Quadratic Tree Kernel
RE Relation Extraction
SMO Sequential Minimal Optimization
Sofa Subject of Analysis
SVM Support Vector Machine
TK Tree Kernel
TKNB Tree Kernel Naïve Bayes
TKSVM Tree Kernel Support Vector Machine
UIMA Unstructured Information Management Architecture

Appendix A

Operator Reference

A.1 Tokenizers

In this section we will describe the parameters and the input and output ports of the `SentenceTokenizer` in detail. These parameters and ports are extended from the super-class `TokenizerImpl` and they are the same for `WordTokenizer` and `LineTokenizer`. A new tokenizer should be extend the class `TokenizerImpl` and the method `String[] tokenization(String text)` should be implemented.

SentenceTokenizer This tokenizer splits texts into sentences. The parameters for this operator are presented in Table A.1 and the I/O ports are presented in Table A.2.

Parameter	Description
attribute	Select the attribute here which has to be used for extracting the text to be tokenized from each example.
new token-name	Type a new attribute name in here which will be created to write the new tokens to.

Table A.1: Parameters for *SentenceTokenizer*

I/O	port-name	Description
I	**example set input**	The example set which will be used for tokenization.
O	**example set output**	The example set containing the new tokens.
O	**original example set output**	The original example set got at the input port.

Table A.2: I/O-ports for *SentenceTokenizer*

WordTokenizer This tokenizer splits texts into words. The parameters and the I/O ports are the same as for the `SentenceTokenizer`.

LineTokenizer This tokenizer splits texts into lines. It splits texts contained in examples by using the linebreaks in the texts, one could say. The parameters and the I/O ports are the same as for the `SentenceTokenizer`.

A.2 Visualizer

The visualizers are used for the annotation and the visualization of textual data. The parameters and the ports of the `ParseTreeVisualizer` and the `TextVisualizer` are comparable, and therefore we only present the parameters and ports of the `ParseTreeVisualizer`.

ParseTreeVisualizer This operator creates a visualization of parse trees. For each example the parse tree and the corresponding sentence will be displayed. Only one example is displayed at a time, but one can switch from one example to another during the visualizing process. Table A.3 describes the parameters of the operator and Table A.4 describes the ports.

Parameter	Description
sentence-attribute	Select the attribute here which has to be used for extracting the text to be displayed in addition to the parse tree from each example.
parsetree-atrribute	Select the attribute here which has to be used for extracting the parse tree to be displayed from each example.

Table A.3: Parameters for *ParseTreeVisualizer*

I/O	port-name	Description
I	**example set input**	The example set which will be used for tokenization.
O	**parsetree visualization output**	The visualization component which becomes visible in the results workspace.
O	**original example set output**	The original example set got at the input port.

Table A.4: I/O-ports for *ParseTreeVisualizer*

TextAnnotator The `TextAnnotator` is used for annotating a textual dataset. The textual data of the `text-attribute` are displayed. The user can annotate the tokens by marking them and by selecting formerly created labels. During the annotation process it is possible to create new label types which can directly be used for annotating. The labels are stored in the `label-attribute` for each token. After finish

the annotation process the dataset is stored in a repository which is selected using the parameter `repository-entry`.

Parameter	Description
repository-entry	Select a repository entry here which has to be used for storing the dataset after having finished the annotation process.
text-attribute	Select the attribute here which has to be used for extracting the text to be displayed in addition to labeling from each example.
label-attribute	Select the attribute here which has to be used for extracting the label to be displayed from each example.

Table A.5: Parameters for *TextAnnotator*

I/O	port-name	Description
I	**example set input**	The example set which will be used for tokenization.
O	**annotation output**	The annotation component which becomes visible in the results workspace.
O	**example set output**	The original example set got at the input port.

Table A.6: I/O-ports for *TextAnnotator*

TextVisualizer This operator creates a visualization of labeled textual data. Like for the `ParseTreeVisualizer` two attributes have to be selected. The nominal values of the first attribute will be displayed as text whereas the nominal values of the second attribute will be used as labels. The parameters and ports are comparable to the ones of the `ParseTreeVisualizer` and therefore are not specified again.

A.3 Preprocessing

WordVectorPreprocessing This operator creates a BOW representation (see Section 3.1) of the values given by the attributes defined by the parameter `valueAttribute`. The regular expression which can be set using the parameter `splitExpression` is used for splitting the suggested values. Each unique resulting split value is used to create a new attribute containing a 1 if the value in the attribute `valueAttribute` contains the split value and 0, otherwise. The operator is very similar and we disclaim on presenting the parameters and ports in detail.

A.3.1 Named Entity Recognition

The preprocessing operators for NER are extending two types of super classes. The first one is `PreprocessOperatorImpl` and the second one is

`MultiPreprocessOperatorImpl`. `PreprocessOperatorImpl` enriches the dataset by one additional attribute for every value it sees, whereas `MultiPreprocessOperatorImpl` is creating multiple additional attributes for every value.

WordPreprocessing `WordPreprocessing` works like the operator `PrefixPreprocessing`. The difference is that the extracted value from attribute `wordAttributeName` is directly stored (without any manipulation) to the attribute `operatorName`. The parameter `length` is not needed and therefore it is not apparent.

PrefixPreprocessing `PrefixPreprocessing` extends `PreprocessOperatorImpl`. The most important method to implement after extending `PreprocessOperatorImpl` is `String newValueToInsert(String w, int length,int index)`. This method is delivering the new attribute value in a `String` representation and is called using the parameters `w` which contains the value of the attribute `wordAttributeName` of the `index`th example shifted by `position` examples. The parameter `length` can be used if the resulting value should contain a specific length. All operators extending `PreprocessOperatorImpl` until some exceptions are defined by the same parameters which exemplarily are listed in Table A.7. The input and output ports are presented in Table A.8. This operator creates a new attribute containing the prefixes of length `length` of the values extracted from attribute `wordAttributeName`. If the value extracted from attribute `wordAttributeName` is 'Felix', for instance, the prefix of length 3 is 'Fel' and will be stored in the attribute `operatorName`.

Parameter	Description
position	The position of the example to be chosen relative to the current example. (-2 will take the example two examples before, -2_2 will take all examples between two before and two after the current example, $-2, 2$ will take the examples two before and two after the current example)
length	The parameter length is determining the length of the extracted prefix.
operatorName	Select the name of the new attribute here which will contain the created values.
wordAttributeName	Select the attribute which will be used to extract the values to be used for preprocessing.

Table A.7: Parameters for *PrefixPreprocessing*

I/O	port-name	Description
I	**example set input**	The example set which will be used for preprocessing.
O	**example set output**	The example set containing the new attribute(s).
O	**original**	The original example set got at the input port.

Table A.8: I/O-ports for *PrefixPreprocessing*

ngram_preprocessing `ngram_preprocessing` extends `MultiPreprocessOperatorImpl`. The most important method to implement after extending `MultiPreprocessOperatorImpl` is `ArrayList<String> newValueToInsert(String w, int length, int index)`. This method is delivering a list of new attribute values and the processing is comparable to the processing of `PreprocessOperatorImpl`. The resulting values will be converted into a binary attribute, each. This operator will split the values extracted from attribute `wordAttributeName` into pieces of length `length`. Each unique piece will result in a new attribute which contains a 1 for each example containing this piece or a 0 otherwise. The parameters and input and output ports are comparable to the already presented operators.

IOBPreprocessing This operator cuts off the prefixes 'B-' and 'I-' of `wordAttributeName`.

IndexPreprocessing This operator just takes the index of the relatively chosen example and creates an additional attribute containing this information.

SuffixPreprocessing Works like `PrefixPreprocessing` but with the difference that the suffixes are extracted.

WordCountPreprocessing This operator counts the words of the current sequence and creates an attribute containing this information.

GeneralizationPreprocessing This operator generalizes the currently chosen value of `wordAttributeName` by replacing capital letters by 'A', small letters by 'a' and digits by 'x'. The resulting value is stored in the newly created attribute `operatorName`.

WikipediaPreprocessing This operator creates a new attribute and puts the wikipedia-category of the `wordAttributeName` of the relatively chosen example as a value into it.

LetterCountPreprocessing This operator counts the number of letters of the value extracted of `wordAttributeName` of the relatively chosen example and stores them as a value into a newly created attribute.

RegExPreprocessing `RegExPreprocessing` also extends `MultiPreprocessOperatorImpl`. The chosen value is checked against a number of regular expression. If an expression is matched, the corresponding attribute of the example is allocated with a 1 and otherwise it is allocated with a 0.

TagListPreprocessing This operator checks the `wordAttributeName` of the relatively chosen example against a given tag list. If the current value is contained in the list, the newly created attribute is allocated with 1 and otherwise it is allocated with 0.

StartOfSequencePreprocessing This operator creates a new attribute that contains a 1 if the current token shifted by `location` is at the beginning of the sequence/sentence and 0 otherwise.

EndOfSequencePreprocessing Works like `StartOfSequencePreprocessing` but for the end of sequences/sentences.

A.3.2 Relation Extraction

In this section we present the preprocessing operators related to *Relation Extraction*.

TreeCreatorAndPreprocessor The operator `TreeCreatorAndPreprocessor` reads the values out of a particular attribute and creates a `Tree`-representation out of it. The resulting tree-structured value is stored in a newly created attribute. The creation of the tree structure is done by parsing a machine-readable or natural language sentence. Additionally, a tree structure given in `String`-representation can be read and converted into a tree-structured attribute value. The parameters and the input and output ports of this operator are shown in Tables A.9 and A.10.

Parameter	Description
valueAttribute	The attribute which contains the tree or the sentence to be parsed.
needParsing	Does the attribute value need to be parsed?
modelfile	The file containing a parser model
parseTreeType	The trees have to be pruned for special tasks (see Chapter 5). Select pruning type here.
FTK	Selecting this will activate the list-creation needed for the FTK.

Table A.9: Parameters for *TreeCreatorAndPreprocessor*

I/O	port-name	Description
I	**example set input**	The example set which will be used for preprocessing.
O	**example set output**	The example set additionally containing the new attribute.

Table A.10: I/O-ports for *TreeCreatorAndPreprocessor*

HTMLTreePreprocessing The `HTMLTreePreprocessing` operator works like the `TreeCreatorAndPreprocessor` operator as it creates tree-structured attribute values. In contrast to the `TreeCreatorAndPreprocessor` the resulting trees are not pruned.

Zhou Features

In this section we present some of the features published in [Zhou and Su, 2004]. According to the two entities creating the relation candidate, several informational units concerning these entities are extracted by the features. The features are extending the class `RelationPreprocessOperatorImpl`. Two particular methods have to be implemented by each operator: `String newValueToInsertTree(Tree t)` and `String newValueToInsert(String w1, String w2, List<String> addAtts, int length, ExampleIteration exIter)`. Both methods are creating the value for the newly created attribute. The first is based on a tree-representation, and the second one is based on the position of the two entities in the sentence.

WBNULLPreprocessing This operator creates a new binary attribute containing a 'true'-value if no word is located between the two entities creating the relation candidate. The parameters and input and output ports of this operator are presented in Tables A.11 and A.12.

Parameter	Description
operatorName	The name of the new attribute that will be created.
use tree	If this is selected the tree-structured attribute value will be used.
firstAttributeName	The name of the attribute containing the name of the first entity must be inserted here (only if tree is used).
secondAttributeName	The name of the attribute containing the name of the second entity must be inserted here (only if tree is used).
additional Attributes	Additional attributes can be selected using this parameter list. These attributes are only needed by several operators.

Table A.11: Parameters for *WBNULLPreprocessing*

I/O	port-name	Description
I	**example set input**	The example set which will be used for preprocessing.
O	**example set output**	The example set additionally containing the new attribute.
O	**original**	The original example set got from the input port.

Table A.12: I/O-ports for *WBNULLPreprocessing*

WBFLPreprocessing This operator creates a new attribute which will contain the word between the two entities creating the relation candidate if there is only one word between. The parameters and input and output ports are the same like for `WBNULLPreprocessing`.

WBFPreprocessing This operator creates a new attribute which will contain the first word between the two entities creating the relation candidate if there is more than one word between. The parameters and input and output ports are the same like for `WBNULLPreprocessing`.

WBLPreprocessing This operator creates a new attribute which will contain the last word between the two entities creating the relation candidate if there is more than one word between. The parameters and input and output ports are the same like for `WBNULLPreprocessing`.

WBOPreprocessing This operator creates a new attribute which will contain the words except the first and the last one between the two entities creating the relation candidate if there is more than one word between. The parameters and input and output ports are the same like for `WBNULLPreprocessing`.

BM1FPreprocessing This operator creates a new attribute which will contain the word before the first entity of the two entities creating the relation candidate. The parameters and input and output ports are the same like for `WBNULLPreprocessing`.

BM1LPreprocessing This operator creates a new attribute which will contain the word two words before the first entity of the two entities creating the relation candidate. The parameters and input and output ports are the same like for `WBNULLPreprocessing`.

AM2FPreprocessing This operator creates a new attribute which will contain the word after the last entity of the two entities creating the relation candidate. The parameters and input and output ports are the same like for `WBNULLPreprocessing`.

AM2LPreprocessing This operator creates a new attribute which will contain the word two words after the last entity of the two entities creating the relation candidate. The parameters and input and output ports are the same like for `WBNULLPreprocessing`.

M1greaterM2Preprocessing This operator creates a new binary attribute which will contain 'true' if the second entity is encapsulated in the first entity. The parameters and input and output ports are the same like for `WBNULLPreprocessing`.

NumberOfMBPreprocessing This operator creates a new attribute which will contain the number of entity mentions between the two entities creating the relation candidate. The parameters and input and output ports are the same like for `WBNULLPreprocessing`.

NumberOfWBPreprocessing This operator creates a new attribute which will contain the number of words between the two entities creating the relation candidate. The parameters and input and output ports are the same like for `WBNULLPreprocessing`.

A.4 Meta

Binary2MultiClassRelationLearner This operator works like the `Polynomial by Binomial Classification` operator already available in *RapidMiner*. If a dataset containing more than two classes should be processed by a binary learner a strategy to use the binary learner will have to be chosen (see Section 2.4.3). The main reason for developing a new operator for this purpose is the fact that some candidates for *Relation Extraction* are defined only for some special relation types. It follows that the amount of relation candidates varies for each type of relation class currently focusing. For each class different parameters have to be adjusted for the internal learning mechanism. The parameters and input and output ports are presented in Tables A.13 and A.14

Parameter	Description
classification strategies	The strategy to be chosen to partition the dataset for applying the internal learner.
use local random seed	An own random seed is used to achieve repeatable results.
event1	The name of the attribute containing the type of the first entity must be inserted here.
event2	The name of the attribute containing the type of the second entity must be inserted here. Using the two types of entities allows to check whether the combination is suitable for particular relation classes.
null-Label	The default-class has to be selected here.
confidence	The confidence-level which has to be achieved to predict a certain class. If the confidence-level is not being achieved by any class the default-class is predicted.
epsilon-list	A particular epsilon-value can be adjusted for every learner/class here.

Table A.13: Parameters for *Binary2MultiClassRelationLearner*

I/O	port-name	Description
I	**training set**	The example set which will be used for learning.
O	**model**	The model created by the internal learning mechanism.
O	**example set**	The original example set got from the input port.

Table A.14: I/O-ports for *Binary2MultiClassRelationLearner*

A.5 Data

CSVbatchedDataReader The `CSVbatchedDataReader` operator extends the `Read CSV` operator which is already available in *RapidMiner*. The `CSVbatchedDataReader` operator allows to read in datasets which are prepared like described in Section 7.2.1.

A.6 Learner

TreeKernel Naive Bayes The `TreeKernel Naive Bayes` operator is the implementation of the approach presented in Section 6. We only present the parameters differing from the `Naive Bayes (Kernel)` operator in Table A.15. Table A.16 contains the input and output ports of the operator.

Parameter	Description
lambda	The first kernel to be used for the *Composite Kernel* (just *Entity* is possible)
sigma	The list of attributes to be used by the *Entity Kernel*
gaussian	The λ-value to be used for QTK or FTK by the *Composite Kernel*
treestructure selection	The λ-value to be used for QTK or FTK by the *Composite Kernel*
distribution	The λ-value to be used for QTK or FTK by the *Composite Kernel*

Table A.15: Parameters for *Treekernel Naive Bayes*

I/O	port-name	Description
I	**training set**	The example set which will be used for learning.
O	**model**	The model created by the internal learning mechanism.
O	**exampleSet**	The original example set got from the input port.

Table A.16: I/O-ports for *Treekernel Naive Bayes*

TreeSVM We enhanced the already available *JMySVM* [Rueping, 2000] implementation in *RapidMiner* by abilities to process tree structures. We implemented the Kernel presented by [Collins and Duffy, 2001], the Fast Tree Kernel by [Moschitti, 2006b, Moschitti, 2006a] and the Composite Kernel by [Zhang et al., 2006]. The operator `TreeSVM` has some additional parameters in contrast to the `Support Vector Machine` operator. These parameters are shown in Table A.17. The input and output ports are presented in Table A.18.

Parameter	Description
kernel type	The kernel type to be used (*Collins and Duffy (trivial)*, *Moschitti (FTK)*, *Composite Kernel*)
CollinsDuffy Kernel Lambda	The λ-value to be used (see Section 5.5) for QTK or FTK
Composite Kernel Alpha	If the *Composite Kernel* is used the α value (see Section 5.4.5) can be adjusted here.
kernel type 1	The first kernel to be used for the *Composite Kernel* (just *Entity* is possible)
attribute list	The list of attributes to be used by the *Entity Kernel*
kernel type 2	The second kernel to be used for the *Composite Kernel* (QTK and FTK possible)
Collins Duffy Kernel Lambda (composite)	The λ-value to be used for QTK or FTK by the *Composite Kernel*

Table A.17: Additional parameters for *TreeSVM*

I/O	port-name	Description
I	**training set**	The example set which will be used for learning.
O	**model**	The model created by the internal learning mechanism.
O	**estimated performance**	The original example set got from the input port.
O	**exampleSet**	The original example set got from the input port.

Table A.18: I/O-ports for *TreeSVM*

Kernel Perceptron The `Kernel Perceptron` operator is the implementation of the approach presented in Section 6.4. The parameters of the operator are presented in Table A.19. The input and output ports are presented in Table A.20.

Parameter	Description
kernel type	The kernel type to be used (*CollinsDuffy*, *FastTree*, *Treeceptron*, *DAGperceptron*, *OneDAGperceptron*)
attribute	The attribute containing the tree-structures.
lambda	The λ-value to be used (see Section 5.5) for the tree kernel.
sigma	The σ-value to be used (see Section 5.5) for the tree kernel.
bootstrap	If this is selected the at each step a randomly chosen example will be selected.
stopping	After this number of iteration training will stop. Selecting -1 will make the perceptron do one run on the complete example set.

Table A.19: Parameters for *Kernel Perceptron*

I/O	port-name	Description
I	**training set**	The example set which will be used for learning.
O	**model**	The model created by the internal learning mechanism.
O	**example set**	The original example set got from the input port.

Table A.20: I/O-ports for *Kernel Perceptron*

ConditionalRandomField This operator implements a CRF which is presented in Section 2.4.2. The parameters and the input and output ports of this operator are presented in Tables A.21 and A.22. This operator needs an embedded particular optimization operator. The optimizer is presented in the next section (Section A.6.1).

Parameter	Description
text-Attribute name	Select the attribute containing the words of the sentence to be processed

Table A.21: Parameters for *ConditionalRandomField*

I/O	port-name	Description
I	**example set input**	The example set which will be used for learning.
O	**example set output**	The original example set got from the input port.
O	**model output**	The model created by the CRF.

Table A.22: I/O-ports for *ConditionalRandomField*

SVMlight The `SVMlight` operator allows the usage of the $SVMlight$ of [Joachims, 2002] in *RapidMiner*. This is possible by connecting the C-code of the $SVMlight$ with *RapidMiner* via JNI. We used the implementation of Martin Theobald[1] and implemented the possibility to perform tree kernel calculations in addition.

A.6.1 Optimizer

LBFGS_optimizer The `LBFGS_optimizer` can be used as an optimization method for the `ConditionalRandomField`-operator. The optimization technique is presented by [Nocedal, 1980, Liu and Nocedal, 1989].

Parameter	Description
maximum number of iterations	After this number of iterations the optimization is stopped.
eps for convergence	The solution is assumed to be optimal if a smaller ϵ value than this one is achieved during optimization.
features sorted in array	Select this if the features are sorted in an array.

Table A.23: Parameters for *LBFGS_optimizer*

I/O	port-name	Description
I	**feature set input**	The feature set which will be used for optimization.
O	**feature set output**	The feature set containing the optimized weights.

Table A.24: I/O-ports for *LBFGS_optimizer*

A.7 Validation

If particular examples or tokens are grouped into sequences or sentences these sequences should be respected by operators which split the example set like `SplittedXBatchValidation` and `BatchSplitValidation` or `SequenceSampling`.

[1] `http://www.mpi-inf.mpg.de/~mtb/svmlight/JNI_SVM-light-6.01.zip`

PerformanceEvaluator This operator is a special performance evaluator for NER. It calculates precision, recall and f-measure for each class except the default-class. Additionally, the accumulated values are calculated because those values are used very often in the *Information Extraction* community.

Parameter	Description
precision	The precision will be calculated for each class except the default class (`not-NER-tag`) if this box is selected.
recall	The recall will be calculated for each class except the default class (`not-NER-tag`) if this box is selected.
f-measure	The f-measure will be calculated for each class except the default class (`not-NER-tag`) if this box is selected.
overall-precision	The precision will be calculated for all classes together except the default class (`not-NER-tag`) if this box is selected.
overall-recall	The recall will be calculated for each class except the default class (`not-NER-tag`) if this box is selected.
overall-f-measure	The f-measure will be calculated for each class except the default class (`not-NER-tag`) if this box is selected.
stopword	A special token represents the end of a sentence in some datasets. This token can be selected here so that it is not used for the calculation of the performance.
not-NER-tag	Type in the default-class here.
iob	Select this box if the dataset is in IOB-format.

Table A.25: Parameters for *PerformanceEvaluator*

I/O	port-name	Description
I	**example set input**	The example set which will be used for evaluating the performance.
O	**example set output**	The original example set from the input port.
O	**simplePerformance**	Delivers the `SimpleResultObject` from *RapidMiner*.
O	**performance**	Delivers the performance calculated by this operator.

Table A.26: I/O-ports for *PerformanceEvaluator*