Karlsruhe Institute of Technology

Automatic Topic Classification of Media Reports Mentioning KIT

Bachelor Thesis of

Robert Marx

At the Department of Informatics
Institute for Anthropomatics and Robotics
Interactive Systems Labs

Reviewers:	Prof. Alex Waibel
	Dr. Sebastian Stüker
Advisors:	Dr. Sebastian Stüker
	Dipl.-Inform. Jan Niehues

Time Period: 1st Decembre 2015 – 31st March 2016

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, 31st March 2016

Abstract

This thesis evaluates if state-of-the-art text classification methods are able to accurately predict the categories of media reports that mention the "Karlsruher Institut für Technologie"(KIT). It describes methods to prepare the media reports for further processing and then focuses on improving the baseline classification methods, which mainly are a Support Vector Machine (SVM) with linear kernel, Chain Augmented Naive Bayes (CAN) and the Maximum Entropy method. Based on the results of multiple experiments it concludes that the SVM ist the best approach with a F-score of 81.8%.

Deutsche Zusammenfassung

Diese Bachelorarbeit evaluiert, ob moderne Methoden der Textklassifikation in der Lage sind, die Kategorien von Presseartikeln, die das "Karlsruher Institut für Technologie" erwähnen, akkurat vorauszusagen. Dabei wird die Methodik beim Vorbereiten der Presseberichte für die weitere Verarbeitung beschrieben. Ein Großteil der Bachelorarbeit erklärt die Experimente mit der die grundlegenden Methoden verbessert wurden. Die hauptsächlich beschriebenen Methoden sind die Support Vector Machine (SVM) mit linearem Kernel, der Chain Augmented Naive Bayes (CAN) und die Maximum Entropy Methode. Auf Basis der verschiedenen Experimente wird gerschlussfolgert, dass die SVM die beste Methode ist mit einem F-score von 81.8%.

Contents

1. Introduction

1.1 Motivation

Automatic *text classification* is one of many things that *machine learning* (ML) made possible. ML tries to simulate all types of learning processes with a computer. Learning is a phenomenon every human is capable of but it has many facets like "acquisition of new declarative knowledge, the development of motor and cognitive skills through instruction or practice, the organization of new knowledge into general, effective representations, and the discovery of new facts and theories through observation and experimentation" ([Carbonell et al., 1983]).

Many scientists have the goal to create an *artificial intelligence* (AI) that works similar to the human brain and can learn on its own as a consequence. The first step to accomplishing that was the invention of the *perceptron* by Frank Rosenblatt 1958. The perceptron is not enough to correctly model a human brain, but provides the basis for *artificial neural network*s (ANN). ANN's are a popular method for ML nowadays, but AI was and still is in an early stadium.

In the mean time many other ML approaches were discovered, but before the early 90s the most popular *text categorization* (TC) method was *knowledge engineering* (KE). KE classifies documents with the help of rules that are manually defined by domain experts. For example [Hayes and Weinstein, 1990] used such an approach to categorize news stories. Research was focused on developing new and better methods from the 90s on, because the increasing use of the Internet and growing number of electronically stored documents induced a need for methods that can deal with large amounts of data.

[Maron, 1961] introduced probabilistic TC on the basis of Bayes' theorem, but it took over 30 years until it was general consensus that probabilistic approaches to ML could outperform KE and other methods while being easier to use and develop in contrast to KE and ANN's. Other ML approaches simultaneously started to gain traction. ML approaches differ on how they classify documents. They also use different representations of the documents. Under such circumstances it becomes hard to choose one of the many state-of-the-art methods, because their performance highly depends on the data used or the particular task they have to fulfill. Each of these methods has an environment where it performs good. Due to that, research in the last few years has focused more on the preprocessing of documents and on building hybrids of established methods.

1.2 Goal

The goal of this bachelor thesis is to evaluate if it is possible to transfer the task of classifying the category of a media report from human to computer. Human experts can classify documents like media reports with high accuracy, but have to invest a lot of time, in order to create and maintain a database with tens of thousands of documents. A computer is able to complete the classification of many documents in a much shorter time, but in order to make manual work obsolete, the computer needs to achieve results that are as close as possible to manual classification. As measurement for results alongside accuracy, F-score is introduced for the purpose of assessing the quality of the classifier. Ultimately, the aspiration is to obtain high accuracy and F-score. A lot of established and easy-to-use libraries, which provide users fast access to ML, exist. Even though these almost trivialize the creation of multiple classifiers, the performance depends on the selected parameters and the suitability of a classifier for the available data.

1.3 Outline

First of all, this thesis gives an overview of research that has already been done in terms of ML and in particular TC. In the course of it, state-of-the-art technology is revealed. In Chapter 4 the database, consisting of media reports about the KIT, is reviewed. Chapter 3 lists and describes many of the methods that are used in this thesis. The execution and results of those methods are discussed and improved upon afterwards in Chapter 5. The final results of this thesis are presented in Chapter 6.

2. Related work

2.1 Machine learning

In general, machine learning describes the process of an algorithm, often called *learner*, building a model from data to make predictions about more data. There are multiple types of machine learning, but the most interesting type for this thesis is *supervised learning*. It means that the learner has examples with their intended output, called labels, at disposal, so it can learn how to make predictions about new data. Supervised learning is used in this thesis, because a manually built database is available which has all the information a supervised learner needs. *Unsupervised learning* tries to find structure in given examples to build a model describing the data. Because there are no labels available, special techniques are required to get useful predictions, e.g., clustering algorithms, which separate examples into different categories. If the guidelines for manually annotating data with labels are not precise, unsupervised learning could find patterns that allow, e.g., a better separation of data or a different amount of categories.

2.2 Text classification

Automatic classification is a branch of supervised learning and entails *text classification*. Most text documents have a large amount of unique words. This characteristic of data is called dimensionality. High dimensionality can be a problem for general classification methods, but a lot of research into text classification has been done that suggests methods to deal with high dimensionality.

The endeavor of this thesis is to assign the best fitting category to an unseen text and is a special case of text classification named *text categorization*, which is referred to as TC in the remainder of the thesis.

The basic steps to classifying a text are:

1. Data preparation
2. Feature extraction
3. Feature selection or dimensionality reduction
4. Model creation with learner
5. Prediction with model

This thesis evaluates different supervised learning algorithms to find the best approach to text categorization with the database at disposal. The database is described in Chapter 4. The following section will shed light on standard procedures in TC and assess what recent research is focused on.

2.3 State of the art

2.3.1 Preprocessing

Preprocessing includes all the steps that happen before training a model. How exactly data preparation is done depends on the available data and differs widely. In Chapter 5 data preparation for this thesis is described (also see Figure 5.1).

However, there are many different options to consider for feature extraction and selection. Most classifiers can not process raw text data. They need numerical values, which are called *features*. These can be binary, but the most common method to extract features is the *bag of words*, which uses word counts as features. In the last few years research focused on alternative document representations, e.g., [Elberrichi et al., 2008] use WordNet to capture relations between words.

TC works without feature selection, but it is commonly included, because it can improve the performance of many classifiers, e.g., the Naive Bayes classifier, drastically. Feature selection tries to select the features that are the most valuable for classification and fewer features mean lower dimension. Though a dimensionality reduction does not necessarily have a positive impact on the performance of a classifier, it at least accelerates creating it and making predictions. Measures are needed to determine the value of various features. With their aid a feature ranking is created to determine the best features. [Lux, 2012] describes some of the commonly used measures:

- **Term frequency** measures how often a term appears in the data.
- **Mutual information** measures how much features contribute to making the correct classification.
- **Information gain** is based on the concept of *Entropy*, which measures the purity of a category.
- **Chi-squared test**, also seen as χ^2, uses the measure of independence between term and category.

More complex alternatives are, e.g.: Principal Component Analysis (PCA) ([Pearson, 1901]) and Latent Semantic Analysis (LSA) ([Deerwester et al., 1990]). [Yu et al., 2008] report dramatic dimensionality reduction and good classification results with LSA.

2.3.2 Supervised learning

Naive Bayes is a core technique in information retrieval. It has been used in a lot of research over a long period of time and is despite its simplicity and assumptions a viable option in text classification as shown by [Lewis, 1998]. [Domingos and Pazzani, 1997] find it to be optimal for some datasets. But [Kohavi, 1996] indicates that more complex classifiers are better suited for larger amounts of training data and proposes a fusion of Naive Bayes with *decision trees*.

The theory of decision trees and their representation as a graph is easy to understand, and they can perform quite well. A popular method for synthesizing decision trees is the ID3 Algorithm from [Quinlan, 1986] and its successors C4.5 ([Quinlan, 1993]) and C5. In TC they are often used as baseline classifiers, but are commonly outperformed.

[Nigam et al., 1999] show that *maximum entropy* models, also called *logistic regression*, can sometimes outperform Naive Bayes significantly in text classification tasks and their results indicate that it is a promising technique for text classification.

The *nearest-neighbors* algorithm also is a very simple method for TC, but it has an obvious disadvantage in the time it takes to make predictions. It was introduced to TC by [Masand et al., 1992]. This thesis will refer to it as k-NN, because it classifies a document on basis of the k documents that are closest to it. In a high dimensionality space it is time expensive and the accuracy also tends to deteriorate.

Two *neural network* (NN) approaches to TC were made by [Wiener et al., 1995] and [Ng et al., 1997]. A disadvantage of NNs is their time consuming training, but Wiener et al. remark that they can predict multiple topics simultaneously and model a higher order interaction between terms. They also use *Latent Semantic Indexing*(LSI), which is similar to LSA, which was mentioned before, for feature selection.

Support vector machines (SVMs) were introduced by [Vapnik, 1995] and are one of the most promising models for TC, because they outperform many of the older models. They can perform very well without tuned parameters, which is a big advantage. [Joachims, 1998] provides theoretical and empirical evidence which supports that SVMs are suited for text categorization while being easy to use.

2.3.3 Unsupervised learning

Clustering techniques try to divide the data into groups with high similarity. A basic procedure is the *K-means algorithm* which splits the data into k clusters. An efficient K-means algorithm is described by [J. A. Hartigan, 1979] who also published the first iteration in 1975. There are multiple ways to incorporate clustering into text classification. It can be used for feature selection or can supply additional features ([Kyriakopoulou and Kalamboukis, 2006], [Takamura, 2003], [Yong et al., 2009]).

Latent Dirichlet Allocation (LDA), introduced by [Blei et al., 2003], is an unsupervised learning method that is able to find distinct topics in documents and can match the most appropriate topic to a document. [Bíró et al., 2008] use LDA in combination with supervised methods to classify spam.

2.3.4 Ensembles

There are various options for using more than one classifier at the same time. [Rokach, 2010] gives a very detailed overview of ensembles. Ensembles are a very active field of research. Creating an ensemble of classifiers can result in improved performance or more stable results. Negative effects include an increase in memory usage and calculations proportional to the amount of classifiers. The classifiers have to be different to achieve a positive effect. The three basic ensemble options are:

- Train classifiers with various learning methods:

  The resulting predictions are combined and either an algorithm or an additional classifier, sometimes called *president*, has to make a final prediction. This procedure is called *stacking*. *Majority voting*, which returns the most frequent prediction, is an example for an algorithm. [Sakkis et al., 2001] stack a naive Bayes classifier and a k-NN classifier and uses another k-NN classifier as president. [Dzeroski and Zenko, 2004] show that always stacking classifiers is a viable option in comparison to seeking the best single classifier.

- Use different parameters for each classifier

- Train classifiers on different training subsets:

  [Utami et al., 2014] use this approach in combination with SVMs and decision trees. *Boosting* is a more refined approach. It tries to create a strong classifier out of many weak ones by adjusting the weights of misclassified documents. A popular boosting algorithm is AdaBoost developed by [Freund and Schapire, 1995]. [Zhu et al., 2006] augmented it for use with multiple categories. [Xue and Li, 2015] describes the *random forest* classifier, which is a famous ensemble method that combines multiple decision trees trained on random subsets.

3. Text classification

3.1 Naive Bayes

The Naive Bayes classifier is based on Bayes' theorem. [Sebastiani, 2002] gives a detailed overview of probabilistic classifiers like the Naive Bayes classifier. For a given category c and document d, the theorem is:

$$P(c|d) = \frac{P(c) * P(d|c)}{P(d)} \tag{3.1}$$

This theorem calculates the *posterior probability* with knowledge of a *prior probability* and a *likelihood*. In the case of TC, the prior probability of c is the probability of randomly choosing a document of category c. The denominator P(d) can be ignored, because it is a constant for every category. The Naive Bayes classifier gets his name from Bayes' theorem and from making the naive assumption that all of the features of d, which are words or n-grams in TC, are conditionally independent given c. So the likelihood P(d|c) for a document vector $d = (v_1, v_2, ...v_K)$ can be calculated by multiplying the likelihood of each feature. This leads to:

$$P(c|d) = P(c) * \prod_{j=1}^{K} P(v_j|c) \tag{3.2}$$

Then the optimal prediction c* would be the category that maximizes the posterior probability.

$$\text{c}^* = \arg\max_{c \epsilon C} \{P(c|d)\} \tag{3.3}$$

Another assumption can be made. There are different possible prior distributions, e.g., *Dirichlet distribution* and *uniform distribution*. A *maximum likelihood* Naive Bayes classifier is the result of assuming an uniform distribution, which makes the prior probability obsolete:

$$\text{c}^* = \arg\max_{c \epsilon C} \{\prod_{j=1}^{K} P(v_j|c)\} \tag{3.4}$$

3.2 N-gram language models

Speech recognition made up most of the research regarding statistical language models for a long time. Since the beginning of the 21st century they are used for information retrieval tasks frequently. [Hiemstra, 2001] summarizes the expansion of language models and recent research in information retrieval.

3.2.1 Calculating probabilities

A language model should assign high probability to words or n-grams that occur often and vice versa. N-grams are sequences of n words. They can overlap and, for the sentence "This is a nice example", following n-grams can be extracted:

- **1-grams / unigrams**: This, is, a, nice, example
- **2-grams / bigrams**: This is, is a, a nice, nice example
- **3-grams / trigrams**: This is a, is a nice, a nice example

The probability of a sequence of words $w_1 w_2 ... w_T$ is calculated on basis of the *chain rule of probability*:

$$P(w_1 w_2 ... w_T) = \prod_{i=1}^{T} P(w_i | w_1 w_2 ... w_{i-1}) \tag{3.5}$$

This is not a proper approach, because for training the language model only a limited amount of text is available. Thus, a long and unique sequence will be rare and its probability very low. A possible solution to this problem is the *Markov n-gram independence assumption* which approximates the probability of a word by only considering the last n-1 words. The probability of the example above can be calculated with the extracted n-grams.

Using unigrams:

$$P(\text{This is a nice example}) \approx P(This) * P(is) * P(a) * P(nice) * P(example) \tag{3.6}$$

Using bigrams:

$$P(\text{This is a nice example}) \approx P(is|This) * P(a|is) * P(nice|a) * P(example|nice) \tag{3.7}$$

It is easy to calculate these probabilities. For unigrams it is the ratio of occurrences of the word to the total amount of words. N-grams of order n > 1 are estimated with:

$$P(w_i | w_{i-n+1} ... w_{i-1}) = \frac{\#(w_{i-n+1} ... w_i)}{\#(w_{i-n+1} ... w_{i-1})} \tag{3.8}$$

Trigrams are a good compromise between computational effort and accuracy, but usage of 4-gram and 5-gram language models is not uncommon.

3.2.2 Smoothing

Finally, the language model needs a mechanism that accounts for n-grams that were not learned with the training data, because if it would assign a probability of zero to an unknown n-gram, Equation 3.5 would result in a probability of zero for the whole sequence. This mechanism is called *smoothing* and multiple techniques are described in detail by [Peng and Schuurmans, 2003].

3.2.3 Perplexity

The quality of a language model for a certain document is measured by its perplexity (3.9). A small perplexity value indicates high quality.

$$Perplexity = \sqrt[T]{\prod_{i=1}^{T} \frac{1}{P(w_i | w_1 ... w_{i-1})}} \tag{3.9}$$

3.2.4 SRILM toolkit

[Madnani, 2009] also explains n-gram language models and additionally shows how to implement them with Python. For trigrams and higher he proposes using the SRILM toolkit. [Stolcke, 2002] gives an overview of the software's possibilities and its design philosophy. It is mainly for building and applying statistical language models and is widely used in natural language processing tasks like speech recognition or machine translation. Development started in 1995 and is still ongoing. The programming language used is C++ and SRILM runs on UNIX and Windows. Furthermore it provides multiple smoothing algorithms.

3.3 Chain Augmented Naive Bayes

The work of [Peng and Schuurmans, 2003] tried combining Naive Bayes and n-gram language models for text classification and achieved very good results even though it is a simpler technique than many other of the methods in the following sections, which is why his approach is used in this thesis.

The basic idea behind his approach is to use the approximated probability of a sentence, calculated with Equation 3.5, as likelihood in the maximum likelihood Naive Bayes classifier (Equation 3.4). At first separate language models are generated with the training documents of each category. Then a new document will be evaluated by each of the models. The model achieving the highest likelihood represents the most probable category for the given document.

One advantage over the standard Naive Bayes classifier is the consideration of words that are unknown. Also language models can employ advanced smoothing techniques. Another advantage is the involvement of Markov dependencies between adjacent words while the simplifying assumption of Naive Bayes considers words conditionally independent given a category. These differences are noticeable in performance in favor of the proposed approach.

3.4 Scikit-learn

Scikit-learn is a library for the programming language Python and is utilized in this thesis, because it integrates many state-of-the-art machine learning algorithms allowing this thesis to evaluate multiple algorithms in shorter time. It is an open source project aiming at making machine learning accessible to non-specialists and inter-operates with NumPy and SciPy, which are commonly used Python libraries for numerical and scientific purposes. [Pedregosa et al., 2011] give a detailed overview of the still ongoing project.

3.4.1 Feature extraction

The goal of feature extraction is to take the data, in this case text, and process it to get numerical features that can be used by the learner, because most of them can not handle raw text with variable length. Scikit provides utilities for *tokenizing*, which in this case means splitting sentences into words and giving each of them an identification number, as well as *counting* occurrences of each token and *normalizing and weighting* them. In Scikit the class with these capabilities is called *CountVectorizer*.

3.4.1.1 Bag of words

The most popular representation of a text in TC is the *bag of words*. It is a simplifying representation, because it just stores every word on its own. Thus it ignores the order of words in a sentence although some phrases could be useful for classification. It is possible to use n-grams (fragment of n consecutive words) with the bag of words approach, but the main part of semantic information lies in the internal structure of a document, which is not represented by this approach.

3.4.1.2 Term weighting

The bag of words also assigns a weight to every word in the document. A simple weighting scheme is the *term frequency* (tf) which indicates how often a term t is mentioned in a document d, but tf does not tell how important the term is for classification of the document. Some terms with high tf could appear in multiple documents from different categories.

$$tf_{t,d} = \frac{\text{occurrences of t in d}}{\text{amount of all terms in d}} \tag{3.10}$$

Inverse document frequency (idf) indicates how common a term is in the available data. It is obtained by calculating the fraction of documents the term appears in and inversing as well as logarithmically scaling that fraction.

$$idf_t = \log \frac{\text{amount of all documents}}{\text{amount of documents that include t}} \tag{3.11}$$

By multiplying tf and idf (tf-idf) a term can only achieve a high weight, if it appears multiple times in a document while being rare in general. Tf-idf weights can be calculated in Scikit with the *TfidfTransformer* class. Instead of using CountVectorizer and TfidfTransformer, *TfidfVectorizer* can be used.

3.4.2 Feature selection

Although tf-idf weighting should weigh non-informative features low, there is still room for explicit *feature selection*. Due to the high dimensionality of text data the learning process of a classifier can be a computationally expensive task. Reducing the dimensions with feature selection can reduce memory usage and processing time greatly.

There are a lot of words that lack significant information in English as well as in German though they can be important for phrases. These are mostly function words, e.g., conjunctions and articles. In information retrieval they are called *stop words* and are often removed from the data. Instead of removing them beforehand, Scikit also allows passing a list of stop words as parameter to the Count- or TfidfVectorizer class.

Scikit provides feature selection with χ^2 as scoring function, which ranks features according to their score. On basis of this ranking, features are selected. This thesis uses *SelectKBest* to select a variable amount of features.

3.4.3 Support Vector Machines

The support vector machines (SVMs) used nowadays made their first appearance in the work of [Cortes and Vapnik, 1995] about structural risk minimization. Since then they quickly became one of the state-of-the-art methods for classification. [Joachims, 1998] applied SVMs to text categorization with good results and also explained why SVMs are good at text categorization. His research showed that feature selection is not needed, when using SVMs, because they can handle high dimensionality. [Taira and Haruno, 1999] achieved the best results in their work without feature selection, which supports the results of Joachims. Furthermore, parameter tuning is obsolete, because the default parameters theoretically provide the best effectiveness.

A linear SVM has the goal of separating positive and negative examples with a hyperplane. Simply separating the examples is not enough, so it tries to maximize the margin between hyperplane and the closest examples. The best separating hyperplane is determined by the

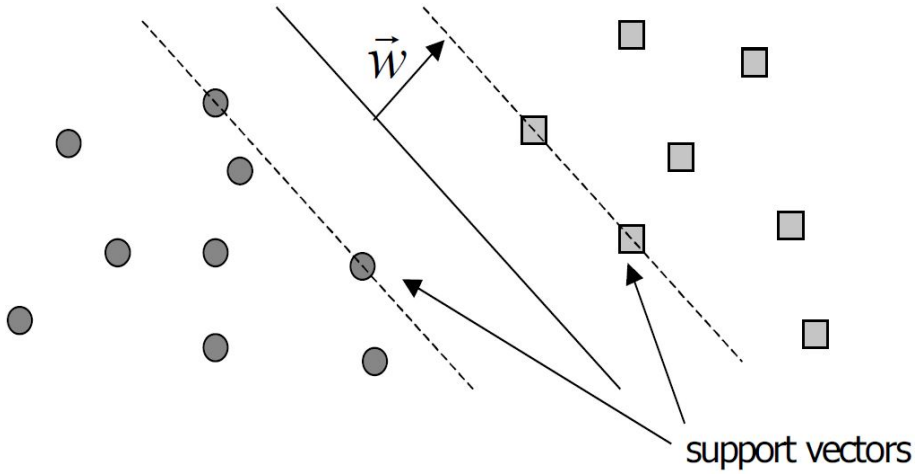


Figure 3.1: Linear support vector machine from [Dumais et al., 1998]

support vectors, which are a small subset of the data. Figure 3.1 shows the SVM approach to linearly separable data with two dimensions. Not every data is linearly separable. A SVM penalizes examples that are on the wrong side of the decision boundary. It can also use *kernel functions* to classify nonlinearly. Kernel functions fit the maximum-margin hyperplane into a new feature space that can be nonlinear and high dimensional.

Scikit includes two popular libraries for SVMs. The class *SVC* is based on *libsvm* and accepts different kernel functions. The library *liblinear* is used by the class *LinearSVC*. It is similar to the SVC class with a linear kernel, but because the underlying libraries are different and they handle multi-class data differently, performance can vary from each other.

3.4.4 Maximum entropy

[Nigam et al., 1999] has a detailed look at maximum entropy classifiers and applies them to TC. On the basis of a data set maximum entropy is a technique that estimates probability distributions. These should be as uniform as possible, because of the principle of maximum entropy. It claims that the distribution which represents the available information best is also the one with maximal entropy. In order to model the available information, constraints are derived from the data. Multiple algorithms are able to use these constraints to find a maximum entropy distribution, e.g., *Improved Iterative Scaling* (IIS) and *Limited-memory BFGS* (LBFGS). Features are needed to create constraints and in TC word counts can be used.

A big difference to the Naive Bayes classifier is, that maximum entropy classifiers do not assume independence between words. These circumstances imply easier use of n-grams as features, but it does not make it a better classification method.

In Scikit the class implementing maximum entropy classifier is called *LogisticRegression*. The included algorithms for the optimization problem are either based on the *liblinear* library or implement *Newton's conjugate gradient* method, LBFGS or the *Stochastic Average Gradient* method.

3.5 Cluster analysis

The goal of *cluster analysis* or *clustering* is to create groups called *clusters* for similar objects and assign every object to one of these groups. In TC possible objects are either

words or documents. This task can be solved by many algorithms with completely different strategies and cluster definitions.

The word clustering tool used in this thesis is named *mkcls* and belongs to the *Giza++* distribution. In [Och, 1999] the creator of the tool explains the theory which it is based on.

3.6 Evaluation

As shown in this chapter, multiple different classification techniques are at issue. It is desirable to use the classifier providing the best outcome in a finished product and it is important to know how it would perform in the future. Therefore methods or formulas for estimating performance are needed.

3.6.1 Accuracy

Accuracy indicates the probability that a random sample is classified correctly and is calculated by dividing the amount of correctly classified samples by the complete amount of samples. It definitely can be a good measure of performance, but can lead to misjudging the quality of a classifier. If a classifier always predicts the most frequent category in an imbalanced dataset it can achieve high accuracy without being able to classify any other categories. To avoid problems like this other measurements can be used.

3.6.2 F-score

The *F-score* or *F-measure* (3.14) is the *harmonic mean* of two other measurements named *precision* (3.12) and *recall* (3.13). A *confusion matrix* (see Figure 3.2) simplifies the calculation of these measures, e.g.: the prediction is a true positive, if the prediction and the actual label of the data are both positive.
If a classifier has high precision for a category c and predicts c, it will probably be a correct prediction. If the classifier also has high recall for c, it will identify a high percentage of documents belonging to c. High precision and low recall will lead to rarely predicting c, but predicting c correctly. In contrast, low precision and high recall will result in often predicting c leading to wrong classifications.

Figure 3.2

		Prediction		
		p	**n**	**total**
Actual value	**p′**	**T**rue **P**ositive	**F**alse **N**egative	P′
	n′	**F**alse **P**ositive	**T**rue **N**egative	N′
	total	P	N	

(a) Confusion matrix

$$Precision = \frac{\#TP}{\#TP + \#FP} \tag{3.12}$$

$$Recall = \frac{\#TP}{\#TP + \#FN} \tag{3.13}$$

$$F - score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.14}$$

(b) Measures

3.6.3 Averaging

Applying these measures for single categories is trivial, but there are two possible methods to get an overall performance estimation:

1. **Micro-averaging:**
   Stack the predictions of each category and then calculate the measures.
2. **Macro-averaging:**
   Calculate the measures for each category and then calculate the average of these measures.

If the data is imbalanced, micro-averaging is the preferred method for estimating the future performance, because macro-averaging assumes a uniform distribution.

3.6.4 Holdout method

The holdout method is splitting the available data into two sets at the minimum. One set is for training a classifier and the other set is for testing it. In order to build a good classifier as much training data as possible is needed so this thesis makes use of 90% of the data as training material and 10% for testing purposes. Unfortunately a wider confidence interval for the performance evaluation is a drawback of a smaller test set. Tuning parameters of classifiers and selecting the best one should not be done by interpreting the results achieved with the test set, because it might improve performance only for that specific dataset. Therefore another 10% of the training set is split off to receive a validation set. Now if performance of classifiers is tested on the validation set there will still be the test set which consists of purely unseen data and can be used to evaluate final performance of the selected classifier. Since the available data is very unbalanced, every set of documents has to have the same distribution of categories. If one category would be underrepresented in one of the sets, it would simultaneously be overrepresented in the others which in turn could severely impact the performance estimation.

3.6.5 Stratified k-fold cross validation

A single validation set can consist of samples that are better or worse to classify than the average sample. So instead of using a single validation set, stratified k-fold cross validation is used in order to get a better representation of the average performance.

In k-fold cross validation the data is split into k mutually exclusive partitions, called folds, and in stratified cross validation the folds share the same distribution of categories as the original dataset. For every single fold a classifier is trained on the data set excluding the fold which in turn represents the validation set to be used for testing. A more detailed background to cross validation methods can be found in [Kohavi, 1995]. His results indicate that 10-fold stratified cross validation is the best method for model selection. K values between 10 and 20 reduce variance sufficient and are not as computationally expensive as higher k values.

4. Database

4.1 Description

The database this thesis is based on is manually administered with the purpose of gathering all media reports mentioning the Karlsruhe Institut für Technik (KIT). At the time of the first classification experiments, it contained reports from July 2011 to November 2015. All types of background information of a report are stored in the database in German. As a result it gets easier to search for specific coherences.

4.2 Content

Apart from the data supplied by each report, e.g., title and authors, the database contains other interesting annotations. The category of a report is the most important annotation for this thesis, but the others can support classification or could be subjects for similar experiments. Tables 4.1 and 4.2 show a few examples of reports with the following annotations:

1. **Category**
   There are 8 main categories and 22 subcategories (see Figure 4.3). The subcategories represent fields of research, e.g., informatics, and are independent of the main categories. It is mandatory that a media report belongs to a single main category. Subcategories however are optional and multiple subcategories can be assigned to a report.

2. **Tendency**
   The tendency is the perceived sentiment of the report towards the KIT. It can be positive, neutral or negative. *Sentiment analysis* is a form of TC, but uses sentiment instead of categories.

3. **Genre**
   This is the literary genre of a report, e.g., interview, news, commentary, blog, gossip or even radio and video.

4. **Thematic priority**
   Arts, politics, economics, science and miscellaneous are the 5 possible themes.

5. **Media attention**
   This considers if the KIT is the main theme, a sub-theme or just a side issue in the

report. Nearly a quarter of the reports have the KIT as main theme and 15% as sub-theme. The majority only mentions the KIT incidentally.

6. **Language**
   A lot of different languages are represented, but for this thesis German is the main concern. Figure 4.2b shows how many reports are German.

7. **Elite**
   Roughly 0.5% of the reports are about the status of the KIT in the *German Universities Excellence Initiative*.

8. **Event**
   Around 11.75% of the reports are about an event.

9. **Ranking**
   Approximately 0.75% are about the the ranking of KIT with respect to other universities.

10. **Content edited**
    The content of a quarter of all reports was edited, which leaves a lot of data unprepared for processing. Data preparation is covered in Section 5.1.

11. **Keywords**
    Keywords like "Fukushima" are either active or not active. They simplify searching for reports about relevant topics.

4.3 Imbalanced data

[Japkowicz and Stephen, 2002] studied the effect of imbalanced data on performance and deduced that it significantly decreases performance of classifiers. Especially for binary classification a lot of methods exist, which can improve performance on imbalanced data. The easiest being over- and under-sampling of data. [He and Garcia, 2009] describe a lot of these methods and multiple others (e.g., [Abe et al., 2004] and [Zhou and Liu, 2010]) try to do the same for multi-class problems, which are an active field of research. Imbalance is not the only problem the available data has. Classifying with a balanced training set will still perform bad, if the training set is too small.

Figure 4.1a shows that the portion of reports with negative sentiment is very small in comparison to the other sentiments. After removing non-German reports there would be even less negative articles. If a classifier for sentiment analysis were built, the classification accuracy of negative reports would probably be very low, because there are not enough examples the classifier could be trained on.

Similarly are the problems with the categories of the reports. The distribution of categories is very imbalanced as Figure 4.3 shows. 42,345 is the sum of reports that are labeled with one of the eight categories, but not every report in the database has its original content available and many reports are not German (see Figure 4.1b). Because this thesis wants to classify the category of German text, the amount of usable reports shrinks significantly. In Table 5.1 the remaining reports per step of data cleaning are listed. Categories 3 and 4 are underrepresented and might hinder the performance of a classifier. A possibility to remedy this, is uniting categories that are underrepresented.

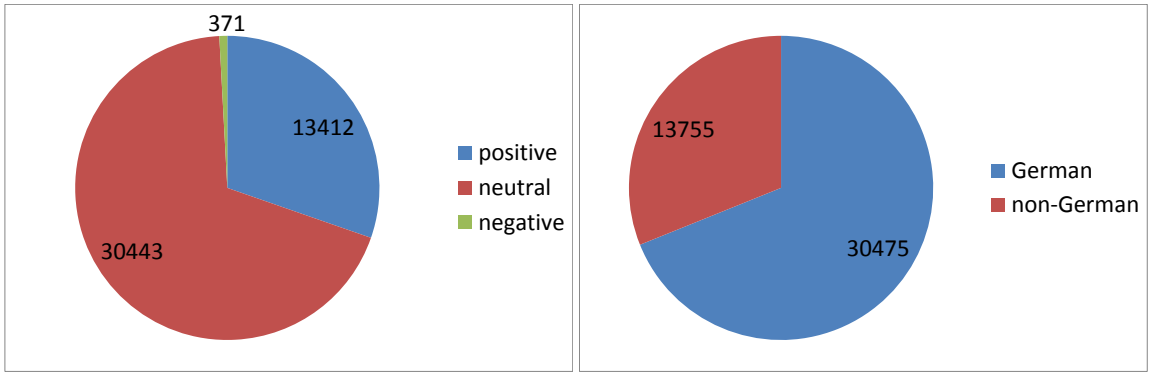
Table 4.1: Excerpt from the content table.

ID	Genre	Thematic Priority	Media attention	Tendency	Text	Language	Elite	Event	Ranking	Edited
276	Nachricht/Bericht	Politik	Randthema	neutral	Befinden sich im ...	deutsch	0	0	0	-1
277	Nachricht/Bericht	Wirtschaft	Randthema	neutral	Von Alfred Dürr ...	deutsch	0	0	0	-1
282	Kommentar/Glosse	Wissenschaft	Randthema	neutral	Ihr Name hat ...	deutsch	0	0	0	-1
283	Nachricht/Bericht	Wissenschaft	Hauptthema	positiv	Karlsruhe. Morgen ...	deutsch	0	-1	0	-1

Table 4.2: Excerpt from the categories table.

ID	Category
276	7. Forschungs- und Hochschulpolitik
276	Energie, Nukleare
277	2. Studium und Lehre
277	Architektur und Bauwesen
282	1. Wissenschaft und Forschung
282	NanoMikro
283	1. Wissenschaft und Forschung
283	Architektur und Bauwesen
283	Katastrophenmanagement

Figure 4.1



(a) Sentiment of media reports in the database.

(b) German reports in the database.

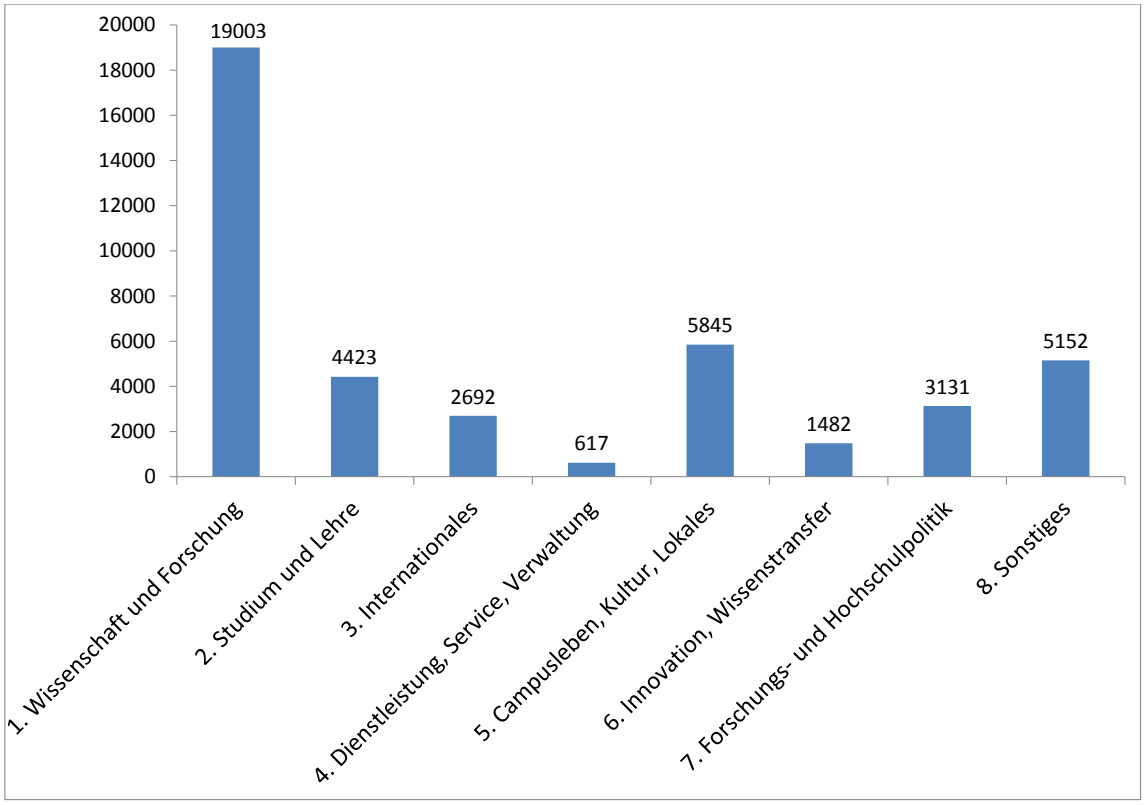


Figure 4.3: Distribution of categories in the database.

5. Experimental results

5.1 Data preparation

Preparing the data for further processing needs to be done before text classification. If the data is in a consistent format, it is easier to read it and store it in the computer's memory. In this thesis two formats are used. For one every line in a file represents a whole media report and for the other every line represents a single sentence of a report.

The database was provided as a Microsoft Excel table, but only a few columns are necessary in the scope of this thesis. Specifically important are content, title, category, language and the content identification number (ID) of each report. They were extracted and saved as three comma-separated values (CSV) files to preserve the structure of rows and columns. Figure 5.1 is a flowchart of the whole data preparation process.

Initially the content of each report had to be unified, because a lot of reports that originated from websites still had their original formatting, meaning that line breaks had to be removed and indentation had to be corrected in order to get one report or sentence per line. The extracted language and category files were prepared by keeping only IDs of German reports for the former and removing all subcategories for the latter. Afterwards the files had to be joined on every content ID to get a file where every report is German and is labeled with its corresponding category.

Since not every report is German nor has a corresponding text in the database, the amount of documents shrinks by 57% overall. The distribution of the documents over the 8 main categories is presented in Table 5.1. In this table step 1 of data preparation is the extraction of labeled reports. Step 2 is the removal of reports without stored text and step 3 is the removal of non-German reports.

Next every document was searched for punctuation characters, e.g., periods, and on every occurrence a line break was inserted in order to create a file with the beforehand mentioned format of one sentence per line. Due to the fact that periods are not only used as end of sentences but in multiple other cases like abbreviations and dates, compromises had to be made with imperfect but sufficient results. Without further need for punctuation, both files underwent a cleaning process stripping all punctuation, e-mail addresses, Internet links and other noise. It replaced German umlauts with their English equivalents and transformed each word into lowercase. Both of the cleaned files were then split to create 10 stratified cross validation sets according to Section 3.6.5.

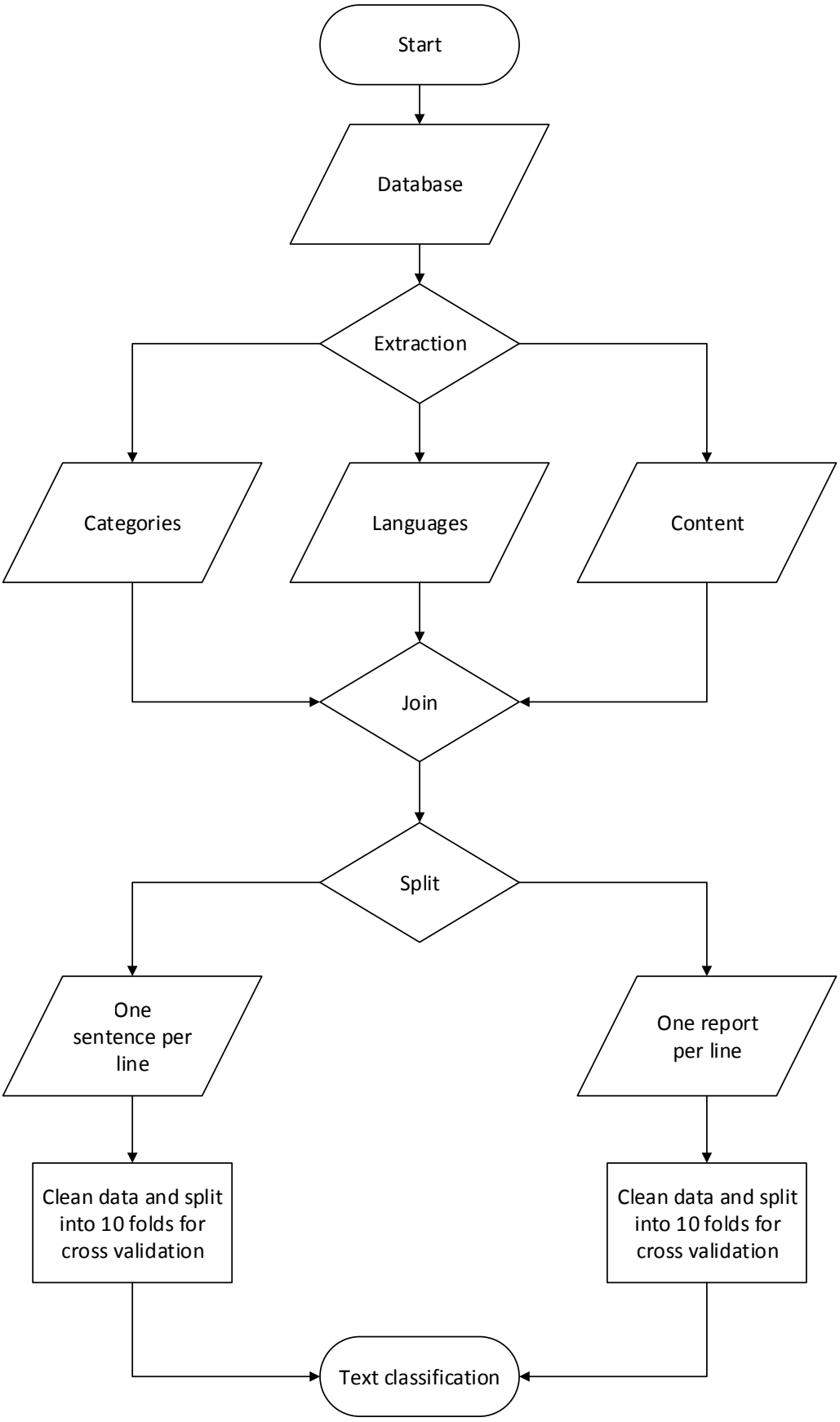


Figure 5.1: Flowchart of data preparation

	Category								Overall	
Step	1	2	3	4	5	6	7	8	Amount	Loss
1	19003	4423	2692	617	5845	1482	3131	5152	42345	-
	45%	10.5%	6.4%	1.5%	14%	3.5%	7.4%	12%	100%	
2	17061	3070	534	306	3195	529	910	2363	27968	33%
	61%	11%	2%	1.1%	11.4%	2%	3.2%	8.5%	100%	
3	11001	2051	170	148	2111	426	724	1741	18372	57%
	60%	11.2%	1%	0.8%	11.5%	2.3%	4%	9.5%	100%	

Table 5.1: Distribution of documents after data preparation.

5.2 Baselines

The major baseline approaches are SVMs, maximum entropy and CAN. SVM was chosen, because it is easy to deploy and supposedly does not need tuning of parameters. Maximum entropy and CAN are used, because the former focuses more on the connection between words and the latter uses language models. All in all they cover a lot of different approaches to text classification and might lead experiments in a certain direction, if one greatly outperforms the others.

Randomly guessing a category without the distribution in mind would only achieve an accuracy of 12.5%. If Category 1 is always predicted instead, the prediction would be right 60% of the time. The goal of the experiments is to maximize classification accuracy and therefore at the minimum excel the 60%. It is important to note that in tables that show results of following experiments the *recall measure equals the accuracy*. Additionally *support* is the amount of media reports, whose category was evaluated.

5.2.1 Chain Augmented Naive Bayes (CAN)

The approach with language models was to train an n-gram language model for each category. SRILM demands a training file with one sentence per line, which was already created, but had to be split into one file per category. On basis of these files and a vocabulary, trigram language models with *Kneser-Ney smoothing*, which is considered the most effective smoothing method, were generated. The vocabulary was created by extracting all unique words from the training files.

Then the generated language models were used to evaluate every media report. The input consisted of one file, containing multiple sentences, per report. The evaluation results were the likelihood and perplexity of each sentence and the entire report. This evaluation procedure took approximately one second for each combination of language model and report. Circa 1,600 reports in each validation set and 8 language models add up to many hours of evaluation.
The resulting likelihood of a report was used in a maximum likelihood naive Bayes classifier (see Equation 3.4), which was very fast.

The baseline experiment was done with the holdout method instead of cross validation and only the achieved accuracy was measured. This lead to the results in Table 5.2 that seemed acceptable, but in reality were pessimistic (see Section 5.3.1). The accuracy on

Category 1 stands out, which was not surprising, because the language model of category 1 had by far the most training documents at disposal. Categories 2, 5 and 8 have good to medium accuracy. All the categories with a very low amount of training documents are seemingly not classifiable. The first idea to solve this was to unite the smaller categories into a single category.

	Category								
	1	2	3	4	5	6	7	8	Overall
Support	890	147	12	14	188	21	44	132	1448
Accuracy	0.931	0.728	0	0.071	0.489	0.381	0.16	0.425	0.759

Table 5.2: CAN baseline.

5.2.2 Scikit

The library Scikit makes it easy to evaluate many classification methods, which is why next to linear SVMs and maximum entropy other similar methods were evaluated at the same time. This allows to make assumptions about the qualities of the different data sets that are used throughout the experiments. The following methods performed best after tuning parameters and thus will be included in following experiments:

- Ridge regression, which is an improved version of linear regression, with a higher tolerance than default.
- Passive-Aggressive (PA), which also is a linear model introduced by [Crammer et al., 2006], with default parameters.
- Multinomial Naive Bayes (MNB) with less smoothing than default.
- K-nearest-neighbors (k-NN), which is different from the others, because it does not construct an internal model of the data, with a k of 10 and weights based on the distance to neighbors.
- Perceptron, a linear model mentioned in chapter 1, with 10 iterations over the trainings data.

The linear SVM did not need parameter tuning, but the maximum entropy classifier improved a lot by weakening the regularization. For the optimization problem of maximum entropy the Stochastic Average Gradient method performed best. In Table 5.3 the exact results with the tuned classifiers can be seen. The linear SVM achieved the highest F-score while needing the least amount of parameter tuning. This is consistent with the research of [Joachims, 1998].

Classifiers	Precision	Recall	F-score
Linear SVM	0.801	**0.807**	**0.799**
Maximum Entropy	**0.802**	**0.807**	0.797
Ridge	0.800	0.805	0.794
PA	0.796	0.802	0.797
MNB	0.789	0.793	0.787
k-NN	0.787	0.795	0.786
Perceptron	0.780	0.787	0.782
Average	0.793	0.799	0.791

Table 5.3: Baselines.

5.3 Improvements

This section lists the experiments that lead to a higher F-score or a decreased computation time without impairing classification results.

5.3.1 Uniting categories and cross validation

For the next experiment all documents labeled with categories 3, 4, 6, 7 and 8 were instead assigned to category 3, which effectively reduced the amount of categories from eight to four. Category 5 was changed to category 4.

With CAN the first noticeable effect was the shorter amount of time needed for likelihood calculation. This is attributable to the use of four instead of eight language models. As Table 5.4 shows the accuracy did not change.

Category	Precision	Recall	F-score	Support
1	0.826	0.926	0.873	890
2	0.613	0.667	0.638	147
3	0.860	0.394	0.540	188
4	0.495	0.453	0.473	223
avg / total	0.758	0.758	0.744	1448

Table 5.4: Results of uniting categories with CAN.

The experiment was repeated, but with cross validation. The results in Table 5.5a show the advantages of stratified cross validation in comparison with the holdout method. The overall accuracy and F-score are about 5% higher with cross validation. This could be attributed to certain folds performing much better and thus increasing the average, but there is no fold standing out positively or negatively. The most likely explanation for such a difference is that the validation set of the holdout method did not contain random reports, but rather the first 10%, which might be especially bad examples. From now on only 10-fold cross validation with random reports was used. Validating ten folds takes much more time, but it is unavoidable, if results as close as possible to reality are desired.

Table 5.5b shows all cross validation results of this experiment. The average f-score improved by 0,8% and the linear SVM achieved the highest f-score again, which increased by 0,6%. It is questionable, if these minor performance gains justify effectively removing four categories. For this reason the original and the united categories were both used in the following experiments.

Fold	Precision	Recall	F-score
1	0.798	0.798	0.789
2	0.798	0.805	0.798
3	0.792	0.800	0.792
4	0.814	0.820	0.812
5	0.800	0.806	0.796
6	0.796	0.802	0.792
7	0.786	0.797	0.787
8	0.798	0.805	0.795
9	0.799	0.806	0.798
10	0.790	0.798	0.787
Average	0.796	0.804	0.795

(a) CAN cross validation results

Classifiers	Precision	Recall	F-score
CAN	0.796	0.804	0.795
Linear SVM	**0.804**	**0.811**	**0.805**
MaxEnt	0.802	0.809	0.803
Ridge	0.801	0.808	0.800
PA	0.799	0.805	0.801
MNB	0.800	0.804	0.801
k-NN	0.791	0.799	0.793
Perceptron	0.787	0.793	0.789
Average	0.797	0.804	0.799

(b) Overall results

Table 5.5: Performance with 4 categories.

5.3.2 Inclusion of titles

The data preparation process was repeated to include the title of each report. Unfortunately not every report had a corresponding title. The experiment was interesting, because it showed how much the performance is affected by additional training data. In the end the inclusion of titles increased the size of the training data by 1.75%. Based on the average results in Table 5.6 the F-score with four categories was 0,1% higher and with eight categories 0,2% higher. These minor improvements however are not statistically significant and the stagnation of the linear SVM's performance indicates that the contribution of titles is too small.

5.3.3 Removal of stop words

It is a common procedure in TC to remove stop words from the data. The expected results are a significant reduction in time needed for feature extraction and training of the model. Additionally it can sometimes improve performance. A list of German stop words from [Doyle, 2016] was used to remove all occurrences of stop words in the data. This reduced the size of the data by more than 20%. Table 5.7 presents mixed results. The experiment affected some classifiers positively and some negatively, but the differences in performance were negligible. Interestingly the linear SVM improved with four categories and worsened with eight categories. The main improvements of the experiment become apparent when reviewing the computation times, which are significantly lower. It can be concluded that the removal of stop words is beneficial for all classifiers, which is why further experiments will operate on the reduced data.

	4 Categories			8 Categories		
Classifiers	Precision	Recall	F-score	Precision	Recall	F-score
CAN	0.798	0.806	0.797	0.794	0.802	0.790
Linear SVM	**0.804**	**0.811**	**0.805**	0.802	**0.809**	**0.800**
Maximum Entropy	0.803	0.810	0.804	**0.803**	0.808	0.799
Ridge	0.803	0.810	0.803	0.802	0.807	0.796
PA	0.801	0.807	0.803	0.798	0.803	0.797
MNB	0.803	0.807	0.803	0.790	0.794	0.788
k-NN	0.792	0.800	0.793	0.788	0.796	0.787
Perceptron	0.791	0.796	0.793	0.781	0.788	0.784
Average	0.799	0.805	0.800	0.794	0.801	0.793

Table 5.6: Results of adding titles to data.

	4 Categories			8 Categories		
Classifiers	Precision	Recall	F-score	Precision	Recall	F-score
Linear SVM	**0.805**	**0.812**	**0.806**	**0.802**	**0.808**	**0.800**
Maximum Entropy	0.802	0.809	0.803	**0.802**	**0.808**	0.798
Ridge	0.802	0.810	0.802	0.801	0.806	0.795
PA	0.800	0.806	0.802	0.797	0.803	0.797
MNB	0.802	0.806	0.803	0.790	0.794	0.788
k-NN	0.792	0.800	0.793	0.789	0.798	0.789
Perceptron	0.793	0.799	0.795	0.782	0.790	0.785
Average	0.799	0.806	0.800	0.796	0.801	0.793

Table 5.7: Performance without stop words.

5.4 Miscellaneous experiments

The following experiments include the use of meta features and feature selection, but did not improve the accuracy and F-score of the various classifiers.

5.4.1 Interpolating language models

It is a common technique to interpolate language models to combine, e.g., a unigram model with a bigram model in order to achieve better performance. A short attempt was made to combine the language models of each category with the model of the entire training data. The motive was the small amount of training data for some of the categories.
SRILM provides linear interpolation of language models with a interpolation weight. Experimenting with multiple different weights always decreased overall performance, but sometimes increased performance on a specific category. All in all the results were negligible.

5.4.2 LM probabilities as features

Instead of using the maximum likelihood Naive Bayes to classify with the calculated likelihood of each category, the scores can be used as feature for other classifiers. In a first experiment only these features were used for training. The idea here is that always choosing the maximum likelihood might be outperformed by other methods, if they can learn distributions where the highest likelihood would classify the wrong category and instead predict the right one. Depending on the amount of categories a single feature vector consisted of only 4 or 8 features, which is a big difference to the high dimensional data used in the other experiments. Most of the classification methods used before did not perform well on the new and low dimensional data. The only two methods coming close to the baseline approach were k-NN and Random Forest.

5.4.3 Chi-squared test

Further feature selection was attempted with the chi-squared test to only use the best features for classification. The original data consists of about 180,000 words or in this case features. In steps of at first 10,000 then 20,000 the original amount of features was reduced and a proportional but low decline in performance could be observed. Unlike stop words, a lot of the removed features are not abundant in the data, so a lot of unique or rare words have to be removed to achieve a similar reduction of data as in the previous experiment. Mostly the linear classification models still performed well. With only 20,000 features the f-score of the linear SVM, which was consistently the best classifier so far, decreased by only 1.2% while training was 60% faster. However, because the predictions worsened and faster classification is only subsidiary, the data without stop words remained the basis for experiments.

5.4.4 Clustering words

At first similar words were grouped into one of 1,000 clusters with the mkcls tool. Then for one experiment the words were exchanged with their corresponding clusters and for the other the clusters were used as meta features. Former can be helpful, if the available data is sparse, because unique words normally have low weights, but the corresponding clusters, that appear multiple times in the data, do not. Both experiments were not successful. The best F-score achieved in the first experiment, with a k-NN classifier (k=5), was 1,8% worse than the linear SVM and the meta features did not increase performance of the SVM either.

5.5 Stacking classifiers

After trying to improve performance by tuning parameters and features of single classifiers, the next approach was to use the best performing classifiers in an ensemble. During cross validation each of the classifiers had predicted every document of the trainings set. These predictions were combined into a new meta trainings set with the same labels as the former set. So one of the new feature vectors contained the category of the original document and the predictions the different classifiers made for that document. The ensemble needs an additional classifier called president to make a final prediction. Cross validation was used again to choose the best performing president and to evaluate the best possible setup for the meta trainings set, because using the predictions from a large amount of classifiers is not necessarily good for performance. Linear models performed poorly on the meta data. In Table 5.8 the two best presidents are listed namely a Random Forest classifier, which used an ensemble of 100 decision trees, and a SVM with a *radial basis function* (rbf) kernel. The best performing ensembles:

- Four categories: CAN, k-NN, linear SVM, MNB and Maximum Entropy
- Eight categories: CAN, linear SVM, MNB and Maximum Entropy

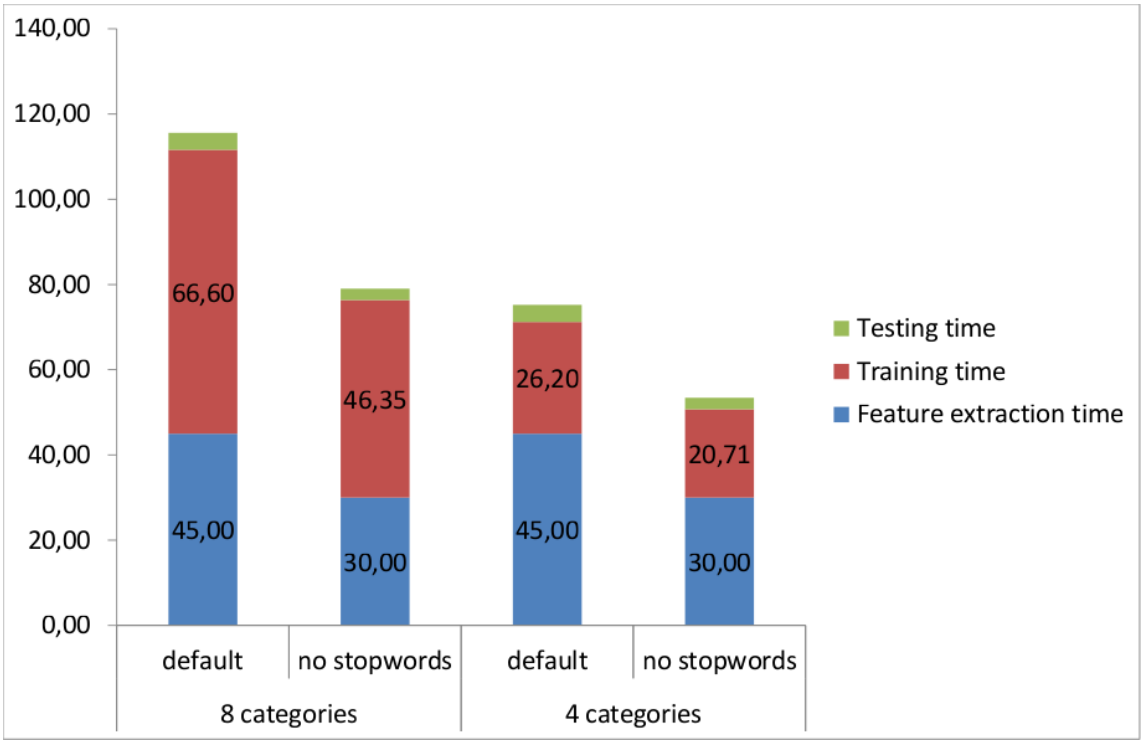
	4 Categories			8 Categories		
Classifiers	Precision	Recall	F-score	Precision	Recall	F-score
Random Forest	**0.813**	**0.818**	**0.813**	0.806	0.812	0.805
rbf SVM	0.811	0.817	0.812	**0.810**	**0.815**	**0.807**

Table 5.8: Performance of ensembles.

5.6 Computation time

Most of the experiments were done on a computer with an Intel i5 2500k 4x3.3GHz processor and 12 gigabytes of memory. Figure 5.2 depicts the average elapsed time of the classifiers using Scikit. There are clear differences between the data sets. The use of less categories did not shorten feature extraction, but training was much faster in comparison to the original data set. Removing stop words was equally beneficial for both sets.

Figure 5.2: Elapsed time after cross validation in seconds.



In Table 5.9 the training and classification times of single classifiers is listed. The linear SVM classifies fastest and k-NN slowest, because it is a *lazy learner* and classifies without building a model beforehand. Therefore it almost does not need time for training. The maximum entropy method however is the slowest in training a model. Training language models and evaluating sentences is even slower, e.g., evaluating a single validation set takes about 1,200 seconds if done parallel on multiple cores.

	4 Categories		8 Categories	
Classifiers	Training	Test	Training	Test
Linear SVM	16.89	**0.040**	26.736	**0.058**
Maximum Entropy	**85.77**	0.071	**227.772**	0.115
Ridge	28.937	0.073	55.636	0.124
PA	2.683	0.073	5.133	0.111
MNB	0.549	0.070	0.924	0.115
k-NN	0.213	**18.081**	**0.212**	**18.032**
Perceptron	4.06	0.072	8.059	0.120

Table 5.9: Elapsed time in seconds.

5.7 Summary

Until now all experiments were done to choose the best classification method and the best feature set. The best single classifier was the linear SVM, but the ensemble performed slightly better. The experiments also revealed that the data with titles and without stop words performed best. Uniting categories also lead to a small performance gain, but predicting only four categories is not desired for real data as long as the database has eight categories. Both of the methods were tested as well as the dataset with four and with eight categories.

The final results in Table 5.10 show that the linear SVM outperformed the ensemble in every aspect. There are several possible reasons for that. The linear SVM might scale better to a larger training set or the test set is an outlier where the SVM can perform especially good. It is also possible that the president of the ensemble was overfitted and only performs good on the validation sets. Even if the ensemble would have better performance, the linear SVM is a lot easier to use, because it did not need tuning, and the training and classification times are much faster.

	4 Categories			8 Categories		
Classifiers	Precision	Recall	F1-score	Precision	Recall	F1-score
Linear SVM	**0.818**	**0.823**	**0.818**	**0.808**	**0.812**	**0.803**
Ensemble	0.815	0.820	0.817	0.805	0.807	0.800

Table 5.10: Final test results.

6. Conclusion

The goal of this thesis was to evaluate if the task of classifying the categories of media reports mentioning the KIT can be automated. Because the categories are manually annotated at the moment, a text classifier needs too achieve very high accuracy to replace manual work. The approach to reach the goal of the thesis was composed of multiple parts. At first the data had to be prepared, because it needed to be in a format that is easy to process. Many of the reports were not manually edited after copying them from web pages. This lead to a lot of unusable data inside of the reports. After data preparation only half of the available reports were left, because they were not German or because they did not have stored text data available.
The beforehand selected classification baseline methods were then used with the prepared data to obtain a first baseline. The linear SVM performed good right from the beginning. To further improve the baseline the amount of categories was reduced, which improved overall performance. Other experiments like word clustering and interpolation of language models failed, but the inclusion of titles and the removal of stop words enhanced speed of classification and increased the F-score of many classification methods. Finally the best classifiers were chosen to create an ensemble. In experiments with cross validation this stacking procedure turned out to perform better than each of the single classifiers.

The goal of the thesis was partly met. The best result obtained was a F-score of 81.8% and an accuracy of 82.3% with a linear SVM. In comparison with the minimum accuracy of about 60%, which is the proportion of reports belonging to the first category, the results are good. The SVM operated on a modified data set that had four categories instead of the original eight and no stop words. Additionally the titles of reports were included. With the original eight categories the same classifier achieved a F-score of 80.3% and an accuracy of 81.2%. These are the outcomes that could be expected, if manual categorization would be automated, but the results were not good enough to support automating categorization. All of the classification methods performed poorly on five of the eight categories. The only category that was correctly predicted most of the times was the first category. This can be attributed to the massively imbalanced data set. Another problem was the size of the test set, which was too small to back the results in a statistically solid manner. The cross validation results show that, with more data or a different holdout split, the presented stacking classifier might have performed better than the linear SVM. Also, the linear SVM needed the least amount of tuning compared to all other applied methods and was one of the fastest methods. This matches the claims of many researchers that SVMs are one of the best text classification methods to date.

The experimental results imply that additional data has a positive effect on prediction accuracy. That is why the approach of this thesis could obtain considerably better results, if it would be applied to a larger corpus of media reports in the future. Minor improvements could be, e.g., the advanced usage of meta features or other ensemble methods than stacking. It could also be beneficial to deal with the data imbalance with under- and oversampling or superior methods, but without compromising the amount of categories.

Bibliography

[Abe et al., 2004] Abe, N., Zadrozny, B., and Langford, J. (2004). An iterative method for multi-class cost-sensitive learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 3–11. ACM.

[Bíró et al., 2008] Bíró, I., Szabó, J., and Benczúr, A. A. (2008). Latent dirichlet allocation in web spam filtering. In *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web*, AIRWeb '08, pages 29–32, New York, NY, USA. ACM.

[Blei et al., 2003] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022.

[Carbonell et al., 1983] Carbonell, J. G., Michalski, R. S., and Mitchell, T. M. (1983). Machine learning: A historical and methodological analysis. *AI Magazine*, 4(3).

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[Crammer et al., 2006] Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585.

[Deerwester et al., 1990] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, 41(6):391–407.

[Domingos and Pazzani, 1997] Domingos, P. and Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2):103–130.

[Doyle, 2016] Doyle, D. (2016). German stopwords. `http://www.ranks.nl/stopwords/german`. Accessed: 12.03.2016.

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

[Dzeroski and Zenko, 2004] Dzeroski, S. and Zenko, B. (2004). Is combining classifiers better than selecting the best one? In *MACHINE LEARNING*, pages 255–273. Morgan Kaufmann.

[Elberrichi et al., 2008] Elberrichi, Z., Rahmoun, A., and Bentaallah, M. A. (2008). Using wordnet for text categorization. *Int. Arab J. Inf. Technol.*, 5(1):16–24.

[Freund and Schapire, 1995] Freund, Y. and Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting.

[Hayes and Weinstein, 1990] Hayes, P. J. and Weinstein, S. P. (1990). Construe/tis: A system for content-based indexing of a database of news stories. In *IAAI*, volume 90, pages 49–64.

[He and Garcia, 2009] He, H. and Garcia, E. A. (2009). Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284.

[Hiemstra, 2001] Hiemstra, D. (2001). *Using language models for information retrieval*. Taaluitgeverij Neslia Paniculata.

[J. A. Hartigan, 1979] J. A. Hartigan, M. A. W. (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108.

[Japkowicz and Stephen, 2002] Japkowicz, N. and Stephen, S. (2002). The class imbalance problem: A systematic study. *Intell. Data Anal.*, 6(5):429–449.

[Joachims, 1998] Joachims, T. (1998). *Machine Learning: ECML-98: 10th European Conference on Machine Learning Chemnitz, Germany, April 21–23, 1998 Proceedings*, chapter Text categorization with Support Vector Machines: Learning with many relevant features, pages 137–142. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Kohavi, 1995] Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143. Morgan Kaufmann.

[Kohavi, 1996] Kohavi, R. (1996). *Scaling up the accuracy of Naive-Bayes classifiers: A decision-tree hybrid*. AAAI Press, Menlo Park, CA (United States).

[Kyriakopoulou and Kalamboukis, 2006] Kyriakopoulou, A. and Kalamboukis, T. (2006). Text classification using clustering. In *In Proceedings of the ECML-PKDD Discovery Challenge Workshop*.

[Lewis, 1998] Lewis, D. D. (1998). Naive (bayes) at forty: The independence assumption in information retrieval. pages 4–15. Springer Verlag.

[Lux, 2012] Lux, E. (2012). Feature selection for text classification with naive bayes.

[Madnani, 2009] Madnani, N. (2009). Querying and serving n-gram language models with python. *The Python Papers*, 4(2).

[Maron, 1961] Maron, M. E. (1961). Automatic indexing: an experimental inquiry. *Journal of the ACM (JACM)*, 8(3):404–417.

[Masand et al., 1992] Masand, B., Linoff, G., and Waltz, D. (1992). Classifying news stories using memory based reasoning. *Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'92)*, (15):59–64.

[Ng et al., 1997] Ng, H. T., Goh, W. B., and Low, K. L. (1997). Feature selection, perceptron learning, and a usability case study for text categorization. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '97, pages 67–73, New York, NY, USA. ACM.

[Nigam et al., 1999] Nigam, K., Lafferty, J., and Mccallum, A. (1999). Using maximum entropy for text classification.

[Och, 1999] Och, F. J. (1999). An efficient method for determining bilingual word classes. In *Proceedings of the Ninth Conference on European Chapter of the Association for Computational Linguistics*, EACL '99, pages 71–76, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Pearson, 1901] Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2(11):559–572.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Peng and Schuurmans, 2003] Peng, F. and Schuurmans, D. (2003). *Combining naive Bayes and n-gram language models for text classification.* Springer.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Rokach, 2010] Rokach, L. (2010). Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39.

[Sakkis et al., 2001] Sakkis, G., Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Spyropoulos, C. D., and Stamatopoulos, P. (2001). Stacking classifiers for anti-spam filtering of e-mail. In *in 'Empirical Methods in Natural Language Processing*, pages 44–50.

[Sebastiani, 2002] Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47.

[Stolcke, 2002] Stolcke, A. (2002). Srilm – an extensible language modeling toolkit. In *Proc. Intl. Conf. on Spoken Language Processing*, volume 2, pages 901–904.

[Taira and Haruno, 1999] Taira, H. and Haruno, M. (1999). Feature selection in svm text categorization. In *AAAI/IAAI*, pages 480–486.

[Takamura, 2003] Takamura, H. (2003). *Clustering Approaches to Text Categorization.* PhD thesis.

[Utami et al., 2014] Utami, I. T., Sartono, B., and Sadik, K. (2014). Comparison of single and ensemble classifiers of support vector machine and classification tree. *Journal of Mathematical Sciences and Applications*, 2(2):17–20.

[Vapnik, 1995] Vapnik, V. N. (1995). The nature of statistical learning theory.

[Wiener et al., 1995] Wiener, E., Pedersen, J. O., and Weigend, A. S. (1995). A neural network approach to topic spotting.

[Xue and Li, 2015] Xue, D. and Li, F. (2015). Research of text categorization model based on random forests. In *Computational Intelligence Communication Technology (CICT), 2015 IEEE International Conference on*, pages 173–176.

[Yong et al., 2009] Yong, Z., Youwen, L., and Shixiong, X. (2009). An improved knn text classification algorithm based on clustering. *Journal of Computers.*

[Yu et al., 2008] Yu, B., ben Xu, Z., and hua Li, C. (2008). Latent semantic analysis for text categorization using neural network. *Knowledge-Based Systems*, 21(8):900 – 904.

[Zhou and Liu, 2010] Zhou, Z.-H. and Liu, X.-Y. (2010). On multi-class cost-sensitive learning. *Computational Intelligence*, 26(3):232–257.

[Zhu et al., 2006] Zhu, J., Rosset, S., Zou, H., and Hastie, T. (2006). Multi-class adaboost.