Hierarchical Text Classification using Methods from Machine Learning

Michael Granitzer

Hierarchical Text Classification using Methods from Machine Learning

Master's Thesis

at

Graz University of Technology

submitted by

Michael Granitzer

Institute of Theoretical Computer Science (IGI),
Graz University of Technology
A-8010 Graz, Austria

27th October 2003

Advisor: Univ.-Prof. DI Dr. Peter Auer

Hierarchische Textklassifikation mit Methoden des maschinellen Lernens

Diplomarbeit

an der

Technischen Universität Graz

vorgelegt von

Michael Granitzer

Institut für Grundlagen der Informationsverarbeitung (IGI),
Technische Universität Graz
A-8010 Graz

27. Oktober 2003

Diese Arbeit ist in englischer Sprache verfaßt.

Betreuer: Univ.-Prof. DI Dr. Peter Auer

Abstract

Due to the permantently growing amount of textual data, automatic methods for organizing the data are needed. Automatic text classification is one of this methods. It automatically assigns documents to a set of classes based on the textual content of the document.

Normally, the set of classes is hierarchically structured but today's classification approaches ignore hierarchical structures, thereby loosing valuable human knowledge. This thesis exploits the hierarchical organization of classes to improve accuracy and reduce computational complexity. Classification methods from machine learning, namely BoosTexter and the newly introduced Centroid-Boosting algorithm, are used for learning hierarchies. In doing so, error propagation from higher level nodes and comparing decisions between independently trained leaf nodes are two problems which are considered in this thesis.

Experiments are performed on the Reuters 21578, the Reuters Corpus Volume 1 and the Ohsumed data set, which are well known in literature. Rocchio and Support Vector Machines, which are state of the art algorithms in the field of text classification, serve as base line classifiers. Comparing algorithms is done by applying statistical significance tests. Results show that, depending on the structure of a hierarchy, accuracy improves and computational complexity decreases due to hierarchical classification. Also, the introduced model for comparing leaf nodes yields an increase in performance.

Kurzfassung

Durch die starke Zunahme textueller Daten entsteht die Notwendigkeit automatische Methoden zur Datenorganisation einzusetzten. Automatische Textklassifikation ist eine dieser Techniken. Sie ordnet Textdokumente auf inhaltlicher Basis automatisch einer definierten Menge von Klassen zu.

Die Klassen sind meist hierarchisch strukturiert, wobei die meisten heutigen Klassifikationsansätze diese Struktur ignorieren. Dadurch geht a priori Information verloren. Die vorliegende Arbeit beschäftigt sich mit dem Ausnützen hierarchischer Strukturen zur Verbesserung von Genauigkeit und Zeitkomplexität. BoosTexter und der hier neu vorgestellte CenroidBooster, Algorithmen aus dem Bereich des maschinellen Lernens, werden als hierarchische Klassifikationsmethoden eingesetzt. Die bei hierarchischer Klassifikation entstehenden Probleme der Fehlerfortpflanzung von hierarchisch höheren Knoten und das Vergleichen von Entscheidungen aus unahängig trainierten Blättern werden dabei berücksichtigt.

Die Verfahren werden anhand bekannter Datensätze, dem Reuters-21578, Reuters Corpus Volume 1 und Ohsumed Datensatz analysiert. Dabei dienen Support Vector Maschinen und Rocchio, beides State of the Art Techniken als Vergleichsbasis. Die Vergleiche zwischen Ergebnissen erfolgen anhand statistischer Signifikanztests. Die Ergebnisse zeigen, daß abhängig von der hierarchischen Struktur, Genauigkeit und Zeitkomplexität verbessert werden können. Der Ansatz zum Vergleich von unabhängig trainierten Blättern verbessert die Genauigkeit ebenfalls.

I hereby certify that the work presented in this thesis is my own and that work performed by others is appropriately cited.

Ich versichere hiermit, diese Arbeit selbständig verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Danksagung

Ich möchte an diesem Punkt meinen Eltern und Großeltern danken. Sie haben es mir ermöglicht, mein Studium und somit auch diese Arbeit in Angriff zu nehmen. Danke.

Mein Dank gilt auch Professor Dr. Peter Auer, der mir die Gelegenheit gab, eine Diplomarbeit im Bereich des maschinellen Lernens zu verfassen und mir mit guten Ratschläge und Hinweisen zur Seite stand.

Vielen herzliche Dank auch an meine Freundin Gisela Dösinger, auf deren Hilfe ich immer zählen konnte und daß sie, sowie meine Arbeitskollegen Wolfgang Kienreich und Vedran Sabol, immer ein offenes Ohr für mich hatte.

Die letzte Danksagung gilt meinem Arbeitgeber, dem Know-Center, fr das zu Verfgung stellen von technischen und zeitliche Ressourcen.

Der Weg ist das Ziel

Michael Granitzer
Graz, Austria, Oktober 2003

Contents

List of Figures

List of Tables

1. Introduction and Problem Statement

This chapter introduces the need for text classification in today's world and gives some examples of application areas. Problems of flat text classification compared to hierarchical text classification and how they may be solved by incorporating hierarchical information are outlined. Given this motivation, a general mathematical formulation on flat and hierarchical text classification, which is the problem formulation for this thesis, concludes the chapter.

1.1. Introduction

One common problem in the information age is the vast amount of mostly unorganized information. Internet and corporate Intranets continue to increase and organization of information becomes an important task for assisting users or employees in storing and retrieving information. Tasks such as sorting emails or files into folder hierarchies, topic identification to support topic-specific processing operations, structured search and/or browsing have to be fulfilled by employees in their daily work. Also, available information on the Internet has to be categorized somehow. Web directories like for example Yahoo are build up by trained professionals who have to categorize new web sites into a given structure.

Mostly this tasks are time consuming and sometimes frustrating processes if done manually. Categorizing new items manually has some drawbacks:

1. For special areas of interest, specialists knowing the area are needed for assigning new items (e.g. medical databases, juristic databases) to predefined categories.

2. Manually assigning new items is an error-prone task because the decision is based on the knowledge and motivation of an employee.

3. Decisions of two human experts may disagree (inter-indexing inconsistency)

Therefore tools capable of automatically classifying documents into categories would be valuable for daily work and helpful for dealing with today's information volume. A number of statistical classification and machine learning techniques like Bayesian Classifier, Support Vector Machines, rule learning algorithms, k-NN, relevance feedback, classifier ensembles, and neural networks have been applied to the task.

Chapter 2 introduces traditional indexing and term selection methods as well as state of the art techniques for text classification. Multi-class classification using binary classifier and performance measurements are outlined.

Issues of hierarchical text classification and the proposed model for this thesis are illustrated in chapter 3. Finally experiments and their results are presented in chapter 4 and the conclusion of this thesis is given in chapter 5.

To give a motivation for text classification, this section concludes with application areas for automatic text classification.

1.1.1.   Automatic Indexing

Automatic Indexing deals with the task of describing the content of a document through assigning key words and/or key phrases. The key words and key phrases belong to a finite set of words called *controlled vocabulary*. Thus, automatic indexing can be viewed as a text classification task if each keyword is treated as separate class. Furthermore, if this vocabulary is a thematic hierarchical thesaurus this task can be viewed as hierarchical text classification.

1.1.2.   Document Organization

Document organization uses text classification techniques to assign documents to a predefined structure of classes. Assigning patents into categories or automatically assigning newspaper articles to predefined schemes like the IPTC Code (International Press and Telecommunication Code) are examples for document organization.

1.1.3.   Text Filtering

Document organization and indexing deal with the problem of sorting documents into predefined classes or structures. In text filtering there exist only two disjoint classes, relevant and irrelevant. Irrelevant documents are dropped and relevant documents are delivered to a specific destination. E-mail filters dropping "junk" mails and delivering "serious" mails are examples for text filtering systems.

1.1.4.   Word Sense Disambiguation

Word Sense Disambiguation tries to find the sense for an ambiguous word within a document by observing the context of this word (e.g. bank=river bank, financial bank). WSD plays an important role in machine translation and can be used to improve document indexing.

1.2.   Definitions and Notations

The following section introduces definitions and mathematical notations used in this thesis. For easier reading this section precedes the problem formulation.

1.2.1.   Notation

- $\vec{x}$ Vectors are written lower case with an over lined arrow.
- D Sets are written bold, italic and upper case.
- H High dimensional (vector) spaces are written italic and upper case.
- $\mathfrak{M}$ Matrices are written as German "Fraktur" characters and upper case.
- $\mathcal{G}$ Graphs are written calligraphic and upper case
- C, D Classes and Documents are written San Serif and upper case.
- $\mathrm{sgn}(\cdot)$ returns 1 if the sign is positive, -1 else.
- $\pi[\cdot]$ returns 1 if the argument of π is true, 0 else.
- $\vec{w} \cdot \vec{d}$ denotes the inner product of two vectors

1.2.2. Definitions

Since the implemented algorithms are used to learn hierarchies some preliminary definitions describing properties of such hierarchies and their relationship to textual documents and classes are given.

Hierarchy ($\mathcal{H}$): A Hierarchy $\mathcal{H} = (\boldsymbol{N}, \boldsymbol{E})$ is defined as directed acyclic graph consisting of a set of *nodes* $\boldsymbol{N}$ and a set of ordered pairs called *edges* $(N_p, N_c) \in \boldsymbol{E} \subseteq \{\boldsymbol{N} \times \boldsymbol{N}\}$. The direction of an edge (N_p, N_c) is defined from the *parent node N_p* to the *direct child node N_c*, specified through the relational operator $N_p \to N_c$ which is also called *direct path* from N_p to N_c.

A path $N_p \dashrightarrow N_c$ with length n is therefore an ordered set of nodes $\{N_1 \to N_2 \to, \ldots, N_{n-1} \to N_n\} \subseteq \boldsymbol{N}$ where each node is the parent node of the following node. In a hierarchy $\mathcal{H}$ with a path $N_p \dashrightarrow N_c$ there exists no path $N_c \dashrightarrow N_p$ since the hierarchy is acyclic.

Additionally there exists exactly one node called *root node N_r* of a graph $\mathcal{H}$ which has no parent. Nodes which are no parent nodes are called *leaf nodes*. All nodes except leaf nodes and the root node are called *inner nodes*.

Classes (C): Each node N_i within a hierarchy $\mathcal{H}$ is assigned exactly to one class C_i ($\boldsymbol{C} \equiv \boldsymbol{N} \in \mathcal{H}$). Each class C_i consists of a set of documents $\boldsymbol{D} \in \mathsf{C}_i$.

If not stated otherwise within this thesis, for each class C_i a classification hypothesis h_i is calculated. The form of h_i is given by the classification approach.

Documents (D): Documents of a hierarchy $\mathcal{H}$ contain the textual content and are assigned to one or more classes. The classes of a document are also called *labels* of a document $\boldsymbol{L} = \{\mathsf{C}_1 \ldots \mathsf{C}_l\}$.

In general each document is represented as *term vector* $\vec{d_i} = <d_{1,i}, d_{2,i} \ldots d_{|T|,i}>$ where each dimension $d_{j,i}$ represents the weight of a term obtained from preprocessing. Preprocessing and indexing methods are discussed in Section 2.1

1.3. Problem Formulation

Since hierarchical text classification is an extension of flat text classification, the problem formulation for flat text classification is given first. Afterwards the problem definition is extended by including hierarchical structures (as defined in 1.2.2) which gives the problem formulation for this thesis.

1.3.1. Text Classification

Text Classification is the task of finding an approximation for the unknown *target function* $\Phi : \boldsymbol{D} \times \boldsymbol{C} \to \{T, F\}$, where $\boldsymbol{D}$ is a set of documents and $\boldsymbol{C}$ is a set of predefined classes. Value T of the target function $\Phi : \boldsymbol{D} \times \boldsymbol{C} \to \{T, F\}$ is the decision to assign document $\mathsf{D}_j \in \boldsymbol{D}$ to classes $\boldsymbol{C}_i \in \boldsymbol{C}$ and value F is the decision not to assign document $\mathsf{D}_j \in \boldsymbol{D}$ to classes $\mathsf{C}_i \in \boldsymbol{C}$. Φ describes how documents ought to be classified and in short assigns documents $\mathsf{D}_j \in \boldsymbol{D}$ to classes $\mathsf{C}_i \in \boldsymbol{C}$. The target function Φ is also called *target concept* and is element of a *concept class* $\Phi \in \mathcal{C}$.

The approximating function $\tilde{\Phi} : \boldsymbol{D} \times \boldsymbol{C} \to \{T, F\}$ is called *classifier* or *hypothesis* and should coincide with Φ as much as possible. This coincidence is called effectiveness and will be described in 2.3. A more detailed definition considering special constraints to the text classification tasks is given in [68].

For the application considered in this thesis the following assumptions for the above definition are made:

- The target function Φ is described by a document corpus. A corpus is defined through the set of classes $\boldsymbol{C}$, the set of documents $\boldsymbol{D}$ and the assignment of classes to documents $D_j \in C_j$. No additional information for describing Φ is given. The document corpus is also called classification scheme.

- Documents $\boldsymbol{D}$ are represented by a textual content which describes the semantics of a document.

- Categories $\boldsymbol{C}$ are symbolic labels for documents providing no additional information like for example meta data.

- Documents $D_j \in \boldsymbol{D}$ can be assigned to $1 \ldots |\boldsymbol{C}|$ categories (multi-label text classification). Since this is a special case of binary text classification, where a document is assigned to a category $C_i \in \boldsymbol{C}$ or not, $C_i \notin \boldsymbol{C}$, algorithms and tasks for binary text classification are also considered.

For classifying documents automatically, the approximation $\tilde{\Phi}$ has to be constructed.

1.3.2. Hierarchical Text Classification

Supplementary to the definition of text classification a graph $\mathcal{H}$ is added for defining the unknown target function Φ, such that

$$\Phi(D, C_i) = T \Rightarrow \Phi(D, C_j) = T$$

if

$$C_j \dashrightarrow C_i, \ C_j, C_i \in \mathcal{H}.$$

$\mathcal{H}$ is a *hierarchical structure* defining relationships among classes. The assumption behind these constraints is, that $C_i \rightarrow C_j$ defines a IS-A relationship among classes whereby C_i has a broader topic than C_j and the topic of a parent class C_i covers all topics $\sum_{\forall C_i \dashrightarrow C_k} C_k$ of its child classes. Additionally topics from siblings differ from each other, but must not be exclusive to each other. Thus, topics from siblings may overlap[1]. The IS-A relationship is asymmetric (e.g. all dogs are animals, but not all animals are dogs) and transitive (e.g. all pines are evergreens and all evergreens are trees; therefore all pines are trees). The goal is, as before, to approximate the unknown target function by using a document corpus. Additionally the constraints defined by the hierarchy $\mathcal{H}$ have to be satisfied.

Since classification methods depend on the given hierarchical structure including classes and assigned documents, the following basic properties can be distinguished:

- Structure of the hierarchy:

  Given the above general definition of a hierarchy $\mathcal{H}$, two basic cases can be distinguished. (i) A *tree* structure, where each class (except the root class) has exactly one parent class and (ii) a *directed acyclic graph* structure where a class can have more than one parent classes.

---

[1]Which must be true for allowing multilabel classification

4

- Classes containing documents:

  Another basic property is the level at which documents are assigned to classes within a hierarchy. Again two different cases can be distinguished. In the first case, documents are assigned only to leaf classes which is defined here as *virtual hierarchy*. In the second case a hierarchy may also have documents assigned to inner nodes. Note that the latter case can be extended to a virtual hierarchy by adding a virtual leaf node to each inner node. This virtual leaf node contains all documents of the inner node.

- Assignment of documents

  As done in flat text classification, it can be distinguished between multi label and single label assignment of documents. Depending on the document assignment the classification approach may differ.

The model proposed here is a *top-down* approach to hierarchical text classification by using a *directed acyclic graph*. Additionally, multi label documents are allowed. A top down approach means that recursively, starting at the root node, at each inner node zero, one or more subtrees are selected by a local classifier. Documents are propagated into these subtrees till the correct class(es) is/are found.

From a practical point of view, this thesis focuses on hierarchical text classification in document management systems. In such systems, text classification can be used in two ways:

- Documents are automatically assigned to $n = [0 \dots |C|]$ classes

- The users are provided with a ranked list of classes to which a document may belong to. The user can choose one of these classes to store the document. This task can be viewed as semi automatic classification. Additionally to the former way, a ranking mechanism between classes has to be applied.

Whereas the first task is similar to automatic indexing (where the hierarchy is a controlled vocabulary) the latter task may be viewed as a query, returning a ranked list of classes having the most suitable classes ranked top. Note that the latter task can be used to perform the former one, but not vice versa.

This tasks can also be achieved by flat text classification methods, where the hierarchical relationship between classes is ignored when building the classifier. But in application areas like document organization and automatic indexing this may be a major loss of information. Document organization is mostly done in hierarchical structures, built up by humans using their knowledge on a specific domain. Using this knowledge might improve the classification.

Viewed from the point of machine learning, having a lot of possible classifications usually leads to a complex concept class, thereby increasing learning time and decreasing generalization accuracy. Beneath this aspect, classification time in a flat model is linear with the amount of classes whereas a hierarchical model might allow only logarithmic classification time. This can be useful in settings where a large amount of documents has to be classified automatically into a big hierarchical structure.

All these aspects point toward hierarchical text classification whereby two major problems arise in the machine learning setting:

1. Reducing error propagation

2. The validity of comparing classifications from different leaf nodes

Comparing results from different leaf nodes can not be achieved easily if the training of these leaf nodes is independent from each other. Leaf nodes dealing with an easier classification problem may produce higher confidence values[2] than leaf nodes dealing with harder problems. So, if no additional mechanism regulates the comparison of leaf nodes, then results are hardly comparable.

Furthermore, decisions for selecting the appropriate sub hierarchy have to be highly accurate. If wrong decisions are made early in the classification process[3] the error is propagated through the whole hierarchy leading to higher error rates than in a flat model.

Solving these problems is a non-trivial task and the main focus of this thesis.

---

[2]e.g. achieving a higher margin in the case of linear classifiers
[3]under the assumption of a weak heuristic for this sub hierarchy

2. State of Art Text Classification

As stated in chapter 1.3 text classification is the task to approximate a unknown target function Φ through inductive construction of a classifier on a given data set. Afterward, new, unseen documents are assigned to classes using the approximate function $\tilde{\Phi}$. Within this thesis, the former task is referred to as "learning" and the latter task is called "classification".

As usual in classification tasks, learning and classification can be divided into the following two steps:

1. Preprocessing/Indexing is the mapping of document contents into a *logical view* (e.g. a vector representation of documents) which can be used by a classification algorithm. Text operations and statistical operations are used to extract important content from a document. The term logical view of a document was introduced in [4].

2. Classification/Learning: based on the logical view of the document classification or learning takes place. It is important that for classification and learning the same preprocessing/indexing methods are used.

Figure 2.1 illustrates the classification process. Each step above is further divided into several modules and algorithms.

This chapter is organized as follows: an overview on algorithms and modules for document indexing is given in section 2.1. Various classification algorithms are introduced in section 2.2. Section 2.3 introduces performance measures for evaluating classifiers.

2.1. Preprocessing-Document Indexing

As stated before, preprocessing is the step of mapping the *textual content* of a document into a *logical view* which can be processed by classification algorithms. A general approach in obtaining the logical view is to extract meaningful units (*lexical semantics*) of a text and rules for the combination of these units (*lexical composition*) with respect to language. The lexical composition is actually based on linguistic and morphological analysis and is a rather complex approach for preprocessing. Therefore, the problem of lexical composition is usually disregarded in text classification. One exception is given in [15] and [25], where Hidden Markov Models are used for finding the lexical composition of document sections.
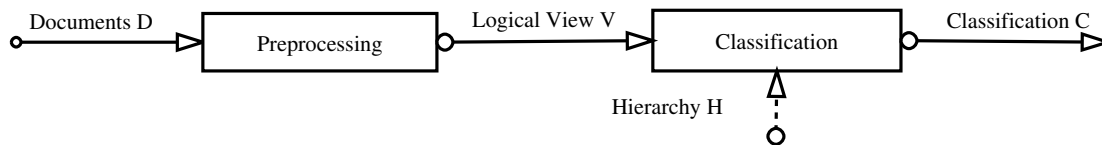
Figure 2.1.: Document classification is divided into two steps: Preprocessing and classification.
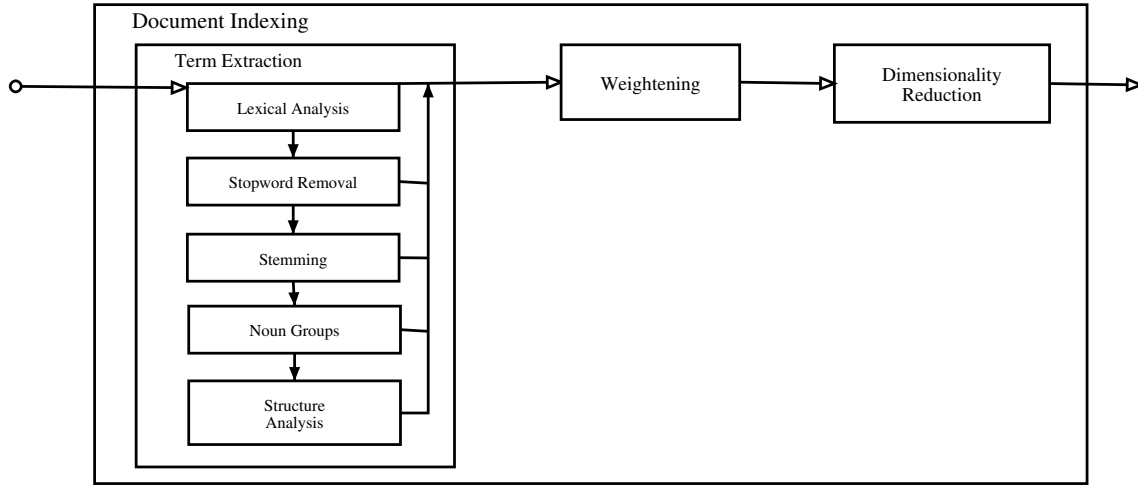
Figure 2.2.: Document indexing involves the main steps term extraction, term weighting and dimensionality reduction.

By ignoring lexical composition the logical view of a document D_j can be obtained by extracting all meaningful units (terms) from all documents D and assigning weights to each term in a document reflecting the importance of a term within the document. More formally, each document is assigned an n-dimensional vector $\vec{d_j} = <w_1, w_2, \ldots, w_n>$ whereby each dimension represents a term from a term set T. The resulting n-*dimensional space* is often referred to as *Term Space* of a document corpus. Each document is a point within this Term Space. So by ignoring lexical composition, preprocessing can be viewed as transforming character sequences into an n-dimensional vector space.

Obtaining the vector representation is called *Document Indexing* and involves two major steps:

1. *Term Extraction*:

   Techniques for defining meaningful terms of a document corpus (e.g. lexical analysis, stemming, word grouping etc.)

2. *Term Weighting*

   Techniques for defining the importance of a term within a document (e.g. Term Frequency, Term Frequency Inverse Document Frequency)

Figure 2.2 shows the steps used for document indexing and their dependencies.

Document Indexing yields to a high dimensional term space whereby only a few terms contain important information for the classification task. Beside the higher computational costs for classification and training, some algorithms tend to *overfit* in high dimensional spaces. Overfitting means that algorithms classify all examples of the training corpus rather perfect, but fail to approximate the unknown target concept Φ (see 2.1.3). This leads to poor effectiveness on new, unseen documents. Overfitting can be reduced by increasing the amount of training examples. It has been shown in [28] that about 50-100 training examples may be needed per term to avoid overfitting. For this reasons dimensionality reduction techniques should be applied.

The rest of this section illustrates well known techniques for term selection/extraction (see 2.1.1), weighting algorithms (see 2.1.2) and the most important techniques for applying dimensionality reduction (see 2.1.3).

2.1.1. Term Extraction

Term extraction, often referred to as feature extraction, is the process of breaking down the text of a document into smaller parts or terms. Term extraction results in a set of terms T which are used for the weighting and dimensionality reduction steps of preprocessing.

In general the first step is a *lexical analysis* where non letter characters like sentence punctuation and styling information (e.g. HTML Tags) are removed. This reduces the document to a list of words separated by whitespace.

Beneath the lexical analysis of a document, information about the document structure like sections, subsections, paragraphs etc. can be used to improve the classification performance, especially for long documents. Incorporating structural information of documents has been done in various studies (see [39], [33] and [72]). Doing a *document structure analysis* may lead to a more complex representation of documents making the term space definition hard to accomplish (see [15]). Most experiments in this area have shown that performance over larger documents can be increased by extracting structures and subtopics from documents.

Identifying terms by words of a document is often called *set of words* or *bag of words* approach, depending on whether weights are binary or not. Stopwords, which are topic neutral words such as articles or prepositions contain no valuable or critical information. These words can be safely removed, if the language of a document is known. Removing stopwords reduces the dimensionality of term space. On the other hand, as shown in [58], a sophisticated usage of stopwords (e.g. negation, prepositions) can increase classification performance.

One problem in considering single words as terms is the semantic ambiguity (e.g. river bank, financial bank) which can be roughly categorized in:

- Synonyms:

  A synonym is a word which means the same as another word (e.g. Movie$\longleftrightarrow$Film).

- Homonym:

  A homonym refers to a word which can have two or more meanings (e.g. lie).

Since only the context of the word within a sentence or document can dissolve this ambiguity, sophisticated methods like morphological and linguistic analysis are needed to diminish this problem. In [23] morphological methods are compared to traditional indexing and weighting techniques. It was stated, that morphological methods slightly increase classification accuracy for the cost of higher computational preprocessing. Additionally, these methods have a higher impact on morphologically richer languages, like for example German, than simpler languages, like for example English. Also, text classification methods have been applied to this "Word Sense Disambiguity" problem (see [30]).

Beside synonymous and homonymous words, different syntactical forms may describe the same word (e.g. go, went, walk, walking). Methods for extracting the syntactical meaning of a word are *suffix stripping* or *stemming* algorithms[1]. Stemming is the notation for reducing words to their word stems. Most words in the majority of Western languages can be "stemmed" by deleting (stripping) language dependent suffixes from the word (e.g. CONNECTED,CONNECTING$\Rightarrow$CONNECT). On the other hand, stripping can lead to new ambiguities (e.g. RELATIVE,RELATIVITY) so that more sophisticated methods performing linguistic analysis may be useful. The performance of stripping and stemming algorithms depends strongly on the simplicity of the used language. For English a lot of stripping and stemming algorithms exist, the Porters Algorithm [55] being the most popular one.

---

[1]Which are language dependent algorithms

Recently a German stemming algorithm [9] has been incorporated into the Lucene Jakarta project and is also freely available.

Taking *noun groups*, which consist of more than one word as term, seems to capture more information. In a number of experiments single word terms were replaced by word grams [2] or phrases. As stated in [3],[19], [40] and [8] this did not give a significantly better performance.

A language independent method for extracting terms is called *character n-grams*. This approach was first discussed by Shannon [69] and further extended by Suen [71]. A character n-gram is a sequence of n characters occurring in a word and can be obtained by simply moving a window of length n over a text (sliding one character at the time) and taking the actual content of a window as term. N-gram representation has the advantage of being language independent and of learning garbled messages. As stated in [24] stemming and stopword removal are superior for word-based systems but are not significantly better for an n-gram based system. The major drawback of n-grams is the amount of unique terms which can occur within a document corpus. Additionally, character n-grams in Information Retrieval (IR) systems yield to the incapability of replacing synonymous words within a query. In [45] it is stated, that the number of unique terms for 4-grams is around equal to the number of unique terms in a word based system. Experiments in [10] have shown that character n-grams are suitable for text classification tasks. Also, character n-grams have been sucessfully applied to clustering and visualizing search results (see [31]).

2.1.2. Term Weighting

After extracting the term space from a document corpus the influence of each term within a document has to be determined. Therefore each term t_i within a document is assigned a weight w_i leading to the above described vector representation $\vec{d_j} = < w_1, w_2 \ldots, w_n >$ of a document. The most simple approach is to assign binary values as weights indicating the presence or absence of a term. A more general approach for weighting is counting the occurrences of terms within a document normalized by the amount of words within a document, the so called *term frequency* $freq(t_k, \mathsf{D_i}) = \frac{occ(t_k, \mathsf{D_i})}{N}$. Thereby N is the number of terms in $\mathsf{D_i}$ and $occ(t_k, \mathsf{D_i})$ is the number of occurrences of term t_k in $\mathsf{D_i}$.

The term frequency approach seems to be very intuitive, but has a major drawback. For example function words occur often within a document and they have a high frequency, but since these words occur in nearly all documents they carry no information about the semantics of a document. This circumstances correspond to the well known Zip-Mandelbrot law [42] which states, that the frequency of terms in texts is extremely uneven. Some terms occur very often, whereas as a rule of thumb, half of the terms occur only once. Similar to term frequencies, logarithmic frequencies as

$$freq_{log}(t_k, \mathsf{D_i}) = \log(1 + freq(t_k, \mathsf{D_i}))$$

may be taken, which is a more common measure in quantitative linguistics (see [23]). Again, logarithmic frequency suffers from the drawback, that function words may occur very often in the text. To overcome this drawback, weighting schemes are applied for transforming these frequencies into more meaningfull units. One standard approach is the *inverse document frequency (idf)* weighting function which has been introduced by [62]

$$w_{k,i} = freq(t_k, \mathsf{D_i}) * \log \frac{|\boldsymbol{D}|}{|\{\mathsf{D_i} \in \boldsymbol{D} \mid t_k \in \mathsf{D_i}\}|}$$

and is know as *Term Frequency Inverse Document Frequency* (TFIDF) weighting scheme. Thereby $freq(t_k, \mathsf{D_i})$ denotes the term frequency of term t_k within document i, $\boldsymbol{D}$ denotes the set of available

---

[2] n-word grams are a sequence of n words consequently occurring in a document

documents and $\{D_i \in \boldsymbol{D} \mid t_k \in D_i\}$ denotes the set of documents containing term t_k. In other words a term is relevant for a document if it (i) occurs frequently within a document and (ii) discriminates between documents by occurring only in a few documents.

For reducing the effects of large differences between frequencies of terms a logarithmic or square root function can be applied to the term frequency leading to

$$w_{k,i} = freq_{log}(t_{k,i}) * \log \frac{|\boldsymbol{D}|}{|\{D_i \in \boldsymbol{D} \mid t_k \in D_i\}|}$$

or

$$w_{k,i} = \sqrt{freq(t_{k,i})} * \log \frac{|\boldsymbol{D}|}{|\{D_i \in \boldsymbol{D} \mid t_k \in D_i\}|}$$

TFIDF weighting is the standard weighting scheme within text classification and information retrieval.

Another markable weighting technique is the *entropy weighting* scheme which is calculated as

$$w_{k,i} = freq_{log}(t_{k,i}) * (1 + entropy(t_k, \boldsymbol{D}))$$

where

$$entropy(t_k, \boldsymbol{D}) = \frac{1}{\log |\boldsymbol{D}|} \sum_{j=1}^{|\boldsymbol{D}|} \frac{occ(t_k, D_j)}{occ(t_k, \boldsymbol{D})} * \log \frac{occ(t_k, D_j)}{occ(t_k, \boldsymbol{D})}$$

is the entropy of term k. As stated in [20] the entropy weighting scheme yields better results than TFIDF or other ones. A comparison of different weighting schemes is given in [62] and [23]. Additionally, weighting approaches can be found in [11], [12] and in the AIR/X system [29].

After having determined the influence of a term by using frequency transformation and weighting, the length of terms has to be considered by normalizing documents to unique length. From a linguistic point of view normalizing is a non trivial task (see [43],[51]). A good approximation is to divide term frequencies by the total number of tokens in text which is equivalent to normalize the vector using the one norm:

$$\vec{w}_k = \frac{\vec{w}}{\|\vec{w}\|_1}$$

Since some classification algorithms (e.g. SVM's) yield better error bounds by using other norms (e.g. euclidean), these norms are frequently used within text classification.

2.1.3. Dimensionality Reduction

Document indexing by using the above methods leads to a high dimensional term space. The dimensionality depends on the number of documents in a corpus, for example the 20.000 documents of the Reuters 21578 data set (see section 4) have about 15.000 different terms. Two major problems arise having a that high dimensional term space:

1. Computational Complexity:

   Information retrieval systems using cosine measure can scale up to high dimensional term spaces. But the learning time of more sophisticated classification algorithms increases with growing dimensionality and the volume of document copora.

2. Overfitting:

Most classifiers (except Support Vector Machines [35]) tend to overfit in high dimensional space, due to the lack of training examples.

To deal with these problems, dimensionality reduction is performed keeping only terms with valuable information. Thus, the problem of identifying irrelevant terms has to be solved for obtaining a reduced term space $T' \subset T$ with $|T'| \ll |T|$. Two distinct views of dimensionality reduction can be given:

- Local dimensionality reduction:

  For each class C_i, a set $T'_i, |T'_i| \ll |T|$ is chosen for classification under C_i.

- Global dimensionality reduction:

  a set $T', |T'| << |T|$ is chosen for the classification under all categories C

Mostly, all common techniques can perform local and global dimensionality reduction. Therefore the techniques can be distinguished in another way:

- by term selection:

  According to information or statistical theories a subset T' of terms is taken from the original space T.

- by term extraction

  terms in the new term space T' are obtained through a transformation $T' = \sigma(T)$. The terms of T' may be of a complete different type than in T.

2.1.3.1. Dimensionality Reduction by Term Selection

The first approach on dimensionality reduction by term selection is the so called filtering approach. Thereby measurements derived from information or statistical theory are used to filter irrelevant terms. Afterwards the classifier is trained on the reduced term space, independent of the used "filter function".

Another approach are so called wrapper techniques (see [48]) where term selection based on the used classification algorithm is proposed. Starting from an initial term space a new term space is generated by adding or removing terms. Afterwards the classifier is trained on the new term space and tested on a validation set. The term space yielding best results is taken as term space for the classification algorithm. Having the advantage of a term space tuned on the classifier, the computational cost of this approach is a huge drawback. Therefore, wrapper approaches will be ignored in the rest of this section.

Document Frequency: A simple reduction function is based on the document frequency of a term t_k. According to the Zipf-Mandelbrot law, the highest and lowest frequencies are discarded. Experiments indicate that a reduction of a factor 10 can be performed without loss of information.

Function	$f(\mathsf{C_i}, t_k)$	Mathematical form
DIA association factor	$z(t_k, \mathsf{C_i})$	$Pr[(\mathsf{C_i}\vert t_k)]$
Information gain	$IG(t_k, \mathsf{C_i})$	$\sum_{\mathsf{C_k}\in\{\mathsf{C_i},\bar{\mathsf{C}}_i\}} \sum_{t\in\{t_k,\bar{t}_k\}} Pr[t,\mathsf{C_k}] * \log \frac{Pr[t,\mathsf{C_k}]}{Pr[t]*Pr[\mathsf{C_k}]}$
Mutual information	$MI(t_k, \mathsf{C_i})$	$\log \frac{Pr[t_k,\mathsf{C_i}]}{Pr[t_k]*Pr[\mathsf{C_i}]}$
Chi-Square	$\chi^2(t_k, \mathsf{C_i})$	$\vert T\vert \frac{[Pr[t_k,\mathsf{C_i}]*Pr[\bar{t}_k,\bar{\mathsf{C}}_i]-Pr[t_k,\bar{c}_i]*Pr[\bar{t}_k,\mathsf{C_i}]]^2}{Pr[t_k]*Pr[\bar{t}_k]*Pr[\mathsf{C_i}]*Pr[\bar{c}_i]}$
NGL coefficient	$NGL(t_k, \mathsf{C_i})$	$\sqrt{\vert T\vert}\frac{[Pr[t_k,\mathsf{C_i}]*Pr[\bar{t}_k,\bar{\mathsf{C}}_i]-Pr[t_k,\bar{c}_i]*Pr[\bar{t}_k,\mathsf{C_i}]]}{\sqrt{Pr[t_k]*Pr[\bar{t}_k]*Pr[\mathsf{C_i}]*Pr[\bar{c}_i]}}$
Relevancy score	$RS(t_k, \mathsf{C_i})$	$\log \frac{Pr[t_k\vert\mathsf{C_i}]+d}{Pr[t_k\vert\bar{c}_i]+d}$
Odds ratio	$OR(t_k, \mathsf{C_i})$	$\frac{Pr[t_k\vert\mathsf{C_i}]*(1-Pr[t_k\vert\bar{\mathsf{C}}_i])}{(1-Pr[t_k\vert\mathsf{C_i}])*Pr[t_k\vert\bar{\mathsf{C}}_i]}$
GSS coefficient	$GSS(t_k, \mathsf{C_i})$	$Pr[t_k,\mathsf{C_i}] * Pr[\bar{t}_k,\bar{c}_i] - Pr[t_k,\bar{c}_i] * Pr[\bar{t}_k,\mathsf{C_i}]$

Table 2.1.: Important term selection functions as stated in [68] given with respect to a class $\mathsf{C_i}$. For obtaining a global criterion on a term these functions have to be combined (e.g. summing). Terms yielding highest results are taken.

Statistical and Information-Theoretic Term Selection Functions: Sophisticated methods derived from statistics and information theory have been used in various experiments yielding to a reduction factor of about 100 without loss. Table 2.1 lists the most common term selection functions as illustrated in [68].

A term selection function $f(t_k, \mathsf{C_i})$ selects terms t_k for a class $\mathsf{C_i}$ which are distributed most differently in the set of positive and negative examples[3]. For deriving a global criterion based on a term selection function, these functions have to be combined somehow over the set of given classes C. Usual combinations for obtaining $f(t_k)$ are

- sum: calculates the sum of the term selection function over all classes as

$$f_{sum}(t_k) = \sum_{i=1}^{C} f(t_k, \mathsf{C_i})$$

- weighted sum: calculates the sum of the term selection function over all classes weighted with the class probability:

$$f_{wsum}(t_k) = \sum_{i=1}^{C} Pr[\mathsf{C_i}] f(t_k, \mathsf{C_i})$$

- maximum: takes the maximum of the term selection function over all classes:

$$f_{max}(t_k) = \max_{i}^{C} f(t_k, \mathsf{C_i})$$

Terms yielding highest results with respect to the term selection function are kept for the new term space, other terms are discarded. Experiments indicate that $\{OR_{sum}, NGL_{sum}, GSS_{max}\} > \{\chi^2_{max}, IG_{sum}\} > \{\chi^2_{wsum}\} >> \{MI_{max}, MI_{sum}\}$ where ">" means "performs better than".

---

[3]Based on the assumption, that if a term occurs only in the positive or negative training set, it is a good feature for this class.

2.1.3.2. Dimensionality Reduction by Term Extraction

Term extraction methods create a new term space T' by generating new synthetic terms from the original set T. Term extraction methods try to perform a dimensionality reduction by replacing words with their concept.

Two methods were used in various experiments, namely

- Term Clustering

- Latent Semantic Indexing (LSI)

These methods will be discussed in the rest of this section.

Term Clustering: Grouping together terms with a high degree of pairwise semantic relatedness, so that these groups are represented in the Term Space instead of their single terms. Thus, a similarity measure between words must be defined and clustering techniques like for example k-means or agglomerative clustering are applied. For an overview on Term Clustering see [68] and [16].

Latent Semantic Indexing: LSI compresses document term vectors yielding to a lower dimensional term space T'. The axes of this low dimensional space are linear combinations of terms within the original term space T. The transformation is done by a singular value decomposition (SVD) of the document term matrix of the original term space. Given a term-by-document matrix $\mathfrak{D}_{m \times n}$ where $m = |T|$ is the number of terms and $n = |D|$ is the number of documents, the SVD is done by

$$\mathfrak{D} = \mathfrak{U}\mathfrak{S}\mathfrak{V}$$

where $\mathfrak{U}_{m \times r}$ and $\mathfrak{V}_{r \times n}$ have orthonormal columns and $\mathfrak{S}_{r \times r}$ is the diagonal matrix of singular values from the original matrix $\mathfrak{D}$, where $r \leq min(m, n)$ is the rank of the original term-by-document matrix $\mathfrak{D}$

Transforming the space means that the $r - k$ smallest singular values of $\mathfrak{S}$ are discarded (set to zero), which results in a new term-by-document matrix

$$\tilde{\mathfrak{D}}_{m \times n} = \tilde{\mathfrak{U}}_{m \times k}\tilde{\mathfrak{S}}_{k \times k}\tilde{\mathfrak{V}}_{k \times n}^{T}$$

which is an approximation of $\mathfrak{D}$. Matrix $\tilde{\mathfrak{S}}$ is created by deleting small singular values from $\mathfrak{S}$, $\tilde{\mathfrak{U}}$ and $\tilde{\mathfrak{V}}$ are created by deleting the corresponding rows and columns.

After having obtained these results from the SVD based on the training data, new documents are mapped by

$$\vec{d'} = \tilde{\mathfrak{S}}^{-1}\tilde{\mathfrak{U}}^{T}\vec{d}$$

into the low dimensional space (see [14] and [5]).

Basically, LSI tries to capture the latent structure in the pattern of word usage across documents using statistical methods to extract these patterns. Experiments done in [67] have shown that terms not selected as best terms for a category by χ^2 term selection, were combined by LSI and contributed to a correct classification. Furthermore, they showed that LSI is far more effective than χ^2 for linear discriminant analysis and logistic regression, but equally effective for neural networks classifiers. Additionally, [22] demonstrated that LSI used for creating category specific representation yields better results than creating a global LSI representation.

2.2. Classification Methods

As stated in section 1.3 text classification can be viewed as finding a approximation $\tilde{\Phi} : \boldsymbol{D} \times \boldsymbol{C} \rightarrow \{T, F\}$ of an unknown target function $\Phi : \boldsymbol{D} \times \boldsymbol{C} \rightarrow \{T, F\}$. The function values F and T can be used in two ways:

- Hard Classification

  A hard classification assigns each pair $< D_i, C_i >$ a value T or F.

- Soft Classification

  A soft classification assigns a ranked list of classes $\boldsymbol{C} = \{C_1, C_2 \ldots C_n\}$ to a document D_i or assigns a ranked list of documents $\boldsymbol{D} = \{D_1, D_2 \ldots D_n\}$ to a class C_i.

Hard classification can be achieved easily by the definition of $\tilde{\Phi}$. Usually, the inductive construction of a classifier for class $C_i \in \boldsymbol{C}$ consists of a function $h_i : \boldsymbol{D} \rightarrow [0, 1]$ whereby a document D_j is assigned to class C_i with confidence h_i.

Given a set of classifiers $\boldsymbol{h} = \{h_1, h_2 \ldots h_n\}$, ranked classification can be easily achieved by sorting classes (or symmetrically documents) by their h_i values.

The following subsections describe classification approaches implemented in this thesis and outline their general theoretical properties. Afterwards, other commonly used classification approaches are discussed roughly. If not stated otherwise, the discussed algorithms take a term vector $\vec{d_j}$ as input for a document D_j which is obtained by some document indexing methods described in section 2.1.

2.2.1. Linear Classifiers

One of the most important classifier family in the realm of text classification are linear classifiers. Linear classifiers have, due to their simple nature, a well founded theoretical background. One problem at hand is, that linear classifiers have a very restricted hypothesis class.

A linear classifier is a linear combination of all terms t_k from a term or feature space T. Formally, given the above notations, a linear classifier $h(\vec{d_j}) \rightarrow \{-1, 1\}$ can be written as

$$h(\vec{d_i}) = sign(\vec{w} \cdot \vec{d_i} - \theta) = sign\left(\sum_{k=1}^{|T|} w_k * d_{k,i} - \theta \right)$$

where w_k is the weight for term t_k and $d_{k,i}$ is the value of term t_k in document D_i. Thus, each class C_j is represented by a weight vector $\vec{w_j}$ which assigns a document $\vec{d_i}$ to a class if the inner product $w_j \cdot d_i$ exceeds some threshold θ_j and does not assign a document otherwise.

Figure 2.3 illustrates a linear classifier for the two dimensional case. The equation $\vec{w} \cdot \vec{d} = \theta$ defines the decision surface which is a hyperplane in a $|T|$-dimensional space. The weight vector $\vec{w}$ is a normal projection of the separating hyperplane whereas the distance of the hyperplane from the origin is

$$\frac{\theta}{\|\vec{w}\|}$$

and the distance of a training example to the hyperplane is given by

$$r = \frac{\vec{w} \cdot \vec{d} - \theta}{\|\vec{w}\|}$$

15
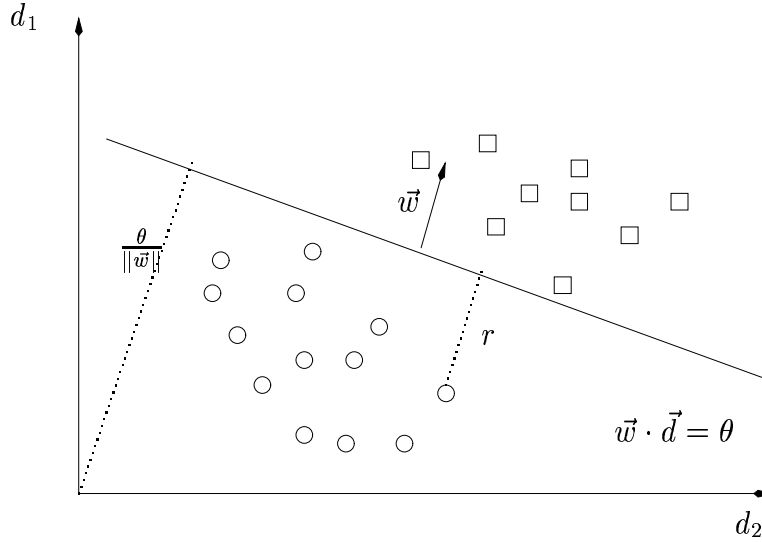
Figure 2.3.: Linear classifier separating the trainings data for the binary case. Square and circles indicate positive and negative training examples.

in the case of an euclidean space.

Learning a linear classifier can be done in various ways. One well known algorithm is the Perceptron algorithm which is a gradient decent algorithm using additive updates. Similar to the Perceptron algorithm, Winnow is a multiplicative gradient descent algorithm. Both algorithms can learn a linear classifier in the linear separable case. Section 2.2.1.1 illustrates an alternative to the Perceptron algorithm, called Support Vector Machines. SVM's are capable of finding a optimal separating hyperplane in the linear separable case and in an extension for the linear non separable case they are able to minimize a loss function. Other important learning algorithms are for example Minimum Squared Error procedures like Widrow Hoff and linear programming procedures. An introduction to them is given in [18].

2.2.1.1. Support Vector Machines

This section gives an introduction to support vector machines. SVM's are covered in more detail because they are used as baseline classifiers for some experiments done in the experimental section of this thesis. SVM's are todays top notch methods within text classification. Their theory is well founded and gives insight in learning high dimensional spaces, which is appropriate in the case of text classification.

SVM's are linear classifiers which try to find a hyperplane that maximizes the margin between the hyperplane and the given positive and negative training examples.

Figure 2.4 illustrates the idea of maximum margin classifiers. The rest of this section gives a brief overview on the properties of SVM's. For a more detailed introduction see [7], and [49]. The theory of SVM's was introduced by in the early works of Vapnik [74], which is written in russian[4]. For SVM's applied to text classification see [19], and [35].

---

[4]Additionally, more information on SVM's may be obtained from http://www.kernel-machines.org
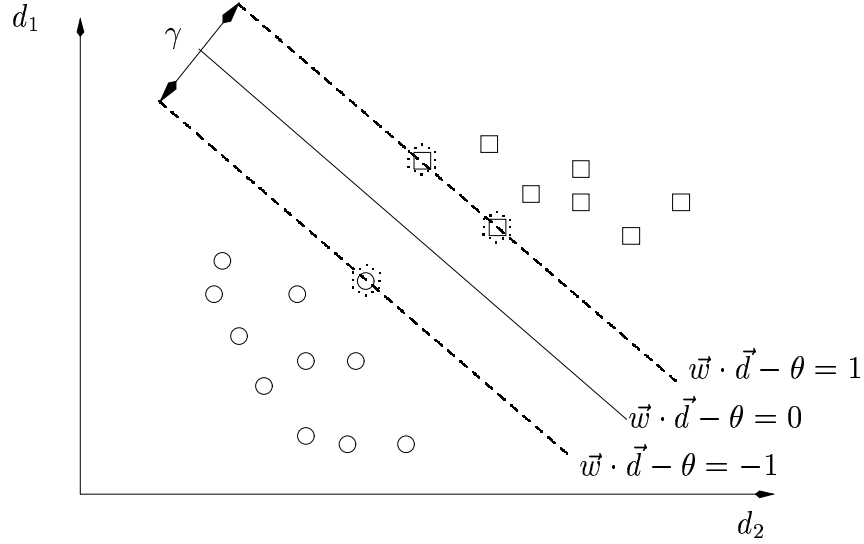
Figure 2.4.: Maximum margin linear classifier separating the training data for the binary case. Squares and circles indicate positive and negative training examples respectively. Support vectors are surrounded by doted circles.

Linear Separable Case: Let $S = \{< \vec{d_1}, y_1 > \cdots < \vec{d_m}, y_m >\}$ be a set of training examples. Examples are represented as feature vectors $\vec{d_i} \in T$ obtained from a term space T. For this section binary labels $y_i \in Y = \{-1, 1\}$ are assumed to indicate whether a document is assigned to a class or not (a extension to the multi-label case is given in section 2.2.1.3).

A linear classifier which maximizes the margin can be formulated as set of inequalities over the set of training examples, namely

$$y_i(\vec{d_i} \cdot \vec{w} + \theta) - 1 \geq 0 \ \ \forall d_i \in S$$

subject to

$$\|\vec{w}\| \to \min.$$

Vector $\vec{w}$ is the normal vector on the separating hyperplane. From this formulation it can be obtained, that all positive examples, for which equality holds, lie on the hyperplane $H_1 : \vec{d_i} \cdot \vec{w} + \theta = 1$. Similar, all negative examples, for which equality holds, lie on the hyperplane $H_2 : \vec{d_i} \cdot \vec{w} + \theta = -1$. Thus, the maximum margin is $\gamma = \frac{2}{\|w\|_2}$ in the separable case. So maximizing margin γ is the same as minimizing $\frac{1}{2}\|w\|_2^2$. Those data points, for which equality holds are called *support vectors*. Figure 2.4 shows a solution for the two dimensional case. Support Vectors are surrounded by dotted circles.

The above set of inequalities can be reformulated by using a Lagrangian formulation of the problem. Positive Lagrange multipliers α_i, $i = 1 \ldots |S|$ are introduced, one for each of the inequality constraints. To form the Lagrangian, the constraint equations are multiplied by the Lagrange multipliers and subtracted from the objective function which gives:

$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{|S|} \alpha_i(y_i(\vec{d_i} \cdot \vec{w} + \theta) - 1)$$

Minimizing $L(\vec{w}, b, \vec{\alpha})$ with respect to $\vec{w}, b$ and maximizing it with respect to $\vec{\alpha}$ yields the solution for the above problem and can be found at an extremum point were

$$\frac{\partial L}{\partial \theta} = 0 \;\; and \;\; \frac{\partial L}{\partial \vec{w}} = 0$$

which transforms into

$$\sum_{i=1}^{|\boldsymbol{S}|} \alpha_i y_i = 0 \quad and \quad \vec{w} = \sum_{i=1}^{|\boldsymbol{S}|} \alpha_i y_i \vec{d_i}$$

for $L(\vec{w}, b, \vec{\alpha})$. The dual quadratic optimization problem can be obtained by plugging these constraints into $L(\vec{w}, b, \vec{\alpha})$ which gives

$$\max_{\alpha} \sum_{i=1}^{|\boldsymbol{S}|} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \vec{d_i} \vec{d_j}$$

$$\text{subject to} \quad \alpha_i \geq 0 \; \forall_i \quad and \quad \sum_{i=1}^{|\boldsymbol{S}|} \alpha_i y_i = 0$$

From this optimization problem all α_i can be obtained which gives a separating hyperplane defined by $\vec{w}$, maximizing the margin γ between training examples in the linear separable case. Note that the formulation of these optimization problem replaces $\vec{w}$ with the product of the Lagrangian multipliers and the given training examples $\sum_{i=1}^{|\boldsymbol{S}|} \alpha_i y_i \vec{d_i}$. Thus, the separating hyperplane can be defined only through the given training patterns $\vec{d_i}$. Additionally, support vectors have a Lagrange multiplier of $\alpha_i > 0$ whereas all other training examples have a Lagrange multiplier of zero ($\alpha_i = 0$). Support vectors lie on one of the hyperplanes H_1, H_2 and are the critical examples in the training process. So testing or evaluating a SVM means evaluating the inner product of a given test example with the support vectors obtained from the training process, written as

$$h(\vec{d_j}) = \text{sgn} \left(\sum_{\alpha_i \neq 0} \alpha_i y_i \vec{d_i} \cdot \vec{d_j} \right).$$

Also, if all training examples with $\alpha_i = 0$ would be removed, retraining the SVM yields to the same separating hyperplane.

Calculating the separating hyperplane implicitly through the given training patterns gives two big advantages. First, learning infinite concept classes is possible, since the hypothesis is only expressed by the given training patterns. Second, SVM's can be easily extended to the nonlinear case by using kernel functions. A short introduction to kernel functions and their implications is given below.

Non Separable Case: The above formulation holds for linear separable training data. Given non linear separable training data, the above defined dual optimization problem does not lead to a feasible solution. Also, in terms of statistical learning theory, if a solution is found this might not be the solution minimizing the risk on the given training data with respect to some loss function (see [74],[49]). Thus, the minimization problem is reformulated by relaxing the constraints on the trainings data if necessary. This can be done by introducing positive *slack variables* $\xi_i, i = 1 \ldots |\boldsymbol{S}|$ which relax the hard margin constraints. Formally this is

$$y_i(\vec{d_i} \cdot \vec{w} + \theta) \geq 1 - \xi_i, \;\; \xi_i \geq 0 \;\; \forall \vec{d_i} \in \boldsymbol{S}$$
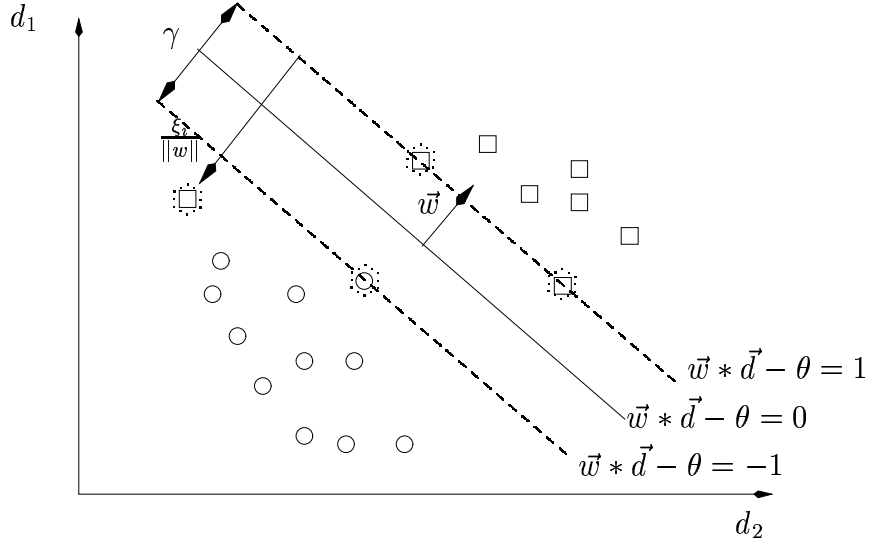
Figure 2.5.: Maximum margin linear classifier extended to the non separable case by using slack variables ξ_i.

Figure 2.5 illustrates a SVM extended to the non separable case. Thus, for an error to occur, the corresponding ξ_i must exceed unity. So $\sum_i \xi_i$ is an upper bound on the number of training errors. One way for addressing the extra cost for errors is to change the objective function to

$$\min_{\vec{w}, b, \vec{\xi}} \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^{|\boldsymbol{S}|} \xi_i$$

where $C > 0$ is a parameter assigning a higher/lower penalty to errors [5].

Again, by applying Lagrangian multipliers, the dual optimization problem can be formulated as

$$\max_{\alpha} \sum_{i=1}^{|\boldsymbol{S}|} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \vec{d_i} \cdot \vec{d_j}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C \ \forall_i \quad \text{and} \quad \sum_{i=1}^{|\boldsymbol{S}|} \alpha_i y_i = 0$$

The only difference from the linear separable definition is, that α_i is now bound by the trade off parameter C.

Non-Linear SVM's: Another possibility for learning linear inseparable data is to map the training data into a higher dimensional space by some mapping function Φ[6]. As stated in [1], by applying an appropriate mapping linear inseparable data become separable in a higher dimensional space. Thus, a mapping of the form

$$\Phi : \Re^N \to F$$
$$x :\to \Phi(x)$$

---

[5]In statistical learning theory C can also be viewed as a trade off between the empirical error and the complexity of the given function class.

[6]By mapping the training data the decision function is no longer a linear function of the data
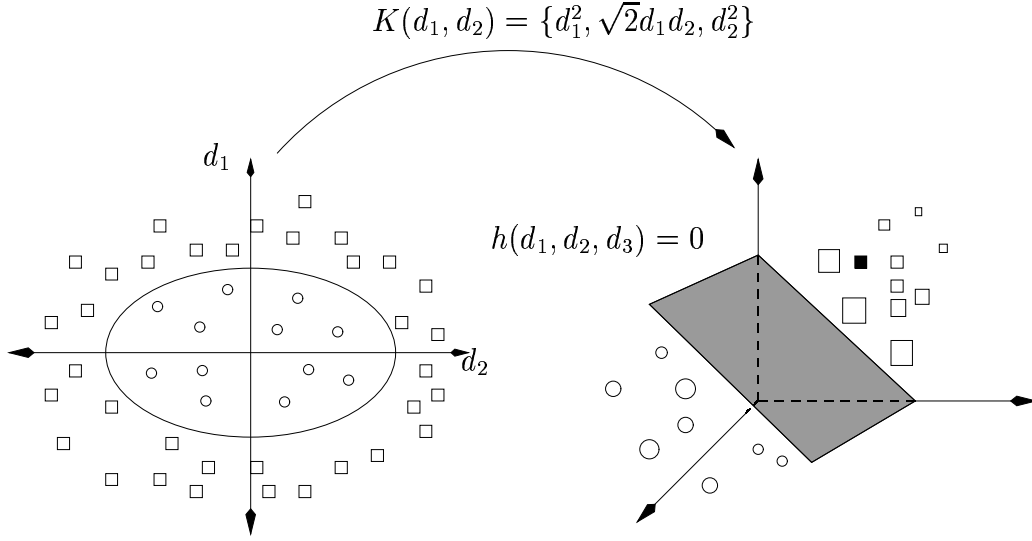
Figure 2.6.: Transformation from two dimensional to three dimensional space by using the kernel $K(d_1, d_2) = \{d_1^2, \sqrt{(2)}d_1d_2, d_2^2\}$. The shaded plane right shows the separating plane in the three dimensional space.

is applied on the sequence $\boldsymbol{S}$ of training examples transforming them into $\boldsymbol{S'} = \{< \Phi(\vec{d_1}), y_1 > \cdots < \Phi(\vec{d_m}), y_m >\}$. Thereby, F is the new feature space obtained from the original space through the mapping Φ. This is also implicitly done by neural networks (using hidden layers which map the representation) and Boosting algorithms (which map the input to a different hypothesis space).

Figure 2.6 illustrates a two dimensional classification example mapped to the three dimensional space by $\Phi(d_1, d_2) = < d_1^2, \sqrt{2}d_1 * d_2, d_2^2 >$. The mapping makes the examples linearly separable.

One drawback of applying a mapping into a high dimensional space may be the so called *curse of dimensionality*. According to statistics, the difficulty of an estimation problem increases drastically (in principle exponential in terms of training examples) with the dimensions of the space. Fortunately, it was shown in [75] that, by applying the framework of statistical learning theory, the complexity of the hypothesis class of a classifiers matters and not the dimensionality. Thus, a simple decision class like linear classifiers is used in high dimensional spaces resulting in a complex decision rule in the low dimensional space (see again Figure 2.6).

One drawback of the mapping is the algorithmic complexity arising from the high dimensional space F, making learning problems virtually intractable. But since learning and testing SVM's is defined by evaluating the inner product between training examples, so called *kernel functions* can be used to reduce algorithmic complexity and makeing infinite spaces tractable. A Kernel function $K(\vec{d_i}, \vec{d_j})$ for a mapping Φ is defined as

$$K : \Re^d \times \Re^d \to \Re$$

whereby for all feature space examples $\vec{d_i}, \vec{d_j} \in \Re^d$ equation

$$\Phi(\vec{d_i})\Phi(\vec{d_j}) = K(\vec{d_i}, \vec{d_j})$$

holds. Common Kernel functions are

$$\text{Gaussian RBF} \quad K(\vec{d}_i, \vec{d}_j) = exp\left(\frac{-\|\vec{d}_i - \vec{d}_j\|^2}{c}\right)$$

$$\text{Polynomial} \quad K(\vec{d}_i, \vec{d}_j) = ((\vec{d}_i\vec{d}_j) + b)^d$$

$$\text{Sigmoidal} \quad K(\vec{d}_i, \vec{d}_j) = \tanh(\kappa(\vec{d}_i\vec{d}_j) + b)$$

Training of SVM's: Training a SVM is usually a quadratic programming (QP) problem and therefore algorithmically complex and expensive in terms of computation time and memory requirements if applied to a huge amount of training data. To increase speed and decrease memory requirements, three different approaches have been proposed.

Chunking methods (see [6]) start with a small, arbitrary subset of the data for training. The rest of the training data is tested on the resulting classifier. The test results are sorted by the margin of the training examples on the wrong side. The first N of these and the already found support vectors are taken for the next training step. Training stops at the upper bound of the training error. This method requires the number of support vectors N_S to be small enough so that a Hessian matrix of size N_S by N_S will fit in memory.

A *decomposition* algorithm not suffering from this drawback was introduced in [52]. Thereby only a small portion of the training data is used for training in a given time. Additionally, only a subset of the support vectors (which are currently needed) have to be in the actual "working set". This method was able to easily handle a problem of about 110,000 training examples and 100,000 support vectors. An efficient implementation of this method, including some extension on working set selection and successive shrinking can be found in [36]. It was implemented in the freely available SVM^{light} package of Joachims. This implementation was also used as baseline classifier in the practical part of this thesis.

Beneath these two algorithms another variant of fast training algorithms for SVM's, the *sequential minimal optimization (SMO)*, was introduced in [53]. SMO is based on the convergence theorem provided in [52]. Thereby, the optimization problem is broken down in simple, analytically solvable problems which are problems involving only two Lagrangian multipliers. Thus, the SMO algorithm consists of two steps:

1. Using a heuristic to choose the two Lagrangian multipliers

2. Analytically solving the optimization problem for the chosen multipliers and updating the SVM

The advantage of SMO lies in the fact, that numerical QP optimization is avoided entirely. Additionally, SMO requires no matrix storage since only two Lagrangian classifiers are solved at a time. As stated in [53], SMO performs better than the chunking method explained above.

2.2.1.2.   Rocchio

The Rocchio classification approach (see [56]) is motivated by Rocchio's well known feedback formula (see [59]) which is used in vector space model based information retrieval systems. Basically Rocchio is a linear classifier, defined as aprofile vector, which is obtained through a weighted average of training examples. Formally, the profile vector $\vec{c}_i = < c_{1,i}, c_{2,i}, \ldots, c_{|T|,i} >$ for a category C_i is calculated as

$$c_{k,i} = \beta * \sum_{d_j \in \boldsymbol{D}_{\mathsf{C}_i}} \frac{d_{k,j}}{|\boldsymbol{D}_{\mathsf{C}_i}|} - \gamma * \sum_{d_j \in \boldsymbol{D}_{\overline{\mathsf{C}}_i}} \frac{d_{k,j}}{|\boldsymbol{D}_{\overline{\mathsf{C}}_i}|}$$

where $\boldsymbol{D}_{\mathsf{C}_i}$ is the set of positive training example of category C_i and $\boldsymbol{D}_{\overline{\mathsf{C}}_i}$ is the set of negative training examples for category C_i.

β and γ are control parameters for defining the relative importance or influence of negative and positive examples. So, if γ is set to zero and β is set to 1 the negative examples are not taken into account. The resulting linear prototype vector is the so called centroid vector of its positive training examples which minimizes the sum squared distance between positive training examples and the centroid vector.

For the classification of new examples the closeness of an example to the prototype vector is used. Usually the cosine similarity measure defines this closeness:

$$similarity(\mathsf{D_j}, \mathsf{C_i}) = \frac{\vec{d_j} * \vec{c_i}}{\|\vec{d_j}\| * \|\vec{c_i}\|}$$

In words, the cosine similarity is the cosine of the angle between the category prototype vector and the document vector. A document is assigned to the category if it has the closest angle to the category prototype vector.

Beneficial in the Rocchio approach is the short learning time, which is actually linear with the number of training examples. Effectiveness in terms of error rates or precision and recall suffers from the simple learning approach. It has been shown by [56], that including only negative examples, which are close to the positive prototype vector (so called near negatives) can improve classification performance. In information retrieval this technique is known as query zoning (see [70]). Furthermore, by additionally applying "good" term selection techniques and other enhancements (e.g. dynamic feedback optimization) Schapire and Singer [56] have found that Rocchio performs as good as more complex techniques like boosting (see 2.2.2), and is 60 times quicker to train. Rocchio classifiers are often used as baseline classifiers comparing different experiments with each other.

2.2.1.3. Multi-label Classification using Linear Classifiers

So far only binary classification problems were considered where a hyperplane separates two classes. Normally, more than one class exists in text classification. This leads to the question how binary decisions may be adapted to multi label decisions. Formally, the set of possible labels for a document $\mathsf{D_i}$ is extended to $\boldsymbol{Y}_i = \{\mathsf{C}_1 \ldots \mathsf{C_k}\}$.

The following approaches were suggested by [32] and [76]:

1. Using k one-to-rest classifiers

2. Using $\frac{k(k-1)}{2}$ pairwise classifiers with one of the following voting schemes:

   a) Majority Voting
   b) Pairwise Coupling
   c) Decision Directed Acyclic Graph

3. Error Correction Codes

All these methods rely on combining binary classifiers. However, learning algorithms may be adapted for directly learning a k-class problem. Adaption varies from learning algorithm to learning algorithm. One example of adapting a learning algorithm is given in section 2.2.2.3, where Boosting is modified to learn a k-class problem directly.
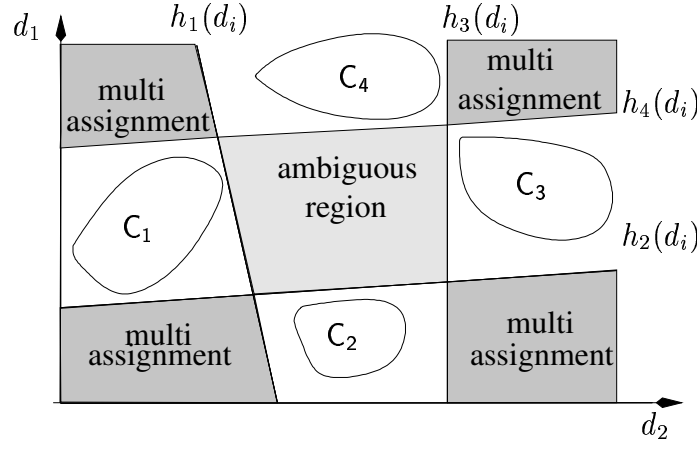
Figure 2.7.: Multiclass classification using 1 vs. rest approach

One-to-rest classification: A 1 vs. rest classification has k independent classifiers $h_j(\vec{d}) \rightarrow \{-1, 1\}$. A classifier h_j should output 1 if a document is assigned to class C_j, -1 otherwise, which is written as

$$h_j(\vec{d_i}) = \begin{cases} 1 & if \ D_i \in C_j \\ -1 & if \ D_i \notin C_j \end{cases}$$

Training classifier h_j means selecting all examples $\vec{d_k} \in C_j$ as positive examples and all other examples $\vec{d_k} \notin C_j$ as negative examples.

Doing so, classification may be undefined in some cases. For linear classifiers this happens, if none of the k linear classifiers exceeds the given threshold, which is

$$(\vec{w_k} \cdot \vec{d_i} + \theta) \leq 0 \ \forall_{c_k \in C}$$

Additionally, assignments to more than one class are possible if i out of k linear classifiers exceed the given threshold. In the case of single class classification only one class has to be chosen. This can be done by taking the class which has the largest margin to the separating hyperplane written as $\arg\max_{C_k \in C}(\vec{w_k} \cdot \vec{d_i} + \theta)$ [7]. Figure 2.7 illustrates 1 vs. rest classification for a two dimensional feature space.

Approaches using 1 vs. rest classifiers are given in [18] and with focus on SVM's and Boosting in [66] and [2]. One problem arising with one-to-rest classifiers is, that usually there are more negative examples than positive ones which is a asymmetric problem and could bias the classifier.

Pairwise Classifiers: By using pairwise classifiers $\frac{k(k-1)}{2}$ different (linear) classifier are trained, whereby each classifier determines whether an example corresponds to one or another class out of all available k classes, formally written as

$$h_{k,j}(\vec{d_i}) = \begin{cases} 1 & if \ D_i \in C_k \\ -1 & if \ D_i \in C_j \end{cases}$$

For finding the final class, different voting schemes can be applied. Max Wins is a voting scheme introduced by Friedman [27].

---

[7]Note that the scales of different, independently trained classifiers may not be directly comparable
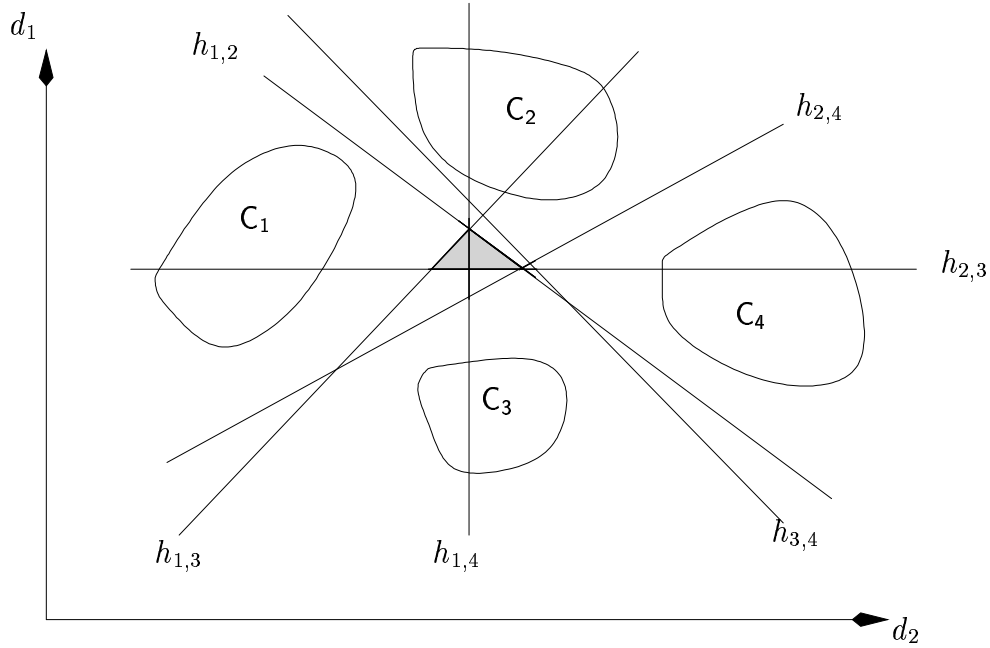
Figure 2.8.: Multiclass classification using pairwise classification. The shaded region defines the region of ambiguity if using a majority vote as decision base.

Thereby a majority vote over all classifiers given by $\arg\max_{C_k \in C} \sum_{C_i \neq C_k} \pi[h_{k,i} = 1]$ is calculated. In case of a tie the decision is rejected. Figure 2.8 shows pairwise classification of 4 classes.

Pairwise coupling, another voting scheme, is based on probabilistic models for estimating the correct class. The pairwise probability between two classes is calculated for this voting scheme, which is the probability of $\vec{d}$ belonging to class C_i, given that it can only belong to class C_i or C_j. Usually, the probability is based on the output of a classifier. A more detailed discussion on voting schemes is given in [32].

Another well known approach relies on building up a directed acyclic graph. Each node in the graph is assigned a 1 vs. 1 classifier. The path taken in the graph, i.e. which test should be performed next, depends on the outcome of the test. It was shown by [54], that for large margin classifiers a generalization bound on this decision DAG's can be given.

Normally pairwise classifiers do not suffer from the drawback of asymmetric example distribution (which for sure depends on the given training data). Learning and classification time depends on the learning algorithm and problem at hand. Since more classifiers have to be learned, one could suggest that learning time must increase. Studies have shown, that especially for SVM's, the learning and classification time can be reduced by using pairwise classification. The argument is, that 1 vs. 1 classifiers are simpler and therefore, in the case of SVM's, fewer kernel evaluations have to be performed (see [54]).

Error Correcting Codes: Error correcting codes are a more general approach for obtaining multi class decisions from binary ones. This approach was introduced in [17]. The general idea behind error correcting codes is to assign each class $C_i \in C$ a row in a coding matrix $\mathfrak{M} \in \{-1, 1\}^{|C|, l}$. The length of the binary code is given by l. Each column of matrix $\mathfrak{M}$ is assigned a binary classifier $h_k(\vec{d}) \rightarrow \{-1, 1\}$, $1 \leq k \leq l$. Training examples for a binary classifier h_k are examples corresponding to classes C_i having $\mathfrak{M}_{i,k} = 1$ as positive and $\mathfrak{M}_{i,k} = -1$ as negative examples.

For classifying a new example all binary decisions $h_{1...l}$ are evaluated, yielding a code in form of a vector $\vec{b}$ with length l. The class with the closest corresponding row of matrix $\mathfrak{M}$ is chosen. Closeness can be determined by some appropriate distance measures. Usually, the Hamming Distance, which is the number of different bits between codes, is used.

A unifying approach for margin based classifiers as well as an analysis of the resulting generalization performance by using one of these techniques is given in [2]. Within these experiments, concerning classification accuracy the one-to-rest approach is inferior compared to the other coding techniques. Also the output coding of error correcting codes depends strongly on the task at hand.

2.2.2. Boosting

Boosting algorithms define a special subfield of classifier committees. The idea behind classifier committees is that k independent expert hypotheses are combined to form the classification hypotheses. For classifier committees the experts are often different classes of hypothesis (e.g. Decision Tree, Probabilistic Classifier, Linear Classifier etc.). Typically Boosting uses only one type of hypothesis, the so called *weak* or *base* hypothesis.

In contrast to classifier committees, Boosting does not combine independent classifiers. Instead, Boosting combines hypotheses generated from the same base learning algorithm on different subsets of training examples. Therefore in each round, where a weak hypothesis is generated, an appropriate subset of training examples has to be chosen and finally all calculated weak hypotheses have to be combined. The choice of a weak hypothesis is not captured in the general definition of Boosting algorithms.

AdaBoost [26] is today's most common approach to Boosting with a well developed theoretical framework based on statistical learning theory. The rest of this section gives a brief introduction into AdaBoost focusing on theoretical bounds, multiclass classification, and the choice of weak learning algorithms. At last, AdaBoost applied to Text Classification is introduced.

2.2.2.1. AdaBoost

AdaBoost was developed in 1994 by Freund and Schapire. As shown in various studies AdaBoost is a fast and reliable classification method for a large field of applications. Also boosting applied to text classification performs quite successful, providing similar results as today's top notch methods like SVM's and LLSF ([79],[65]). A brief introduction to boosting is also given in [63][8].

Let $\boldsymbol{S} = \{< \vec{d_1}, y_1 > \cdots < \vec{d_m}, y_m >\}$ be a sequence of training examples $\vec{d_i}$ drawn from an instance space I and y_i be the corresponding label of example $\vec{d_i}$ from a finite label set $\boldsymbol{Y}$. For the sake of simplicity $\boldsymbol{Y} = \{-1, 1\}$ yields a binary classification problem which can be easily extended to a multiclass problem (see section 2.2.2.3 and 2.2.1.3)[9]

Training examples $\boldsymbol{S}$ and a distribution $\boldsymbol{B}$ over the training examples are used as input to a *weak* or *base* learning algorithm $\Phi(\boldsymbol{S}, \boldsymbol{B})$, which calculates a weak hypothesis $h : I \rightarrow \Re$. Thereby $\mathrm{sgn}(h(\vec{d}))$ is interpreted as predicted label for instance $\vec{d}$ and $|h(\vec{d})|$ reflects the confidence in this prediction. Although $h(\vec{d})$ can include all real numbers. If not stated otherwise it is assumed that $h(\vec{d})$ is bound to $[-1, 1]$ without loss of generality. The distribution $\boldsymbol{B}$ of examples is represented through weights $B(i) \in [0, 1]$ for each example.

---

[8]Boosting related information can be obtained from http://www.boosting.org

[9]Since regression problems are not covered within this thesis logistic regression versions of boosting like stated in [13] are not covered.

Algorithm 1 A generalized version of AdaBoost

Require: $S = <\vec{d_1}, y_1> \cdots <\vec{d_m}, y_m>, \vec{d_i} \in D, y_i \in Y = \{-1, 1\}$

Definitions:

　　S...Set of trainings examples with according labels

　　Y...Labels for examples, 1 for positives and -1 for negative examples

　　B...Distribution of trainings examples whereby $B_t(i)$ is the weight for $\vec{d_i}$ in Round t

　　Φ...Base Learner which takes S, B as input and outputs h_t

　　$h_t : D \rightarrow \{-1, 1\}$...Weak hypothesis generated by Φ under B_t

Function *AdaBoost* **begin**

　　Initialize $B_1(i) = \frac{1}{m}$

　　for all $t = 1 \ldots T$ **do**

　　　　Calculate weak hypothesis: $h_t = \Phi(S, B)$

　　　　Choose $\alpha_t \in \Re$

　　　　for all $B_t(i) \in B_t$ **do**

　　　　　　Update distribution:

　　　　　　$B_{t+1}(i) = \frac{B_t(i) * \exp(-\alpha_t y_i h_t(\vec{d_i}))}{Z_t}$

　　　　end for

　　　　where $Z_t = \sum_i B_t(i) * \exp(-\alpha_t y_i h_t(\vec{d_i}))$ norms B_{t+1} to a distribution

　　end for

　　Output final hypothesis $H(\vec{d}) = \text{sgn}(\sum_{t=1}^{T} \alpha_t h_t(\vec{d}))$

end function

Given the above definitions, in each round of boosting a weak hypothesis $h_t = \Phi(S, B_t)$, based on the current example distribution, is calculated. Afterwards the example distribution B_t is updated as

$$B_{t+1}(i) = \frac{B_t(i) * \exp(-\alpha_t y_i h_t(\vec{d_i}))}{Z_t}$$

where

$$Z_t = \sum_i B_t(i) * \exp(-\alpha_t y_i h_t(\vec{d_i}))$$

norms B_{t+1} to a distribution.

Let

$$f(\vec{d}) = \sum_{t=0}^{T} \alpha_t h_t(\vec{d})$$

be the weighted sum of weak hypothesis h_t. Than the final hypothesis $H(\vec{d})$ is calculated as

$$H(\vec{d}) = \text{sgn}\left(f(\vec{d})\right)$$

Intuitively, each weak hypothesis h_t is weighted based on its accuracy on the training examples with respect to their distribution in round t. Updating the example distribution (weights) can be viewed as making "hard" examples more and "easy" examples less important for the next round of boosting.

The training error of AdaBoost can be bound by

$$\frac{1}{m}|\{i : H(\vec{d_i}) \neq y_i\}| \leq \prod_{t=1}^{T} Z_t$$

which has been proved by Schapire and Singer in [64].

As a consequence of this bound, greedily minimizing Z_t in each round of boosting tends to minimize the training error of AdaBoost. This can be used for deriving α_t and for choosing the weak learner h_t, which will be described in the next section.

2.2.2.2. Choice of Weak Learners and α_t

Given the above bound on the training error, the goal is to find α_t which minimizes Z_t. So, lets assume that $u_i = y_i h_t(\vec{d_i})$ and a weak hypothesis $h_t(\vec{d})$ with range $[-1, 1]$ is given. If t is fixed, Z_t can be bound by:

$$Z = \sum_i B(i) e^{-\alpha u_i} \leq \sum_i B(i) \left(\frac{1 + u_i}{2} e^{-\alpha} + \frac{1 - u_i}{2} e^{\alpha} \right)$$

.

Next α_t can be chosen to minimize the right hand side of this equation as

$$\alpha = \frac{1}{2} \ln \left(\frac{1 + r}{1 - r} \right)$$

where $r = \sum_i B_t(i) u_i$. This choice gives the upper bound for Z of

$$Z \leq \sqrt{1 - r^2}$$

The quantity r_t is a natural measure of the correlation between the predictions of h_t and the y_i's with respect to the distribution $\boldsymbol{B}_t$.

For the case of a binary weak hypothesis h_t with range $\{-1, 1\}$, r_t is closely related to the ordinary error through $\epsilon_t = Pr_{i \sim \boldsymbol{B_t}}[h_t(\vec{d_i}) \neq y_i] = \frac{1 - r_t}{2}$. Thus, for binary h_t, the choice of α_t can be written in terms of the weak hypothesis error, as

$$\alpha = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

.

Choosing α_t this way minimizes the training error within each round of boosting through minimizing $Z_t = \sum_i B_t(i) * e^{-\alpha_t y_i h_t(\vec{d_i})}$. By folding α_t into h_t this expression can be simplified to

$$Z_t = \sum_i B_t(i) * e^{-y_i h_t(\vec{d_i})}$$

where it is assumed that the weak learner can freely scale every weak hypothesis by a constant factor $\alpha_t \in \Re$. In this case, besides providing weak hypotheses for the base learning algorithm, the weak learner is also responsible for minimizing Z_t. Some weak learners, as for example gradient based algorithm, can be easily adapted to minimize Z_t rather than the traditional mean squared error. Also, having weak learners which minimize Z_t can be beneficial in some practical settings for designing weak learning algorithms. This issue will be outlined in section 2.2.2.4 where weak learners for text classification are discussed.

Algorithm 2 AdaBoost.MH: AdaBoost extended to the multiclass, multi-label case using Hamming loss

---

Require: $\boldsymbol{S} = \langle \vec{d_1}, y_1 \rangle \cdots \langle \vec{d_m}, y_m \rangle, \vec{d_i} \in D, y_i \in \boldsymbol{Y} = \{-1, 1\}$

Definitions:

    $\boldsymbol{S}$...Set of trainings examples with according labels

    Y...Labels for examples, 1 for positives and -1 for negative examples

    $\boldsymbol{B}$...Distribution of training examples where $B_t(i, \mathsf{C_j})$ is the weight for $\vec{d_i}$ and class $\mathsf{C_j}$ in round t

    Φ...Base Learner which takes $\boldsymbol{S}$, $\boldsymbol{B}$ as input and outputs h_t

    $h_t : \boldsymbol{D} \times \boldsymbol{C} \rightarrow \{-1, 1\}$...Weak hypothesis generated by Φ under $\boldsymbol{B}_t$.

Function *AdaBoost* **begin**

    Initialize $\forall_{\vec{d_i} \in D, \mathsf{C_j} \in \boldsymbol{C}} B_1(i, \mathsf{C_j}) = \frac{1}{m*k}$

    for all $t = 1 \ldots T$ **do**

      Calculate weak hypothesis: $h_t = \Phi(\boldsymbol{S}, \boldsymbol{B})$

      Choose $\alpha_t \in \Re$

      for all $B_t(i, \mathsf{C_j}) \in \boldsymbol{B}_t$ **do**

        Update distribution:

        $B_{t+1}(i, \mathsf{C_j}) = \frac{B_t(i, \mathsf{C_j})*\exp(-\alpha_t y_i h_t(\vec{d_i}), \mathsf{C})}{Z_t}$

      end for

      where $Z_t = \sum_l \sum_i B_t(i, \mathsf{C_j}) * \exp(-\alpha_t y_i h_t(\vec{d_i}, \mathsf{C}))$ norms $\boldsymbol{B}_{t+1}$ to a distribution

    end for

    Output final hypothesis $H(\vec{d}, \mathsf{C_j}) = \mathrm{sgn}(\sum_{t=1}^{T} \alpha_t h_t(\vec{d}, \mathsf{C}))$

end function

2.2.2.3. Boosting applied to multi-label Problems

Till now only the binary case, where a document is in a given class or not, was considered. As discussed in section 2.2.1.3 the binary case can be extended to the multi label case by applying some voting schemes. The same holds for boosting. Additionally, Boosting can be easily extended to learn multi label assignments directly.

Formally, each example $\vec{d_i}$ is assigned to a finite set of classes or labels $\boldsymbol{C}_i \subset \boldsymbol{C}$ where $k = |\boldsymbol{C}_i|$ is the number of assigned classes. Thus, each example presented to the learning algorithm is a pair $< \vec{d_i}, \boldsymbol{C}_i >$, where $\boldsymbol{C}_i \subset \boldsymbol{C}$ is the set of labels assigned to $\mathsf{D_i}$. Whereas the goal of a learning algorithm for the single class case is to find a hypothesis h which minimizes the probability $Pr[y_i \neq h(\vec{d_i})]$ of a wrongly classified example, the goal of a learning algorithm for the multi label class is somehow unclear and may depend on the problem at hand.

One possibility is to seek a hypothesis $h(\vec{d_i})$ which predicts one of the assigned classes $h(\vec{d_i}) \in \boldsymbol{C_i}$ correctly. This measure is called *one error* of hypothesis h which can formally be written as

$$one - error(h) = Pr[\vec{d_i}, \boldsymbol{C}_i \mid h(\vec{d_i}) \notin \boldsymbol{C}_i]$$

For single label classification the one-error is identical to the ordinary error measure.

Another possibility is to predict all of the correct labels. The learning algorithm generates a hypothesis which predicts sets of labels. The loss or error of a hypothesis depends on how this predicted set differs from the one that was observed. Thus, we seek a hypothesis $h : \boldsymbol{D} \rightarrow 2^{\boldsymbol{C}}$ where the loss is

$$\frac{1}{n} E_{(\vec{d_i}, \boldsymbol{C_i})}[|h(\vec{d_i}) \triangle \boldsymbol{C}|]$$

where $\triangle$ denotes symmetric difference and $\frac{1}{n}$ is a normalization factor to ensure a value of $[0, 1]$. Here, n is the number of different classes $|\boldsymbol{C}|$. This measure is called *Hamming loss* of h.

For applying the hamming loss to boosting, each example $< \vec{d_i}, \boldsymbol{C_i} >$ is extended to $n = |\boldsymbol{C}|$ examples where each example is of the form $< \vec{d_i}, Y[\mathsf{C}] >$. Thereby $Y[\mathsf{C}]$ denotes that example $\vec{d_i}$ has label C assigned or not which can be formally written as

$$Y[\mathsf{C}] = \begin{cases} +1 & if\ \mathsf{C} \in \boldsymbol{C}_i \\ -1 & if\ \mathsf{C} \notin \boldsymbol{C}_i. \end{cases}$$

Using this setting the multi label case is again reduced to several binary classifications. Different from the multi label classification in section 2.2.1.3, the binary hypotheses are not learned independent from each other[10]. This is done by maintaining a distribution (i.e. a set of weights) over examples and classes. Algorithm 2 shows this extension of the general AdaBoost algorithm, called AdaBoost.MH, to the multi label case.

For the training error the following bound based on the hamming loss (denoted by $hloss(h)$) can be given:

$$hloss(H) \leq \prod_{t=1}^{T} Z_t$$

.

Again this bound indicates that for minimizing the hamming loss within each training round, Z_t has to be minimized. Z_t is given as

$$Z_t = \sum_{i, \mathsf{C_j} \in \boldsymbol{C}} B_t(i, \mathsf{C_j}) e^{-\alpha_t Y[\mathsf{C_j}] h_t(\vec{d_i}, \mathsf{C_j})}$$

As before Z_t can be minimized by

$$\alpha = \frac{1}{2} ln \left(\frac{1 + r_t}{1 - r_t} \right)$$

$$r_t = \sum_{i, \mathsf{C_j} \in \boldsymbol{C}} B_t(i, \mathsf{C_j}) Y[\mathsf{C_j}] h_t(\vec{d_i}, \mathsf{C_j})$$

for binary h_t with range over $\{-1, 1\}$. So the only difference between the single and multi-label case is the maintained distribution and the calculation of Z_t.

Additionally, AdaBoost.MH can be extended by other error measures. Advanced measurements are given in [64], where a ranking loss measurement and output coding for multiclass problems is introduced.

2.2.2.4. BoosTexter: Boosting applied to Text Classification

So far, the general boosting framework and the application to multi label problems have been illustrated. Furthermore, a brief introduction to the influence of α_t and the design of the weak learner was given above. This general framework can be applied to any classification task at hand, also for text classification. As mentioned above, weak classifiers may be any kind of classifier like for example neural networks or linear classifiers. This section discusses weak classifiers for the domain of text classification used with AdaBoost.MH, which are well known within literature.

---

[10]Although, in section 2.2.1.3 the learned classifier are not strictly independent from each other, since they share the same training data.

Since text classification deals with high dimensional spaces and in general with a large amount of given training data, a weak classifier should be fast and reliable. Thus this section focuses on simple classifiers. Specially, all of the methods illustrated in this section are one-level decision trees (see section 2.2.3.2 for a discussion on decision trees). Given terms t_k obtained from the indexing step (see section 2.1), the weak learner seeks the best term for partitioning the training data into correct classes (with respect to distribution $\mathbf{B}$). Formally speaking, the weak classifier returns a hypothesis of the form

$$h(\vec{d}, l) = \begin{cases} c_{0,l}^k & if\ d_k = 0 \\ c_{1,l}^k & if\ d_k \neq 0 \end{cases}$$

where c is a real number, $\vec{d}$ is the term vector of a document[11] D, j indicates whether term t_k exists in document D and k indicates the term which was chosen by the weak classifier.

The weak learners search all possible terms t_k of the given term space T. For each term values $c_{j,l}^k$ are chosen as discussed below. Once all terms have been searched, the weak hypothesis with the lowest score, which minimizes Z_t as shown below, is chosen. Training examples are given as sequence $\mathbf{S} = \{< \vec{d_1}, y_1 > \cdots < \vec{d_m}, y_m >\}$ where $Y_i[\mathsf{C_l}] = \{-1, 1\}$ indicates whether example $\vec{d_i}$ belongs to class l or not.

The binary case: For the binary case $c_{j,l}^k$ takes on the range of $\{-1, 1\}$ and thus the hypothesis is of the form $h(\vec{d}, l) \rightarrow \{-1, 1\}$.

All terms are searched and the term t_k with the lowest error on the training set is selected. Formally, with $c_{j,l}^k \in \{-1, 1\}$ the hypothesis for term t_k is

$$h_k(\vec{d}, l) = \begin{cases} -1 & if\ d_k = 0 \\ 1 & if\ d_k \neq 0 \end{cases}$$

The weak learner returns the hypothesis $h(\vec{d}, l) = \arg\min_{h_k} \epsilon_{t_k, l}$ which minimizes the error on the training set, given by

$$\epsilon_{t_k, l} = \sum_{i:d_k \neq 0} B_t(i) \pi[Y_i[\mathsf{C_l}] = -1] + \sum_{i:d_k = 0} B_t(i) \pi[Y_i[\mathsf{C_l}] = 1]$$

with respect to distribution $\mathbf{B}$.

Next α has to be chosen for minimizing Z_t. As stated above, this is achieved by choosing $\alpha = \frac{1}{2} ln \left(\frac{1 - \epsilon_{t_k, l}}{\epsilon_{t_k, l}} \right)$.

AdaBoost.MH with real valued predictions: For this weak learner $c_{j,l}^k$ is a unrestricted real-valued number. For a given term t_k the training data is partitioned into sets of examples containing term t_k or not, which can be formulated as $\mathbf{D}_0 = \{\vec{d} \mid d_k = 0\}$ and $\mathbf{D}_1 = \{\vec{d} \mid d_k \neq 0\}$. Given the current distribution $\mathbf{B}$ and each possible label l, weights $W_b^{j,l}$ are calculated as

$$W_b^{j,l} = \sum_{i=1}^m B_t(i, l) \pi[\vec{d_i} \in \mathbf{D}_j \wedge Y_i[\mathsf{C_l}] = b]$$

where $\pi[x]$ is 1 if x is true and 0 else, $j = \{0, 1\}$ and $Y_i[\mathsf{C_l}] = \{-1, 1\}$ indicates whether example $\vec{d_i}$ belongs to class $\mathsf{C_l}$ or not. Thus, $W_{+1}^{j,l}$ ($W_{-1}^{j,l}$) is the weight of documents in partition $\mathbf{D}_j$ which are

---

[11]Note that $d_k = 0$ if the term does not occur in the document (with respect to the used indexing method) and $d_k \neq 0$ if the term occurs in the document

(are not) labeled by l. Partition $\boldsymbol{D}_1$ ($\boldsymbol{D}_0$) is the set of documents which contain (contain not) term t_k. As shown in [65], Z_t can be minimized for a particular term t_k by choosing

$$c^k_{j,l} = \frac{1}{2} \ln \left(\frac{W^{j,l}_{+1}}{W^{j,l}_{-1}} \right)$$

and setting $\alpha_t = 1$.

These settings imply that

$$Z_t = 2 \sum_{j \in \{0,1\}} \sum_{\mathsf{C}_\mathsf{I} \in \boldsymbol{C}} \sqrt{W^{j,l}_{+1} W^{j,l}_{-1}}.$$

Thus, the term t_k for which Z_t is smallest will be chosen.

AdaBoost.MH with real valued predictions and abstaining: The above hypothesis makes the assumption that the presence and the absence of a term carries information on the class a document belongs to. Within text classification it may be useful that the weak hypothesis abstains from the decision if a term is not contained in a document. This can be accomplished by letting the hypothesis output a value of zero in the absence of a term which is

$$h(\vec{d}, l) = \begin{cases} 0 & if\ d_k = 0 \\ c^k_{1,l} & if\ d_k \neq 0 \end{cases}$$

with $\alpha_t = 1$ and $c^k_{1,l}$ given from above as

$$c^k_{1,l} = \frac{1}{2} \ln \left(\frac{W^{1,l}_{+1}}{W^{1,l}_{-1}} \right)$$

So let

$$W_0 = \sum_{i | \vec{d}_i \in \boldsymbol{D}_0} B_t(i, l)$$

be the weight of all documents not containing term t_k. Thus, it can be shown (see [64]) that

$$Z_t = W_0 + 2 * \sum_{\mathsf{C}_\mathsf{I} \in \boldsymbol{C}} \sqrt{W^{1,l}_{+1} W^{1,l}_{-1}}$$

and as before the term minimizing Z_t is chosen.

The advantage of abstaining is an improvement in running time and, as stated in [65], the performance is comparable to the version without abstaining. The improvement in running time results due to that only weights $B_t(i, l)$ for documents $\vec{d}_i \in \boldsymbol{D}_1$ have to be updated.

The main hypothesis returned by AdaBoost.MH, using real valued predictions (with and without abstaining) as weak classifier, is a linear classifier with binary input vectors.

Hence, the main hypothesis of AdaBoost is

$$H(\vec{d}, l) = \sum_{t=1}^{T} h_t(\vec{d}, l) = \sum_{t=1}^{T} c^k_{1,l}(t) * \pi[d_k \neq 0] - \sum_{t=1}^{T} c^k_{0,l}(t) * \pi[d_k = 0]$$

where $c^k_{j,l}(t)$ indicates the value output by the weak classifier in round t for class l, if term t_k was chosen by the weak classifier in round t. Note that if term t_k was not chosen by the weak classifier in round t than $c^k_{0,l}(t) = 0$ and $c^k_{1,l}(t) = 0$ and that only one term can be selected in one round.

31

So, if the weight vector $\vec{w}_l = < w_1, \dots \vec{w}_{|T|} >$ for a class l is considered as

$$w_{k,l} = \sum_{t=1}^{T} c_{1,l}^{k}(t) - \sum_{t=1}^{T} c_{0,l}^{k}(t)$$

and the document vector $\vec{d}$ is given as binary vector, the main hypothesis for a label l can be written as

$$H(\vec{d}, l) = \sum_{k} \left[\sum_{t=1}^{T} c_{l,k}^{k}(t) * \pi[d_k \neq 0] - \sum_{t=1}^{T} c_{0,l}^{k}(t) * \pi[d_k = 0] \right] = \sum_{k} w_{k,l} * d_k = \vec{w}_l \cdot \vec{d}$$

which is actually a linear classifier with binary input vectors $\vec{d}$. For improving speed, an inverted index mapping terms to documents is created before learning a node.

Furthermore, in each round

$$W^{j,l} = \sum_{i=1}^{m} D_t(i, l)$$

is precomputed before scanning the inverted list of indexes. Afterwards, for each term the value $W_{+}^{j,l}$ is calculated by summing over the documents in which the term appears. Finally, $c_{j,l}^{k}$ and Z_t is calculated using $W_{i}^{j,l} = W^{j,l} - W_{+}^{j,l}$.

2.2.2.5. CentroidBoosting: An extension to BoosTexter

Encouraged from good accuracy and fast learning of the Rocchio algorithm, a modified version of Rocchio was used as weak classifier. Within each boosting round a positive $\vec{c}^{+}$ and negative $\vec{c}^{-}$ centroid vector are calculated with respect to the current example distribution. The negative centroid vector is subtracted form the positive one, yielding to a separating hyperplane between the given training examples. So let T_{i}^{+} be the positive training examples for class C_i and let T_{i}^{-} be the negative training examples for class C_i. Formally, if the positive centroid vector of class C_i is calculated as

$$\vec{c}_i^{+} = \frac{1}{|T_i^{+}|} \sum_{\vec{d}_j \in T_i^{+}} B_t(j) \vec{d}_j$$

and the negative centroid vector is calculated as

$$\vec{c}_i^{-} = \frac{1}{|T_i^{-}|} \sum_{\vec{d}_j \in T_i^{-}} B_t(j) \vec{d}_j$$

then the classification hypothesis returned by the weak classifier is

$$h_i(\vec{d}) = \left(\frac{\vec{c}_i^{+}}{\|\vec{c}_i^{+}\|} - \frac{\vec{c}_i^{-}}{\|\vec{c}_i^{+}\|} \right) \cdot \vec{d} = \vec{w}_i \cdot \vec{d}$$

Here, $sgn(h_i(\vec{d}))$ determines whether an example is assigned to node C_i or not. Updating the distribution weight $B_t(j)$ of an example $\vec{d}_j$, given the current weak hypothesis $\vec{w}_t$, is based on the distance of $\vec{d}_j$ to the separating hyperplane. So, if α_t is viewed as being folded into the weak classifier (see section 2.2.2.2) then explicitly calculating α_t can be written as

Algorithm 3 Centroid Booster

Require: $S = <\vec{d_1}, y_1 > \cdots < \vec{d_m}, y_m >, \vec{d_i} \in \boldsymbol{D}, y_i \in \boldsymbol{Y} = \{-1, 1\}$

Definitions:

    S...Set of trainings examples with according labels

    Y...Labels for examples, 1 for positives and -1 for negative examples

    B...Distribution of trainings examples whereby B_i is the weight for $\vec{d_i}$

    T...Number of Boosting rounds

Function *CentroidBooster* **begin**

    Initialize $B_1(i) = \frac{1}{m}$

    for all $t = 1 \ldots T$ **do**

        $c^+ = \frac{1}{|\boldsymbol{T_i^+}|} \sum_{\vec{d_j} \in \boldsymbol{T_i^+}} B_t(j) \vec{d_j}$

        $c^- = \frac{1}{|\boldsymbol{T_i^-}|} \sum_{\vec{d_j} \in \boldsymbol{T_i^-}} B_t(j) \vec{d_j}$

        $\vec{w_t} = \frac{\vec{c_i^+}}{\|\vec{c_i^+}\|} - \frac{\vec{c_i^-}}{\|\vec{c_i^+}\|}$

        $\vec{w_t} = \frac{\vec{w_t}}{\|\vec{w_t}\|}$

        for all $B_t(i) \in \boldsymbol{B_t}$ **do**

            $B_{t+1}(i) = \frac{B_t(i) * \exp(-y_i * \vec{w_t} \cdot \vec{d_i})}{Z_t}$

        end for

        where $Z_t = \sum_i B_t(i) * e^{-y_i * \vec{w_t} \cdot \vec{d_i}}$ norms $\boldsymbol{B_{t+1}}$ to a distribution

    end for

    Output final hypothesis $H(\vec{d}) = sign(\sum_{t=1}^{T} \vec{w_t} \cdot \vec{d})$

end function

Centroid Boosting algorithm. In each round a Rocchio classifier is calculated based on the current example distribution. Weights of examples close to the hyperplane are increased, otherwise decreased.

$$\alpha_t = \vec{w} \cdot \vec{d}$$

and thereby the update for an example $\vec{d_j}$ is

$$B_{t+1}(j) = B_t(j) * e^{-y_i * \eta * \vec{w} \cdot \vec{d_j}}$$

Intuitively this update rule implies that the weights of examples having a large distance from the hyperplane are decreased and the weights of wrongly classified examples or examples close to the hyperplane are increased. In the next round of boosting, examples which were hard to classify have a higher contribution to the separating hyperplane than those which had been classified easily. The positive and negative prototype vectors are calculated from the new distribution and so a Rocchio classifier focusing on those "hard" to classify examples is calculated in the next round. Taking the weighted average of all examples as centroid vectors can also be viewed as calculating the mean of a normal distribution with respects to the distribution weights. Furthermore, if all terms are considered to be statistically independent and having the same variance σ^2 (which yields to a covariance matrix of $\Sigma = \sigma^2 \mathfrak{I}$), then the hyperplane $\vec{w} \cdot \vec{d} = 0$ separates the two distributions halfway between their means and is orthogonal to $\vec{w}$. Thus, the resulting weak classifier is a minimum distance classifier with respect to the given weight distribution. In [18] a more detailed illustration of these aspects is given. The factor η is be used to optimize the update of the distribution weights $\boldsymbol{B}$ with respect to minimizing Z_t. In this thesis, η is set to 1 since the optimal value of η is hard to find and beyond the scope of this thesis.

As described above, updating the example distribution using BoosTexter can be viewed as binary decision. The weight of an example is updated by a real number $\alpha_t = c_{i,j}^k$ if it contains the chosen term, otherwise no update is performed (abstaining) by setting $\alpha_t = 0$. Centroid boosting updates each example according to the distance from the current hyperplane. In general, each example has a different α_t for updating the example weight.

The main hypothesis for Centroid boosting is a linear classifier, which is an averaged Rocchio classifier on different example distributions. Algorithm 3 shows the centroid boosting algorithm.

Replacing the main hypothesis by a correct weak hypothesis: Since the main hypothesis obtained by centroid boosting has the same hypothesis class as the weak hypothesis, the main hypothesis given after a specific number of iterations may be replaced by the current weak hypothesis. The rational therefore is, that with the growing number of boosting rounds, the Rocchio hypothesis obtained from the actual example distribution may linearly separate the given data. In the case of noise free data, this weak hypothesis yields to a better classification hypothesis than the actual main hypothesis. This is because of the current main hypothesis being an average of weak hypotheses before, which do not necessarily linearly separate the data. After taking this weak hypothesis as new main hypothesis, boosting is performed as usual. The performance of this technique is evaluated within the experiments.

2.2.3. Other Classification Methods

This section provides a rough overview on classification methods not covered within this thesis.

2.2.3.1. Probabilistic Classifiers

Probabilistic classifiers assign a document $\vec{d}$ to a class C_i by evaluating the propability that class C_i is drawn given document $\vec{d}$, which is $Pr[C_i \mid \vec{d}]$ and thresholding the probability. For estimating this probability from given training data Bayes theorem is used:

$$Pr[C_i \mid \vec{d_j}] = \frac{Pr[\vec{d_j} \mid C_i] * Pr[C_i]}{Pr[\vec{d_j}]}$$

Here, $Pr[C_i]$ is the prior probability for class C_i which is simply the class frequency $Pr[C_i] = \frac{|\{D_j | D_j \in C_i\}|}{|D|}$. The propability $Pr[\vec{d_j}]$, for a document drawn at random from a given distribution $\boldsymbol{D}$, can be ignored if only the single class maximizing $Pr[C_i \mid \vec{d}]$ is needed. This is because $Pr[\vec{d_j}]$ is independent of the classes $C_i \in \boldsymbol{C}$.

Estimating $Pr[\vec{d_j} \mid C_i]$ is difficult, since $\vec{d_j}$ is a high dimensional vector and considering all dependencies between vector components is computational expensive. Thus, assumptions for calculating the Likelihood $Pr[\vec{d_j} \mid C_i]$ have to be made. A common way to do this is to assume that coordinates (i.e. each term) are independent of each other leading to

$$Pr[\vec{d_j} \mid C_i] = \prod_{k=1}^{|T|} Pr[t_{k,j} \mid C_i]$$

where $Pr[t_{k,j} \mid C_i]$ is the probability that, given class C_i term $t_{k,j} \in D_j$ is drawn. Due to the naive nature of this assumption the approach is called Naive Bayes classifier.

$$t = Sport$$

$t \in d_j \qquad t \notin d_j$

$C = Sport \qquad t = politics$

$t \in d_j \qquad t \notin d_j$

$C = \overline{Sport} \qquad t = competition$

$t \in d_j \qquad t \notin d_j$

$C = Sport \qquad C = \overline{Sport}$
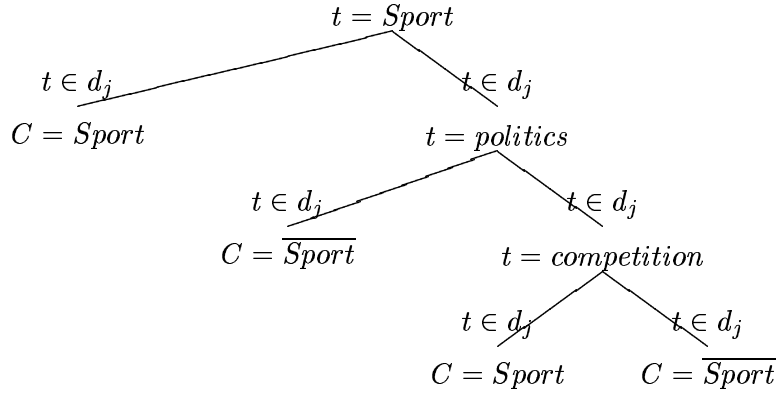
Figure 2.9.: A simple example decision tree. Branches are taking on binary decisions depending on whether a given term exists within a document or not. Leaf nodes indicate if the document belongs to class "Sport" or not (denoted by "$\overline{Sport}$").

Training of a Naive Bayes classifier is done by evaluating the probabilities of a term t_k on the training data. One approach for approximating the factors $Pr[t_k \mid \mathsf{C_i}]$ is to use term occurrences, formally written as

$$Pr[t_k \mid \mathsf{C_i}] = \frac{1 + occ(t_k, \mathsf{C_i})}{|\boldsymbol{T}| + \sum_{l=1}^{|\boldsymbol{T}|} occ(t_l, \mathsf{C_i})}$$

where $occ(t_k, \mathsf{C_i})$ is the number of times term k occurs in documents of class $\mathsf{C_i}$ in the training set. Adding 1 in the numerator and $|\boldsymbol{T}|$ in the denominator is know as Laplace smoothing which adds pseudo counts to all word frequencies. If Laplace smoothing is omitted, a term which occurs not in the training data but in a unseen document which has to be classified, would yield to probability $Pr[\vec{d_j} \mid \mathsf{C_i}] = 0$, regardless if the document belongs to this class or not. For a detailed discussion on probabilistic text classifier and the impact of the independence assumption as well as different distribution models see [41] and [68].

2.2.3.2. Decision Tree Classifier

So far, all described classifiers were quantitative in nature, which means that their classification hypothesis is based on an approximated real valued or probabilistic function. Decision tree and decision rule classifiers are different, since their classification hypothesis is based on a symbolic representation.

A decision tree text classifier is a binary tree $\mathcal{T}$ where each inner node is labeled by a term t_k and each branch defines a test on this term deciding if a branch should be taken or not. Leaf nodes are representing classes $\mathsf{C_i} \in \boldsymbol{C}$ of documents $\boldsymbol{D}$. Figure 2.9 shows a simple decision tree.

Classifying a document $\mathsf{D_j}$ means recursively traversing the decision tree by deciding in each inner node which branch should be taken. The decision is based on the representation of a document $\mathsf{D_j}$ (i.e. the term vector $\vec{d_j}$) and the decision rule for this branch. Classification stops if a leaf node is reached. The class corresponding to the leaf node is assigned.

There are several well known training algorithms for decision trees like ID3, C4.5 and C5 (see [46]). Usually these algorithms follow a *divide and conquer* strategy which could be generalized to the following rule

1. check if all training examples of an inner node have the same label and if so make this node a

leaf node with the documents label.

2. if not, take term t_k which partitions the current training examples of an inner node best to a given criterion and t_k becomes the label of the created inner node.

These steps are performed recursively till no more inner nodes are created. Selecting term t_k is the important step whereby *information gain* or *entropy* are commonly used as decision criteria.

Decision trees tend to overfit on the training data, since a fully grown tree consists of very specific branches resulting from the training data. Therefore, pruning of branches using a validation set is performed. For more details on decision trees see [46].

2.2.3.3. Example Based Classifier

All above illustrated classification approaches calculate an explicit classification hypothesis using a training set. In contrast, example based classifiers classify unseen documents based on the similarity to all training documents. The class having the highest number of similar training documents is chosen as correct class. Since no training is performed, example based classifiers are also called *lazy learners*.

One of the most popular example based classifier is the *k-nearest neighbor* (k-NN) algorithm. For deciding whether an unseen document D_j corresponds to a class C_i, k-NN tests if the majority of k nearest neighbor documents are more often in class C_i than in any other class:

$$h(\vec{d}_j) = \arg\max_{C_i \in C} \sum_{d_k \in D_{k,\vec{d}_j}} \pi[\Phi(\vec{d}_k, C_i) = T]$$

where $D_{k,\vec{d}_j}$ is the set of k training examples closest to the unclassified example $\vec{d}_j$ based on some distance measure $\Delta(\vec{d}_i, \vec{d}_j)$.

This formulation of the k-NN algorithm performs single label text classification. In [78] a *distance-weighted* version of k-NN is introduced, which calculates hypothesis of the form $h_{C_i}(\vec{d}_j) \rightarrow \Re$,

$$h_{C_i}(\vec{d}_j) = \sum_{d_k \in D_{k,\vec{d}_j}} \frac{1}{\Delta(\vec{d}_k, \vec{d}_j)} \pi[\Phi(\vec{d}_k, C_i) = T]$$

where $\Delta(\vec{d}_k, \vec{d}_j)$ is again some distance measure for defining $D_{k,\vec{d}_j}$. The distance measure $\Delta(\cdot, \cdot)$ may be replaced by some similarity measure (e.g. cosine similarity) and by appropriately transforming the above functions. On the distance weighted version of k-NN, thresholding must be performed for obtaining a binary decision.

Since k-NN classifiers do not explicitly generate a hypothesis they are not restricted to a specific class of hypothesis like for example linear classifiers. But since they are not restricted to a specific class of hypothesis they may tend to overfit, especially for wrongly chosen numbers of k. As shown in [78],[39] k-NN yields good results within the text classification domain.

2.3. Performance Measures

Various performance measures within text classification exist, covering different aspects of the task. This multitude makes comparisons between various experiments difficult. This section covers the most used performance measures, their benefits and drawbacks.

Category		Target Function Φ	
C_i		True	False
Approximation	True	TP_i	FP_i
$\tilde{\Phi}$	False	FN_i	TN_i

Table 2.2.: Contingency table for a class C_i.

2.3.1. Precision and Recall

Performance within text classification is usually given in terms of precision and recall, as in information retrieval.

As stated in section 1.3 a classifier approximates the unknown target function $\Phi : \boldsymbol{D} \times \boldsymbol{C} \to \{T, F\}$. Therefore a classifier makes a binary decision for each pair $< D_j, C_i >$. Each decision has four possible outcomes with respect to the correct classification and can be outlined using a contingency table (see table 2.2).

Given the contingency table precision $prec_i$ and recall rec_i for class C_i can be estimated as follows:

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

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

In words recall is an estimator for the "degree of how many documents of a class are classified correctly" and precision is an estimator for the "degree that if a document is assigned to the class, this assignment will be correct". In terms of conditional probabilities with resp. to C_i this can be viewed as

$$prec_i = Pr[\Phi(D_j, C_i) = T | \tilde{\Phi}(D_j, C_i) = T]$$

$$rec_i = Pr[\tilde{\Phi}(D_j, C_i) = T | \Phi(D_j, C_i) = T]$$

So precision and recall are class related measurements and have to be averaged somehow for obtaining a global measurement. Two different methods are generally used:

- *microaveraging*

  Precision and recall are averaged over all taken decisions, independent from the participating class:

  $$prec_i^{\mu} = \frac{\sum_{i=1}^{|\boldsymbol{C}|} TP_i}{\sum_{i=1}^{|\boldsymbol{C}|} (TP_i + FP_i)}$$

  $$rec_i^{\mu} = \frac{\sum_{i=1}^{|\boldsymbol{C}|} TP_i}{\sum_{i=1}^{|\boldsymbol{C}|} (TP_i + FN_i)}$$

- *macroaveraging*

  Precision and recall are evaluated for each class and averaged afterward:

  $$prec_i^{M} = \frac{\sum_{i=1}^{|\boldsymbol{C}|} prec_i}{\boldsymbol{C}}$$

  $$rec_i^{M} = \frac{\sum_{i=1}^{|\boldsymbol{C}|} rec_i}{\boldsymbol{C}}$$

Usually in text classification experiments microaveraged precision and recall values are used. Microaveraging mostly yields to better results than macroaveraging in the text classification domain. This circumstance seems to be quite natural since classes having a low class frequency (and therefore less positive trainings examples) are usually harder to learn.

2.3.2. Other measurements

Beneath precision and recall other measurements can be derived using the contingency table. Accuracy

$$acc_i = \frac{TP_i + TN_i}{TP_i + FP_i + FN_i + TN_i}$$

and error

$$err_i = \frac{FP_i + FN_i}{TP_i + FP_i + FN_i + TN_i} = 1 - acc_i$$

are not widely used within text classification due to that accuracy (resp. error) has the tendency to favor classifiers behaving like trivial rejectors $\forall_{\mathsf{C}_i \in \boldsymbol{C}} \tilde{\Phi}(\mathsf{D}_j, \mathsf{C}_i) = F$. By considering a trivial rejector as classification hypothesis and given $m = |\boldsymbol{D}|$ and $n_i = |\{\mathsf{D}_k \mid \mathsf{D}_k \in \mathsf{C}_i\}|$, accuracy and error can be written as

$$acc_i = \frac{m - n_i}{m} = 1 - \frac{n_i}{m}$$

$$err_i = \frac{n_i}{m}$$

So err_i is the class frequency which is usually very low in text classification tasks. Therefore, the trivial rejector tends to outperform many non trivial classifiers in terms of error and accuracy (see [11]) and it follows that accuracy and error is not an aproperiate measure for text classification. For additional performance measures, based on user defined costs, see [68].

2.3.3. Combination of Precision and Recall

Classifiers within text classification are often of the form $h_i : \boldsymbol{D} \rightarrow [0, 1]$. As introduced in section 2.2, for obtaining a binary decision from h_i for the pair $< \mathsf{D}_j, \mathsf{C}_i >$ a threshold Θ_i is used. If Θ_i is set to 0 one would receive a trivial acceptor for each class, assigning all documents $\mathsf{D}_j \in \boldsymbol{D}$ to all classes $\mathsf{C}_i \in \boldsymbol{C}$. A trivial acceptor would yield a recall of 1 having a precision of

$$prec^\mu = prec^\mu_{acceptor} = \frac{\sum_{\mathsf{C}_i \in |\boldsymbol{C}|} \boldsymbol{D}^+_{\mathsf{C}_i}}{|\boldsymbol{C}| * |\boldsymbol{D}|}$$

which is the average distribution of positive test examples over all classes ($\boldsymbol{D}^+_{\mathsf{C}_i}$ are the positive documents for class C_i). Precision and recall are not strictly symmetric, since for the trivial rejector case precision is undefined. By using the contrapositive precision and recall, symmetry can be shown (see [68]).

So for ranking classifiers isolating precision from recall makes no sense. An exception to this are single label classifiers, which return only one class (e.g. $\mathsf{C}_k = \arg\max_{\mathsf{C}_i}(h_i(\mathsf{D}_j))$ in the case of the above definition of h_i. Due to the fact that if a document is assigned to one class (thereby increasing TP or FP) it can not be assigned to another class (thereby increasing FN or TN) either precision or recall can be used as performance measure.

In practice ranking classifiers can be tuned toward precision or recall by setting an appropriate threshold. Hence classifiers should be evaluated by a measure which combines precision and recall. Methods for combining precision and recall are outlined in the next paragraphs.

F_β **Function:** The F_β measure is defined as

$$F_\beta = \frac{(\beta^2 + 1) * prec * rec}{\beta^2 * prec + rec}$$

and was first introduced in [57].

Thereby, β defines the importance of precision vs. recall. If $\beta = 1$ precision and recall are equally important, for $\beta = 0$, F_β coincides with precision and for $\beta = \infty$ F_β coincides with recall. Usually, β is set to 1. It can be shown (see [48]) that the F_1 value is always greater or equal to the precision/recall breakeven point of a classifier.

Precision/Recall Breakeven Point: For evaluating the precision/recall breakeven point (PRB) a classifier's threshold is tuned till precision is equal to recall. Various precision and recall values are evaluated repeatedly with varying threshold so that recall raises from 0 to 1 and precision drops from 1 to $prec^\mu_{acceptor}$. If there exists no threshold Θ for which $prec = rec$ the breakeven point is interpolated between the closest precision recall pairs.

As stated by Yang [79] the interpolated breakeven point may not be a reliable measure if the precision/recall pairs are not close enough. Additionally, the PRB is an artificial measure (if interpolated) due to the fact that no parameter settings of a classifier can achieve the breakeven point. Furthermore, the breakeven point is not immediately desirable from a users point of view.

11-Point Average Precision: For 11-point average precision the threshold Θ is adapted to obtain recall values of $0, 0.1, 0.2 \ldots 1$. For each recall step a precision value is calculated. The 11-point average precision is obtained by averaging over all precision values. For classifiers making hard classifications (see section 2.2) recall may not be varied at will. Therefore, precision and recall is defined in another way. Let $\boldsymbol{R}_j = \{\mathsf{C_k} \mid \tilde{\Phi}(\mathsf{D_j}, \mathsf{C_k}) = T\}\}$ be the ranked result set of classes for document $\mathsf{D_j}$ and let $\boldsymbol{R}_j^+ = \{\mathsf{C_k} \mid \Phi(D_j, \mathsf{C_k}) = T\}$ be the correct result set for a document $\mathsf{D_j}$. The ranking on $\boldsymbol{R}_j$ is defined as $rank_h(\mathsf{C_j}) < rank_h(\mathsf{C_k}) \cdots < rank_h(\mathsf{C_m}), rank_h(\cdot) \in \mathbb{N}$ where $rank_h(\mathsf{C_j}) < rank_h(\mathsf{C_k})$ means that $\mathsf{C_j}$ achieved a better (lower) ranking then $\mathsf{C_k}$ with respect to hypothesis h. Then precision and recall are calculated as (see [79])

$$rec_j = \frac{|\boldsymbol{R}_j \cap \boldsymbol{R}_j^+|}{|\boldsymbol{R}_j^+|}$$

$$prec_j = \frac{|\boldsymbol{R}_j \cap \boldsymbol{R}_j^+|}{|\boldsymbol{R}_j|}$$

The recall values of $0, 0.1, 0.2 \ldots 1$ are obtained by thresholding the result set based on the rank $rank_h(\mathsf{C_k})$, which means that the classifier decides how far down the ranked list one has to go. Precision and recall are averaged over all test documents which leads to a microaveraged precision/recall estimation.

2.3.4. Measuring Ranking performance

An approach for measuring ranking effectiveness in multi-label systems is given by Schapire and Singer in [65]. They introduced three different performance criteria covering different aspects of a ranking classifier. The intuition behind these criteria is similar to query estimation in information retrieval systems. Given a query the returned documents should be ranked according to a given relevance measurement. For text classification this means that in a ranked list of possible categories the true categories should be ordered first (i.e. having the lowest rank). As for 11-point average precision, a classifier returns a set of classes (ranked list) $\boldsymbol{R}_j$ as defined before.

One-error: The one error evaluates how many times the top-ranked label was not in the set of correct labels. It can be formally written as

$$one_{err} = \frac{1}{|\boldsymbol{D}|} \sum_{j=1}^{|\boldsymbol{D}|} \pi[\mathsf{C_i}^1 \cap \boldsymbol{R}_j^+ = \emptyset]$$

where $\mathsf{C_i}^1$ denotes the class having $rank_h(\mathsf{C_i}) = 1$ for document D_j. Straightforward the one-error estimates the performance of a system for the top ranked document. In single label classification the one-error is the ordinary error.

Coverage: Coverage measures how a system correctly identifies all correct labels of a document. Thus, it calculates the average length of the list of ranked classes that contains all correct classes for a document. Formally coverage can be written as:

$$coverage_{\boldsymbol{D}} = \frac{1}{|\boldsymbol{D}|} \sum_{j=1}^{|\boldsymbol{D}|} \max_{\mathsf{C_i} \in \boldsymbol{R}_j^+} \left(rank_h(\mathsf{C_i})\right) - 1$$

For single-label classification problems, coverage is the average rank of the correct label and is zero if the system does not make any classification errors.

Average Precision: Coverage and one-error are not complete measures for the multi-label case. A low one-error is achievable but at the same time the coverage may be high and vice versa. To assess both aspects non interpolated average precision may be used. Usually this performance measure is used in information retrieval systems to measure the document ranking performance of a query. Adapted to multi-label text classification average precision measures the effectiveness of label ranking. Formally, average precision is written as

$$avgprec_{\boldsymbol{D}} = \frac{1}{|\boldsymbol{D}|} \sum_{j=1}^{|\boldsymbol{D}|} \frac{1}{|\boldsymbol{R}_j^+|} \sum_{\mathsf{C_i} \in \boldsymbol{R}_j^+} \frac{|\{\mathsf{C_i}' \in \boldsymbol{R}_j^+ \mid rank_{h(\vec{d}_j)}(\mathsf{C_i}') \leq rank_{h(\vec{d}_j)}(\mathsf{C_i})|\}}{rank_{h(\vec{d}_j)}(\mathsf{C_i})}$$

which is the average fraction of correct classes $\boldsymbol{R}_j^+$ ranked before a correct class $\mathsf{C_k} \in \boldsymbol{R}_j^+$. Thus, $avgprec() = 1$ if $h(\vec{d}_j)$ perfectly ranks the correct classes of D_j.

3. Hierarchical Classification

Hierarchical classification is performed top down. Classification of new examples is done by starting at the root node and traversing the hierarchy till correct classes are found. Recursively at each node a classifier decides which branches i.e. which edges of the graph should be traversed further down. By doing so, two problems have to be considered:

1. Error Propagation

   If an upper level decision node makes a wrong decision by erroneous forwarding or not forwarding a document to a sub hierarchy, this document may be wrongly classified. Thus, this erroneous decisions may lead to invalid classifications.

2. Confidence of a decision

   Classifiers in different subtrees are trained independent by each other. Results between those classifiers can not be directly compared. So all decisions along the classification path have to be taken into account and a mechanism for comparing independent classification decisions has to be applied.

In addition to the proposed hierarchical classification model, this chapter focuses on these two problems. The chapter is organized as follows: First a formal description of the basic classification model is given. A model for estimating the confidence of a decision is introduced in section 3.2 and training set selection for reducing error propagation is described in the section 3.3. The chapter concludes with describing related work in the area of hierarchical text classification.

3.1. Basic Model

The approach taken in this model is to calculate classifiers within each node. Given a document vector as input, the classifier decides which, if any, path from the parent node should be taken for further traversing the hierarchy. Additionally, the classifier should output a confidence for taking this decision. Classification stops if no more paths are selected by any node classifiers. As stated in the introduction, this approach is know as *top down* approach (see section 1.3.2).

For a detailed formulation, the concept of decision nodes is introduced. Within a decision node the selection of the branches takes place. So let us define a decision node E^k of a hierarchy $\mathcal{H}$ as a set of nodes $\boldsymbol{N}_k = \{N_k\} \cup \{N_i \mid N_k \to N_i\}$ where N_k is the root node of E^k and $\boldsymbol{N}_k \backslash N_k$ are the direct children of N_k (see figure 3.1). Node N_i^k is equivalent to node N_i of the graph, whereby the superscript indicates that N_i is used in decision node E^k. Thus, the superscript k resolves the overlap between decision nodes. Each decision node E^k is assigned a classifier $h^k(\vec{d})$ which outputs a set of nodes $\boldsymbol{N}_c \subseteq \boldsymbol{N}_k$ for a given document $\vec{d}$. Additionally, each node $N_j \in \boldsymbol{N}_c$ is assigned a real valued number ranging between $[0, 1]$, reflecting the confidence for the correctness of the decision.

As mentioned in section 2.2.1.3, various approaches can be taken for performing multi-label classification. Possible approaches depend on the classification technique used. In this thesis two approaches for multi-label classification are considered:
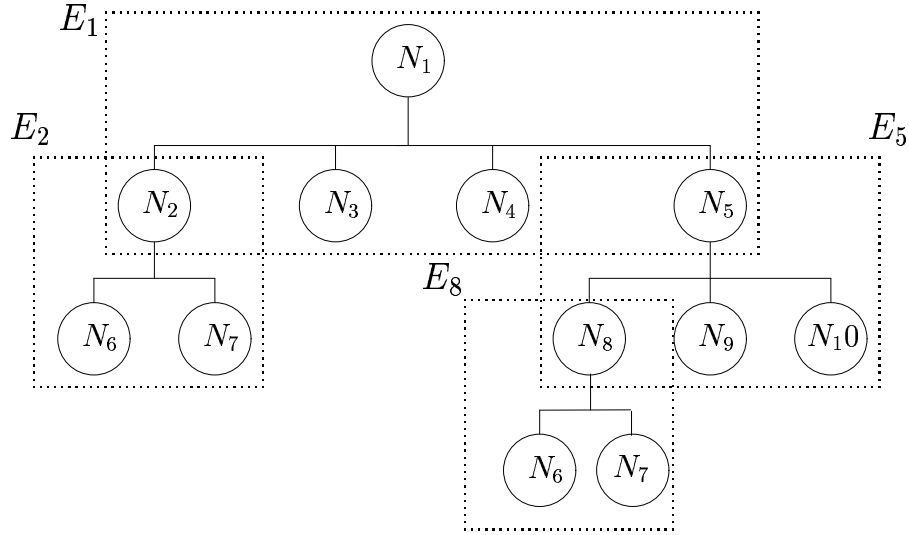
Figure 3.1.: Definition of decision nodes in a hierarchy. As it can be seen, decision nodes overlap. Note that N_8 and N_2 have the same childs, indicating a directed acyclic graph as hierarchical structure.

- One vs. rest approach

- Learning multi-label classification directly

The one vs. rest approach is implemented by using linear classifiers learned through the CentroidBooster algorithm (see section 2.2.2.5). Direct learning of multi-label classification is done by AdaBoost.MH. The hypothesis constructed by AdaBoost.MH can be viewed as linear classification hypothesis (see section 2.2.2.4). Since AdaBoost.MH differs only in the learning of the linear classifier, categorizing new unseen examples is the same for CentroidBooster and AdaBoost.MH. For the rest of this section it is assumed, that a decision node classifier $h^k(\vec{d})$ can be divided into n classifiers $h_i^k(\vec{d}) \rightarrow \{-1, 1\}$, one for each node $N_i^k \in E^k$. Each node classifier is assigned a weight vector $\vec{w}_i^k$, and by assuming a threshold $\Theta = 0$, the classifier outputs 1 if $\vec{w}_i^k \cdot \vec{d} > 0$ and -1 otherwise[1].

Given a decision node E^k, a document $\vec{d}$ is assigned to a node N_i^k (and therefore to a class), if $h_i^k = 1$ and N_i is a leaf node of E^k. For considering the assignment of documents to the root of a decision node, a virtual leaf is added to each decision node. This virtual leaf is equivalent to the root node N_k. If N_i^k is an inner node (and therefore the root node of a decision node E^i), the document is propagated further down if $h_i^k = 1$. For $h_i^k = -1$ the document is not assigned to N_i^k or any children $\{N_j | N_i^k \dot{\rightarrow} N_j\}$ via the path containing N_i^k. Note that children $\{N_j | N_i^k \dot{\rightarrow} N_j\}$ may be selected via another path of the hierarchy in the case of a directed acyclic graph. Figure 3.2 illustrates classification within a decision node using one vs. rest linear classifiers.

Starting at the root node, classification for a branch stops if

- no node N_i^k is selected (i.e. no classifier outputs 1).

- node N_i^k is a leaf node of $\mathcal{H}$ and $h_i^k(\vec{d}) = 1$

---

[1] For AdaBoost.MH, binary input vectors are assumed

E^1
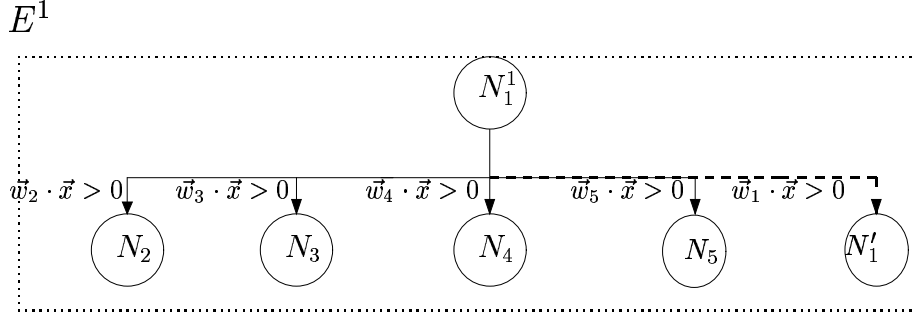
Figure 3.2.: Classification model in a decision node E^k. A linear classifier w_i is used as decision function for each node N_i. For considering the assignment of documents to the root of a decision node, a virtual leaf is added to each decision node (indicated by the dashed line). This virtual leaf is equivalent to the root N_k of the decision node.

Classification for the whole hierarchy stops, if classification within all decision nodes is stopped.

Thus, classification $h_{\mathcal{H}} : \boldsymbol{D} \to \boldsymbol{C}$ of a document D_j into a hierarchical structure $\mathcal{H}$ returns a set of classes denoted $\boldsymbol{R}_j$.[2] Ideally this set of classes coincide with the correct classes $\{C_k \mid \Phi(D_j, C_k) = T\}$ of a document D_j. This should be achieved by the node classifiers, which are responsible for correctly rejecting documents not belonging to a class and/or a sub hierarchy.

3.2. Confidence of a decision

Providing users with proposals of classes a document belongs to (see section 1.3.2) is one application of hierarchical text classification. All correct classes of a document should be returned yielding to a recall close to 1. Increasing recall will decrease precision and the user is provided with a large list of possible classes. Thus it would be appropriate, if most of the correct classes are ranked top, reducing the time for users browsing through the list.

So the point of interest is to obtain a ranking of the returned classes in a way that correct classes appear first in the result list, which can be achieved by estimating a confidence for the classification decision. Since assigning a class is based on one or more independent decisions, an overall confidence for all decisions has to be calculated.

Each decision node E^k outputs a set of nodes (which are equivalent to classes) with a corresponding confidence or score. Linear classifiers are used in each decision node E^k. For linear classifiers the distance of a document D_j to the separating hyperplane indicates the confidence of a decision. Larger distances reflects higher confidence, small distances lower confidence.

Mapping the output range of a linear classifier from $[-1, 1]$ to $[0, 1]$ is done by applying a sigmoid function to the output of the classifier. So the score is defined as

$$score_{N_i^k}(\vec{d_j}) = \frac{1}{1 + \exp\left(-h_i^k(\vec{d})\right)}$$

Applying a sigmoid function to the output of a classifier has the impact that changes in the distance which are closer to the hyperplane lead to larger changes in the confidence than changes in the distance which are farther away.

---

[2]since each node N_k is assigned a class C_k the classified nodes yield to a set of classes.
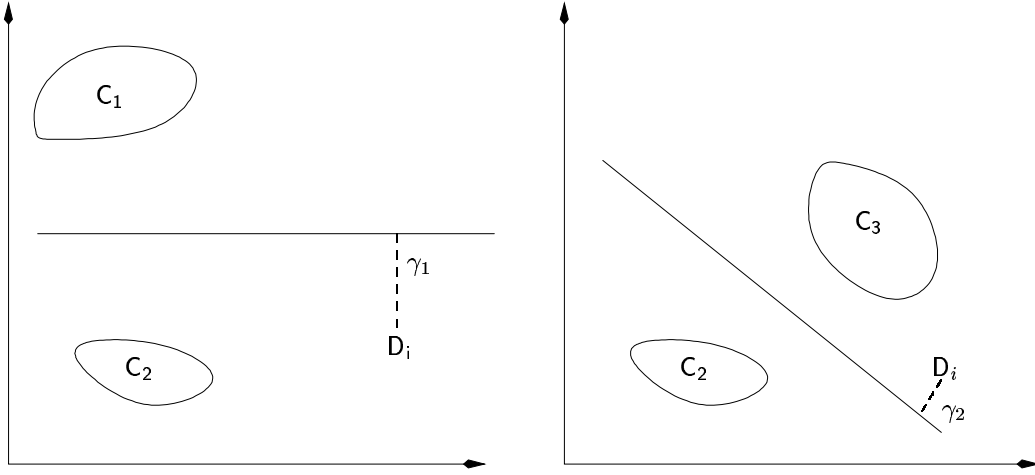
Figure 3.3.: The classes C_1, C_2 (left) and C_2, C_3 (right) are trained independently of each other. Since margin γ_1 is higher than margin γ_2, ranking based on the distance to the classifier is higher for class C_2 than for class C_3. So if C_3 is the correct class, the decision is wrong.

So let us define $\boldsymbol{R}_j = \{C_k \mid \tilde{\Phi}(D_j, C_k) = T\}$ as the set of classes for document D_j returned by the hierarchical classifier and let $path_{C_i}$ be the set of nodes which have been traversed to reach class C_i. Assuming that the probability $Pr[N_i^k \mid D_j]$ for a node given document D_j is reflected by $score_{N_i^k}(\vec{d}_j)$ and that the probability is independent from other decision nodes, then the score for assigning document D_j to class C_i is calculated as

$$score_{\mathcal{H}}(C_i, D_j) = \prod_{N_s^k \in path_{C_i}} Pr[N_s^k \mid D_j] = \prod_{N_s^k \in path_{C_i}} score_{N_i^k}(\vec{d}_j)$$

In words, the score for a class C_i over the hierarchy is the product of probabilities for all classifiers along the classification path. So if a node classifier h_i^k rejects a document this can be viewed as assigning a confidence/score $score_{N_i^k}(\vec{d}_j) = 0$, thereby reducing the overall $score_{\mathcal{H}}$ to zero.

Deriving (linear) classifiers which output a reasonable probability $Pr[N_i^k \mid D_j]$ for a node must be achieved to obtain good scoring values with the above definition of $score_{\mathcal{H}}$. The problem at hand is, that each decision node is trained independent from the others. Thereby, the complexity of a learning problem may differ between decision nodes (e.g. different margins between classes, non linear separability for linear classifiers, different amount of training examples etc.).

For illustrating the problem of independently trained classifiers, consider the following example. Let us assume two decision nodes N^1 and N^2 are trained independent from each other. Node N^1 consists of two classes, which are linear separable and node N^2 consists of ten classes, which are not linear separable. Furthermore, document D_i should originally belong to a class C_j in decision node N^2 but due to hierarchical classification it is also wrongly propagated to node N^1. So for obtaining a ranked list of classes, the distances of the linear classifier in node N^1 has to be compared to that of node N^2. Since N^1 has been never trained on positive examples of C_j (where document D_i belongs to), the output of the linear classifier will be arbitrary with respect to C_j. Also, it is likely that the distance to the separating hyperplane is large, since the training data in node N^1 has a large margin on average (due to linear separability) in contrast to training data in node N^2. Thus, it is likely that the wrongly chosen class in N^1 obtains a higher ranking than the correct class of node N^2. Figure 3.3 illustrates this problem. Thereby, document D_i is originally assigned to class C_3, but since the
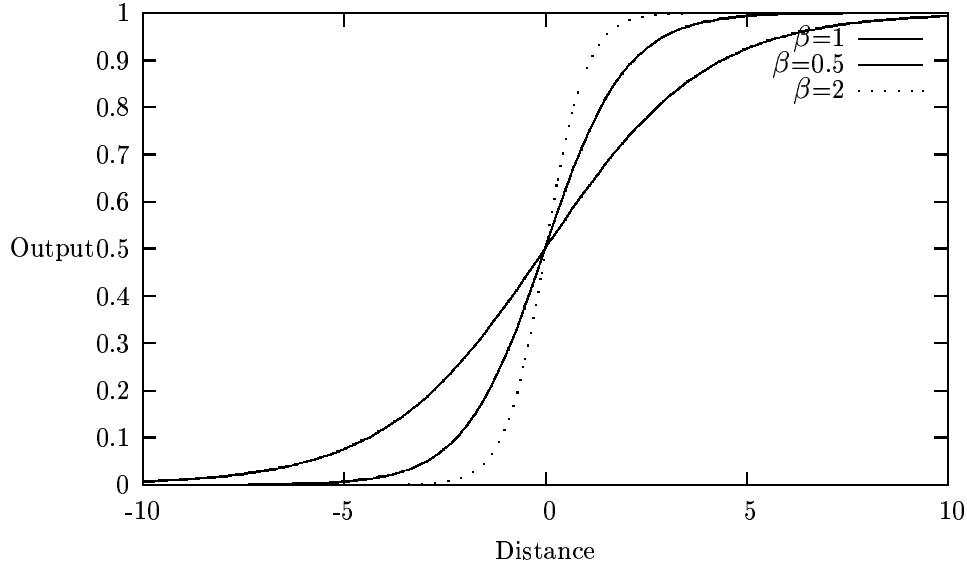
Figure 3.4.: Sigmoid functions with different β. It can be seen that for larger values of β the sigmoid function is steeper.

ranking is based on the margin from the separating hyperplane and the classifier for C_1 vs. C_2 yields a larger margin (which can be viewed as easier classification problem) than C_3 vs. C_2, the ranking for class C_2 is higher than for class C_3.

So, if ranking is based on the distance to the separating hyperplane, a mechanism for comparing independently trained classifiers has to be applied. The model proposed here, adds a regularization parameter β to the sigmoid function, namely

$$score_{N_i^k}(\vec{d_j}) = \frac{1}{1 + \exp\left(-\beta_i^k h_i^k(\vec{d})\right)}$$

Parameter β determines the steepness of the sigmoid function and reflects the influence of the distance to the score. A low β indicates lesser influence of changes in the distance close to the hyperplane. A high beta indicates higher influence of changes in the distance close to the hyperplane and lesser influence of changes in the distance far away from the hyperplane. Figure 3.4 illustrates this circumstance.

Calculating β_i^k for classifier i in decision node k is done on a validation set. Therefore, the average margin $\overline{\gamma}_i^k$ of examples on the validation set is calculated. Additionally, the error ϵ_i^k of a classifier on the validation set is calculated. Then β_i^k is estimated by

$$\beta_i^k = \frac{1}{\overline{\gamma}_i^k} \log \frac{1 - \epsilon_i^k}{\epsilon_i^k}$$

Given this equation, β_i^k is higher if the average margin on the validation set is low, making distances of independent classifiers comparable. Additionally β_i^k is higher if the classifier has a low error on the validation set, which biases toward classes having a low generalization error.

3.3. Learning the Classification Hypothesis

Till now only classification of new examples was considered. This section introduces how the training and validation set is selected. Let us recall from before that learning a hierarchy $\mathcal{H}$ is subdivided in learning classifiers for each decision node $N_r^k \in \mathcal{H}$. So the learning process is divided into the following 3 steps, considering a split of the training examples of a hierarchy $\mathcal{H}$ into a training set $\boldsymbol{T}^{\mathcal{H}}$ and a validation set $\boldsymbol{V}^{\mathcal{H}}$:

1. Calculate weight vector on the training set $\boldsymbol{T}^{\mathcal{H}}$ for each node N_r^k of a decision node E^k.

2. Evaluate parameters to obtain the score for each node N_r^k of a decision node E^k on the validation set $\boldsymbol{V}^{\mathcal{H}}$.

3. Relearn the weight vector in each node N_r^k of a decision node E^k by using training and validation set $\boldsymbol{T}^{\mathcal{H}} \cup \boldsymbol{V}^{\mathcal{H}}$.

Step 1 learns a classifier for each node on the training set $\boldsymbol{T}^{\mathcal{H}}$, yielding to an approximation of the final classification model. This approximated classification model is used for evaluation (step 2) of the score parameters. This evaluation is based on $\boldsymbol{V}^{\mathcal{H}}$. Afterward the whole model is retrained using test and validation set for obtaining the final classification model. Due to the lack of (positive) training examples, the model has to be retrained for obtaining a optimal classification hypothesis.

The training set $\boldsymbol{T}_i^k$ for a node N_i^k, are all documents assigned to any child node $\{N_j \mid N_k^k \dot{\to} N_j\}$ of root node N_k^k (of decision node E^k). Positive examples for node N_i^k are those which are assigned to node N_i^k and all children $\{N_j \mid N_i^k \dot{\to} N_j\}$ of node N_i^{k3}. So if $\boldsymbol{T}_j = \{\vec{d_l} \mid \vec{d_l} \in N_j\}$ is the set of training examples for node N_j (and thereby class C_j), the training set for a decision node E^k can be defined as

$$\boldsymbol{T^k} = \bigcup_{\{N_i \mid N_k^k \dot{\to} N_i\}} \boldsymbol{T}_i$$

whereby the positive training examples $\boldsymbol{T}_j^{k,+}$ of a node N_j^k are defined as

$$\boldsymbol{T}_j^{k,+} = \boldsymbol{T^k} / \boldsymbol{T_j^{k,-}}$$
$$\boldsymbol{T_j^{k,-}} = \bigcup_{\forall N_i^k \neq N_j^k} \{\boldsymbol{T}_l \mid N_i^k \dot{\to} N_l\}$$

The validation set $\boldsymbol{V}$ is defined in the same way as the training set. Note thtat training set and validation set are independent of each other, $\boldsymbol{T}^{\mathcal{H}} \cap \boldsymbol{V}^{\mathcal{H}} = \emptyset$.

Robust Training Set: As stated before, error propagation is a problem concerning hierarchical text classification. If a wrong classification is made in top decision nodes, a document may be assigned to a class in the according sub hierarchy. Thus, one has to deal with wrongly propagated documents along the decision path. Within this thesis this problem is overcome by selecting a robust training set, so that wrongly propagated documents are rejected in a decision nodes of the sub hierarchy.

---

[3]Note that each decision node is extended by a virtual node as substitution for the root node.
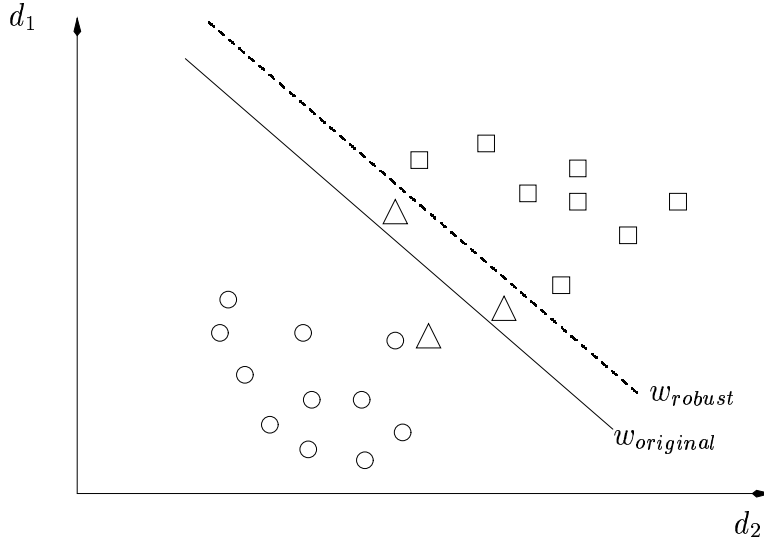
Figure 3.5.: An example for the impact of robust training set selection. Circles indicate negative examples and squares positive ones. Triangles are those examples, selected from the parent node. The dotted hyperplane is obtained by incorporating those robust training examples.

Let us assume E^j is the parent decision node of E^k and that decision node E^j propagates a document wrongly into decision node E^k. If all nodes within E^k are trained with negative examples taken from the training set $\boldsymbol{T}_k^{j,-}$, the probability that wrongly propagated documents will be rejected increases. Since the training set $\boldsymbol{T}^k$ contains all examples from the sub hierarchy defined by E^k, additionally taking all examples from the parent decision node E^j would yield to large number of training examples making computation expensive.

This problem can be overcome by applying query zone selection to the training examples of E^j. The following steps illustrate the selection of robust training examples for a decision node E^k with parent E^j:

1. Train decision node E^j by using training set $\boldsymbol{T}^j$

2. Classify $\boldsymbol{T}^j$ into E^j by using the learned classifiers from step 1

3. Take the n examples from the negative training set of node N_k^j, which have the lowest confidence for not belonging to N_k^j. This set is denoted by $\boldsymbol{T}^{j,robust} \subset \boldsymbol{T}_k^{j,-}$

4. Use $\boldsymbol{T}^{j,robust}$ as negative training examples for all nodes within E^k

Using robust training set selection, those examples which are likely to be propagated wrongly into decision node E^k are used as negative examples in E^k, probably yielding to a rejection of wrongly propagated documents.

3.4. Related Work

As stated in the introduction, today's methods for text classification are usually based on a flat model. Recently, some approaches for hierarchical text classification, overcoming the restrictions of flat models, were introduced. These approaches are based on classification and indexing methods as described

Author	Hierarchical Model	Classification Method	Datasets
Dumais et. al.[21]	top down	SVM	LookSmart Web Directory
Sun and Lim [73]	top down	SVM	Reuters 21578
Ruiz and Srinivasan [61]	top down	Neural Networks (HME), Rocchio	Ohsumed
Ng et al.[50]	top down	Perceptron	Reuters 22173 (modified)
Koller and Sahami [38]	top down, single label	Baysean Network	Reuters 22173 (modified)
Dhillon et al [16]	top down	Naive Bayes and Word Clustering	Demoz Web Directory
McCallum et al [44]	big bang	Naive Bayes and Shrinkage	Company Web Pages, Newsgroups and Yahoo
Mladenic [47]	top down	Naive Bayes	Yahoo

Table 3.1.: Related work in hierarchical text classification.

above. This section provides an overview on how indexing and classification methods were combined and which results were obtained.

The used classification model depends basically on the hierarchical structure of the given training data and on the document assignment to nodes of the structure. These properties, which are closely related to those given in [73], are outlined in section 1.3.

In literature two different learning models are reported. The first model follows a *top-down* strategy for classification. It is the model used in this thesis which was introduced above.The second approach is to construct a flat model of classifiers. Different to the normal flat classification models, hierarchical dependencies are taken into account for estimating parameters of the learning algorithm. Sun [73] referred to this approach as the *big bang*. Most work currently done in hierarchical text classification focuses on top-down strategies, which will be discussed first here. Table 3.1 gives an overview on related work and will be described in the rest of this section.

Dumais et al. [21] trained a 1 vs. m linear SVM for each parent node (starting at the root node). As test hierarchy a tree structure with depth 2 of web documents taken from the LookSmart web directory was used. It consists of 13 top level classes and 150 second level classes with documents assigned only to the second level classes. Two different scoring types were used. A boolean scoring function assigns a document to a category if the top level category and the second level category exceed a given threshold. The second scoring type multiplies probabilities of the first and second level category decision. If the resulting score exceeds a threshold, the document is assigned to a category. The probabilities were obtained by summing the weights for features present in the document and applying a sigmoid function to this sum. For comparing results between different categories the sigmoid function is fitted to the output of a SVM using a regularized maximum likelihood fitting. Thus, the SVM produces posteriori probabilities which are suitable for comparison. Hierarchical classification yields an advantage of about 4% (regarding F_1 values) compared to the flat approach.

A SVM was also used by Sun and Lim [73]. Their setting differs from that of Dumais in allowing document assignments also to inner nodes. They applied two different classifier types for each node, a sub tree classifier and a local classifier. Additionally two novel performance measures, tak-

ing class relationships into account, were introduced. The first measurement is based on the cosine similarity between classes. Here the problem is, that performance is measured within the same term space where classification takes place and the performance measurement is biased toward the used indexing methods. Distances between classes in the hierarchical structure is the second performance measure used. Thus, the edges of the graph between the correct and the assigned class are counted as performance measure. Experiments are performed on the Reuters 21578, but the hierarchical model was not compared to the flat model.

A similar classification model was proposed by Ruiz and Srinivasan [61]. A modified Hierarchical Mixture of Experts model was used for hierarchical text classification. Hierarchical Mixture of Experts is a Neural Network (as introduced in [37]) consisting of gating and expert networks. Inner nodes are consisting of expert and gating networks, leaf nodes have only expert networks assigned. Within each inner node the expert network decides if the document belongs to the current node and the gating networks decides if a concept from a child is present within the document. If so, the document is further propagated top down. Comparison between categories of different branches is implicitly given through the definition of the hierarchical mixture model. At each inner node a Softmax function smooths the output for the next layer. Since the performance (in terms of accuracy and speed) of Neural Networks is sensitive to the feature space size, two term selection methods, namely Mutual Information and Odds Ratio were applied and evaluated. Beneath these term selection methods query zoning is used for training set selection (see 2.2.1.2). Like stated in [56], query zoning improves classification in training speed and accuracy. Evaluation was done on different subsets of the OHSUMED corpus. Results indicate an improvement of hierarchical compared to flat models in most cases. Improvements are ranging between 3% and 8% in terms of macroaveraged F_1 values. Similar to Ruiz and Srinivasan, Wiener et al. [22] also introduced the use of Hierarchical Mixture of Experts in hierarchical text classification. Different to Ruiz, they used only a model covering a two level hierarchy. Additionally, no query zoning mechanism was applied.

Ng et al.[50] build their hierarchical classifier using Perceptrons and distinguishing between leaf nodes and non-leaf nodes. Again, a top down approach is used with the output of the classifier being the final set of leaf nodes reached in the recursion (zero, one or more). Feature selection is based on the correlation coefficient, χ^2 and manually. Also, as Ruiz and Srinivasan, Ng et al. performed training set selection by only selecting the 200 most relevant negative examples for a category. Their experiments were performed on the Reuters 22173 collection. Results indicate good performance of the classifier, whereby the best results where obtained by using manually feature selection.

Both, the SVM approach ([21] ,[73]) and NN ([61],[50]) approach allow multi label classification, since more than one branch may be taken at each parent node. Different to this, Koller and Sahami [38] proposed a hierarchical approach that greedily selects the best child class (winner takes it all approach). Thus, their method selects a single path and assigns all the categories on the path to the document. The hierarchical structure is used as a filter that activates only a single best classification path. Also errors in classification at the higher levels are irrecoverable in the lower levels. As classification method the so called *KDB* algorithm was used, whereby feature selection is based on cross entropy between categories. The KDB algorithm is able to learn a Bayesian Network where the features of a class depend upon each other in a hierarchical manner. To keep learning computational efficient, the KDB algorithm applies a heuristic based on class conditional mutual information. In their experiments a modified Reuters 22173 collection was used. Two different hierarchies resulted from their modification. Both hierarchies are of 2-level depth and had 9 and 6 classes. The training set size was about 700 examples, which is a drawback in their experiments. Nevertheless, it was shown that the hierarchical classifier outperforms the flat one and that a fewer number of features are needed in the former case.

The work of Dhillon et al. [16] focused on the usage of word clustering for hierarchical text

classification. They reduced the given term space by divisive clustering and compared the word clustering to agglomerative clustering and other feature selection approaches like mutual information and χ^2. In addition, they used their divisive clustering for hierarchical classification of Demoz web directory data. As hierarchical classification approach a naive Bayes classifier with an underlying multinomial distribution was used. Each inner node was assigned a naive Bayes classifier using the documents of the child classes as training documents. Classification is again performed by using a top down strategy and stops if a leaf node is reached. One drawback of their experiments is that they used accuracy as performance measure and that they performed random splitting as evaluation strategy. They report an increase in accuracy of about 6% for the hierarchical model. Additionally, a smaller number of features are needed in the hierarchical model.

McCallum et al proposed a big bang approach. They [44] exploit the hierarchical model for parameter estimation, enhancing single node classifiers. A Bayesian classifiers and a statistical technique called shrinkage for improving accuracy are used. First, parameters for each node are calculated using a standard Naive Bayes method. Afterwards, shrinkage smooths parameter estimates of a child node with its ancestors in order to obtain more robust estimates. Testing on the training set is performed by evaluating posterior probabilities of each class and selecting the class having the highest probability. Their experiments were conducted using three different data sets. One contains hierarchical (by industry code) ordered company web pages, the second dataset was a self made hierarchy obtained from 20 Newsgroups and the third was the Yahoo Science hierarchy. Their experiments showed, that, if the training data is sparse, shrinkage improves classification accuracy by up to 29%.

Similar to McCallum, Mladenic [47] approached hierarchical classification by building Bayesian classifiers for each node in the hierarchy. Positive examples of a node are positive examples for this node and all descendants, weighted according to the position in the hierarchy. These examples were used for estimating the parameters of the Bayesian classifiers. The classification process works by evaluating the posterior probabilities of each class. Classes for which classifiers predict a probability greater than 0.95 are assigned. Mladenic's main effort is in creating a weighting scheme for combining probabilities obtained at different nodes of the tree. Experiments were performed on the Yahoo Science hierarchy. No comparison to flat classification was performed.

4. Experiments and Results

This section covers the experimental settings and results for the introduced algorithms. A major point of interest lies in the comparison of the hierarchical to the flat text classification in terms of accuracy, learning and classification time. As stated in 1.3.2, including hierarchical structures should decrease learning and classification time. On the other hand, accuracy should increase on hierarchical data sets, since more information is available. Impacts of robust training set selection, as well as the performance of Boosting compared to the base line classifiers are evaluated. At last, ranking performance of the introduced scoring model is evaluated.

The section starts with describing algorithm parameters and preprocessing methods, as well as the used performance measures (taken from section 2.3). Afterwards test methods and datasets are outlined. This chapter concludes with detailed experimental results.

4.1. Parameters, Algorithms and Indexing Methods

Three different algorithms are evaluated by experiments, namely SVM, Rocchio and Boosting. Furthermore, three different weak classifiers are used in the Boosting approach, whereby two weak classifiers are well known in text classification and one weak classifier is fairly new. SVM's and Rocchio are used as base line classifiers. Rocchio is used only on flat data sets. No robust training set selection is performed for both. This section introduces the used parameters for these algorithms and describes the abbreviations used in the result tables. The section concludes with the indexing methods used in the experiments.

4.1.1. Boosting

Boosting algorithms are used in hierarchical and flat manner. For hierarchical classification robust training set selection is performed. Robust training set selection is given in percent of the positive examples of a class. This number of examples is added to the training set of a class as negative examples. The examples are taken from the set of negative parent training examples (see section 3.3).

BoosTexter: AdaBoost.MH with real valued predictions and AdaBoost.MH with real valued predictions and abstaining (see section 2.2.2.4) are used as weak classifiers. *BoosTexter.RV* refers to the former, *BoosTexter.RVA* to the later algorithm. Training was performed for 10,000 iterations.

CentroidBooster: *CentroidBooster* is used as third Boosting classifier. Since CentroidBooster is based on a stronger weak hypothesis, only 1,000 iterations were performed. Furthermore, experiments with and without replacing the main hypothesis by a linear separating weak classifier are performed. The former algorithm is named *CentroidBooster*, the later algorithm which replaces the main hypothesis is named *CentroidBooster.rep*.

4.1.2. Base Line Classifiers

Within this section two base line classifiers are outlined. Base line classifiers serve the task of comparing results obtained from the implemented algorithms to those obtained from current state of the art techniques. Support vector machines and Rocchio based classification are chosen as baseline classifiers.

Support vector machines (see section 2.2.1.1) are reliable classification approaches yielding to very good results in text classification. Additionally, a fast and efficient implementation exists which can be used freely. Thus, support vector machines serve as performance criterion for the implemented algorithms.

Rocchio on the other hand is a very simple classifier which can be computed in a small amount of time. Thus, comparing complex algorithms to Rocchio gives insight in the trade off between computational complexity and accuracy. Rocchio is only used in a flat setting, incorporating no hierarchical information.

The implementation of those algorithms is outlined in the next subsections. Robust training set selection were not evaluated for the base line algorithms.

4.1.2.1. Rocchio

Rocchio's classification approach was chosen as base line algorithm for comparing results obtained from Boosting. Calculating the prototype vector $\vec{c}_i$ for a class C_i is done only by using the positive examples of this class, which corresponds to setting $\gamma = 0$ (see section 2.2.1.2). All training examples are normed to unit length by using the euclidean norm. The resulting prototype vector is also known as *centroid* of a class and is the average vector of all documents D_j assigned to this class with respect to an euclidean space. For Rocchio, rejecting decisions for an example if it does not suite any class is not possible in this setting. Thus, for applying multi label classification and for incorporating reject decisions a threshold must be calculated, which was not done in this experiments.

4.1.2.2. SVM^{light}

Support vector machines (see section 2.2.1.1) are used as another base line algorithm. Since an efficient implementation is difficult and beyond the scope of this thesis the freely available SVM^{Light} package from Thorsten Joachims was used.

Different to the Rocchio approach, SVM^{Light} was also used in a hierarchical manner. Since the output of the SVM^{Light} package can not be seamlessly integrated in the framework for training and testing, no robust training set selection has been done in the experiments. Also, the computational complexity could not be measured in a fair manner due to implementation difficulties. Nevertheless, comparing results to support vector machine should reveal the performance of the above introduced algorithms.

4.1.3. Document Indexing

As described in section 2.1 several document indexing approaches exist. This section introduces the preprocessing approaches used in the experiments.

The lexical analysis is done by removing all sentence punctuation and special characters or tags (e.g. HTML, XML tags) from a document. The resulting text of a document corresponds to the bag of word approach mentioned in section 2.1. Lexical analysis precedes all other indexing steps.

After the lexical analysis, stopwords were removed from the document. The used stopword list is given in appendix A. Since the test data is in English, Porter's algorithm [55] is used for suffix stripping.

Performing feature selection is based on Zipf Mandelbrot's law. Terms not occurring within n (where n is set to 3 in the experiments) documents are discarded before learning. Since more sophisticated methods are not considered within this thesis, it has to be kept in mind that performance and accuracy of classifiers can be increased by good feature selection algorithms (see 2.1.3).

After having determined the terms for a document, TFIDF weighting is applied. TFIDF weighting is described in section 2.1.2.

4.2. Used Performance Measures

For measuring classification accuracy, different aspects of the implemented algorithms have to be investigated. First, classification performance of node classifiers is evaluated. As stated in section 2.3 the binary decision whether a document D_j belongs to a class C_i or not is evaluated by the F_1 measure.

Additionally, to estimate the performance of the used scoring model, ranking based performance measures are applied. For comparing ranking performance, a recall of 1 must be achieved by the hierarchical classifier.

4.2.1. Classification Performance

Classification performance is given as F_1 measure which is defined in 2.3.3. A classifier $h_{hierarchical}$ returns an ordered set of classes $\boldsymbol{R}_j \subset \boldsymbol{C}$ for an unknown document D_j. Each pair $< D_j, C_i >$, with $C_i \in \boldsymbol{R}_j$, is treated as positive binary decision of the classifier h_i on document D_j. All other pairs $< D_j, C_k >$, $C_k \notin \boldsymbol{R}_j$ are false decisions for classifier h_k. All documents $D_j \in \boldsymbol{D}_{test}$ within the test set are tested on the hierarchical classifier whereof the contingency table for each class is derived. Macroaveraging classes with no positive examples in the test set will give $F_1 = 0$ if a document is assigned to the class or $F_1 = 1$ if no documents are assigned to the class.

For macroaveraged performance values it has to be taken into account, that the algorithms (except Rocchio) are biased toward classes having a larger number of training examples. Thus, the macroaveraged F_1 value could be increased by considering the distribution of training examples over classes during learning. This was not done within this thesis.

4.2.2. Ranking Performance

Ranking performance should cover aspects of obtaining a good ranking for a given multi label problem. The applied mechanism for comparing decisions of independently trained classifiers can be evaluated by ranking performance measurements. Experiments using the adaptive sigmoid function should outperform others by achieving higher ranking performance.

Again the classification hypothesis $h_{hierarchical}(D_j)$ returns a ranked list of classes $\boldsymbol{R}_j \subset \boldsymbol{C}$. Note that the F_1 measurement is independent from the order of the returned list. The goodness of ranking is evaluated by:

- one-error

  As described in 2.3.3 the one-error is defined as average error of top ranked documents, i.e. how often a correct document is not ranked first.

- coverage:

  Coverage is used as defined in section 2.3.3. For comparing coverage a recall of 1 has to be achieved. Therefor, in experiments evaluating coverage, the threshold within each classification node is set to zero.

- average precision:

  Average precision is defined as in section 2.3.3. Equal to coverage, all correct classes must be returned by the classifier. Thus, no thresholds are applied in test runs evaluating this measure.

For evaluating coverage and average precision, recall has to be 1. This is achieved by ignoring the threshold of each node classifier. Since this is not suitable for an application, threshold adaptation should be performed on a validation set. Threshold adaptation was not applied in this thesis, since it is beyond the scope.

4.2.3. Comparing Experiments

Comparing different experiments is done using statistical significance tests as proposed in [79]. Two different test categories are distinguished. A comparing proportions test is used for testing one error, microaveraged precision and microaveraged recall. Macroaveraged F_1 values are tested by using a macro sign test, macro t-test and a macro t-test after rank transformation. These tests and their application are outlined in this section. For a detailed discussion see [79].

Comparing Proportions (p-test): The performance scores (e.g. precision, recall and error) of two systems A and B are compared as follows:

- p_a is the performance score of System A

- p_b is the performance score of System B

- n_a and n_b are the number of trials in the two samples that are used for evaluating system A and B. The definition of n_a and n_b depends on the used performance measures:

  - for recall, it is the number of documents assigned to categories
  - for precision, it is the number of documents assigned to categories by a system
  - for error, it is the number of category-document pairs

- Compute $p = \frac{n_a * p_a + n_b * p_b}{n_a + n_b}$, the observed proportion of the total of $n = n_a + n_b$ trials

The null hypothesis is $p_a = p_b = p$. The alternative hypothesis is $p_a > p_b$. The 1-sided P-Value is computed as

$$Z = \frac{p_a - p_b}{\sqrt{p(1 - p)(1/n_a + 1/n_b)}}$$

Macro Sign Test (S-test): This test compares two systems (A and B) based on the paired F_1 value for each category. Thereby

- $a_i \in [0, 1]$ is the F_1 value of system A on the category $\mathsf{C_i}$, $i = 1 \ldots |\mathbf{C}|$.

- $b_i \in [0, 1]$ is the F_1 value of system B on the category $\mathsf{C_i}$, $i = 1 \ldots |\mathbf{C}|$.

- n is the number of times that a_i and b_i differ

- k is the number of times that $a_i > b_i$

The null hypothesis is $k = 0.5n$, the alternative hypothesis is that k has a binomial distribution of $B(n, p)$ where $p > 0.5$, meaning that system A is better than system B. If $n > 12$, the 1-sided P-value can be approximated by using the standard normal distribution for

$$Z = \frac{k - 0.5n}{0.5\sqrt{n}}$$

Macro t-Test (T-test): A t-test is used for comparing the paired F_1 values for individual categories. The same notation as defined in the S-test is used. Additionally

- $d_i = a_i - b_i$ is the difference of a_i and b_i

- $\overline{d}$ is the average of the d_i values for $i = 1 \ldots n$

The null hypothesis is $\overline{d} = 0$, the alternative hypothesis is $\overline{d} > 0$. For $n < 40$ the P-value is computed using the t-distribution with the degree of freedom of $n - 1$ for

$$T \geq \frac{\overline{d}}{s.e.(\overline{d})}.$$

where $s.e.(\overline{d}) = \sqrt{\frac{\sigma^2}{n}}$ is the standard error.

Macro t-Test after rank transformation (R-test): A rank transformation is performed on the F_1 values of system A and B before applying a t-test. The F_1 values of both systems are pooled together and sorted. The following notation is used on the rank transformed values:

- a'_i is the rank of the F_1 score of system A on the ith category

- b'_i is the rank of the F_1 score of system B on the ith category

- $d'_i = a'_i - b'_i$ is the rank difference on the ith category

- $\overline{d'}$ is the average of the d'_i values for $i = 1 \ldots n$

- n is the number of times where $a'_i \neq b'_i$

The null hypothesis is $\overline{d'} = 0$. The alternative hypothesis is $\overline{d'} > 0$ and the 1-sided P-value is calculated as

$$T \geq \frac{\overline{d'}}{s.e.(\overline{d'})}.$$

As stated in [79] the macro tests have different properties regarding robustness and sensitivity for outliers. Thus, the macro tests are combined for obtaining reliable results.

Appendix B.2 summarizes the results of statistical testing over all experiments. Testing is performed for significance levels of $\alpha = 0.005$, $\alpha = 0.01$ and $\alpha = 0.05$. The following notation is used indicating significance in tables:

- $\sim$ indicates no significant differences

- $\ggg$ indicates that the row algorithm achieves a significantly higher measurement then the column algorithm. The significance level α is 0.005

- $\gg$ indicates that the row algorithm achieves a significantly higher measurement then the column algorithm. The significance level α is 0.01

- $>$ indicates that the row algorithm achieves a significantly higher measurement then the column algorithm. The significance level α is 0.05

- $\lll$ indicates that the row algorithm achieves a significantly lower measurement then the column algorithm. The significance level α is 0.005

- $\ll$ indicates that the row algorithm achieves a significantly lower measurement then the column algorithm. The significance level α is 0.01

- $<$ indicates that the row algorithm achieves a significantly lower measurement then the column algorithm. The significance level α is 0.05

Significance testing is done for microaveraged error, microaveraged precision, microaveraged recall and macroaveraged F_1 values. For displaying the results, the following abbreviations are used:

- one error: *err.*

- precision: *prec.*

- recall: *rec.*

- S-Test for F_1 value: F_1 *S-Test*

- T-Test for F_1 value: F_1 *T-Test*

- R-Test for F_1 value: F_1 *R-Test*

4.3. Datasets

This section introduces the data sets used within the experiments. The standard Reuters 21578 is used for comparing text classification to results within literature. Evaluation of hierarchical classification performance is done by using the new Reuters Corpus Volume 1, where only a subset of this corpus is used. Additionally, experiments are performed on the well know Ohsumed data set. Details of these data sets are outlined in this section.

4.3.1. Reuters-21578

To provide an overview on the base line accuracy of the classifiers and for comparison with various studies, the Reuters 21578 corpus was taken.[1] (see [79] for a comparison of different classifiers on the Reuters 21578). Various splits of the Reuters 21578 can be used, whereas the ModeApte Split was chosen within this thesis. By using the ModeApte split comparisons to other studies are possible, especially for the SVM^{light}, where the precision/recall break-even point was published by Joachims.

---

[1]The Reuters-21578 test collection is available at http://www.daviddlewis.com/ressources/testcollections/reuters21578

The Mode Apte split of the Reuters 21578 corpus results in 9602 trainings and 3299 test documents. From these split, all categories (including the documents not assigned to any category) which have no training or test document were deleted. The resulting data set has 90 different categories and is the same as used by Joachims in [35].

In [65] ranking performance of AdaBoost.MH was evaluated on a subset of the Reuters 21578. For comparing ranking performance of the implemented algorithms within these experiments, the same subset was used. Therefore classes which include at least 100 articles were chosen, yielding to a subset of 19 different classes. This split is referred to as "Reuters 19". Experiments were performed with subsets of growing example size for the 3, 10 and 19 largest classes. A detailed description of the data set is given in appendix B.1, table B.1. Experiments on this corpus are performed by using 3-folded cross validation.

4.3.2. Reuters Corpus Volume 1

The Reuters Corpus Volume 1 (RCV 1) is an archive of 806,791 English language news stories and can be obtained from http://about.reuters.com/researchandstandards/corpus. It includes all English language stories produced by Reuters journalists between 20/8/1996 and 19/8/1997. The stories are formated using a consistent XML schema that is based on NewsML. The RCV 1 is organized by three different coding schemes.

- Topic Codes

- Industry Codes

- Region Codes

In [60] a more detailed analysis of the corpus is given. Since the implemented algorithms are not capable of handling 800,000 documents, a small subset was extracted containing about 28,000 training and 26,000 test documents. The split was obtained by taking all documents in the period between 21/8/96 and 31/8/96 as training set and all documents between 1/09/96 and 10/09/96 as test set.

The extracted hierarchy is organized using topic codes, it is about four levels deep and has four top nodes. Classes having no positive training examples were discarded leading to 102 classes, 38,050 documents and 55,226 labels. Thereof, 18,252 documents resp. 26,792 labels are in the test set. On average a document is assigned to 1.45 classes. For documents assigned to more than one class, the average path length between these classes is 3.29 on the training and 3.22 on the test set. The hierarchical structure of the data set is given in appendix B.3.

4.3.3. Ohsumed

The OHSUMED test collection was created by Hersh and colleagues at the Oregon Health Sciences University [34]. It is a subset of the MEDLINE database and consists of 348,566 documents from 270 medical journals in the period from 1987 to 1991. Only 233,445 documents contain title and abstract (the remaining documents contain only a title). The documents are manually indexed using the MESH (Medical Subject Headings) thesaurus, which consists of 18,000 categories whereof 14,321 categories are used in the OHSUMED corpus.

Since using all documents leads to a computational intractable data set only the "Heart Diseases" sub hierarchy of the MESH Thesaurus was used. This sub hierarchy has 119 categories. Documents

Algorithm	One Error	Microaveraged			Macroaveraged		
		Prec.	Rec.	F_1	Prec.	Rec.	F_1
SVM^{light}	13.04	0.948	0.794	*0.864*	0.586	0.368	*0.425*
Rocchio	23.83	0.848	0.648	*0.757*	0.615	0.549	*0.580*
CentroidBooster	15.53	0.885	0.771	*0.824*	0.662	0.472	*0.551*
CentroidBooster.rep	11.18	0.912	0.825	*0.866*	0.670	0.428	*0.561*
BoosTexter.RV	14.44	0.903	0.811	*0.855*	0.573	0.393	*0.466*
BoosTexter.RVA	14.24	0.859	0.792	*0.824*	0.448	0.352	*0.394*
Schapire [65] AdaBoost.MH	-	-	-	*0.853*	-	-	-
Schapire [65] kNN	-	-	-	*0.852*	-	-	-
Schapire [65] LLSF	-	-	-	*0.855*	-	-	-

Table 4.1.: Shows the microaveraged and macroaveraged F_1 results for the Reuters 21578 Mode Apte split.

in the period from 1987 to 1990 were used as training set and documents from 1991 as test set. Additionally, all categories having no positive training examples were discarded, leading to 102 categories, 16,592 documents and 23,669 labels. Therefrom, 5,263 labels are in the test set and 18,406 labels are in the training set. The test set contains 3,764 unique test documents. For documents assigned to more than one class, the average path length between these classes is 2.26 on the training and 2.39 on the test set.The hierarchical structure and the distribution of training/test examples and number of classes is shown in appendix B.4.

4.4. Results

This section shows the results obtained from experiments with the above mentioned data sets. First, the Reuters 21578 is evaluated for comparison of the implemented algorithms with those reported in literature. Afterwards, hierarchical classification performance is compared to flat classification performance on the RCV 1 and Ohsumed data set. Finally, the impact of robust training set selection and the computational complexity is evaluated.

4.4.1. Standard Data sets

Evaluation of the Reuters 21578 is done for comparing the obtained results with those reported in literature. From this the correctness of base line classifiers and the Boosting algorithm is shown. Table 4.1 shows the results. A document frequency threshold of 3 was used as feature selection method.

Comparing the implemented algorithms with literature, BoosTexter.RV achieves about the same microaveraged F_1 value as in [65]. SVM^{light} has a slightly higher F_1 value than the reported precision-recall breakeven point of 0.842, in [35][2]. One reason for this difference are different preprocessing methods, i.e. stopwords and stemming, used in this thesis. Nevertheless, both algorithms perform as well as in literature.

Regarding one-error, CentroidBooster.rep and SVM^{light} perform best, followed by BoosTexter.RV, BoosTexter.RVA and CentroidBooster. Rocchio has the highest one-error. The similar holds

---
[2]As stated in section 2.3, the precision recall breakeven point is alway less or equal to the F_1 measure

Algorithm	Flat Classification			Hierarchical Classification		
	One Error	Micro. F_1	Macro. F_1	One Error	Micro. F_1	Macro. F_1
BoosTexter.RV	35.86	0.661	0.449	33.90	0.686	0.477
BoosTexter.RVA	36.04	0.658	0.441	33.72	0.676	0.475
CentroidBooster	33.85	0.698	0.500	31.13	0.709	0.528
CentroidBooster.rep	33.69	0.698	0.502	31.08	0.709	0.528
SVM^{light}	34.14	0.698	0.473	30.11	0.722	0.509
Rocchio	46.96	0.538	0.423	-	-	-

Table 4.2.: Hierarchical vs. flat classification on the Reuters Corpus Volume 1.

for microaveraged F_1 values. Thereby, CentroidBooster.rep, SVM^{light} and BoosTexter.RV are achieving the highest scores. BoosTexter.RVA and CentroidBooster have a slightly lower F_1 score. Again, Rocchio attains the lowest microaveraged F_1 value.

The macroaveraged F_1 values for Rocchio is the highest, followed by CentroidBooster.rep and CentroidBooster. BoosTexter.RV and SVM^{light} have a significant lower macroaveraged F_1 value and BoosTexter.RVA performs worst. For Rocchio it has to be taken into account, that no negative examples are considered during training of a class. Thus, Rocchio is not biased toward classes having a larger number of training examples and therefor achieves a higher macroaveraged value.

As conclusion it can be stated, that CentroidBooster.rep performs quite well on micro- and macroaveraging. Results for one-error and microaveraging are comparable to Support Vector Machines. Additionally, CentroidBooster.rep achieves a significant higher macroaveraged F_1 value than SVM^{light}. The performance of BoosTexter and SVM^{light} are the same as stated and literature. Hence, the implementation of both seems to be correct.

4.4.2. Flat vs. Hierarchical

This section compares flat versus hierarchical classification on the Reuters Corpus Volume 1 and the Ohsumed data set. For all experiments, a document frequency threshold of 3 was used. Table 4.2 and 4.4 shows the experimental results. The Ohsumed results are compared to results given in [77] and [61]. Comparing flat to hierarchical classification by using statistical tests is summarized in table 4.5 and table 4.3, detailed results are given in appendix B.2.

RCV 1: By comparing results on the RCV 1 it can be seen that for all classifiers, the one error in the hierarchical case is significantly lower than in the flat case. Interestingly, hierarchical classification seems to decrease microaveraged precision. One explanation would be error propagation. Documents are propagated into the wrong sub hierarchy and wrongly assigned to a class in this sub-hierarchy. So, for hierarchical classification it is more likely that documents are assigned to wrong classes, thereby decreasing precision. On the other hand, microaveraged recall is increased significantly, canceling the decrease in precision. Moreover, the increase in recall is higher than the decrease in precision, resulting in a higher F_1 score. SVM^{light} achieves the highest increase in terms of microaveraged F_1 values due to hierarchical classification. The increase in microaveraged recall for the hierarchical case may be due to dividing a complex classification problem (flat case) into smaller, easier problems. Thus, in the hierarchical case it is more likely that classifiers achieve a higher margin on the separating hyperplane, allowing that the threshold for assigning a document to a class is exceeded more often.

RCV 1	Flat					
	Micro Testing			Macro Testing		
Hierarchical	error	prec.	rec.	F_1 S-Test	F_1 T-Test	F_1 R-Test
CentroidBooster Hierarchical	≪	≪	≫	≫	≫	≪
CentroidBooster.rep Hierarchical	≪	≪	≫	≫	>	≪
BoosTexter.RV Hierarchical	≪	≪	≫	≫	≫	≪
BoosTexter.RVA Hierarchical	≪	≪	≫	≫	≫	≪
SVM Hierarchical	≪	≪	≫	≫	≫	≪

Table 4.3.: Shows results from significance testing on the hierarchical and flat RCV 1 data set. No robust training set selection is performed. Note that the F_1 R-Test evaluates the ranking of classes regarding F_1 measurements. So < indicates that the row algorithm achieves a better ranking of classes and therefor performs better.

Regarding macroaveraged F_1 values, all 3 statistical tests indicate a significant increase for all algorithms. For CentroidBooster and CentroidBooster.rep the T-test shows, that the increase is not as significantly high as for the other algorithms. Comparing the macroaveraged F_1 values between algorithms on the flat data set, it can be seen that CentroidBooster and CentroidBooster.rep achieve already a significant higher macroaveraged F_1 value than all other algorithms. Thus, it can be assumed that CentroidBooster can not increase the F_1 value as much as other algorithms on the hierarchical data set.

Comparing all algorithms on the flat data set, Rocchio is outperformed by all other algorithms in all microaveraged measurements. For macroaveraged F_1 values the difference is slightly lower, but still significant. There are no differences between CentroidBooster and CentroidBooster.rep (which holds also for the hierarchical case). Thus, distinguishing both algorithms is skipped for the remaining analyze of RCV 1. SVM^{light} and CentroidBooster yield equal performance for microaveraged F_1 values, with the difference that SVM^{light} has a higher precision and CentroidBooster achieves a higher recall. As mentioned above, CentroidBooster outperforms all other algorithms regarding macroaveraging. BoosTexter.RV attains lower microaveraged and macroaveraged values than CentroidBooster and SVM^{light}.

On the hierarchical data set, SVM^{light} outperforms all other algorithms in terms of microaveraged precision/recall and one-error. Again, precision is higher for SVM^{light}, whereas recall is higher for CentroidBooster. As stated above that hierarchical classification increases recall and because CentroidBooster achieves a higher recall, CentroidBooster can not benefit that much from the hierarchical data set. Additionally, the difference between macroaveraged F_1 values for SVM^{light} and CentroidBooster decreases. Again, CentroidBooster can not benefit that much from the hierarchical classification as SVM^{light}. Even than, CentroidBooster has a significantly higher macroaveraged F_1 value than SVM^{light}. BoosTexter.RV and BoosTexter.RVA attain about the same micro and macro values. Thus, for the rest of the analysis the distinction between them is skipped. Comparing CentroidBooster and BoosTexter on the hierarchical data set shows, that the performance of BoosTexter gets closer to CentroidBooster for microaveraged values and stays about the same for macroaveraged values.

As conclusion for the RCV 1 data set it can be stated that SVM^{light} performs better in the case of microaveraged measurement than the other algorithms and benefits more from hierarchical classification than CentroidBooster. Nevertheless, the performance of CentroidBooster is comparable to SVM^{light} and exceeds it for macroaveraged measurements. CentroidBooster can not benefit that much from hierarchical data sets. BoosTexter results improve on the hierarchical data set, but can not

Algorithm	Flat Classification			Hierarchical Classification		
	One Error	Micro. F_1	Macro. F_1	One Error	Micro. F_1	Macro. F_1
BoosTexter.RV	47.32	0.593	0.436	46.50	0.607	0.476
BoosTexter.RVA	48.10	0.589	0.412	48.11	0.594	0.479
CentroidBooster	43.17	0.609	0.533	45.30	0.615	0.537
CentroidBooster.rep	43.30	0.608	0.541	45.91	0.611	0.541
SVM	45.40	0.632	0.503	41.55	0.652	0.523
Rocchio	48.35	0.537	0.492	-	-	-
LLSF [77]	-	-	-	-	-	0.55
ExpNet [77]	-	-	-	-	-	0.54
STR [77]	-	-	-	-	-	0.38
HME [61]	-	-	-	-	-	0.557
Flat NN [61]	-	-	0.564	-	-	-
Opt. Rocchio[61]	-	-	0.558	-	-	-

Table 4.4.: Hierarchical vs. flat classification on the Ohsumed Data Set.

Ohsumed	Flat					
	Micro Testing			Macro Testing		
Hierarchical	error	prec.	rec.	F_1 S-Test	F_1 T-Test	F_1 R-Test
CentroidBooster	>	~	~	~	~	<
CentroidBooster.rep	≫	~	~	~	~	~
BoosTexter.RV	~	<	≫	≫	>	<
BoosTexter.RVA						
SVM	≪	≪	≫	≫	≫	≪

Table 4.5.: Shows results from significance testing on the hierarchical and flat Ohsumed data set. The used algorithm is given in the first column of each row. Also, hierarchical experiments are assigned to rows, flat experiments are assigned to columns. A comparison between all algorithms for the flat and hierarchical case is given in the appendix. Note that the F_1 R-Test evaluates the ranking of classes regarding F_1 measurements. So < indicates that the row algorithm achieves a better ranking of classes and therefor performs better.

achieve the performance of CentroidBooster or SVM^{light}.

Ohsumed: CentroidBooster and CentroidBooster.rep have a significant higher one-error on the hierarchical data set than on the flat data set and both algorithms yield about the same results, with a minor advantage for CentroidBooster. Thus, they are not distinguished for the rest of this section. Microaveraged precision and recall are about the same for the flat and hierarchical case, which is also true for the macroaveraged F_1 values. So CentroidBooster can not benefit from the hierarchical structure. In contrast, the performance of SVM^{light} and BoosTexter increases in the hierarchical setting. As for RCV 1, microaveraged precision decreases and recall increases. The increase on the hierarchical data set is lower for BoosTexter than for SVM^{light}. The same holds for macroaveraged F_1 values.

Again, Rocchio performs worst compared to all other algorithms on the flat data set and SVM^{light}

yields the highest microaveraged results. As for the RCV 1 data set, microaveraged precision is significantly higher for SVM^{light} compared to all other algorithms. CentroidBooster has the lowest precision of all algorithms except of Rocchio. SVM^{light}'s microaveraged recall is significantly lower than the recall of CentroidBooster and equal to BoosTexter's recall. Also, CentroidBooster is best in terms of macroaveraged F_1 values, whereas the difference between SVM^{light} is not as large as for the RCV 1 data set. BoosTexter's macroaveraged F_1 value is lowest of all algorithms.

Comparing algorithms on the hierarchical data set, things are changing. As mentioned above, CentroidBooster does not benefit from the hierarchical structure and so microaveraged performance becomes equal to BoosTexter.RV. Again SVM^{light} outperforms all other algorithms on microaveraged performance measures. The high microaveraged F_1 value results from the high precision of SVM^{light}. Recall is comparable to that of BoosTexter.RV.

Results in literature are given only in terms of macroaveraged F_1 values. The performance of CentroidBooster is comparable to results reported for ExpNet in [77] and slightly lower than Neural Network classifier and the optimized Rocchio in [61]. Note that no attention was paid in performing good feature set selection. So it can be stated that CentroidBooster achieves a comparable performance to other classification methods. SVM^{light} is the second best classifier, but achieves a significant lower performance compared to literature. As pointed out before, SVM^{light} achieves good microaveraged performance, but is worse in the case of macroaveraging. For macroaveraging it has to be taken into account, that all algorithms are not tuned towards this error measure.

Summary: On both data sets the conclusion can be drawn, that using hierarchical classification methods improves microaveraged recall of a classifier and decreases precision due to error propagation. SVM^{light} achieves the highest microaveraged F_1 value, but lacks in terms of macroaveraging. Additionally, SVM^{light} benefits from hierarchical data sets more than the other algorithms. Also, BoosTexter benefits from hierarchical data sets. The smallest increase in performance is obtained by CentroidBooster, which is mainly due to CentroidBooster is already achieving a higher recall than the other algorithms. Additionally, CentroidBooster is best in attaining high macroaveraged recall values.

4.4.3. Robust Training Set Selection

This section evaluates the impact of robust training set selection (see section 3.1) on the RCV 1 and the Ohsumed data set. The number of negative examples added to the training set of a node is given in percent of the number of positive examples for this node. The examples are taken from the negative training set of the parent node. Percentages used are 0%, 33%, 66% and 100%.

Table 4.7 summarizes the results for the Ohsumed data set and table 4.6 outlines results for the RCV 1 corpus. Results of significance testing are given in appendix B.2, table B.5 and table B.9.

RCV 1: For CentroidBooster and CentroidBooster.rep there are no differences by using robust training set selection. Statistical testing reveals a $\alpha = 0.05$ significant increase in microaveraged precision for CentroidBooster using 66% and 100% robust training examples. This has to be taken with care, since the significance level is very low and because the same absolute precision value is reached by using 33% robust boosting. Thus, the significance is on the edge. The same holds for the macroaveraged F_1 values, where only the S-test, which may be insensitive in performance comparison, indicates significant differences at a level of $\alpha = 0.05$. Thus it follows, that robust training set selection has no impact on CentroidBooster.

For BoosTexter.RV and BoosTexter.RVA, robust training set selection increases microaveraged precision significantly as robust training set selection should do. But on the other hand, recall is

Algorithm	Robust [%]	One Error	Microaveraged			Macroaveraged		
			Prec.	Rec.	F_1	Prec.	Rec.	F_1
CentroidBooster	0	*31.13*	0.789	0.643	*0.709*	0.642	0.449	*0.528*
CentroidBooster	33	*31.30*	0.795	0.641	*0.710*	0.647	0.446	*0.528*
CentroidBooster	66	*31.33*	0.795	0.640	*0.709*	0.647	0.445	*0.528*
CentroidBooster	100	*31.35*	0.796	0.640	*0.710*	0.647	0.445	*0.527*
CentroidBooster.rep	0	*31.08*	0.787	0.644	*0.709*	0.636	0.450	*0.528*
CentroidBooster.rep	33	*31.26*	0.793	0.643	*0.710*	0.636	0.450	*0.527*
CentroidBooster.rep	66	*31.30*	0.793	0.642	*0.710*	0.638	0.448	*0.527*
CentroidBooster.rep	100	*31.35*	0.794	0.641	*0.710*	0.642	0.447	*0.527*
BoosTexter.RV	0	*33.90*	0.763	0.624	*0.686*	0.584	0.404	*0.477*
BoosTexter.RV	33	*34.05*	0.778	0.602	*0.679*	0.633	0.394	*0.485*
BoosTexter.RV	66	*34.70*	0.781	0.595	*0.676*	0.629	0.391	*0.482*
BoosTexter.RV	100	*35.59*	0.793	0.581	*0.671*	0.635	0.379	*0.475*
BoosTexter.RVA	0	*33.72*	0.762	0.608	*0.676*	0.593	0.396	*0.475*
BoosTexter.RVA	33	*34.00*	0.783	0.599	*0.679*	0.631	0.389	*0.481*
BoosTexter.RVA	66	*34.72*	0.788	0.592	*0.676*	0.641	0.387	*0.490*
BoosTexter.RVA	100	*35.72*	0.794	0.582	*0.672*	0.643	0.381	*0.478*

Table 4.6.: Robust Training Set Selection on the Reuters Corpus Volume 1.

Algorithm	Robust [%]	One Error	Microaveraged			Macroaveraged		
			Prec.	Rec.	F_1	Prec.	Rec.	F_1
CentroidBooster	0	*45.30*	0.658	0.577	*0.615*	0.624	0.471	*0.537*
CentroidBooster	33	*45.27*	0.664	0.574	*0.616*	0.643	0.474	*0.546*
CentroidBooster	66	*45.35*	0.665	0.572	*0.615*	0.631	0.466	*0.536*
CentroidBooster	100	*45.56*	0.668	0.570	*0.615*	0.635	0.470	*0.540*
CentroidBooster.rep	0	*45.91*	0.659	0.569	*0.611*	0.634	0.471	*0.541*
CentroidBooster.rep	33	*45.51*	0.654	0.574	*0.611*	0.642	0.477	*0.547*
CentroidBooster.rep	66	*45.78*	0.656	0.571	*0.611*	0.626	0.468	*0.535*
CentroidBooster.rep	100	*45.91*	0.660	0.569	*0.611*	0.634	0.471	*0.541*
BoosTexter.RV	0	*46.50*	0.667	0.557	*0.607*	0.565	0.411	*0.476*
BoosTexter.RV	33	*47.40*	0.680	0.554	*0.610*	0.595	0.429	*0.498*
BoosTexter.RV	66	*47.02*	0.681	0.553	*0.610*	0.605	0.425	*0.499*
BoosTexter.RV	100	*47.61*	0.685	0.546	*0.608*	0.621	0.425	*0.504*
BoosTexter.RVA	0	*48.11*	0.662	0.539	*0.594*	0.600	0.399	*0.479*
BoosTexter.RVA	33	*48.54*	0.674	0.532	*0.594*	0.588	0.383	*0.460*
BoosTexter.RVA	66	*48.57*	0.683	0.534	*0.600*	0.603	0.388	*0.472*
BoosTexter.RVA	100	*49.23*	0.680	0.529	*0.595*	0.595	0.384	*0.467*

Table 4.7.: Robust Training Set Selection on the Ohsumed Dataset.

decreased yielding to a lower microaveraged F_1 value for BoosTexter.RV and an equal F_1 value for BoosTexter.RVA for robust training set selection. So the effect of decreasing error propagation (which can be estimated via increasing precision) is canceled by the effect that fewer documents are assigned correctly to a class. The macroaveraged F_1 scores increase, but with no statistical significance.

Ohsumed: As for the RCV 1 corpus, robust training set selection has no significant influence on classification performance. No statistical significant differences between experiments with and without robust training set selection could be obtained on microaveraged performance estimates. There is a slight improvement for BoosTexter.RV on the macroaveraged F_1 value, but the improvement is not statistical significant.

Summary: Robust training set selection has no influence on the performance of a classifier. A few experiments have shown, that microaveraged precision increases, but on the other hand recall decreases. Thus, the effect of reducing error propagation is canceled by identifying less documents correctly.

4.4.4. Ranking Performance

First, ranking performance is evaluated on the flat "Reuters 19" split for comparison with [65]. In [65], experiments are performed using a increasing number of classes, ranging from 3 to 19 classes. Since the point of interest lies in a rough comparison with these experiments, only experiments with 3, 10 and 19 are performed. Table 4.8 lists the results for this data set. Experiments using the the introduced scoring function are indicated by the parameter $\beta = yes$. A validation set size of 20% of the training set was used in this case

For experiments with 3 classes BoosTexter achieves equal results with respect to all three measurements as reported in [65]. As the number of classes increases, the implementation of BoosTexter used in this thesis performs slightly better than reported in literature. Minor differences between experiments may occur due to the following reasons:

- Stemming was used in this thesis, in [65] no stemming was used.

- Since 3-folded CV was used, different splittings may result in slightly different measurements.

Nevertheless it can be stated, that this implementation of BoosTexter achieves the same results as in [65].

Results of BoosTexter.RV, BoosTexter.RVA and CentroidBooster.rep are nearly the same, except for $k = 3$ were CentroidBooster.rep and CentroidBooster achieve a better performance. SVM^{light} performs best in all experiments regarding all measurements. So SVM^{light} can be seen as good base line classifier for the remaining experiments.

Using β for adapting the sigmoid function on the output of a classifier has no positive effect regarding ranking performance. Furthermore, for most experiments there is a minor decrease in performance if β is used. The main reason therefor may be, that the Reuters 19 collection is a flat data set, where no adaption of classifier output is needed (see section 3.2), since all classes are trained on the same feature space. By using β different regions in the same feature space are assigned different weights thereby decreasing performance.

For evaluating the impact of β, experiments on the Ohsumed and RCV 1 dataset are performed. Table 4.9 and Table 4.10 summarizes the results for both data sets, whereas the same notation as before is used. Additionally, the parameter *avg* is added. This parameter indicates, that results of nodes on the classification path are geometrically averaged. In experiments using this averaged score, no sigmoid adaption via β is used. The averaged score is introduced due to results obtained from nodes on the classification path are very low without using β. Thus, the score is biased toward ranking upper level classes first. Averaging should overcome this problem. Geometric averaging is used, due to it is robustness to outliers.

k	Algorithm	β	One Error	Coverage	Avg. Rank. Precision
3	BoosTexter.RV	no	1.13	0.012	0.995
		yes	1.19	0.013	0.994
	BoosTexter.RVA	no	0.97	0.011	0.996
		yes	1.02	0.011	0.995
	CentroidBooster	no	0.49	0.006	0.998
		yes	0.54	0.007	0.998
	CentroidBooster.rep	no	0.43	0.006	0.998
		yes	0.54	0.007	0.998
	SVM^{light}	no	0.54	0.007	0.998
	Schapire [65] BoosTexter.RV	no	1.07	0.016	0.994
10	BoosTexter.RV	no	7.550	0.495	0.951
		yes	7.960	0.511	0.950
	BoosTexter.RVA	no	7.170	0.482	0.954
		yes	7.760	0.503	0.951
	CentroidBooster	no	7.410	0.615	0.945
		yes	7.420	0.625	0.942
	CentroidBooster.rep	no	6.930	0.554	0.951
		yes	6.860	0.586	0.949
	SVM^{light}	no	5.760	0.4527	0.964
	Schapire [65] BoosTexter.RV	no	8.90	0.577	0.943
19	BoosTexter.RV	no	8.500	0.722	0.940
		yes	8.920	0.713	0.939
	BoosTexter.RVA	no	8.140	0.686	0.944
		yes	8.640	0.692	0.941
	CentroidBooster	no	8.280	0.989	0.932
		yes	8.060	0.985	0.933
	CentroidBooster.rep	no	7.780	0.889	0.937
		yes	7.720	0.821	0.941
	SVM^{light}	no	6.610	0.683	0.953
	Schapire [65] BoosTexter.RV	no	10.02	0.837	0.930

Table 4.8.: Ranking performance on the "Reuters 19" data set. Experiments performing the adaption of sigmoid parameter β are marked with "yes" in the β column

It can be seen from previous experiments, that BoosTexter.RV and BoosTexter.RVA perform approximately equal. There are only minor differences between the algorithms. For evaluating ranking performance on the Ohsumed and RCV 1 data set, no distinction between BoosTexter.RV and BoosTexter.RVA is made. The same holds for CentroidBooster and CentroidBooster.rep.

Ohsumed: Results in table 4.9 indicate, that a worse ranking performance is achieved by CentroidBooster and BoosTexter.RV, if no β confidence adaption is performed. One-error and coverage is very high, average precision low. Averaging scores of classifier over the classification path improves the results. Nevertheless, the performance is beneath contempt. Using β to adjust confidence of node classifiers increases all three measurement significantly. One Error drops by about a factor of 3, coverage by a factor of 9 and average precision increases by a factor of 7.

Algorithm	β	One Error	Coverage	Avg. Rank. Precision
BoosTexter.RV	avg	78.75	0.135	0.201
	no	88.34	0.304	0.147
	yes	29.21	0.027	0.774
CentroidBooster	avg	55.23	0.065	0.562
	no	88.47	0.278	0.153
	yes	27.31	0.033	0.779
SVM^{light}	no	26.83	0.064	0.739
	yes	26.14	0.028	0.790

Table 4.9.: Ranking performance on the Ohsumed data set.

Figure 4.1.: Frequency of sigmoid values for the Ohsumed data set with $\beta = 1$. The y-axis shows the sigmoid output of the classifiers and the x-axis shows how often this output was observed. Most sigmoid values for BoosTexter.RV (bottom) are around 0.45 whereas for SVM^{light} (top) the distribution is bimodal with a peak at 0.25 and another peak at around 0.7.

Interestingly, SVM^{light} achieves good ranking performance without adjusting the score. The results on the Ohsumed data set are nearly as good as the results obtained for CentroidBooster and BoosTexter.RV using β-adjustment. In the case of one-error, SVM^{light} exceeds BoosTexter.RV with confidence adjustment, but coverage and average precision are worser for SVM^{light} than for CentroidBooster and BoosTexter.RV. By applying β adjustment to the SVM^{light} the coverage can be decreased and average precision can be increased further. The performance exceeds the results obtained by BoosTexter.RV and CentroidBooster.

Analyzing the differences between SVM^{light} and BoosTexter.RV reveals, that SVM^{light} benefits from the margin distribution over all classes. Figure 4.1 shows the score distribution of SVM^{light}
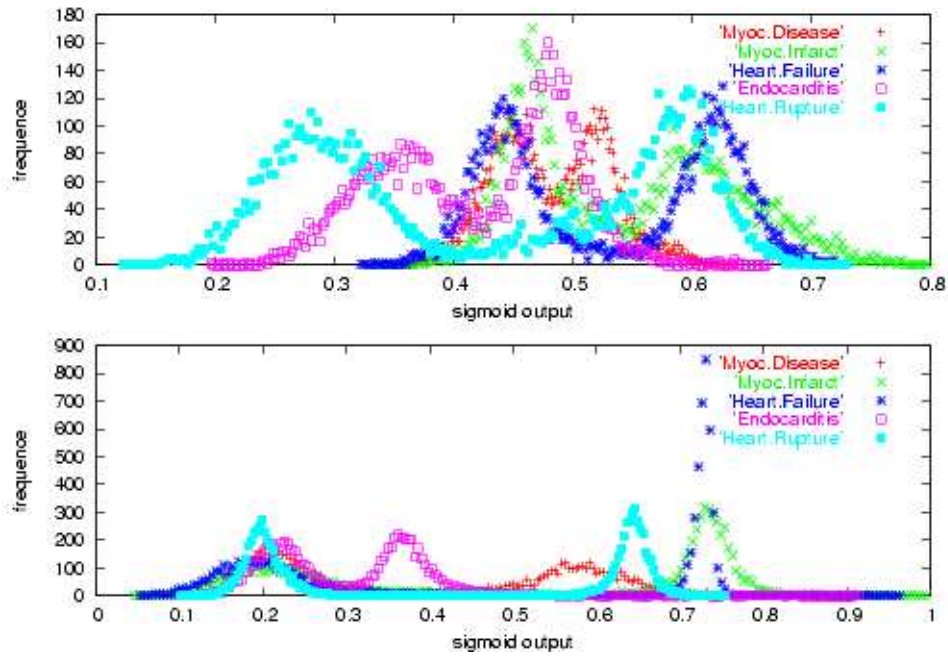
Figure 4.2.: Frequency of sigmoid values for the "Myocardial Infarction", "Heart Rupture", "Endo-carditis", "Myocardial Diseases" and "Myocardial Infarction" class of the Ohsumed data set with $\beta = 1$. The y-axis shows the sigmoid output of the classifiers and the x-axis shows how often this output was observed. The top figure shows the distribution for BoosTexter, the bottom figure for SVM^{light}. It can be seen that for SVM^{light} negative and positive decisions for all classes are grouped closer together as for BoosTexter.RV.

and BoosTexter.RV. The distribution is obtained by discretizing $[0, 1]$ into 1.000 equal intervals and counting the number of scores falling in a interval. The y-axis shows the score and the x-axis shows the number of counts. Scores for each node-document pair are counted. It can be seen that the difference between BoosTexter.RV and SVM^{light} is that SVM^{light} has two peaks, one around 0.25 and one around 0.7 whereas BoosTexter.RV has broader distribution with a peak at around 0.45. For BoosTexter.RV this distribution is generated because different classes have a different margin between the positive and the negative decisions, reflecting harder or easier classification problems for a node. So adding different margin results in a broader distribution. The same holds for CentroidBooster. Due to the constraints $y_i(\vec{d_i} \cdot \vec{w} + \theta) - 1 \geq 0 \quad \forall d_i \in \boldsymbol{S}$ in the learning algorithm things are different for SVM^{light}. For most negative decisions $\vec{d_i} \cdot \vec{w}$ is around -1, yielding to a sigmoid output with $\beta = 1$ around 0.26. For most positive decisions $\vec{d_i} \cdot \vec{w}$ is around 1, yielding to a sigmoid output of 0.73 given $\beta = 1$. So all nodes, independently trained or not, are achieving about the same output on negative and positive decisions, which makes them easier to compare as for BoosTexter. Figure 4.2 shows 5 top level classes and their sigmoid output distribution obtained by BoosTexter.RV and SVM^{light}.

So ranking in BoosTexter.RV and CentroidBooster is biased toward nodes obtaining a low margin between examples. Wrong decisions in such nodes (i.e. propagating a document to the wrong sub-hierarchy) will achieve a score close to 0.5.So if a child node in this wrongly chosen sub-hierarchy achieves a significant higher score than child nodes in the correct sub-hierarchy, the wrong class may be ranked first. Thus, without adapting the score of different nodes ranking performance becomes worse.
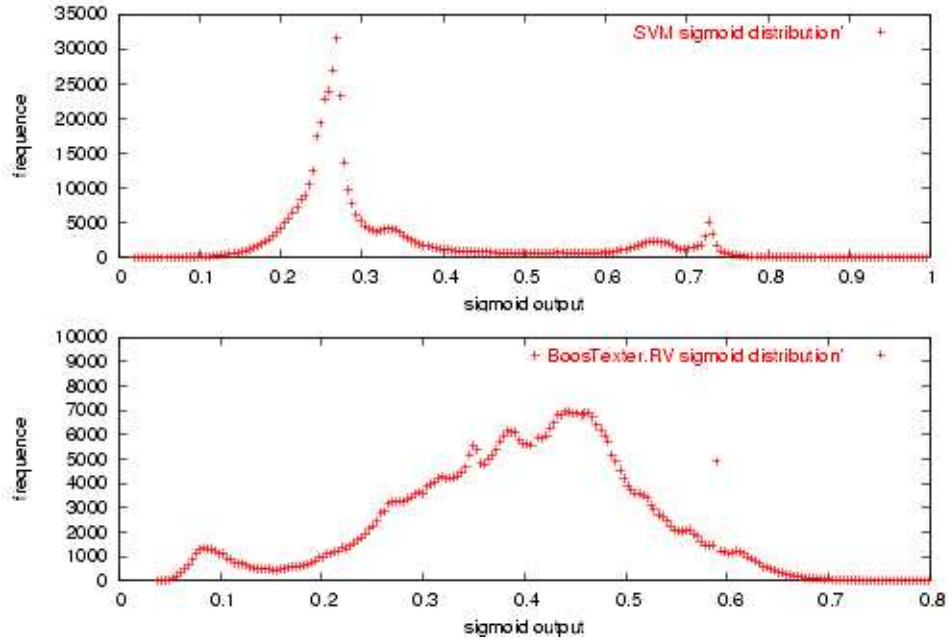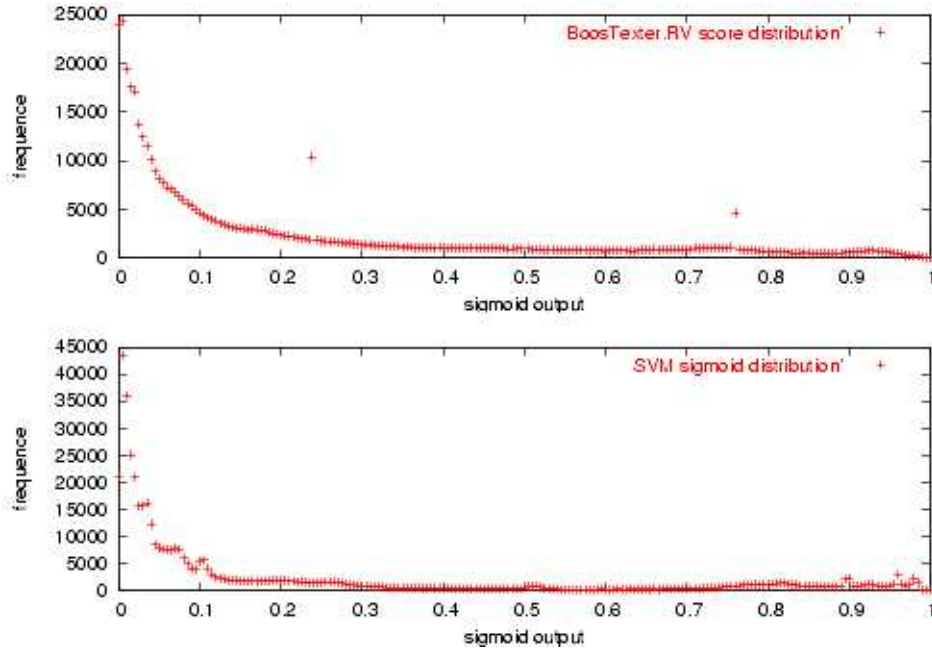
Figure 4.3.: Frequency of sigmoid values for the Ohsumed data set with a β adaption. The y-axis shows the sigmoid output of the classifiers and the x-axis shows how often this output was observed. It can be seen that the majority of decisions is assigned a score close to zero.

Figure 4.3 shows the effect of the adaptive score function. It can be seen, that most negative decisions (score< 0.5) yield a score close to 0. The mass of negative decision is moved toward zero. So, if a document is propagated to the wrong hierarchy it achieves a score close to 0. Positive decision are equally distributed between $[0.5, 0.8]$. So if a high level node propagates a document to the wrong sub-hierarchy, the score will be very low. This low score can not be compensated by a high score of a child node of this sub-hierarchy. So the increase in ranking performance is due to decreasing the score of negative decisions and increasing the score of positive decisions. The sigmoid function is adapted so that negative decisions yield to a saturation of the sigmoid function most time. Additionally it can be seen, that the average margin for positive decision is lower and therefor the score is obtained from the linear area of the sigmoid function. The same holds for SVM^{light}. Additional it can be seen that there is a peak around 0.9 for positive decisions. This analysis shows, that adjusting the score by using the distance of examples improves ranking performance.

RCV 1: From results in table 4.10 it can be seen, that in the case of CentroidBooster and Boos-Texter.RV, a very high one-error, low coverage and low average precision values are obtained if no confidence parameter β is used. Averaging scores from nodes over the classification path increases performance. Coverage decreases and average precision increases significantly. The one-error stays about the same. The results obtained are quite the same as for the Ohsumed data set. In contrast to before, the performance of SVM^{light} without score adjustment is as good as that of Centroid-Booster. If the β-sigmoid function is applied to the output of SVM^{light}, ranking performance is further increased.

From the experiments in this section it can be concluded, that by adjusting the score of a classifier using a β-sigmoid function significantly improves ranking performance in the hierarchical case. Due

68

Algorithm	β	One Error	Coverage	Avg. Rank. Precision
BoosTexter.RV	avg.	58,24	0.113	0.153
	no	56.20	0.329	0.422
	yes	20.57	0.032	0.823
CentroidBooster	avg	54.30	0.057	0.607
	no	55.99	0.308	0.418
	yes	17.38	0.037	0.836
SVM^{light}	no	16.57	0.040	0.840
	yes	16.49	0.027	0.854

Table 4.10.: Ranking performance on the RCV 1 data set.

Data Set	Algorithm	Flat		Hierarchical	
		learn	classify	learn	classify
		[CPU s/it]	[CPU ms]	[CPU s/it]	[CPU ms]
Ohsumed	Centroid Booster	148.10	65.26	49.86	7.24
	BoosTexter.RV	111.12	57.57	32.82	6.45
	BoosTexter.RVA	102.93	60.76	30.48	6.64
	Rocchio	66	65.56	-	-
RCV	Centroid Booster	350.62	55.08	67.20	51.56
	BoosTexter.RV	223.29	50.24	49.42	42.68
	BoosTexter.RVA	217.62	48.90	49.42	40.27
	Rocchio	161	57.56	-	-

Table 4.11.: Shows the learning and classification time for different data sets.

to the constraints on the SVM learning algorithm, SVM's have an advantage for obtaining a good ranking in hierarchical text classification. Nevertheless, ranking performance can be improved for SVM's by the introduced scoring model. Applying a β-sigmoid scoring function in the flat case, does not improve ranking performance.

4.4.5. Computation Time

This section summarizes the learning and classification time for the different algorithms. Differences between between flat and hierarchical classification are outlined. Both measures are quoted for each data set to give insight in the dependency of learning and classification time on the hierarchical structure. Table 4.11 summarizes the results. Since there are no differences for CentroidBooster and CentroidBooster.rep with respect to computational complexity, learning and classification time is showed only for CentroidBooster. The time requirements of algorithms are measured in CPU Seconds. The test system is Athlon XP 2100+ CPU (1700 MHZ) with 512 MB main memory and Linux as operating system. For Boosting, learning time is given in CPU seconds per iteration. For calculating "CPU seconds per iteration", the CPU seconds for learning the whole hierarchy were measured and divided by the number of iterations. The number of iterations is the same for every class/node. For Rocchio the time needed for learning the whole hierarchy is shown. Classification time is given in CPU milliseconds per document.

Flat vs. Hierarchical: As it can be seen, hierarchical learning time is about 3 times smaller for the Ohsumed and about 5 times smaller for the Reuters Corpus Volume 1. Therefrom it can be stated that learning time benefits from hierarchical classification. Looking at the hierarchical structure may explain the difference between the data sets. Classifiers in higher levels of the hierarchy have to be trained with more documents than classifiers at a lower level. Additionally, learning time increases with the number of classes in a node. Comparing the RCV 1 and Ohsumed corpus reveals, that on the second level of the hierarchy the RCV 1 corpus has only 4 and the Ohsumed corpus has 23 classes (see appendix B.4 and appendix B.3). On the third level both data sets have about the same number of classes. So in the hierarchical case the Ohsumed data set has to be trained with all documents on 23 classes whereas for the RCV 1 only 4 classes with all documents are used. Since with each deeper level of the hierarchy the number of training documents decreases, the impact of the number of classes on learning time decreases too. Hence that hierarchical structures having less number of high level nodes benefit more, with respect to learning time, from hierarchical classification than hierarchies having a larger number of high level nodes.

Regarding classification time there is a big difference between the data sets. The Ohsumed data set benefits from hierarchical classification by a factor of 9 whereas there are no big differences regarding the RCV 1 corpus. This implies that on the RCV 1 corpus the number of classifiers evaluated in the hierarchical case are about the same as in the flat case. Additionally, the first three levels of the Ohsumed data set covers about 90% of the test documents and 78% of the classes. On the other hand, only 57% of test documents and 57% of classes are covered by the first 3 levels of the RCV 1 corpus (see table B.10 and table B.11). So one reason for the higher classification time may be, that more documents have to be assigned to lower levels of the RCV 1 hierarchy and more classifiers have to be evaluated on the classification path. But if each document would be perfectly classified by high level nodes of the hierarchy, this circumstance should not matter that much since the number of classifiers which have to be evaluated is lower than in the flat case[3]. This holds also by considering multi-label documents. About 40% of the documents in the RCV 1 are assigned to more than one class. The average path length between labels of multi-label documents is 3.22. So on about 40% of the test examples classifiers on a path having length 3.22 have to be evaluated which should be less than, with resp. to the hierarchical structure of RCV 1, evaluating all classes in the flat case . But since the classification path is on average longer than for the Ohsumed data set, it is more likely that documents are propagated to wrong sub-hierarchies yielding to additional classifier evaluations. This seems to be the main reason for long classification time.

Comparing algorithms: Due to its simple nature, Rocchio is the fastest classifier regarding learning time. It takes only 161 CPU Seconds to learn the RCV 1 hierarchy and about 66 CPU Seconds to learn the Ohsumed hierarchy. Classifying new documents is about the same as for CentroidBooster in the flat case. This seems quite natural because the class of classification hypothesis is the same for both algorithms.

Comparing learning time between the boosting approaches, BoosTexter.RV is about 30-35% faster per iteration than CentroidBooster and about 10% slower than BoosTexter.RVA. Since 10,000 iterations for BoosTexter.RV and 1,000 iterations for CentroidBooster were used, this results in a much higher learning time for BoosTexter. Nevertheless, the optimum number of iterations was not determined in the experiments and so a general conclusion of which algorithm is faster with respect to the best achievable classification result can not be stated and should be the interest of further work.

---

[3]This is not true for an arbitrary hierarchy, especially for the degenerated case of a linear list.

5. Conclusion and Open Questions

An increase in performance by using hierarchical classification depends strongly on the used classifier and the performance measurement of interest. As shown in the experiments, an improvement in terms of microaveraged F_1 values is mainly due to the increase in recall were at the same time precision decreases. This decrease in precision is based on error propagation of high level classifiers. This leads to the conclusion, that classifier achieving high precision values and lower recall benefit more from hierarchical classification than vice versa. SVM are classifiers, at least in the domain of text categorization, which achieve high precision. Thus, Support Vector Machines and BoosTexter benefit more from hierarchical text categorization than the newly introduced CentroidBooster.On macroaveraged F_1 values a slightly higher performance increase from flat to hierarchical classification could be observed.

Corresponding to literature, SVM^{light} gives the best performance on all data sets. Also, CentroidBooster shows good performance on all data sets. On the Reuters-21578 collection CentroidBooster achieves equal scores as SVM^{light} in terms of microaveraged F_1 values. The difference between CentroidBooster and SVM^{light} is, that the former achieves a higher recall whereas the latter achieves a higher precision. One assumption is, that the higher precision of SVM^{light} is achieved because the linear classifier achieves a higher margin on the training data. Evaluating the influence of large margins on hierarchical text categorization would be an issue for further research. Regarding macroaveraged F_1 values, CentroidBooster outperforms SVM^{light} and BoosTexter on all data sets. BoosTexter can not achieve the microaveraged performance of SVM^{light} except at the Reuters-21578 data set. Except for the Ohsumed data set, BoosTexter could not achieve the microaveraged performance of CentroidBooster either. There is a slight advantage for BoosTexter.RV (real valued weak hypothesis) compared to BoosTexter.RVA (real valued weak hypothesis with abstaining). Since BoosTexter shows good performance on standard test collections, further research has to be done for explaining the lack in performance in these experiments. One problem could be the fixed number of 10,000 iterations and has to be investigated further. Rocchio yields to acceptable macroaveraging performance, but performs worst in microaveraged measurements.

Robust training set selection was used to overcome the problem of error propagation. Experiments have shown, that this method did not yield to reliable improvements. In few experiments an increase in precision was achieved, but at the cost of decreasing recall, leading to equal and sometimes worser microaveraged F_1 estimates. So error propagation is still an open problem, and an issue for further research.

Ranking performance is increased by the proposed scoring function. The increase is achieved due to wrong decisions are penalized by a low score. By applying the modified sigmoid function, the mass of the score for wrong decisions is moved towards zero. Thus, propagating a document to the wrong hierarchy achieves such a low score that it can not be compensated by a high score of one child node. For CentroidBooster and BoosTexter, if no confidence adjustment takes place, the training margin is different for different nodes. Thus, some nodes achieve a score close to 0.5 for wrong decisions. If a document is propagated into a wrong sub hierarchy by such a decision, a child node could compensate the low score by a high score yielding worse ranking performance. It was shown,

that SVM's achieve a margin around -1 for most negative and a margin around 1 for most positive decision. The differences between indpendent nodes are not as large as for CentroidBooster or Boos-Texter. Compared to BoosTexter and CentroidBooster, SVM's achieve a better ranking performance on hierarchical data sets in the case where no score adjustment is done. Nevertheless, adjusting the score for SVM's increases ranking performance significantly. One major drawback of adjusting the score function is the usage of a valdiation set. Learning time approximatly doubles. One intersting point would be to evaluate the parameters for adjusting the score without a validation set by using for example theoretical generalization bounds. Another open question is, if ranking performance could be further enhanced by adjusting the score over more than one class, for example over a decision node, a sub-hierarchy or over the whole hierarchy. Hitherto F_1 measures are calculated on the threshold of an example to the hyperplane. Optimizing thresholds on the modified score function may increase hard classification performance and should be further investigated.

Regarding time complexity, hierarchical classification improves learning and classification time. Learning time is improved about 3 to 5 times. Experiments have shown, that the improvement depends strongly on the hierarchical structure, whereas a higher influence is given regarding classification time. Having a larger number of high level nodes and a larger proportion of documents assigned on higher levels of the hierarchy improves classification time. In this setting, learning time is decreased but still outperforms the flat case. Hierarchical structures having more documents assigned to deeper nodes of the hierarchy cancels the improvement of classification speed yielding to about the same classification time as in the flat case. One reason for this is, that documents are propagated to wrong sub hierarchies more likely, resulting in a higher number of node classifiers which have to be evaluated. Learning time benefits from hierarchical structures having proportional few high level nodes.

A. Stopwordlist

The following stopwords were used in the experiments:

a, about, after, all, an, an, and, are,as, at, back, be, been ,but, by, call, came, can, come, could, day, did, do, down, each, every, find, first, for, from, get, give, go, good, had, has, have, he, her, him, his, hot, how, I, if, in, is, it, know, like, little, live, long, look, made, make, many, may, me, more, most, my, new, no, now, of, on, one, only, or, other, our, out, over , part, round, said, see, she sid, so, some, take, than, that, the, their, them, then, there, these, they, thing, this, time, to, two, under, up, use, was, way, we, were, what, when, where, which, who, will, with, wor, word, would, you, your

B. Data Sets

This chapter lists details of the used datasets.

B.1. Reuters 21578

k	Class	#Docs.	Cum. #Docs	Avg. No. Labels/Doc.
1	money-fx	717	717	1
2	grain	582	1299	1
3	crude	578	1877	1.0032
4	trade	486	2363	1.0261
5	interest	478	2841	1.0956
6	ship	286	3127	1.1309
7	wheat	283	3410	1.2319
8	corn	237	3647	1.3176
9	dlr	175	3822	1.3773
10	supply	174	3996	1.3629
11	oilseed	171	4167	1.3835
12	sugar	162	4329	1.3778
13	coffee	139	4468	1.3693
14	gnp	136	4604	1.3682
15	oil	124	4728	1.2842
16	gold	124	4852	1.3625
17	soybean	111	4963	1.3937
18	gas	105	5068	1.3544
19	bop	105	5173	1.4247

Table B.1.: Description of the classes for the Reuters 19 data set

B.2. Significance Test Results

Detailed results of significance testing are outlined in this section. The statistical tests are explained in section 4.2.3. Each cell in the tables indicates the level of significance on the performance measurement:

- $\sim$ indicates no significant differences

- $\ggg$ indicates that the row algorithm achieves a significantly higher measurement then the column algorithm. The significance level α is 0.005

- $\gg$ indicates that the row algorithm achieves a significantly higher measurement then the column algorithm. The significance level α is 0.01

- $>$ indicates that the row algorithm achieves a significantly higher measurement then the column algorithm. The significance level α is 0.05

- $\lll$ indicates that the row algorithm achieves a significantly lower measurement then the column algorithm. The significance level α is 0.005

- $\ll$ indicates that the row algorithm achieves a significantly lower measurement then the column algorithm. The significance level α is 0.01

- $<$ indicates that the row algorithm achieves a significantly lower measurement then the column algorithm. The significance level α is 0.05

Measurements are abbrevated as follows:

- one error: *err.*

- precision: *prec.*

- recall: *rec.*

- S-Test for F_1 value: F_1 *S-Test*

- T-Test for F_1 value: F_1 *T-Test*

- R-Test for F_1 value: F_1 *R-Test*

Ohsumed	CentroidBooster Hierarchical					
	Micro Testing			Macro Testing		
	error	prec.	rec.	F1 S-Test	F1 T-Test	F1 R-Test
CentroidBooster.rep Hierarchical	∼	∼	∼	∼	∼	∼
BoosTexter.RV Hierarchical	∼	∼	<	<	≪	≫
SVM Hierarchical	≪	≫	≪	∼	<	∼
BoosTexter.RVA Hierarchical	≫	∼	≪	≪	≪	≫
CentroidBooster.rep Hierarchical						
CentroidBooster Hierarchical	∼	∼	∼	∼	∼	∼
BoosTexter.RV Hierarchical	∼	∼	∼	<	≪	≫
SVM Hierarchical	≪	≫	<	∼	<	∼
BoosTexter.RVA Hierarchical	>	∼	≪	≪	≪	≫
BoosTexter.RV Hierarchical						
CentroidBooster Hierarchical	∼	∼	>	>	≫	≪
CentroidBooster.rep Hierarchical	∼	∼	∼	>	≫	≪
SVM Hierarchical	≪	≫	∼	>	∼	<
BoosTexter.RVA Hierarchical	>	∼	<	≪	∼	∼
SVM Hierarchical						
CentroidBooster Hierarchical	≫	≪	≫	∼	>	∼
CentroidBooster.rep Hierarchical	≫	≪	>	∼	>	∼
BoosTexter.RV Hierarchical	≫	≪	∼	<	∼	>
BoosTexter.RVA Hierarchical	≫	≪	∼	≪	<	≫
BoosTexter.RVA Hierarchical						
CentroidBooster Hierarchical	≪	∼	≫	≫	≫	≪
CentroidBooster.rep Hierarchical	<	∼	≫	≫	≫	≪
BoosTexter.RV Hierarchical	<	∼	>	≫	∼	∼
SVM Hierarchical	≪	≫	∼	≫	>	≪

Table B.2.: Shows results from significance testing on the hierarchical ohsumed data set. No robust training set selection is performed

Ohsumed	CentroidBooster Flat					
	Micro Testing			Macro Testing		
	error	prec.	rec.	F1 S-Test	F1 T-Test	F1 R-Test
CentroidBooster.rep Flat	~	~	~	~	~	~
BoosTexter.RV Flat	≫	>	≪	≪	≪	≫
BoosTexter.RVA Flat	≫	~	≪	≪	≪	≫
Rocchio	≫	<	≪	~	<	>
SVM Flat	>	≫	≪	~	≪	>
CentroidBooster.rep Flat						
CentroidBooster Flat	~	~	~	~	~	~
BoosTexter.RV Flat	≫	≫	≪	≪	≪	≫
BoosTexter.RVA Flat	≫	>	≪	≪	≪	≫
Rocchio	≫	~	≪	~	<	≫
SVM Flat	>	≫	≪	~	≪	≫
BoosTexter.RV Flat						
CentroidBooster Flat	≪	<	≫	≫	≫	≪
CentroidBooster.rep Flat	≪	≪	≫	≫	≫	≪
BoosTexter.RVA Flat	>	~	≪	≪	≪	≫
Rocchio	~	≪	≪	>	~	~
SVM Flat	<	≫	~	>	≫	≪
Rocchio						
CentroidBooster Flat	≪	>	≫	~	>	<
CentroidBooster.rep Flat	≪	~	≫	~	>	≪
BoosTexter.RV Flat	~	≫	≫	<	~	~
BoosTexter.RVA Flat	~	≫	≫	≪	≪	≫
SVM Flat	≪	≫	≫	~	~	~
SVM Flat						
CentroidBooster Flat	<	≪	≫	~	≫	<
CentroidBooster.rep Flat	<	≪	≫	~	≫	≪
BoosTexter.RV Flat	>	≪	~	<	≪	≫
BoosTexter.RVA Flat	≫	≪	<	≪	≪	≫
Rocchio	≫	≪	≪	~	~	~
BoosTexter.RVA Flat						
CentroidBooster Flat	≪	~	≫	≫	≫	≪
CentroidBooster.rep Flat	≪	<	≫	≫	≫	≪
BoosTexter.RV Flat	<	~	≫	≫	≫	≪
Rocchio	~	≪	≪	≫	≫	≪
SVM Flat	≪	≫	>	≫	≫	≪

Table B.3.: Shows results from significance testing on the flat ohsumed data set.

Ohsumed	CentroidBooster Flat					
	Micro Testing			Macro Testing		
	error	prec.	rec.	F1 S-Test	F1 T-Test	F1 R-Test
CentroidBooster Hierarchical	>	∼	∼	∼	∼	<
CentroidBooster.rep Hierarchical	≫	∼	∼	∼	∼	∼
BoosTexter.RV Hierarchical	≫	∼	∼	∼	<	>
SVM Hierarchical	<	≫	∼	>	∼	∼
BoosTexter.RVA Hierarchical	≫	∼	<	<	≪	≫
CentroidBooster.rep Flat						
CentroidBooster Hierarchical	>	∼	∼	∼	∼	∼
CentroidBooster.rep Hierarchical	≫	∼	∼	∼	∼	∼
BoosTexter.RV Hierarchical	≫	∼	∼	∼	≪	>
SVM Hierarchical	<	≫	∼	∼	∼	∼
BoosTexter.RVA Hierarchical	≫	∼	≪	<	≪	≫
BoosTexter.RV Flat						
CentroidBooster Hierarchical	<	≪	≫	≫	≫	≪
CentroidBooster.rep Hierarchical	∼	≪	≫	≫	≫	≪
BoosTexter.RV Hierarchical	∼	<	≫	≫	>	<
SVM Hierarchical	≪	≫	≫	≫	≫	≪
BoosTexter.RVA Hierarchical	∼	≪	>	>	>	<
Rocchio Flat						
CentroidBooster Hierarchical	≪	∼	≫	≫	>	≪
CentroidBooster.rep Hierarchical	≪	∼	≫	≫	>	≪
BoosTexter.RV Hierarchical	<	>	≫	∼	∼	∼
SVM Hierarchical	≪	≫	≫	∼	∼	<
BoosTexter.RVA Hierarchical	∼	>	≫	∼	∼	∼
SVM Flat						
CentroidBooster Hierarchical	∼	≪	≫	∼	≫	≪
CentroidBooster.rep Hierarchical	∼	≪	≫	∼	≫	≪
BoosTexter.RV Hierarchical	∼	≪	≫	∼	∼	∼
SVM Hierarchical	≪	≪	≫	≫	≫	≪
BoosTexter.RVA Hierarchical	≫	≪	≫	∼	∼	>
BoosTexter.RVA Flat						
CentroidBooster Hierarchical	≪	<	≫	≫	≫	≪
CentroidBooster.rep Hierarchical	≪	<	≫	≫	≫	≪
SVM Hierarchical	≪	≫	≫	≫	≫	≪
BoosTexter.RV Hierarchical	≪	∼	≫	≫	≫	≪
BoosTexter.RVA Hierarchical	∼	∼	≫	≫	≫	≪

Table B.4.: Shows results from significance testing on the hierarchical and flat ohsumed data set. No robust training set selection is performed

Ohsumed		CentroidBooster Hierarchical, 0%					
	Robust	Micro Testing			Macro Testing		
	[%]	error	prec.	rec.	F1 S-Test	F1 T-Test	F1 R-Test
CentroidBooster	33	∼	∼	∼	>	∼	<
CentroidBooster	66	∼	∼	∼	∼	∼	∼
CentroidBooster	100	∼	∼	∼	∼	∼	∼
CentroidBooster.rep Hierarchical, 0%							
CentroidBooster.rep	33	∼	∼	∼	∼	∼	∼
CentroidBooster.rep	66	∼	∼	∼	<	∼	∼
CentroidBooster.rep	100	∼	∼	∼	∼	∼	∼
BoosTexter.RV Hierarchical,0[%]							
BoosTexter.RV	33	∼	∼	∼	∼	∼	∼
BoosTexter.RV	66	∼	∼	∼	∼	∼	∼
BoosTexter.RV	100	∼	>	∼	∼	∼	∼
BoosTexter.RVA Hierarchical,0[%]							
BoosTexter.RVA	33	∼	∼	∼	<	<	<
BoosTexter.RVA	66	∼	>	∼	∼	∼	∼
BoosTexter.RV	100	∼	>	∼	∼	∼	∼

Table B.5.: Shows results from significance testing on experiments using robust training set selection on the Ohsumed data set. Classifiers without robust training set selection are compared to classifiers with increasing robust training sets.

RCV 1	CentroidBooster Hierarchical					
	Micro Testing			Macro Testing		
	error	prec.	rec.	F1 S-Test	F1 T-Test	F1 R-Test
CentroidBooster.rep Hierarchical	~	~	~	~	~	~
BoosTexter.RV Hierarchical	⋙	⋘	⋘	⋘	⋘	⋙
SVM Hierarchical	⋘	⋙	⋘	~	⋘	>
BoosTexter.RVA Hierarchical	⋙	⋘	⋘	⋘	⋘	⋙
CentroidBooster.rep Hierarchical						
CentroidBooster Hierarchical	~	~	~	~	~	~
BoosTexter.RV Hierarchical	⋙	⋘	⋘	⋘	⋘	⋙
SVM Hierarchical	⋘	⋙	⋘	~	⋘	>
BoosTexter.RVA Hierarchical	⋙	⋘	⋘	⋘	⋘	⋙
BoosTexter.RV Hierarchical						
CentroidBooster Hierarchical	⋘	⋙	⋙	⋙	⋙	⋘
CentroidBooster.rep Hierarchical	⋘	⋙	⋙	⋙	⋙	⋘
SVM Hierarchical	⋘	⋙	⋙	⋙	⋙	⋘
BoosTexter.RVA Hierarchical	~	~	~	~	~	~
SVM Hierarchical						
CentroidBooster Hierarchical	⋙	⋘	⋙	~	⋙	<
CentroidBooster.rep Hierarchical	⋙	⋘	⋙	~	⋙	<
BoosTexter.RV Hierarchical	⋙	⋘	⋘	⋘	⋘	⋙
BoosTexter.RVA Hierarchical	⋙	⋘	⋘	⋘	⋘	⋙
BoosTexter.RVA Hierarchical						
CentroidBooster Hierarchical	⋘	⋙	⋙	⋙	⋙	⋘
CentroidBooster.rep Hierarchical	⋘	⋙	⋙	⋙	⋙	⋘
BoosTexter.RV Hierarchical	~	~	~	~	~	~
SVM Hierarchical	⋘	⋙	⋙	⋙	⋙	⋘

Table B.6.: Shows results from significance testing on the hierarchical RCV 1 data set. No robust training set selection is performed

RCV 1	CentroidBooster Flat					
	Micro Testing			Macro Testing		
	error	prec.	rec.	F1 S-Test	F1 T-Test	F1 R-Test
CentroidBooster.rep Flat	∼	∼	∼	∼	∼	∼
BoosTexter.RV Flat	⋙	⋘	⋘	⋘	⋘	⋙
BoosTexter.RVA Flat	⋙	⋘	⋘	⋘	⋘	⋙
SVM Flat	∼	⋙	⋘	≪	⋘	>
Rocchio Flat	⋙	⋘	⋘	⋘	⋘	⋙
CentroidBooster.rep Flat						
CentroidBooster Flat	∼	∼	∼	∼	∼	∼
BoosTexter.RV Flat	⋙	⋘	⋘	⋘	⋘	⋙
BoosTexter.RVA Flat	⋙	⋘	⋘	⋘	⋘	⋙
SVM Flat	∼	⋙	⋘	⋘	⋘	>
Rocchio Flat	⋙	⋘	⋘	⋘	⋘	⋙
BoosTexter.RV Flat						
CentroidBooster Flat	⋘	⋙	⋙	⋙	⋙	⋘
CentroidBooster.RV Flat	⋘	⋙	⋙	⋙	⋙	⋘
SVM Flat	⋘	⋙	⋙	⋙	≫	⋘
Rocchio	⋙	⋘	⋘	<	∼	∼
SVM Flat						
CentroidBooster Flat	∼	⋘	⋙	≫	⋙	<
CentroidBooster.rep Flat	∼	⋘	⋙	⋙	⋙	<
BoosTexter.RV Flat	⋙	⋘	⋘	⋘	≪	⋙
Rocchio Flat	⋙	⋘	⋘	⋘	<	⋙
Rocchio Flat						
CentroidBooster Flat	⋘	⋙	⋙	⋙	⋙	⋘
CentroidBooster.rep Flat	⋘	⋙	⋙	⋙	⋙	⋘
BoosTexter.RV Flat	⋘	⋙	⋙	>	∼	∼
BoosTexter.RVA Flat	⋘	⋙	⋙	>	∼	∼
SVM Flat	⋘	⋙	⋙	⋙	>	⋘

Table B.7.: Shows results from significance testing on the flat RCV 1 data set.

RCV 1	CentroidBooster Flat					
	Micro Testing			Macro Testing		
	error	prec.	rec.	F1 S-Test	F1 T-Test	F1 R-Test
CentroidBooster Hierarchical	≪	≪	≫	≫	≫	≪
CentroidBooster.rep Hierarchical	≪	≪	≫	≫	≫	≪
BoosTexter.RV Hierarchical	∼	≪	∼	<	<	>
SVM Hierarchical	≪	≫	>	>	∼	∼
BoosTexter.RVA Hierarchical	∼	≪	<	≪	<	>
	CentroidBooster.rep Flat					
CentroidBooster Hierarchical	≪	≪	≫	≫	>	≪
CentroidBooster.rep Hierarchical	≪	≪	≫	≫	>	≪
BoosTexter.RV Hierarchical	∼	≪	∼	<	<	>
SVM Hierarchical	≪	≫	≫	∼	∼	∼
BoosTexter.RVA Hierarchical	∼	≪	∼	≪	<	>
	SVM Flat					
CentroidBooster Hierarchical	≪	≪	≫	≫	≫	≪
CentroidBooster.rep Hierarchical	≪	≪	≫	≫	≫	≪
BoosTexter.RV Hierarchical	∼	≪	≫	∼	∼	∼
SVM Hierarchical	≪	≪	≫	≫	≫	≪
BoosTexter.RVA Hierarchical	∼	≪	≫	∼	∼	∼
	Rocchio					
CentroidBooster Hierarchical	≪	≫	≫	≫	≫	≪
CentroidBooster.rep Hierarchical	≪	≫	≫	≫	≫	≪
BoosTexter.RV Hierarchical	≪	≫	≫	≫	≫	≪
SVM Hierarchical	≪	≫	≫	≫	≫	≪
BoosTexter.RVA Hierarchical	≪	≫	≫	≫	≫	≪
	BoosTexter.RV Flat					
CentroidBooster Hierarchical	≪	>	≫	≫	≫	≪
CentroidBooster.rep Hierarchical	≪	∼	≫	≫	≫	≪
BoosTexter.RV Hierarchical	≪	≪	≫	≫	≫	≪
SVM Hierarchical	≪	≫	≫	≫	≫	≪
BoosTexter.RVA Hierarchical	≪	≪	≫	≫	≫	≪
	BoosTexter.RVA Flat					
CentroidBooster Hierarchical	≪	>	≫	≫	≫	≪
CentroidBooster.rep Hierarchical	≪	∼	≫	≫	≫	≪
BoosTexter.RV Hierarchical	≪	≪	≫	≫	≫	≪
SVM Hierarchical	≪	≫	≫	≫	≫	≪
BoosTexter.RVA Hierarchical	≪	≪	≫	≫	≫	≪

Table B.8.: Shows results from significance testing on the hierarchical and flat RCV 1 data set. No robust training set selection is performed

RCV 1		CentroidBooster 0%					
	Robust	Micro Testing			Macro Testing		
	[%]	error	prec.	rec.	F1 S-Test	F1 T-Test	F1 R-Test
CentroidBooster	33	~	~	~	>	~	~
CentroidBooster	66	~	>	~	~	~	~
CentroidBooster	100	~	>	~	>	~	~
CentroidBooster.rep 0%							
CentroidBooster.rep	33	~	~	~	>	~	~
CentroidBooster.rep	66	~	~	~	~	~	~
CentroidBooster.rep	100	~	~	~	~	~	~
BoosTexter.RVA 0%							
BoosTexter.RVA	33	~	≫	<	~	~	~
BoosTexter.RVA	66	≫	≫	≪	~	~	~
BoosTexter.RVA	100	~	>	~	~	~	~
BoosTexter.RV 0%							
BoosTexter.RV	33	~	≫	<	~	~	~
BoosTexter.RV	66	>	≫	≪	~	~	~
BoosTexter.RV	100	~	>	~	~	~	~

Table B.9.: Shows results from significance testing on experiments using robust training set selection on the RCV 1 data set. Classifiers without robust training set selection are compared to classifiers with increasing robust training sets.

level	# labels in test set	# labels in training set	# classes
1	0	0	1
2	735	791	4
3	15,366	15,591	54
4	10,463	11,723	42
5	228	329	1

Table B.10.: Classes, training and test documents per level of the RCV 1 hierarchy.

B.3. Reuters Corpus Volume 1-Hierarchical Structure

This section outlines the hierarchical structure of the used Reuters Corpus Volume 1 data set. Top level nodes are marked bold. Additionally, table B.10 counts the number of classes, training and test documents assigned to a level of the hierarchy. The hierarchy is display in a two column format (for saving space).

Economics

- Housing Starts
- Consumer Finance
    - Retail Sales
    - Consumer Credit
    - Personal Income
- Output Capacity
    - Industrial Production
    - Inventories
    - Capacity Utilization
- Goverment Finance
    - Expenditure Revenue
    - Government Borrowing
- Monetary Economic
    - Money Supply
- Economic Performance
- Employment Labour
    - Unemployment
- Trade Reserves
    - Merchandise Trade
    - Reserves
    - Balance of Payments
- Inflation Prices
    - Consumer Prices
    - Wholesale Prices
- Leading Indicators

Goverments Social

- Health
- Domestic Politics
- War, Civil War
- Fashion
- Travel and Toursim
- Sports
- Science and Technology
- Disasters and Accidents
- Obituaries
- European Community
    - EC Corporate Policy
    - EC Competition Subsidy
    - EC External Relations
    - EC Institutioins
    - EC Monetary Economic
    - EC General
    - EC Agriculture Policy
    - EC Internal Market
    - EC Environment Issues
- International Relations
- Crime, Law Enforcement

- Elections
- Arts, Culture, Entertainment
- Religion
- Welfare, Socail Services
- Weather
- Environment and Natural World
- Biographies, Personalities, People
- Labour Issues
- Human Interest
- Defence

Corporate Industrial

- Insolvency Liquidity
- New Products Services
- Labour
- Production Services
- Legal Judicial
- Management
    - Management Moves
- Ownership Changes
    - Asset Transfers
    - Mergers Acquisitions
    - Privatisations
- Contracts Orders
    - Defence Contracts
- Capacity Facitlies
- Markets Marketing
    - Market Share

- – External Markets
    - – Domestic Markets
- Monopolies Competition
- Funding Captial
    - Bonds Debt Issues
    - Loans Credits
    - Share Captial
    - Credit Ratings
- Advertising Promotion
- Share Listings
- Performance
    - Comment Forecasts
    - Accounts Earnings
        * Annual Results
- Regulation Policy
- Strategy Plans
- Research Development

Markets

- Equity Markets
- Commodity Markets
    - Soft Commodities
    - Energy Markets
    - Metals Trading
- Bond Markets
- Money Markets
    - Interbank Markets
    - Forex Markets

B.4. OHSUMED

This section outlines the hierarchical structure of the used Ohsumed data set. Top level nodes are marked bold. Additionally, table B.11 counts the number of classes, training and test documents assigned to a level of the hierarchy. The hierarchy is display in a two column format (for saving space).

Arrhythmia

- Pre-Excitation Syndromes
    - Wolff-Parkinson-White Syndrome

- – Pre-Excitation, Mahaim-Type
- Heart Block
    - Adams-Stokes Syndrome

level	# labels in test set	# labels in training set	# classes
1	673	177	1
2	10,269	2,987	23
3	6,141	1,732	56
4	1.164	329	16
5	159	38	6

Table B.11.: Classes, training and test documents per level of the Ohsumed hierarchy.

- – Sinoatrial Block
- – Bundle-Branch Block
- Long QT Syndrome
- Bradycardia
- Ventricular Fibrillation
- Tachycardia
  - – Tachycardia, Paroxysmal
  - – Tachycardia, Supraventricular
    - ∗ Tachycardia, Atrioventricular Nodal Reentry
    - ∗ Tachycardia, Ectopic Junctional
    - ∗ Tachycardia, Sinoatrial Nodal Reentry
    - ∗ Tachycardia, Sinus
    - ∗ Tachycardia, Ectopic Atrial
- Atrial Fibrillation
- Sick Sinus Syndrome
- Atrial Flutter
- Arrhythmia, Sinus

Heart Rupture

- Heart Rupture, Post-Infarction

Myocardial Infarction

- Shock, Cardiogenic

Heart Failure, Congestive

- Dyspnea, Paroxysmal
- Edema, Cardiac

Heart Defects, Congenital

- Transposition of Great Vessels
  - – Double Outlet Right Ventricle
- Tetralogy of Fallot
- Marfan Syndrome
- Eisenmenger Complex
- Coronary Vessel Anomalies
- Ebstein's Anomaly
- Truncus Arteriosus, Persistent
- Ductus Arteriosus, Patent
- Cor Triatriatum
- Dextrocardia
- Crisscross Heart
- Aortic Coarctation
- Heart Septal Defects
  - – Endocardial Cushion Defects
  - – Aortopulmonary Septal Defect
  - – Heart Septal Defects, Ventricular
  - – Heart Septal Defects, Atrial
- Levocardia

Myocardial Diseases

- Myocarditis
- Endocardial Fibroelastosis
- Kearns Syndrome
- Cardiomyopathy, Restrictive
- Myocardial Reperfusion Injury
- Chagas Cardiomyopathy
- Endomyocardial Fibrosis
- Cardiomyopathy, Alcoholic

Heart Valve Diseases

- Mitral Valve Insufficiency
- Heart Murmurs
- Tricuspid Valve Stenosis
- Aortic Valve Stenosis
    - Cardiomyopathy, Hypertrophic
- Tricuspid Valve Prolapse
- Aortic Valve Prolapse
- Tricuspid Valve Insufficiency
- Pulmonary Valve Insufficiency
- Aortic Valve Insufficiency
- Pulmonary Valve Stenosis
    - Pulmonary Subvalvular Stenosis
- Mitral Valve Stenosis
- Mitral Valve Prolapse

Endocarditis

- Endocarditis, Bacterial
    - Endocarditis, Subacute Bacterial

Cardiac Output, Low

Heart Aneurysm

Coronary Disease

- Angina Pectoris
    - Angina, Unstable
        * Angina Pectoris, Variant
- Coronary Aneurysm

- Coronary Arteriosclerosis
- Coronary Thrombosis
- Coronary Vasospasm

Pneumopericardium

Pericarditis

- Pericarditis, Constrictive
- Pericarditis, Tuberculous

Postpericardiotomy Syndrome

Cardiomyopathy, Congestive

Ventricular Outflow Obstruction

Heart Neoplasms

Carcinoid Heart Disease

Rheumatic Heart Disease

Pericardial Effusion

Cardiac Tamponade

Heart Arrest

Bibliography

[1] M. A. Aizerman, E. A. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. In *Automation and Remote Control,*, number 25, pages 821–837, 1964.

[2] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *Proc. 17th International Conf. on Machine Learning*, pages 9–16. Morgan Kaufmann, San Francisco, CA, 2000.

[3] C. Apte, F. Damerau, and S. M. Weiss. Automated learning of decision rules for text categorization. *Information Systems*, 12(3):233–251, 1994.

[4] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[5] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. Technical Report UT-CS-94-270, 1994.

[6] B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Computational Learing Theory*, pages 144–152, 1992.

[7] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. In *Knowledge Discovery and Data Mining*, number 2, 1998.

[8] M. F. Caropreso, S. Matwin, and F. Sebastiani. Statistical phrases in automated text categorization. Technical Report IEI-B4-07-2000, Pisa, IT, 2000.

[9] J. Caumanns. A fast and simple stemming algorithm for german words.

[10] W. B. Cavnar and J. M. Trenkle. N-gram statistics for natural language understanding and text processing. In *IEEE Trans. on Pattern Analysis and Machine Intelligence*, volume 2, pages 164–172., 1979.

[11] W. W. Cohen. Learning to classify English text with ILP methods. In L. D. Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 3–24. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[12] W. W. Cohen and Y. Singer. Context-sensitive learning methods for text categorization. In H.-P. Frei, D. Harman, P. Schäuble, and R. Wilkinson, editors, *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval*, pages 307–315, Zürich, CH, 1996. ACM Press, New York, US.

[13] M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, adaboost and bregman distances. In *Computational Learing Theory*, pages 158–169, 2000.

[14] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[15] L. Denoyer, H. Zaragoza, and P. Gallinari. Hmm-based passage models for document classification and ranking. In *23rd European Colloquium on Information Retrieval Research*, 2001.

[16] I. Dhillon, S. Mallela, and R. Kumar. Enhanced word clustering for hierarchical text classification, 2002. to be published.

[17] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[18] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, 2001.

[19] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management*, pages 148–155. ACM Press, 1998.

[20] S. I. Dumais. Improving the retrieval of information from external sources. behavior research methods. In *Instruments and Computers*, pages 229–236, 1991.

[21] S. T. Dumais and H. Chen. Hierarchical classification of Web content. In N. J. Belkin, P. Ingwersen, and M.-K. Leong, editors, *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval*, pages 256–263, Athens, GR, 2000. ACM Press, New York, US.

[22] A. W. E. Wiener, J.O. Pedersen. A neural network approach to topic spotting. In *Proceedings of SDAIR '95*, pages 317–332, Las Vegas, NV, US, 1995.

[23] L. Edda and K. Jörg. Text categorization with support vector machines. how to represent texts in input space? *Machine Learning*, 46:423–444, 2002.

[24] F. Ekmekcioglu, M. Lynch, A. Robertson, T. Sembok, and P. Willett. Comparison of n-gram matching and stemming for term conflation in english, malay and turkish texts.

[25] P. Frasconi, G. Soda, and A. Vullo. Hidden markov models for text categorization in multi-page documents. *Journal of Intelligent Information Systems (JIIS)*, 18(1):195–217, 2002.

[26] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[27] J. H. Friedman. Another approach to polychotomous classification. Technical report, Department of Statistics, Stanford University, 1997.

[28] N. Fuhr and C. Buckley. A probabilistic learning approach for document indexing. *ACM Transactions on Information Systems (TOIS)*, 9(3):223–248, 1991.

[29] N. Fuhr, S. Hartmann, G. Knorz, G. Lustig, M. Schwantner, and K. Tzeras. AIR/X – a rule-based multistage indexing system for large subject fields. In A. Lichnerowicz, editor, *Proceedings of RIAO-91, 3rd International Conference "Recherche d'Information Assistee par Ordinateur"*, pages 606–623, Barcelona, ES, 1991. Elsevier Science Publishers, Amsterdam, NL.

[30] W. A. Gale, K. W. Church, and D. Yarowsky. A method for disambiguating word senses in a large corpus. In *Computers and the Humanitzes*, volume 26, pages 415–439, 1993.

[31] M. Granitzer, W. Kienreich, V. Sabol, and G. Dösinger. Webrat: Supporting agile knowledge retrieval through dynamic, incremental clustering and automatic labelling of web search result sets. In *Proceedings of the 1st Workshop Knowledge Management for Distributed Agile Processes (IEEE Workshop Enabling Technologies)*, 2003.

[32] T. Hastie and R. Tibshirani. Classification by pairwise coupling. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[33] M. A. Hearst and C. Plaunt. Subtopic structuring for full-length document access. In *Research and Development in Information Retrieval*, pages 59–68, 1993.

[34] W. Hersh, C. Buckley, T. J. Leone, and D. Hickam. Ohsumed: an interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 192–201. Springer-Verlag New York, Inc., 1994.

[35] T. Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[36] T. Joachims. Making large-scale svm learning practical. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999.

[37] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. Technical Report AIM-1440, 1993.

[38] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In D. H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 170–178, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[39] L. S. Larkey and W. B. Croft. Combining classifiers in text categorization. In H.-P. Frei, D. Harman, P. Schäuble, and R. Wilkinson, editors, *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval*, pages 289–297, Zürich, CH, 1996. ACM Press, New York, US.

[40] D. D. Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the Fifteenth Annual International ACM SIGIR conference on Research and development in information retrieval*, pages 37–50. ACM Press, 1992.

[41] D. D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In C. Nédellec and C. Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pages 4–15, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[42] B. Mandelbrot. On the theory of word frequencies and on related markovian models of discourse. In *Structure of Language and its Mathematical Aspects*, volume XII, pages 190–210, 1953.

[43] E. L. Margulis. Modelling documents with multiple poisson distributions. In *Information Processing and Management*, volume 29, pages 215–228, 1993.

[44] A. K. McCallum, R. Rosenfeld, T. M. Mitchell, and A. Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In J. W. Shavlik, editor, *Proceedings of ICML-98, 15th International Conference on Machine Learning*, pages 359–367, Madison, US, 1998. Morgan Kaufmann Publishers, San Francisco, US.

[45] E. L. Miller, D. Shen, J. Liu, and C. Nicholas. Performance and scalability of a large-scale N-gram based information retrieval system. *Journal of Digital Information (online refereed journal)*, 2000.

[46] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[47] D. Mladenic. *Machine learning on non-homogeneous, distributed text data.* PhD thesis, University of Ljubljana, Faculty of Computer and Information Science, 1998.

[48] I. Moulinier, G. Raskinis, and J. Ganascia. Text categorization: a symbolic approach. In *In Proceedings of the Fifth Annual Symposium on Document Analysis and Information Retrieval*, 1996.

[49] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. In *IEEE Neural Networks*, number 12(2), pages 181–201, 2001.

[50] H. T. Ng, W. B. Goh, and K. L. Low. Feature selection, perceptron learning, and a usability case study for text categorization. In N. J. Belkin, A. D. Narasimhalu, and P. Willett, editors, *Proceedings of SIGIR-97, 20th ACM International Conference on Research and Development in Information Retrieval*, pages 67–73, Philadelphia, US, 1997. ACM Press, New York, US.

[51] J. K. Orlov. Linguostatistics: Establishing langauge norms of analysis of the speech process. In *Sprache, Text, Kunst. Quantitative Analysen*, pages 1–55, Bochum, Germany, 1982.

[52] E. Osuna, R. Freund, and F. Girosi. Support vector machines: Training and applications. Technical Report AIM-1602, 1997.

[53] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines, 1998.

[54] J. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin dags for multiclass classification, 2000.

[55] M. Porter. An algorithm for suffix stripping. *Program*, 14/3:130–137, 1980.

[56] Y. S. R. E. Schapire and A. Singhal. Boosting and rocchio applied to text filtering. In *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval*, pages 215–223, Melbourne, Australia, 1998. ACM Press, New York, US.

[57] C. J. v. Rijsbergen. *Information retrieval*. Butterworths, London, 2 edition, 1979.

[58] E. Riloff. Little words can make a big difference for text classification. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *Proceedings of SIGIR-95, 18th ACM International Conference on Research and Development in Information Retrieval*, pages 130–136, Seattle, US, 1995. ACM Press, New York, US.

[59] J. J. Rocchio. In *In Gerard Salton, editor, The SMART retrieval system: experiments in automatic document processing*, pages 313–323, Englewood Clis, US, 1971. Prentice-Hall.

[60] T. Rose, M. Stevenson, and M. Whitehead. The reuters corpus volume 1 - from yesterday's news to tomorrow's language resources. In J. W. Shavlik, editor, *Proceedings of the Third Internaional Conference on Language Resources and Evaluation*, Las Palmas de Gran Canaria, 2002.

[61] M. E. Ruiz and P. Srinivasan. Hierarchical text categorization using neural networks. *Information Retrieval*, 5(1):87–118, 2002.

[62] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management: an International Journal*, 24(5):513–523, 1988.

[63] R. E. Schapire. A brief introduction to boosting. In *IJCAI*, pages 1401–1406, 1999.

[64] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Computational Learing Theory*, pages 80–91, 1998.

[65] R. E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

[66] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In *First International Conference on Knowledge Discovery and Data Mining*, Menlo Park, 1995. AAAI Press.

[67] H. Schutze, D. A. Hull, and J. O. Pedersen. A comparison of classifiers and document representations for the routing problem. In *Research and Development in Information Retrieval*, pages 229–237, 1995.

[68] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys (CSUR)*, 34(1):1–47, 2002.

[69] C. E. Shannon. Prediction and entropy of printed english. *Bell Systems Technical Journal*, pages 50–64, 1951.

[70] A. Singhal, M. Mitra, and C. Buckley. Learning routing queries in a query zone. In *Proceedings of SIGIR-97, 20th ACM International Conference on Research and Development in Information Retrieval*, pages 25–32, Philadelphia, US, 1997.

[71] C. Y. Suen. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, Las Vegas, US, 1994.

[72] K. Summers. Near-wordless document structure classification. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR '95)*, pages 462 – 465, 1995.

[73] A. Sun and E.-P. Lim. Hierarchical text classification and evaluation. In *ICDM*, pages 521–528, 2001.

[74] V. N. Vapnik. Estimation of dependences based on empirical data (in russian). 1979.

[75] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.

[76] J. Weston and C. Watkins. Multi-class support vector machines, 1998.

[77] Y. Yang. An evaluation of statistical approaches to medline indexing. In *Proceedings of the American Medical Informatic Association (AMIA)*, pages 358–362, 1996.

[78] Y. Yang and C. G. Chute. An example-based mapping method for text categorization and retrieval. *ACM Transactions on Information Systems (TOIS)*, 12(3):252–277, 1994.

[79] Y. Yang and X. Liu. A re-examination of text categorization methods. In M. A. Hearst, F. Gey, and R. Tong, editors, *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval*, pages 42–49, Berkeley, US, 1999. ACM Press, New York, US.