Comparison of Text Classification Techniques Supporting Journalistic Data Structuring

Barbara Sikora

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2017

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, September 15, 2017

Barbara Sikora

Contents

Abstract

The amount of free text data including online news, blogs, e-mails or social media communication is constantly rising through the development of the World Wide Web. These types represent unstructured data and the missing structure complicates searching in it or getting quick information out of it. Thus, the need for efficient methods for analyzing this kind of texts is growing. The thesis will focus on the problem of automated news classification to sort articles according to their topic. This serves to facilitate handling data in online news services for tasks like tagging or categorizing.

The data the thesis will work with are English news articles at the example of articles from the news agency *The Guardian*. A pre-processing of the unstructured texts is generally required including tokenization, stop word reduction and filtering of important words. Subsequently, the main part of the process follows, which is the reimplementation of the algorithms for the text classification task. The thesis major focus lies on the algorithm called Semantic Fingerprinting, which uses a neuroscience rooted mechanism to detect the similarity between natural linguistic documents. It is generally based on the theory of Semantic Folding and the Hierarchical Temporal Memory (HTM) theory of Jeff Hawkins. The theory of HTM describes a machine learning model that has structural and algorithmic properties of the human neocortex. This method will be compared to two classical algorithms, the k-nearest-neighbor classifier and the Naive Bayes algorithm. Finally, the thesis will evaluate the performance and the results of the algorithms for analyzing news articles, with special emphasis on the Semantic Fingerprinting method in comparison to the two classical classifiers.

Kurzfassung

Durch die Entwicklung des World Wide Webs stieg die Anzahl an digitaler Medien in den letzten Jahren stark an. Im Speziellen beschäftigt sich diese Arbeit mit der Zunahme an digitalen Texten, dazu zählen Texte auf online Nachrichtendiensten, Blog-Nachrichten, E-Mails oder Nachrichten in sozialen Netzwerken. Diese Art von textuellen Medien gehört zu der Kategorie der unstrukturierten Daten und durch ihre fehlende Struktur, wird das Herausfiltern sowie das Suchen von darin enthaltenen, konkreten Informationen erschwert. Durch diese Problemantik wächst das Interesse an Technologien zur Analyse von unstrukturierten Texten. Die Arbeit widmet sich dem Problem der automatisierten Nachrichten Klassifizierung und konzentriert sich auf das Sortieren von Artikeln hinsichtlich ihres Themengebietes. Die Ergebnisse der Untersuchung könnten Nachrichtendiensten bei der Beschriftung und Einordnung von Artikeln helfen und deren Arbeit erleichtern und beschleunigen.

Bei den, in dieser Arbeit verwendeten Daten handelt es sich um englische Nachrichtenartikel. Als Beispieldaten für diese Art von Texten wurden Artikel von dem online Nachrichtendienst *The Guardian* genützt. Für die Verarbeitung der Texte ist ein Pre-Processing Schritt notwendig, einschließlich Tokenization, Stop-Word Entfernung und das Selektieren von wichtigen Wörtern. Anschließend folgt der Hauptteil der Arbeit, die eigene Umsetztung der Algorithmen für die Textklassifizierung. Der Fokus liegt dabei auf der Semantic Fingerprinting Methode, die einen neurowissenschaftlichen Ansatz nützt um Gemeinsamkeiten zwischen natürlich sprachlichen Texten herauszufinden. Es basiert auf der Semantic Folding Theorie sowie auf der Hierarchical Temporal Memory Theorie von Jeff Hawkins. Dabei handelt es sich um ein Machine Learning Modell, das strukturelle sowie algorithmische Eigenschaften des menschlichen Neokortex aufweist. Diese Methode wird mit zwei klassischen Algorithmen verglichen, dem k-nearest-neighbor Klassifikator und dem Naive Bayes Algorithmus. Abschließend werden die Leistungen und die Ergebnisse für das Analysieren von Nachrichtenartikeln bewertet, mit besonderem Augenmerk auf die Semantic Fingerprinting Methode im Vergleich zu den klassischen Algorithmen.

Chapter 1

Introduction

The following chapter describes the purpose of the thesis as well as the solution approach and the composition of the chapters included in this work.

1.1 Problem Description

The development of the World Wide Web in recent years caused simultaneously a rapid increase of free text data including online news, social media communication, blogs or e-mails. This kind of data counts in general to unstructured or semi structured data. Unstructured data represents the largest part of all digital data and it cannot be fit in a database without former processing. The missing structure additionally complicates searching in the data or getting quick information out of it until it got filtered, sorted and categorized to finally gain easy access to the contained subjects [59, 64]. Thus, the need for efficient methods for analyzing unstructured or semi-structured texts is growing. The combination of functions from the fields text mining, machine learning and natural language processing enables getting valuable information from the data and to discover structures from the electronic documents, which forms the relevant subjects. Text mining itself consists of diverse sub-fields like information retrieval, text classification or summarization [1, 36], but its main purpose is to offer users the possibility to extract important information from texts and further analyze it with machine learning and natural language processing [18, p. 4].

This thesis focuses on the problem of automated news classification, a specific subfield of text classification. Text classification itself is in general about labeling natural linguistic texts, such as articles, with classes, which can be compared with topics, on basis of a comparison of its content and a predefined set of training data [33, p. 1]. This serves to facilitate handling data in online news services for tasks like tagging or sorting of articles according to their topic. Thus, the thesis is motivated by the following research question: how can journalistic data be classified to facilitate data organization and filtering. To answer this question, three different algorithms for text classification are reimplemented, tested and compared.

1.2 Solution Statement and Contribution

The data, the algorithms work with, are in general English news articles. As an instance for this kind of data articles from the news agency *The Guardian* are used. After collecting sufficient articles per class, a pre-processing of the unstructured texts is generally required including splitting, stop word reduction and filtering of important words. The overall goal of this step is "to convert the words of the documents into numerical representations" [8, p. 2] in order that the algorithms can work with. Subsequently, the main part of the process follows, the reimplementation of the algorithms for the text classification task.

The thesis main focus lies on the algorithm called Semantic Fingerprinting, which uses a neuroscience rooted mechanism to detect the similarity between natural linguistic documents and is generally based on the theory of Semantic Folding and the Hierarchical Temporal Memory theory of Jeff Hawkins. The theory of HTM describes a machine learning model that has structural and algorithmic properties of the neocortex. It generates individual semantic fingerprints for each text and topic, to finally compare these fingerprints in order to detect the most overlaps [38, p. 6].

The two classical algorithms are the k-nearest-neighbor classifier and the Naive Bayes classifier. The first one focuses on a similarity function to detect the most similar article in the training data to the new unseen document and assigns its class to the new one [18, p. 9] [7, p. 1]. The second one is a probabilistic classifier, which uses the conditional and joint probability for calculating the most likely topic by dismembering the texts and working on each word individually [43, p. 3].

Finally, the thesis evaluates the performance and the results of the algorithms for analyzing news articles, with special emphasis on the Semantic Fingerprinting method in comparison to the k-nearest-neighbor and Naive Bayes classifier.

1.3 Outline

The thesis consists of six chapters, including the introduction. The subsequent chapter provides a general overview of the subject text analysis and a selection of related projects. A detailed insight into the field of text classification follows including the preprocessing procedure and the three algorithms used for the comparison, the k-nearest-neighbor classifier, the Naive Bayes and the Semantic Fingerprinting method. In chapter 3 the development environment is described as well as the used technologies and the data source the algorithms work with. In chapter 4 the implementation of the pre-processing procedure and the algorithms are explained in detail, including multiple source code excerpts. The final results as well as the comparison of the algorithms are contained in chapter 5 with specific information about the development process and the different progressions of the methods. Finally, all findings are summed up in chapter 6 with closing remarks and additional information about challenges and potential future work.

Chapter 2

Basics and State of the Art

The subsequent chapter provides a precise insight into the field of text analysis, relevant projects and important steps preceding the prime purpose, the text classification part. In this chapter the three chosen algorithms, k-nearest-neighbor and Semantic Fingerprinting, will be dissected in detail with the main emphasis on the last technique. Finally, the explication of multidimensional scaling, a useful technology utilized in the corresponding thesis project, follows.

2.1 Introduction to Text Analysis

In recent years a rapid increase of free text data on the World Wide Web was witnessed [7]. This data originates from a variety of sources such as online news, blogs, e-mails, digital libraries or social media communication and are counting to unstructured or semi structured data [18, p. 4].

Structured data can be described as information represented in form of columns and rows which can be easily stored in databases and ordered by several parameters or processed via individual keys. This kind of data constitutes only five to ten percent of all digital data. By comparison, unstructured data represents up to 80% [49, 18] and can be explained as a "massive unorganized conglomerate of various objects that are worthless until identified and stored in an organized fashion". It cannot be fit in a database until the information got searched through, sorted and categorized to finally gain easy access to the subjects [64]. Semi structured data can be located between these two extremes. This signifies, this kind of data does not fit in a relational database either, although it possesses some organizational properties that facilitate analyzing it. XML[1] or JSON[2] documents can be mentioned as samples for semi structured data [59].

The absence of structure in text documents complicates searching in it or getting quick overviews. Thus, the need for efficient methods for analyzing texts is growing. The collaboration of techniques from the fields of text mining, machine learning and natural language processing makes it possible to get meaningful information from the data and to automatically discover interesting and non-trivial patterns from the electronic documents. Text mining can be seen as an hypernym, which consists of diverse

[1] See also https://www.w3.org/standards/xml/core.
[2] See also https://www.w3schools.com/js/js_json_intro.asp.

fields like information retrieval, text categorization, visualization or summarization [1, 36]. However, its main purpose is to enable users to extract relevant information from texts and further analyze the extracted information using machine learning and natural language processing [18, p. 4].

For the procedure of extracting important information, functionalities from natural language processing are needed. They are used to linguistically parse sentences and paragraphs with the purpose of picking the texts into single words, key concepts, verbs and proper nouns [18, p. 5]. Summarized, systems using natural language processing have to determine the structure of a text document to define the "Who?", "What?","Why?" and "Where?" [25, p. 48].

When going further to the deeper tasks like text categorization or summarization, a machine learning approach is partly needed, inter alia for calculating weights or similarities [1, p. 43]. Especially in the field of text classification, machine learning methods gained increasing popularity since the increase of digital information started [33, p. 8] and algorithms like the Bayesian classifier, Decision Tree or k-nearest-neighbor (kNN) have been extensively studied for calculating predictions on new and unseen data [1, p. 6] [18, p. 8].

Text classification, also referred to as text categorization or topic spotting, can be described as an operation, which labels natural linguistic texts, such as articles, e-mails or twitter posts, with topical categories on basis of a comparison of its content and a predefined set of training data [33]. A more precise definition of the classification problem, has been defined by Charu C. Aggarwal. At the beginning, we are given a set of data, where each text is related to a specific topic or so called class value. This data is the training set and it describes the features[3] of the text belonging to a relative topic. When searching the class for a new, unknown test data, this training set is used to compare it with the features of the unknown data and subsequently, to predict the class value [1, pp. 163–164]. Summarized, it means grouping a number of documents according to its topic by analyzing its content. This task can be again modified to smaller sub-challenges like language detection, sentiment analysis or as already mentioned topic detection. All of these subdivisions of the classification task, use classifiers, which have the same aim. They sort text by their subject and assign one or more labels or terms to them [16, p. 119].

The problem of classifying text finds a variety of applications in the domain of text mining. Some use cases where text classification is commonly used are

- news filtering and organization,
- document organization and retrieval,
- opinion mining and
- email classification and spam filtering [1, pp. 164–165].

This thesis will be dealing with one of these applications, the news filtering and organization field.

These days the majority of news services, which often provide their content on online platforms as well, are based on an electronic entity. The sorting of articles into categories or the tagging of them, which means assigning a label or a term to a text, is usually done manually, and by the growth rate of information, the process does not scale up well. For

[3] The term "feature" is synonymously used for the term"word" in this field.

this purpose, automated programs and methods can be helpful for news categorization and the labeling in online news agencies [1, 10, 63].

In order that algorithms can deal with the text documents, "one has to convert the words of the documents into numerical representations". This section is named the pre-processing procedure and consists of two tasks, the feature extraction and the feature selection [8, p. 2]. The first step should clearly define the structure of the language and eliminate a maximum of language dependent properties with the aid of tokenization, stop words removal and stemming [18, p. 5]. The difficulty in sum is to generate a list of terms that describes the documents sufficiently [8, p. 3]. The second step in the pre-processing part has its main responsibility in removing features which hold only less important information in it and this is done by allocating each word a weight, which describes its importance, and to finally construct a vector space. It describes the complete training set including all documents as points and all terms as axes. These two tasks are explained in detail in section 2.3.

After pre-processing, the prepared data can be moved forward to the particular algorithms as numerical vectors related to the initial words in the documents. This thesis focuses on the k-nearest-neighbor classifier, the Naive Bayes classifier and the Semantic Fingerprinting method.

The main part of the k-nearest-neighbor is a similarity function, which detects the most similar documents to the new unseen data[18, p. 9] [7]. The second algorithm, the Naive Bayes, is a probabilistic classifier. It uses the joint probability of words and the related class to predict the category for test instances [43, p. 3]. Semantic Fingerprinting is novel and in comparison to the previous mentioned methods which are using word statistics, this one uses a neuroscience rooted mechanism to detect similarity between natural linguistic documents [38, p. 6]. The algorithms are detailed in section 2.4.

2.2 Existing projects

The field of text mining or in particular text classification gained an ever-expanding prominent status in recent years and thereby multiple projects and papers originated around this field. Thorsten Joachims, Charu C. Aggarwal and Yiming Yang can be highlighted as representatives for several papers [1, 17, 44] and in the following section, a selection of projects are explained. The last project is of particular importance for this thesis, because it is the foundation for the reimplementation in the thesis project.

2.2.1 uClassify

The project, uClassify[4], was launched in 2008 in Stockholm by a team including Jon Kågström, the founder of the project, Roger Karlsson and Emil Ingridsson. It is a free online machine learning web service for text classification and translation. It offers the possibility to use classifiers and create classifiers on your own, when creating an account. It also provides APIs via JSON and XML and has established a big community, which already has published up to 3570 classifiers including sentiment analysis, topic spotting, language detection, mood analysis, age detection or gender detection. The free API

[4] Project can be tested on https://www.uclassify.com/.

allows up to 1000 requests a day, but if the calls exceed the given number, the account can be upgraded. An academic license can be requested as well [51].

One interesting type of classifier, made with uClassify, called Typeanalyzer, tries to analyze the personality of blogs and treats them like persons. In the end it refers different characteristics to them, describing their personality [62].

2.2.2 NewsWeeder

NewsWeeder[5] is a project developed by Ken Lang, which he presented on the 12th International Conference of Machine Learning. It is a filtering system for so called netnews on the Usenet[6], which can be compared with a internet forum, where user can post articles. Newsweeder addresses the problem of user dependency when filtering articles by "letting the user rate his or her interest level for each article being read" [20, p. 1]. This theme belongs more to the Information Retrieval field, which is about selecting all relevant data for a users query while preventing as many nonrelevant data as possible [3, p. 4]. Ken Lang compared in his project the classical term-frequency inverse-document-frequency, *tf-idf*, weighting approach with the Minimum Description Length, MDL, principle[7], which is based on the Bayes's rule. In the study beside the project he did an qualitative analysis and compared the results of two people, who tested the NewsWeeder program. It showed that the second technique, MDL, nearly quadruplicated the percentage of relevant articles found for the users query and excelled thereby the first classical one [20]. The *tf-idf* weighting and the Bayes's rule are described in detail in section 2.3 and 2.4.2.

2.2.3 BoosTexter

The next project, BoosTexter[8], is implemented by Yoav Freund and Robert Schapire and is based on a machine learning technique called boosting. Boosting is a method, which improves the accuracy of algorithms by "producing a very accurate prediction rule by combining rough and moderately inaccurate rules" [11]. Thus, its main idea is "to build a highly accurate classifier by combining many "weak" or "simple" base classifiers, each one of which may only be moderately accurate" [28, p. 1].

The basic boosting algorithm used for this project is named AdaBoost, short for "Adaptive Boosting". It was also invented by the two developers and they won already the Gödel Prize in 2003 for this algorithm [61]. In the project BoosTexter, this algorithm got two new extensions to perform multi-class text and speech classification. The purpose of the project is to test the performance of this modified AdaBoost with multiple classical text classification algorithms like Naive Bayes or Rocchio[9] [32].

[5] Related paper can be found on http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.6286 .
[6] See also www.usenet.org
[7] See also http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.159.403&rep=rep1&type=pdf.
[8] Related paper can be found on https://link.springer.com/article/10.1023/A:1007649029923.
[9] See also http://nlp.stanford.edu/IR-book/html/htmledition/rocchio-classification-1.html.

2.2.4 Retina Engine

The Retina Engine[10] is developed by an Austrian company called Cortical.io, which was founded 2011 by Francisco Webber and Daniel Schreiber. The Retina Engine is part of a platform, which offers the possibility to solve multiple challenging natural language tasks such as meaning-based filtering of unstructured text documents, real-time topic spotting on social media sites or search engine problems independent from language. It can be easily accessed via a REST API. The first release of Retina API was 2014 and since then it got multiple awards and commendations [66].

Retina is based on an idea of Francisco Webber, who came to this field via his medical studies and especially when he was participating medical data processing. Finally he concluded that the brain is the only working "Natural Language Understanding system" and he "decided to apply the principles of cortical processing to text processing". With the cooperation of Jeff Hawkins, one of the founders of Numenta[11] and the inventor of the Hierarchical Temporal Memory theory, which is "an online learning system modelled on how the neocortex performs tasks", the so called Semantic Folding theory was developed. The Retina API[12] therefore "converts language into semantic fingerprints, a numerical representation that captures meaning explicitly and operates on it computationally" [66]. Details to this process are explained in section 2.4.3.

2.3 Pre-Processing

The following section describes the process of data preparation and all required tasks in detail, starting with the data splitting and concluding with the weight per word computations. When starting to work with documents, which commonly consist of strings from a technical point of view, the primary goal is to convert these strings into a representation appropriate for the different learning algorithms and the classifiers [17, p. 5]. An approximate process chart of the pre-processing pipeline can be seen in figure 2.1.

Two types of formats algorithms can work with are the bag-of-words representation and the string representation. The bag-of-words approach describes a document as a set of words (e.g. vectors) in which their associated frequency in the document is stored. This type of representation essentially disregards the sequentially composition of words in the document. On the contrary, the second approach preserves each text as a string and by association the sequence of words. However, the majority of text classification methods primarily prefer the first approach "because of its simplicity for classification purposes" [1, p. 167].

At the beginning of text classification it is important to know which kind of text should be processed, with special focus on the structure like title, sections or paragraphs [15, p. 458], and even the language. Some steps of the pre-processing task for instance, may be simplified or aggravated depending on the language the texts are written in. English benefits from its easy space-delimited segmentation compared to languages such as Chinese, Japanese or Thai, which need an extra segmentation process applied before going further [15, pp. 4–5]. Another simplifying property of the English language is the

[10] Project can be tested on http://www.cortical.io/.
[11] See also http://numenta.com/.
[12] See also http://www.cortical.io/product_retina_api.html.

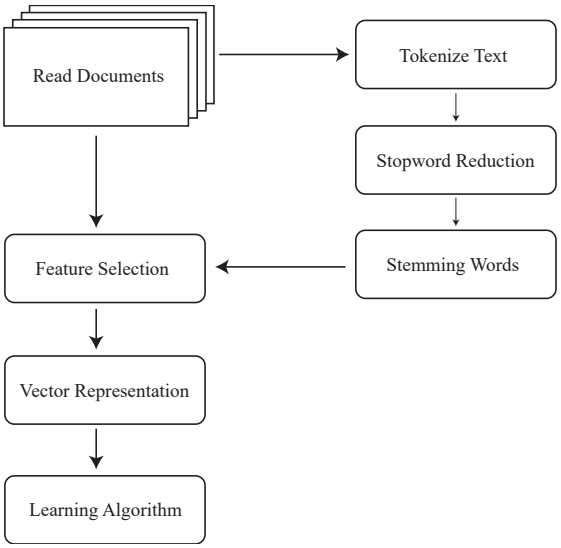


Figure 2.1: Approximation of pre-processing pipeline [18, p. 6].

lower-case notation. Even though, decisions have to be made concerning abbreviations, numbers, special characters, hyphenation [15, p. 458] or compound tokens [39].

As previously refereed in section 2.1, the following three sections belong to the feature extraction step (tokenization, stop word reduction and stemming), whereas weighting belongs to the feature selection step.

2.3.1 Tokenization

The goal of tokenization is to split the string, which represents the text document, into tokens by detecting the word boundaries. In most European languages word boundaries are indicated by the insertion of whitespaces, so including English. Another major segment which must be considered, is the punctuation such as periods, commas, quotation marks, apostrophes or hyphens. These components of a sentence consist of two different types, the ones which are separate tokens and the ones that are part of another token. For instance the expressions *analysts'* and *doesn't*; the first marks the genitive case and the second shows contractions, where letters have been skipped. There are two approaches of how to handle such cases. One opportunity would be to remove only separate tokens, with the difficulty that the system has to detect the difference between these two kinds of tokens [15, pp. 15–16], or the system removes all punctuation marks. The last option raises another question, "the expressions "don't","I'd","John's" do we have one, two or three tokens?" [15, p. 458]. In the related thesis project, the decision is made in favour of removing all punctuation forming one token, because of reasons of simplification.

The next critical point are numbers, for instance in a phrase like "$3.9 to $4 million". The digits can again be treated in two ways, removing it or keeping it as independent token. There is no definite rule for this case, but when keeping it as a token, some other conclusions have to be made. Considering the question whether the phrase mentioned above, would be treated the same if it has been written as "3.9 to 4 million dollars"

or "$3,900,000 to $4,000,000". Finally "the semantics of numbers can be dependent on both the genre and the application" [15, p. 16]. Concerning the project, the classification of news articles, numbers or digits are of low importance for the classification process, therefore they got dropped.

Obviously, English is highly accommodating particularly with regard to the tokenization process. In some languages multi-part words are commonly used, such as German. "Kundenzufriedenheitsabfragen (customer satisfaction survey)" or a noun–noun composition like "Lebensversicherung (life insurance)", to mention just some examples. Multiple other languages "use the hyphen to create essential grammatical structures" like "c'est-à-dire (that is to say)" or "celui-ci (it)" in French [15, p. 18]. All these properties of languages have to be considered and need an expansion of the tokenization task.

2.3.2 Stop Word Reduction

The following step in the natural language processing pipeline concerns the numerous non-informative words in text documents. To extract and separate topic related tokens from the whole mass of words, these non-significant tokens have to be removed. This includes articles, prepositions, conjunctions and some high-frequency words. They belong to the category of stop words and are in general also referred to as noise in natural linguistic texts. The extraction of these words is commonly used to improve the accuracy and to reduce the redundancy of the classifiers [44, pp. 1–3].

To reach this goal, the non-informative words are often condensed in so called stop word lists, with an average length of about 300 to 400 words. In most cases pre-processing system use generic stop word lists for all documents they process, not depending on the category or collection. This decision might be the safest option, due to the fact that the extraction hardly causes a substantial accuracy decrease, but coincident, the increase of it might be only minor [44, p. 3].

Another approach of reducing stop words in texts was developed by Wilbur and Sirotkin [41]. This kind of method enables a more aggressive removal of stop words "from documents without losing retrieval accuracy" and uses a training set of data to detect word significance via a so called "word strength". This value is not calculated on basis of the word frequency, but "on word co-occurrences in pairs of related" or very similar documents. This kind of calculation leads to more efficient and faster computation without accuracy losses. However, the appropriate stop word reduction depends as well on the domain and the application the classifier should work with [44, pp. 3–7]. For the thesis project and the detection of the topic, common stop word lists are adequate and have been chosen.

2.3.3 Stemming

In the field of information retrieval the chance for success depends on the number and frequency of terms the query and the searched document have in common. The same can be applied to text classification, where the overlapping terms are of particular

importance for detecting the topic. Even so the numerous morphological[13] versions of words, contained in the documents, complicate the term matching process and increase the dimensionality of the term space [33, p. 12]. Additionally in most cases these words with the same root, have a similar semantic meaning and could be treated as equivalent. That implies that the purpose of this pre-processing step is to reduce the different morphological versions of words [14, p. 2].

Julie Beth Lovins described this step as "a computational procedure which reduces all words with the same root to a common form". In many cases this is done by eliminating the derivational and inflectional suffixes from each word [24], so for instance to remove the case or the plural. To mention one example, all the words "computes", "computing" and "computer" would be mapped to its common stem "comput". This procedure does not change the document information significantly, but it prevents the increase of features [34, p. 1662].

This approach of suffix removal is just one possibility, but it is used by the two most popular algorithms, the Porter stemmer and the Lovins stemmer [14, p. 2]. There are two major characteristics on which these algorithms differ. The first one is the "significant reduction in the complexity of the rules associated with suffix removal". The Lovins stemmer includes a list of 294 suffixes, 35 recoding rules, which specify how the suffixes are converted, and 29 context-sensitive rules, which define if a suffix should be removed from a word or not. Compared to the Porter Stemmer, which has a list of 60 suffixes, two recoding rules and one context-sensitive rule, the dimensions are reduced remarkably. The second difference concerns the underlying relation of the rules. The rules of the Lovins stemmer are related to the number of characters remaining after removing the suffix, whereas the Porter stemmer is based on the number of remaining consonant-vowel-consonant strings [42, p. 220].

To summarize, the Lovins stemmmer searches the longest match in a large list of endings, while the Porter stemmer uses "an iterative algorithm with a smaller number of suffixes and a few context-sensitive recording rules" [14, p. 2]. Some other possible techniques would be truncation of character strings, word segmentation, letter bigrams or linguistic morphology [14, 21]. In the thesis related project the Porter stemmer is used.

2.3.4 Weighting

The last task describes the process of separating relevant documents, for a specific topic or user query, from unimportant documents. A binary method of elimination, so a decision between "yes" or "no", would be too restrictive. This implies that something between is required [15, p. 459].

The process of defining this value is called term weighting and its purpose is to refer higher weight values to more important terms and lower weight to less important terms. In this context multiple definitions have to be considered. Words which frequently appear in individual documents, can be one indication for a topic-related feature. Thus, one part of a weighting system is the term frequency, tf, the number defining how often a term appears in the documents. Unfortunately, it can happen that the high

[13] In linguistics, morphology is the study of how words are formed and how they are related to other words of the same language [45].

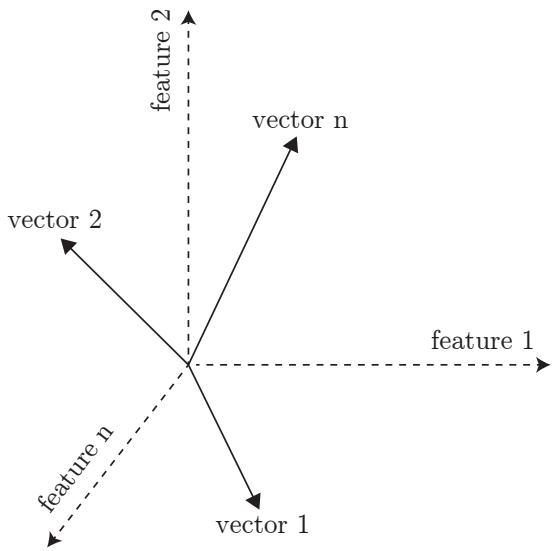


Figure 2.2: Approximation of the vector space model [58].

frequently appearing words are widely spread in the whole set of documents, not only concentrated on a particular number of them. As a consequence all documents would be indexed and marked as relevant, what leads to performance decrease. Hence a factor has to be defined, which "favours terms concentrated in a few documents of a collection". This figure is specified as the inverse document frequency, *idf*. In order to calculate the *idf* value of a word, two other key figures are required, the total number n of documents in the data set and N, the number of documents containing the word. The final factor of a typical *idf* value can be then computed as $\log(N \div n)$. The topic related terms should be able with this extension to distinguish a particular group of documents from the remaining data. Hence, they should have a high term frequency but a low overall collection frequency. Finally, the term weighting can be determined as the product of the term frequency and the inverse document frequency, $tf \cdot idf$ [30, p. 516].

This approach of detecting relevant documents, is called the term discrimination model. However, it has been often opposed with the probabilistic model, which uses a term relevance weight. This weight is defined as "the proportion of relevant documents in which a term occurs divided by the proportion of nonrelevant items in which the term occurs". Unfortunately, it is not "immediately computable without knowledge of the occurrence properties of the terms in the relevant and nonrelevant parts of the document collection". An approximation can be computed with $\log((N - n) \div n)$ for the *idf* factor [30, p. 517].

In conclusion, the term weights form, for each document, a vector filled with its terms and a related weight. All vectors together form the vector space, where the terms are the axes of the space and the documents represent points, more specifically the vectors of weights, in the space. These vectors can now be processed further [31, p. 613]. An approximation of the vector space model can be seen in figure 2.2.

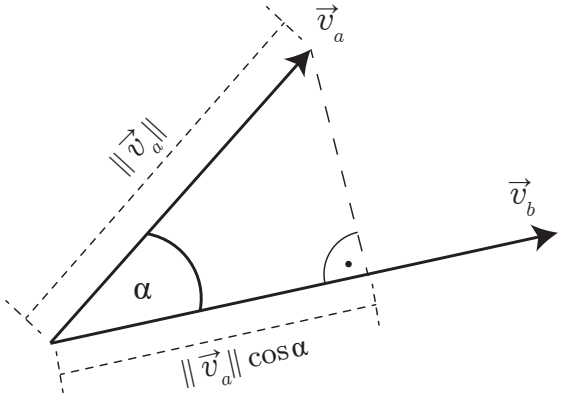


Figure 2.3: Projection of vector $\vec{v}_a$ on vector $\vec{v}_b$ [58].

2.3.5 Similarity

One crucial factor in text classification is similarity, so the detection if two documents are similar or not, to finally define the topic. The k-nearest-neighbor for instance is reliant on the similarity function to detect the k nearest neighbors.

The notion of similarity is often associated with the distance between the two document points in the vector space, which is in general related with the similarity between these vectors. There are several ways to determine a similarity, but first it should be clarified what the value represents. Given two vectors of two comparative documents, the similarity value "reflects the degree of similarity in the corresponding terms and term weights" [31, p. 613].

Just two examples of many to measure this value is via vector analysis, the inner product and the magnitude of the vector between two points. The first option is called the cosine similarity and the second one the euclidean distance [13, p. 51].

The main functionality, which is needed for the cosine similarity is the inner product or also called, dot product, which can be interpreted as the product of the magnitude of the first vector and the magnitude of the projection of the second vector on the first (see equation 2.1). The graphical representation of the interpretation can be seen in figure 2.3. If the angle between the two vectors is 90°, then the length of the projection is zero, thus the value of the dot product is zero [58]

$$\vec{v}_a \cdot \vec{v}_b = \|\vec{v}_a\| \|\vec{v}_b\| \cos\alpha. \tag{2.1}$$

This functionality is crucial important to derive the cosine similarity formula from the cosine theorem[14], whose basic form is

$$c^2 = a^2 + b^2 - 2ab\cos\alpha. \tag{2.2}$$

By changing the variables according to the figure 2.4, the equation can be adjusted to

$$\|\vec{v}_a - \vec{v}_b\|^2 = \|\vec{v}_a\| + \|\vec{v}_b\| - 2 \cdot \|\vec{v}_a\| \cdot \|\vec{v}_b\| \cdot \cos\alpha. \tag{2.3}$$

[14] See also http://www.mathe-online.at/materialien/heike.farkas/files/Vektorrechnung_Ebene/Beweis_Winkel.pdf.

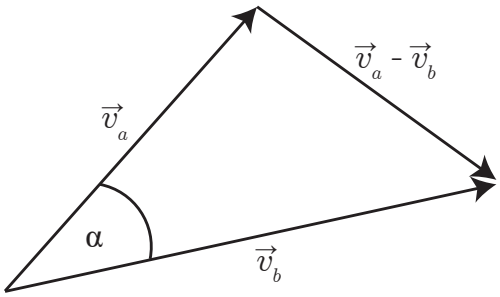


Figure 2.4: Adapting cosine theorem for derivation of cosine similarity

The final formula for the cosine similarity is then as follows

$$sim(\vec{v}_a, \vec{v}_b) = \frac{\vec{v}_a \cdot \vec{v}_b}{\|\vec{v}_a\|\|\vec{v}_b\|}, \tag{2.4}$$

with the two given vectors $\vec{v}_a$ and $\vec{v}_b$, describing the documents, where each value comprises a term and its weight in the related document. The formula computes the similarity factor and the resulting value can be in the range between minus one and one, indicating not similar and very similar. If the value is exactly one, the two vectors are identical [13, p. 51] and the angle between them zero, computable via the inverse cosine. One important advantage of the cosine similarity is its independence of the document length of the comparative vectors, because it normalizes the two vector lengths to one [31, p. 613]. This implies that only overlapping words are considered and short texts will not be declined compared with long texts. However, this unifying has one flaw. "Documents with the same composition but different totals will be treated identically" [13, p. 52]. It means that if one short text is contained in some different and longer text, these two documents would be treated as equal, which can have adverse effects.

Another possibility to calculate the similarity between two documents, is the euclidean distance as mentioned above. This value represents the actual distance between two points and can be calculated by creating the direction vector $\overrightarrow{V_a V_b}$ between them and applying the magnitude. The direction vector itself is computed by subtracting each dimension of the document vectors $\vec{v}_a$ and $\vec{v}_b$ [13, p. 51].

In the thesis project both methods have been tested, but finally the cosine similarity has been chosen, because it worked better for the news articles processed in it.

2.4 Algorithms

The classification of text documents can be done in three different kinds, supervised, semi supervised or unsupervised. However, for classifying a great number of electronic documents, supervised methods are primarily used. This kind of technology needs predefined class labels, so fixed topic categories in advance, and documents, which are assigned to these labels. The training set for the classifier consists of these documents and they are then used to predict a label to a new unseen document. Generally, this process of supervised learning is the core of automatic text classification [18, p. 8]. Some of these methods are described in the following section.

2.4.1 k-nearest-neighbor

One of the oldest and simplest algorithms for pattern recognition is the k-nearest-neighbor algorithm, abbreviated kNN in the following [7]. It classifies each test document, which is not labelled yet, by finding the most frequent label among the k nearest neighbors contained in the training set. Yiming Yang described it as follows: "Given an arbitrary input document, the system ranks its nearest neighbors among the training documents, and uses the categories of the k top-ranking neighbors to predict the categories of the input document." [43, p. 4]. This implies that this algorithm and its performance, crucially depend on the distance measure or similarity function to identify the nearest neighbors [40].

The procedure of many classification methods have two phases, the training phase and the testing phase. K-nearest-neighbor has also these two stages, whereby the first one is restrained and not evident compared to other classification algorithms [8, p. 5]. During the first, all feature vectors, the word vectors with appropriate weights, and the related categories of the training set have to be stored. In the second phase, also referred to as the classification phase, the main process initiates. This stage computes the distances or similarities from the new input vector, which contains the new text document, to all stored train vectors to finally detect the k most similar or closest documents [18, p. 9].

The undoubted advantage of the k-nearest-neighbor algorithm is its simplicity. Furthermore, it performs significantly strong in terms of text data classification and the success remains still stable even when "the category-specific documents form more than one cluster because the category contains, e.g., more than one topic" [8, p. 6]. Though, the method has some restrictions. The major limitation is its high calculation complexity. For detecting the k nearest neighbors, the algorithm has to go through all the documents and features to calculate the similarities and this procedure lasts the longest. This problem can be circumvented by reducing the feature space, by using a smaller data set or by using an improved and accelerated algorithm [18, p. 9]. The next point to consider concerns the not existing weighting between the different document samples. All the texts are treated equally, for instance no preferential treatment of longer text data towards shorter ones. Furthermore, the different number of training documents per class can risk that too many documents from a "large category appear under the k nearest neighbors and thus lead to an inadequate categorization" [8, p. 6]. To solve these problems, various improved versions of kNN have been developed and studied in recent years [7, pp. 1–2] [23].

One approach to overcome the difficulty of differently-sized categories was elaborated in the paper from Li Baoli, Yu Shiwen and Lu Qin. They developed a modified version of kNN, which they describe as follows [23, p. 1]:

> "In the traditional kNN algorithm, the value of k is fixed beforehand. If k is too large, big classes will overwhelm small ones. On the other hand, if k is too small, the advantage of kNN algorithm, which could make use of many experts, will not be exhibited. In practice, the value of k is usually optimalized by many trials on the training and validation sets. But this method is not feasible in some cases where we have no chance to do cross-validation, such as online classification. To deal with this problem, we propose a revised

> k-Nearest Neighbor algorithm, which uses different k values for different classes, rather than a fixed k value for all classes."

To determine the number of nearest neighbors, the method computes the probability for each category that the input document belongs to it, by using only some top n nearest neighbors for the class c, where n is derived from k depending on the size of c in the training data. The result can be summarized as follows, "for larger classes, we used more nearest neighbors; for smaller classes, we used fewer nearest neighbors" [4, p. 218]. This method was tested on a Chinese text classification problem and the results show that their method is less sensitive to the parameter k than the original kNN [23].

Another approach would be combining the traditional kNN with some other algorithm, for instance a genetic algorithm, which uses the evolution strategy to select and combine samples in the training set to find the optimal combination, as described in the paper from N. Suguna and Dr. K. Thanushkodi. This technique has the advantage that it does not consider all training samples and takes only the nearest neighbors; with the addition of the genetic algorithm, it only considers the k nearest neighbors straight away and then computes the similarities to classify the input documents, which improves the computation time [35, p. 18].

Euihong Han et. al [12] developed a weight adjusted modification of the kNN, "which learns the importance of attributes and utilizes them in the similarity measure". This type of extension refers more to the preparation part, so the feature selection step, which finally leads to a preference of specific features in the similarity computation [4, p. 216].

For the thesis related project, which focuses on the comparison of the different algorithms, the traditional kNN was used.

2.4.2 Naive Bayes

This kind of classification algorithm belongs to the group of Bayesian approaches, together with the second section, the Non-Naive Bayes algorithms. The difference between these two types is based on the assumption of word or feature independence. This describes the naive part of the classifiers and it means that the word combination in the documents is irrelevant. This implies further that the presence or absence of a word does not effect the remaining words at all. This approach enables a more efficient computation of the classification process than the non-naive version [8, p. 6] [43, p. 3].

The naive classifiers can be subdivided again into different types of classifiers. Two commonly used examples are the Multivariate Bernoulli Model and the Multinomial Model. Both versions calculate the posterior probability of a category based on the appearance of words in the documents. The actual position or combination of words can be ignored, thus these models work as well with the bag-of-words representation as described in section 2.3. The main differential characteristic of these two approaches is the assumption "of taking (or not taking) word frequencies into account, and the corresponding approach for sampling the probability space" [1, p. 182] [27, p. 2].

Apart from the used model, the features of an given input document can be used in combination with the Bayes rule to compute the joint probability for each class in the data set. The category with the highest probability value will be finally assigned to the

document [1, p. 182] [27, p. 3]. The Bayes rule[15] can be written down as [37, p. 3]

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}. \tag{2.5}$$

It states, applied to the classification problem, the probability that the input document has the class y under the assumption that the word x is contained in the document. This probability is named conditional probability. The total value can be calculated by determining $p(x|y)$, the probability that the word x occurs in the category y, $p(y)$, the probability that the searched document belongs to the class y, and $p(x)$, the probability that the word x is mentioned in a document. The first value $p(x|y)$ is computed by dividing the term frequency of the word x in the class y with the total number of words in the class y. The second value is defined by the ratio of the number of documents in the category y to the total number of documents. The last value can be split as [37, p. 3]

$$p(x) = p(x|y)p(y) + p(x|\neg y)p(\neg y). \tag{2.6}$$

The first part represents the same value as the numerator in the Bayes rule, so the probability that the word x occurs in the category y and the probability that the searched document belongs to the class y. The value gets aggregated with the probability that the word x occurs outside the class y, multiplied with the probability that the searched document is not contained in class y. The $\neg$ sign in the equation indicates the negation of the subsequent term, thus the probability that the word x occurs outside the class y or that the searched document is not contained in class y. Summing up these probabilities for each word in the document by multiplying, results in the complete joint probability for the class [1, p. 182]. When multiplying probabilities, it can cause very small values, which can't be processed further. To solve this problem the logarithm of the likelihood can be used, "because the log-likelihood is monotonically related to the likelihood itself, a maximum on the log-likelihood surface is also a maximum on the likelihood surface" [29, pp. 590–591] and in the field of text classification, only the highest class probability is decisive and the exact computation can be neglected [1, p. 185].

To come back to the two common models of the naive classifiers, the first commonly used one is the Multivariate Bernoulli Model[16][17], which basically uses the presence or absence of words to represent the documents. This implies, the frequency of words is irrelevant in this kind of classification model. A dictionary is created representing the vocabulary of the data set and the representation of the documents finally results in binary vectors, indicating if a word form the dictionary exists in the text or not. The second method, which is often used is the Multinomial Model[18]. This type of classification method stores the frequencies of words in the documents. "As a result, the conditional probability of a document given a class is simply a product of the probability of each observed word in the corresponding class" [1, p. 182].

[15] See also http://stpk.cs.rtu.lv/sites/all/files/stpk/materiali/mi/artificial intelligence a modern approach.pdf.

[16] See also https://www.stat.wisc.edu/sites/default/files/tr1171.pdf.

[17] See also https://nlp.stanford.edu/IR-book/html/htmledition/the-bernoulli-model-1.html.

[18] See also http://www.cs.ubc.ca/~murphyk/Teaching/CS340-Fall06/reading/bernoulli.pdf.

Consequentially, this kind of model works better with data which can easily be transformed into tables of numbers, like word counts in text documents. Additionally, the first version is more suitable for small vocabulary sizes like binary data, so data with only few categories, such as yes or no options; conversely, the second one deals much better with large vocabulary sizes. This key data indicates "that the two models may have different strengths and may therefore be useful in different scenarios" [1, p. 190]. In the case of classifying articles the second approach is more appropriate, because of the various categories and because of the large vocabulary sets. The following section describes the basic operation in detail, inclined towards the basics of the Bayes rule.

In summary, the Naive Bayes works "surprisingly well for many real world classification applications" and its advantages are its simplicity, its insensitivity to deficiencies and it only "requires a small amount of training data to estimate the parameters necessary for classification" [18, p. 10]. Another advantage is its computational efficiency, because it only involves the presence of words in the calculation, not the absent ones [22, p. 5]. Thus, the benefits of the independence assumption can be simultaneously the disadvantage, when the features of the data to process are not independent, which leads to accuracy decrease and needs a remedy to be found [26].

2.4.3 Semantic Fingerprinting

This technology was developed by Francisco Webber and Daniel Schreiber, who additionally founded a company called Cortical.io around this idea. Webber recorded their findings in a white paper [38, pp. 6–40], which is the theoretical foundation for the following section and all information derives from this reference.

The Semantic Fingerprinting method is based on a neuroscience rooted mechanism, which arose during the medical studies of Francisco Webber. The fundamental methodology behind this concept, is named the Semantic Folding Theory, which describes the encoding procedure that transforms textual input data into a Sparse Distributed Representation (SDR), binary representational vectors, which can than be processed further by Hierarchical Temporal Memory (HTM) systems.

HTM Learning Algorithm

This learning algorithm is one part of the Hierarchical Temporal Memory Model and was developed by Jeff Hawkins [2]. Generally, it assumes the structure and functionality of the mammalian neo-cortex, which is from an evolutionary point of view a rather novel structure for managing "the command and control functions of the older (pre-mammalian) parts of the brain". A human being is in general constantly exposed to a stream of sensorial input data, noises, pictures, feelings, and this leads to a continuously learning process of characteristics, which describe the "surrounding environment, building a sensory-motor model of the world". For the optimization of this world-model, deposited in the neo-cortex, already stored information of previous experiences and impressions are used to describe the new and unknown characteristics or to adjust older ones with more details and features [38, p. 10].

The natural condition of the neo-cortex is in general, "a two-dimensional sheet covering the majority of the brain", which consists of "microcircuits with a columnar structure" [38, p. 10]. Furthermore, the surface of the neo-cortex is subdivided into different

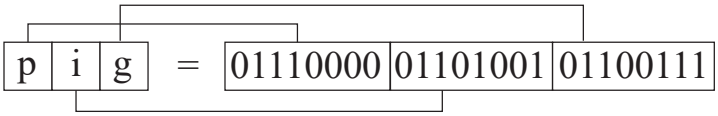


Figure 2.5: Binary representation of the word "pig" without single bit meaning [38, p. 13].

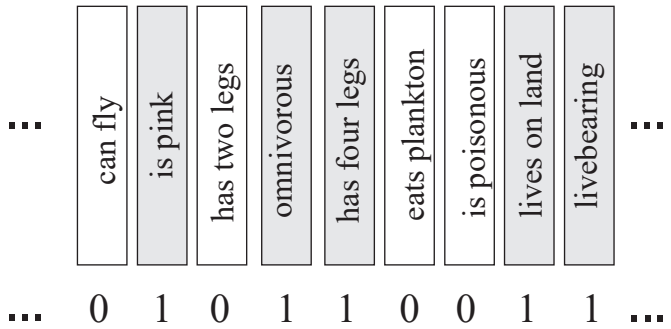


Figure 2.6: Sparse representation of the word "pig" with single bit meaning [38, p. 13].

areas, receiving either inputs from sensory organs or data passed on from another region. All these parts detect frequently reoccurring input sequences, generate a distinct pattern based on them and produce a stable representation of each learned pattern. These types of operations can be described as a memory system as a whole, which processes data just by storing it to a distinct address, a long binary vector, forming the Sparse Distributed Representations, SDRs [38, p. 11]. A system based on the HTM theory assumes this mechanism. It converts input data into Sparse Distributed Representations, saves them and tries to predict and recognize patterns in new and unseen data, based on the already seen ones.

Sparse Distributed Representations

The purpose of SDR representations is to describe various sensory input of objects or impressions in the surrounding environment. The corresponding pre-version of SDRs, are dense binary values (see figure 2.5), where each combination of bits identifies a specific data item, described by a set of stimuli, in the memory system. To detect which kind of data item it is, a dictionary is needed, to keep track of all possible combinations and related data items. The more impressions are stored in this dictionary, the longer it would take to figure out which semantic grounding it has. To overcome this difficulty, the semantic foundation of a binary vector can be directly included in its representation, whereby every bit of it corresponds "to an actual feature of the corresponding data item that has been perceived by one or more senses". This will cause much longer vectors, but in these representations (see figure 2.6) only very few bits are set and by storing only these set bits, a high compression rate can be achieved [38, p. 13].

The only remaining problem is noise, in other words, false activation, shifted bits or false dropping of bits can lead to a wrong or unreadable interpretation of a word [38,

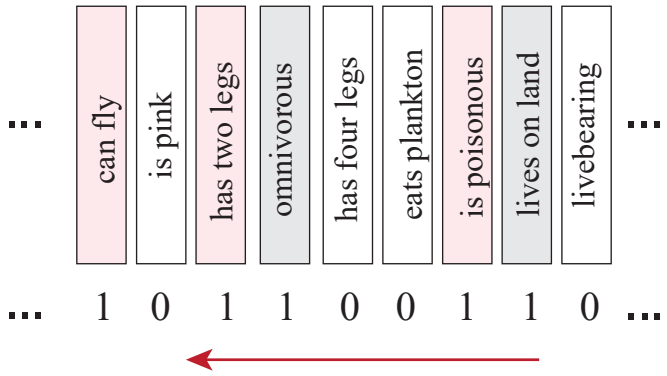


Figure 2.7: Bits shifted to the left cause misinterpretation [38, p. 14].

pp. 13–14]. “Sparse representation are more resistant to dropped bits”, because not all features of the item are needed to identify it correctly. Nevertheless, shifted bits are still troublesome (see figure 2.7). Up to this point, the features of the data item are located at random positions and aren’t ordered. That implies that a 100% match is required to identify a data item. When sorting and grouping similar features together, multiple benefits are gained. The first advantage is the improvement of the noise reduction. Even if a bit is slightly shifted a few places to the left or right, the overall semantic meaning of the whole data still remains the same, because the neighboring features belong to the corresponding group (see figure 2.8). The second improvement concerns the semantic comparison of values. After grouping features, it is possible to calculate a gradual similarity value by comparing separate regions of the fingerprint and this allows a more sensitive comparison. It can be used for disambiguation or conclusions. To summarize the main properties and advantages of SDR encoded input data:

- they can efficiently be stored, because of the very few set bits,
- every bit of them has a semantic meaning,
- similar terms result in a similar SDR,
- they are remarkable noise resistant, because of grouping features and
- they can be combined and merged together without information loss [38, p. 14].

Semantic Folding

With this previous knowledge, Semantic Folding can be described as a “data-encoding mechanism for language semantics”. In general, “language is a creation of the neo-cortex” and all language elements are finally “converted into a inner representation“, the SDRs, “that can be directly used by comprehension circuits” on the neo-cortex [38, p. 17].

Word-SDRs represent the smallest part of language, which contain lexical information and whenever a word sequence is perceived, the neo-cortex detects frequently used patterns, the Word-SDRs, and saves the sequence where they appear in. This sequence can be seen as an instance of a Linguistic Special Case Experience corresponding to a linguistic statement, which consists of sentences. These sentences or text snippets represent a context for each word contained in it. Every time the same Word-SDR appears,

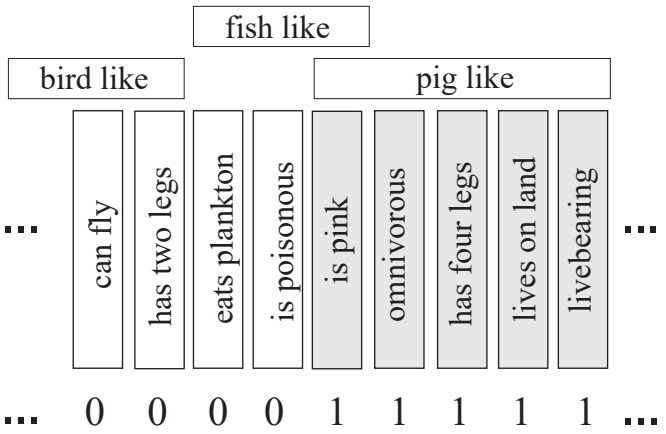


Figure 2.8: Grouping of features for more resistance [38, p. 14].

a new snapshot of the sequence is combined with the stored one, and over time only the bits which "are in common within all states remain active" [38, p. 20]. After some time the learning process progressively switches and uses already known words that have been learned previously for assimilating new words. Thus, "the mature brain depends mostly on existing words to define new ones". Every word gets linked to more and more contexts enhancing its foundation and finally, a word is defined by its list of contexts [38, p. 21].

After this mapping step, a word can be described by a simple one-dimensional word vector. The second mapping procedure concerns the contexts themselves. Their underlying representation are vectors, which can be used to create a two-dimensional map by comparison, where finally "similar context-vectors are placed closer to each other than dissimilar ones". The resulting map is the sum of all received contexts, the Special Case Experiences, so sequences of words and text snippets. When receiving new sequences, the whole map gets dynamically extended [38, pp. 21–22].

This semantic map is used to encode single words, by associating binary vectors to them, containing a one if the word is contained in the underlying context at the specific position or a zero if not. The result is a long sparse vector forming a Word-SDR, which is still quite resistant against noise, because in the underlying context map adjacent contexts have still a similar meaning (see figure 2.9). In general these encodings obtain all advantages of SDR encoded data [38, p. 23].

A set of multiple Word-SDRs can be compared by using a similarity function or distance metric, like the euclidean distance or cosine similarity. The easiest way of calculating the distance is counting binary overlaps between two SDRs. This case it is important to note word-frequencies, because they can lead to misinterpretations. Generally, these functions detect the semantic closeness of words. However, the similarity of words can not be directly equated with word synonymy. This is only a special case of semantic closeness. It is a more flexible concept, determining if two words have similar contexts [38, pp. 23–24]. Another possibility to detect the underlying term or topic of a fingerprint is to look at the overlapping bits and the topic of the shared contexts in the background [38, p. 37].

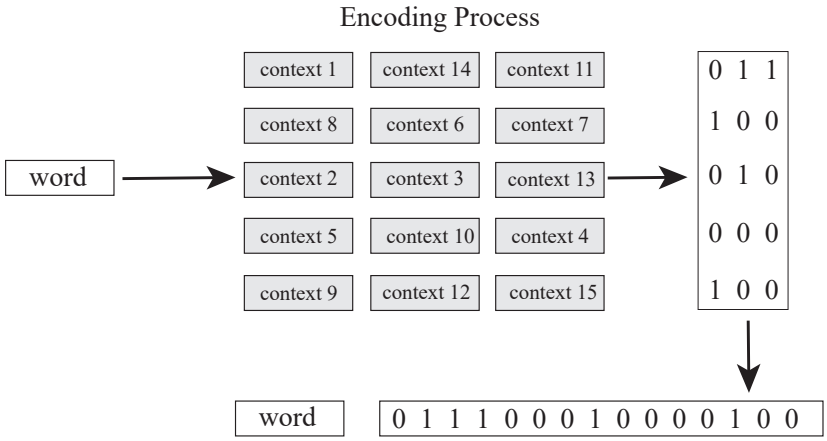


Figure 2.9: Encoding of a word with ordered semantic map [38, p. 23].

Recapping, Semantic Folding converts the incoming data stream into Sparse Distributed Representations in order that the system based on the Hierarchical Temoral Memory theory can make predictions what meaning the unseen SDR has based on the pattern it has seen so far. SDRs can also be referred to as Semantic Fingerprints and the whole process of predicting patterns, as Semantic Fingerprinting.

Classifying Process

In terms of text classification the resources of texts, the articles of the training data, are the reference texts defining the whole Semantic Universe the system has to work in. The texts are converted into word vectors in the pre-processing step and then placed in an array or a 2D matrix, in a way that texts with similar topics are placed closer to each other than different topics. The result of this process is the semantic context map, needed for encoding words. After the map construction, each word of the training or testing texts, gets converted with the aid of this map. This produces a large, binary and sparsely filled vector for each word, the Semantic Fingerprints. The fingerprint can be visualized on a grid, in this case a square grid, by marking the ones as black fields (see figure 2.10) [38, p. 29].

These fingerprints can either be stored in a database, or they can be combined and merged together forming Text-SDRs or Document Fingerprints. This is done by collecting all Semantic Fingerprints of the words contained in a text, stacking them over each other (see figure 2.11) "and the most often represented features produce the highest bit stack" [38, p. 34]. "The bit stacks of the aggregated fingerprint are now cut at a threshold that keeps the sparsity of the resulting document fingerprint at a defined level" [38, p. 35]; in other words, the contexts of the highest stacks will be kept. The same procedure can be applied for the creation of Topic or Category Fingerprints. Finally the new input text which should be classified, can be classified on multiple ways: comparing it with other texts from a specific category, or with a whole Category Fingerprint, or the topic can be read out by looking at the overlapping contexts [38, pp. 39–40]. Details on the classification process and the reimplementation are described in chapter 4.

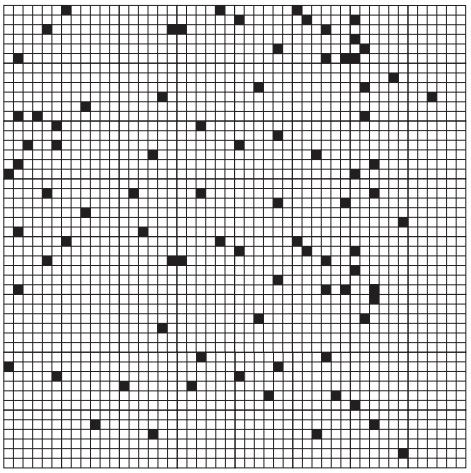
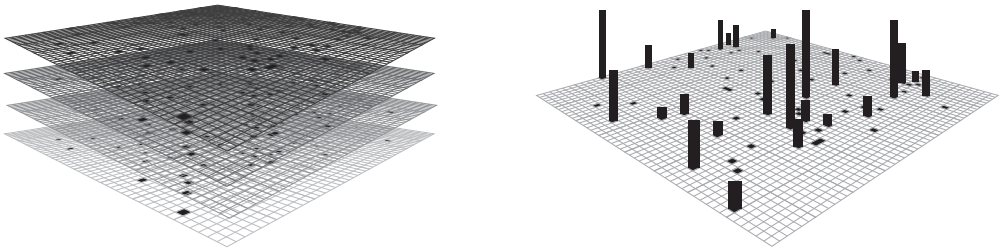


Figure 2.10: Graphical representation of an example Semantic Fingerprint [38, p. 29].



(a) (b)

Figure 2.11: Put Semantic Fingerprints of words on top of each other (a) and detect the highest stacks of ones in the fingerprint to keep it for the final Semantic Fingerprint of a text (b) [38, p. 29].

2.5 Multidimensional Scaling

In the field of text classification, one of the first tasks is the conversion of text into a numeric representation. Therefore numbers have in general a great relevance in classification tasks. Concurrently, this complicates comprehending the workflow of the algorithms, because it is difficult to get an overview of the properties of the numbers: One possibility to overcome this barrier, is multidimensional scaling, a graphical representation of data, preserving the distances between them.

Multidimensional Scaling, in short MDS, is a collection of mathematical proceedings "that enable a researcher to uncover the "hidden structure" of data bases" [19, p. 5] and addresses the reverse problem of distance measuring. When a map is given with a number of cities marked on it, one only have to measure out the distances between them and convert it into real distance. Unfortunately, when a table of distances is given, it requires more effort to develop the related map. In the classification subject, the data vectors the algorithm work with, are usually n-dimensional, with $n > 3$, which means the data is not suited to be visualized directly. Thus, the n-dimensional data have to

Table 2.1: Proximity table of crime rates in 1970 of 50 U.S. states [6, p. 4]. The values show how similar the rates are in this year and if there are some correlations between some crimes.

Crime	**Numb.**	1	2	3	4	5	6
Murder	1	1.00	0.52	0.34	0.81	0.28	0.11
Rape	2	0.52	1.00	0.55	0.70	0.68	0.44
Robbery	3	0.34	0.55	1.00	0.56	0.62	0.62
Assault	4	0.81	0.70	0.56	1.00	0.52	0.33
Burglary	5	0.28	0.68	0.62	0.52	1.00	0.70
Auto theft	6	0.11	0.44	0.62	0.33	0.70	1.00

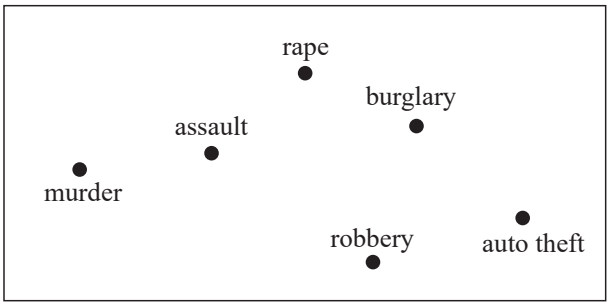


Figure 2.12: A graphical, two-dimensional representation of the proximity values in table 2.1 [6, p. 4]. It visualizes the correlation of each crime, where close crimes are highly correlated and far apart crimes are less connected.

be converted in a m-dimensional one, where $m < n$, but keeping the notions of distance between the data points. Thus, it is a dimension-reduction technique for visualizing data. This set of methods uses a $N \times N$ proximity matrix, which contains similarity values of all objects to each other. These similarities indicate the distances between the objects and how close or far apart two objects are [9, p. 444]. In table 2.1 an example of an proximity table is shown. The final result is a geometric representation of points, where each point corresponds to a distinct object, such as points on a map (see figure 2.12). The representation locates dissimilar objects far apart from each other and similar ones closer, reflecting the proximity values. The map is not necessarily adequate for two dimensions, it can have three, four or even more. This configuration can reveal a hidden structure in the data, which often facilitates comprehending the data [19, p. 7].

In practice, MDS representation are normally always constructed by computer programs due to the complexity of the computation. However, the approximate approach is to determine the dissimilarities between the given objects based on their distance-values, usually with the euclidean distance computation. The whole procedure results in a configuration where the distances between the objects on the map, approximates the corresponding dissimilarities between them. The quality of this configuration or also referred to as approximation error, can be defined with a loss function, which detects the optimal setting to minimize it [5, p. 21] [9, pp. 445–446] and finally adjusts the

configuration. There are numerous versions of loss functions in use, also named stress functions, some well-known representatives are the Raw Stress, the Normalised Stress or the Kruskal's Stress function [5, pp. 22–25].

In general, there are multiple varieties of MDS computations and the variation of the loss function is one aspect of how they can differ. Some other distinguish in the particular type of geometry, into which the method wants to map the object data, or in the mapping function, the algorithms used to determine the optimal representation, or in the possibility to visualize several similarity matrices into one map at the same time [6, p. 3]. Multidimensional Scaling is a very comprehensive field, with a great number of adjustment possibilities and computation effort, but it offers the great opportunity to uncover hidden patterns in data and the prospect to gain a deeper insight into the fundamental substance. More detailed information about the technical background can be found in [5, 6, 9, 19].

Chapter 3

Conceptual Background

The following chapter describes the general set-up of the corresponding project as well as the underlying conceptual idea.

3.1 Used Technology

The main purpose of the thesis correlated project is the creation of a reliable and stable test environment for the comparison of different text classification approaches. In order to achieve this, the whole project is integrated into a Content Management System, CMS, environment and implemented as an extension to make it replicable and reusable. The pre-processing steps are facilitated by means of Sam Hocevar's NlpTools, and the software RStudio enables a closer look on the structure of data the extension uses. The remaining part of the project is implemented independently, based on the respective theory, with the aim of enhancing the understanding of the fundamental principles of text classification.

3.1.1 TYPO3 CMS

TYPO3 is the most commonly used Content Management System with more than 500.000 installations, for diverse web projects since eighteen years, which provides a basic structure for websites, intranets or web and mobile applications [46, 65].

Referring back, TYPO3 was developed by a danish developer, Kasper Skårhøj, in 1997. In those days the term "content management" was not as usual as today, but the more complex the websites became, the more arose the idea of a new system which separates design and content for clearer structures and easier handling. TYPO3 is an open source project, which is one of its key features and offers with version 4.7 and greater, a responsive approach for a variety of devices. The main advantage, in relation to the thesis project, is its extension framework and the possibility to expand the personal TYPO3 application by means of the Extension Manager [65].

Since the 10th November 2015, the seventh version of TYPO3 CMS is released with new features like performance improvements, Bootstrap 3 integration and a visible TYPO3 backend facelift [56]. This version is used for the thesis project. The Extension Manager was added to the main system with the release of version 3.5. This feature is an internal control centre, which manages the new structured architecture, separating the

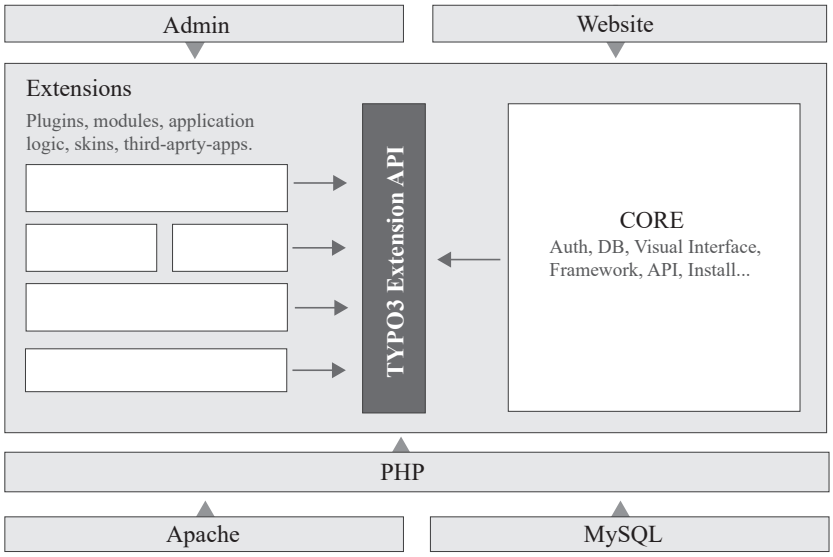


Figure 3.1: Internal TYPO3 structure with primary focus on extensions [65].

core extension from the remaining extensions. Before the introduction of the Extension Manager, everything was mixed together and overlapping, such as framework, plugins, database and API. The structure was unstructured and hard to work with; thus, the Extension Manager was necessitated. The actual internal structure of TYPO3 can be seen in figure 3.1. With the release of TYPO3 4.3 two addtional important extensions have been added to the system, Extbase and Fluid [65].

In the early stages, Extbase and Fluid combined should be an alternative option for the extension development with piBase[1], but over time they gradually assumed this responsibility. Extbase can be described as a framework and enables more complex extensions, whereas Fluid adopts tasks of a template engine and is responsible for in- and output [55]. A detailed view of these two extensions can be found in section 3.2.

3.1.2 PHP NlpTools

These tools consists of multiple PHP classes for natural language processing tasks developed by Sam Hocevar, mainly for its own needs. It is for free and available on GitHub or via Packagist, the PHP package repository, and a detailed documentation can be found on the project's homepage[2]. The main part of the set contains different kinds of tokenizer, stemmers, classification models, clustering models, diverse similarity functions, a *tf-idf* analysis and a stop word filtering method. The repository requires at least PHP 5.3 and for the project, it has to be placed in the TYPO3 `Resources/Private/Libraries` folder.

The thesis project itself uses only some of the classes the library provides. The general underlying idea using a library in the course of the project, is to facilitate the pre-processing part of the classification process. This mainly concerns the preparation

[1] See also https://www.video2brain.com/de/videos-26061.htm.
[2] For the documentation see http://php-nlp-tools.com

of the data, which the extension and the algorithms should process afterwards. For this step following classes are needed: the `WhitespaceTokenizer`, the `PorterStemmer`, the `CosineSimilarity`, the `StopWords` filtering class, the `English` normalizer and several helper classes for calculating the *tf-idf* weighting. A detailed description of the used classes and the pre-processing implementation can be found in the chapter 4.

The homepage of this tool kit, offers a great set of code snippets showing how to use the classes, tutorials, on what can be done and documentations of various projects. On the blog of the page, the author publishes recent innovations or several insights into his projects, presenting his current state of development. Among others, a spam detection service and a programming language detection belong to them[3].

One of the experiments on the blog deals with the comparison of the NlpTools with the popular and commonly used Natural Language Toolkit (NLTK), written in Python. The subject of the contrasting juxtaposition was the hierarchical clustering implementation and mainly its time complexity. After setting up datasets, running and comparing the times for calculation, it turned out, the NLTK "implementation is asymptotically worse than the one in NlpTools". The author's main assumption for the reason why, concerns the calculation of the cluster similarities; NlpTools uses a dissimilarity matrix, but NLTK does not and recalculates the similarities each merge again. After adding a "quick and dirty addition of a similarity matrix to the algorithm" [50] of NLTK, a decent speed improvement has been observed.

3.1.3 RStudio

RStudio's attempt is to provide in general "the most widely used open source and enterprise-ready professional software for the R statistical computing environment" [60]. This includes among others an IDE, an integrated development environment, for the statistical language R and development tools, which facilitates working with this kind of language and enables easy analysis of data. The enterprise was founded in 2008 and contains open source products such as the IDE or Shiny, a framework for web applications using R, as well as a commercial professional version.

However, R is mainly used for statistical computing and graphics. Furthermore it provides a great variety of methods such as classification and clustering methods. In general it is closely related to the S language[4], which is often used for statistical methodology as well, but R offers open source opportunities and a highly extensible access to this field.

In case of the related thesis project, it is used for the construction of multidimensional scaling graphs, displaying the articles used in the text classification in relation to each other. For this purpose the open source desktop version of RStudio IDE was used. Before starting with R for the MDS graph, the data has to be prepared. The calculation needs, as in section 2.5 described, a proximity table containing the similarities between all objects. Thus, a similarity matrix has been created with all articles compared with each other. A small part of the proximity table for a data corpus including the categories `Film` and `Politics` is shown in table 3.1, which is needed to produce the figure 3.2.

After creating a matrix with similarities and the respective article ID as column and

[3] See also http://php-nlp-tools.com.
[4] See also http://www.springer.com/de/book/9780387985039.

Table 3.1: Small cutout of the similarity matrix for a data corpus including the categories `Film` and `Politics`, which has to be created before starting with RStudio, for instance by means of the `Similarity` class of NlpTools.

Article ID	1	2	3	4	5
1	1.0000	0.0226	0.0382	0.0201	0.0429
2	0.0226	1.0000	0.0111	0.0331	0.0815
3	0.0382	0.0111	1.0000	0.0180	0.0296
4	0.0201	0.0331	0.0180	1.0000	0.0432
5	0.0429	0.0815	0.0296	0.0432	1.0000

row names, the euclidean distance matrix can be calculated by means of RStudio with `dist()`. This is needed for RStudio to create the graph with `cmdscale()` as following.

```
1 distances <- dist(mySimilarityMatrix)
2 mds <- cmdscale(distances, eig=TRUE, k=2)
```

To visualize the plot, the x- and y-axes have to be defined and labelled, as well as the type of the graph has to be determined. In addition, the object points on the graph need a label. In the following case they are named with their related category and can be coloured as well.

```
1 xAxis <- mds$points[,1]
2 yAxis <- mds$points[,2]
3 plot(xAxis, yAxis, xlab="x-axis", ylab="y-axis", main="Film vs Politics", type="n")
4 text(xAxis, yAxis, labels = firstCategory, cex=1, col = c("red","blue")[
      firstCategory])
```

The final result might look similar to the following example of the comparison of documents from the film and politics category in figure 3.2, which shows how similar the documents are.

This kind of visualization allows a detailed insight in the structure of data and can help comprehending the working process of algorithms dealing with this fundamental article corpus.

3.2 Working Environment

As already mentioned, Extbase and Fluid, the major milestones for extension development in TYPO3, are integrated in the system since 2009 with version 4.3 and are destined to gradually replace the extensions with piBase. The idea behind is to pave the way to TYPO3 Neos[5] with the introduction of Extbase, because of a better compatibility [65].

3.2.1 Extbase

Extbase itself is a PHP based framework, which ensures a clean separation between different concerns of code sections gaining benefits such as a simpler maintenance. With

[5] See also https://www.neos.io/.

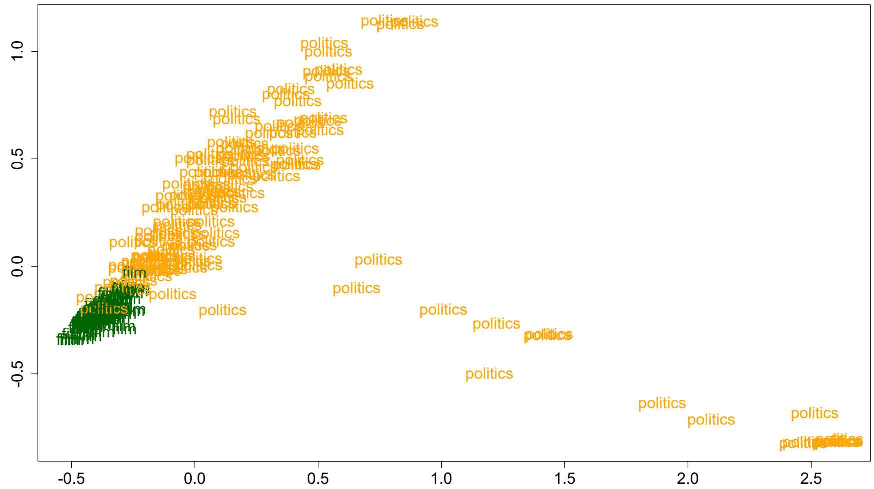


Figure 3.2: MDS graph of the documents from the category `Film` and `Politics`.

the introduction of Extbase, object oriented programming has been introduced to the extension development as well. Before this step, procedural programming, thus joining functions together, was used in 95% of cases for developing extensions [54].

Unfortunately, procedural programming involves some drawbacks like difficulties for re-using code, difficulties in finding errors because of confusing code and the possibility to alter properties everywhere, and properties and methods belonging to the same content could not be united, so encapsulating subjects was not possible. An object orientated approach enables transparent structures and a more realistic view of the code. An object, seen as container of data, mirrors a real object, such as a car, with properties and methods, representing possible functionalities and actions. All the features are present in code, when using object orientation. This heavily facilitates re-using and comprehending code. Extbase's modular approach and modern architecture now requires a different knowledge of developers. Concepts like Domain Driven Design or Model-View-Controller pattern become more decisive during implementation and especially before starting with it [53]. The following section gives a short overview of the relevant concepts in Extbase.

Domain Driven Design

In the field of software development, the usual task is to develop software for a distinct customer. This implies that the first key challenge is to understand the problem of the customer to finally offer him an adjusted solution. The domain driven approach focusses mainly on this understanding of a problem and the definition of it. The crucial factor is to define the problem in close cooperation with the customer to ensure that both sides have the same understanding of the issue. This mainly consists of developing a general model representing the problem, which can be subsequently the base for the final program. In detail, a common language have to be determined to facilitate communication and should be present in the source code as well. This implies core terms, names of entities,

values of objects, such as the types of properties, associations between objects, with regards to the database structure, aggregations of classes, and services defining the life cycle of objects. With resolving these issues, the "domain of the application can be efficiently packed into a software model". Concerning the last issue, when using Extbase the developer does not have to deal with the whole life cycle of objects, so there is no point of direct contact with the database layer and the persisting of objects. The layer, which takes care of the tasks, is named the repository layer and can be imagined as a real life library counter, where the user can get objects of a distinct type, read or edit it, save it or delete it. Thus, for each object type a repository is needed to manage these object instances [53].

Model-View-Controller Pattern

The Model-View-Controller design pattern, abbreviated MVC, divides an application into three rough layers: the Model, which contains the domain model and the corresponding logic, the Controller, which controls the process of the application as well as the communication between the Model and the View, and the View itself, which prepares the data and manages the presentation of the data to the user. The interaction between these three sections starts when the user sends a request object, containing information about the controller and the respective action, which should be executed. One sample action could be listing all properties of car-objects stored in the database. Thus, the object data from the model is required to satisfy the request. The domain objects can be received via the car repository layer, as mentioned above, and after getting the response as an array of car-objects, the data is forwarded to the view. Finally the View displays the car-objects and returns the response to the user. In the common version of the MVC patterns, the View is not only responsible for displaying data, it also listens for changes in the Model to enable an immediate reaction on changes. However, Extbase does not include the client-side of the view, only the server side, so it does not share a persistent connection in order to keep the view even more separate as in the classical MVC pattern. Hence, changes in the Model can not be presented immediately in the browser [53].

General Setup

After the installation of TYPO3, the extension relevant files and folders are located in `typo3conf/ext/`. On this level the extension folder has to be created, named after the extension key, containing all classes and configuration files. Furthermore, the classes are separated in two sections, the domain specific classes, including model and repository classes, and the controller classes. The last part of MVC, the view is situated in the resource folder consisting of various HTML files. Equally important are the three configuration files inside in the extension folder, `ext_emconf.php`, `ext_localconf.php` and `ext_tables.php` [53]. The rough concept can be seen in figure 3.3.

The first file is responsible to supply all crucial data to the Extension Manager to register the new extension including important details such as dependencies. The `localconf` file defines basic properties, the action and the controller name, which should be called when the extension is requested and all remaining actions and controller as well. In the third file, the location of the extension has to be determined with the

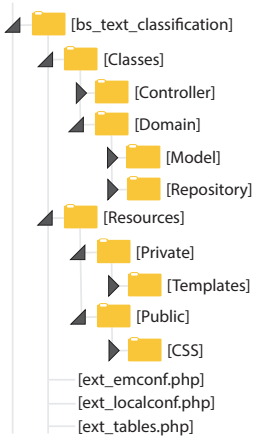


Figure 3.3: Rough concept of folder structure within TYPO3 extension folder.

extension key and a name. Afterwards, the controller can be created as well as the needed actions and variables can be forwarded to the view via `$this->view->assign()` [48]. The system configuration is completed by this and detailed information about the further implementation can be found in chapter 4.

3.2.2 Fluid

Above all, Fluid focuses on the view part of the system. It is a template engine, which is used to display content on a website and was specially developed for TYPO3 and FLOW3. The basic concept of each template engine including Fluid, is to process a template file and replace all placeholders found in it with the content the engine gets from the controller. In principle, Fluid draws upon three major components enabling scalability and flexibility: the Object Accessors, the View Helpers and Arrays. When receiving content from the controller, the template engine has to fill the content into the specific areas, indicated by placeholders. These placeholders are called the Object Accessors, because they are allowed to access the data. The Object Accessors are distinguished by two curly brackets. For instance, {`carColour`} would display the content of the variable `carColour`. The variable itself has to be assigned in the controller with the following syntax.

```
$this->view->assign(variableName, object)
```

To display more complex objects, a View Helper is needed, which are automatically imported in Fluid. They are indicated by the prefix `f:` and can adopt multiple functionalities such as for-loops, if-else conditions, links or a form with several field types. The following code snippet, for instance, shows a for-loop listing car objects with their names and related colour. Via the dot notation the properties, like name and colour, can be accessed [53].

```
<ul>
    <f:for each="{cars}" as="car">
        <li>{car.name} : {car.colour}</li>
    </f:for>
</ul>
```

Arrays, which can be compared to associative arrays in PHP, are fundamental in Fluid to pass multiple variables from the controller to the view and also the other way round. They can contain strings, numbers, objects and sub-objects simultaneously, the only thing they need is a key. In Fluid it is possible to pass a number of variables with a View Helper to a controller. The prime example is a link, where an action and the corresponding controller can be declared, to get retrieved by clicking the link. With the `arguments` property, multiple variables can be passed on to this action. Separating the values by commas, indicates that the values are forwarded in form of an array [53]. The following line shows an example link with action and argument.

```
<f:link.action controller="Cars" action="show" arguments="{car: currentCar, id: 10}"
    >Show current Car</f:link.action>
```

3.2.3 Project Use Case

Manual news organization in nowadays news agencies constitutes of hard effort. A small piece of software can be very useful and lighten someone's workload. The main idea behind this project was to compare different algorithms, which can be used for text classification, to finally build an extension which is able to classify an article based on the learned corpus of training examples. It includes different ideas such as entering an URL and the program logic has to classify the text found on this page, or entering a text directly, which gets classified. Another possibility is to enter a topic and all articles classified to it get listed.

The primary focus of the extension lies on the scientific aspect and on the new approach of the semantic fingerprinting method, compared with other classification techniques.

3.3 Data Source

The aim of the thesis project is to compare the semantic fingerprinting method for classifying articles with the two classical algorithms, k-nearest-neighbour and Naive Bayes. In order to achieve this goal, a great number of articles is needed, which have to live up to certain conditions:

- have to be in English,
- have to be assignable to one specific category,
- have to be real life data and
- have to be of a certain length (more than 300 characters).

Especially to fulfill the third condition, to have realistic data, the decision was made in favour of online news agencies, where a great number of texts are available. Categories are also present in most of the agencies; thus, the main issues are widely satisfied. The decision to chose *The Guardian*, was mainly because of the well structured construction of the page, the possibility to move back and get past articles, and the subdivision into categories (see figure 3.4). With the decision for www.theguardian.com a stable and reliable test environment for the algorithms has been created.

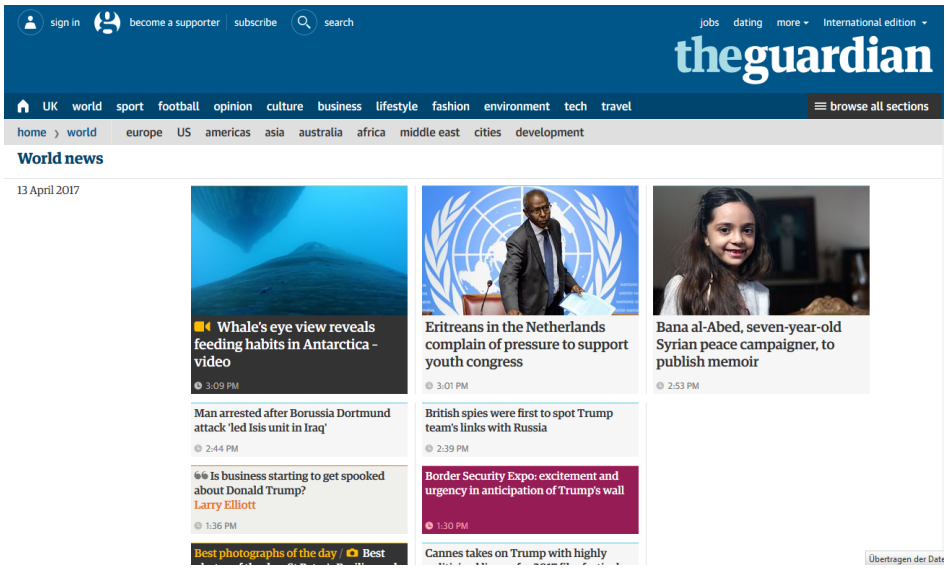


Figure 3.4: General structure and subdivision in categories on www.theguardian.com

Program 3.1: Using cURL for collecting the content from *The Guardian*.

```
1        $ch = curl_init();
2        curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
3        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
4        curl_setopt($ch, CURLOPT_URL, $url);
5        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
6        curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 5);
7        $data = curl_exec($ch);
8        curl_close($ch);
```

In order to get the data from *The Guardian*, the PHP supported cURL library from Daniel Stenberg was used[6]. It enables connections between various servers with diverse protocols. With this library the content of multiple guardian pages has been collected. The cURL commands can be seen in program 3.1.

After receiving the content of a page, this content can be loaded into a `DOMDocument`, a parser object, to parse the string containing the complete HTML from *The Guardian*, back into its DOM tree[7](see program 3.2). This can now be used to select specific elements such as the article text between certain tags (see program 3.4), or to create a `DomXPath`, a path object, and a related query, to get, for example, a list of all links on the page (see program 3.3). These path objects are comparable with directories and folders.

Nevertheless, after extracting the article content, the category, a short description, the title and the publishing date are forwarded to prepare it for further processing. Detailed information about the pre-processing stage continues in chapter 4.

[6] See also http://php.net/manual/en/intro.curl.php.
[7] See also https://www.w3schools.com/js/js_htmldom.asp.

Program 3.2: Create DOMDocument and load HTML content.

```
1 $doc = new \DOMDocument();
2 $doc->loadHTML($data);
```

Program 3.3: Create a query to get all links with "data-link-name".

```
1 $finder= new \DomXPath($doc);
2 $links = $finder->query("//a[@data-link-name='$attr']/@href");
```

Program 3.4: Get content between two specific tags.

```
1 $pattern = "#<$tag>(.*?)</$tag\b[^>]*>#s";
2 preg_match_all($pattern, $string, $matches);
```

The collected data set finally consists of twenty different categories, has the total number of 2295 articles, and each category inheres between 80 and 150 files. The precise breakdown can be seen in table 3.2.

Table 3.2: Distribution of Article Resources per Category.

Category	**Articles**	**Category**	**Articles**
Art and Design	83	Politics	117
Books	103	Science	91
Business	125	Society	113
Culture	102	Sport	142
Environment	125	Technology	104
Fashion	130	Television/Radio	114
Film	115	Travel	133
Football	148	UK News	111
Life and Style	114	US News	102
Opinion	120	World News	103

Chapter 4

Implementation

The following chapter describes the technical process in detail, starting with the pre-processing part, including the usage of the NlpTools, proceeding with the k-nearest-neighbor implementation, the Naive Bayes implementation and concluding with the Semantic Fingerprinting method. The chapter provides detailed insights into the procedure of text classification.

4.1 Pre–Processing

After receiving the raw data of the online articles from the collecting and saving procedure, the primary objective is to split the text into single words, reduce the complexity of features and to convert them further into respective feature vectors. However, one of the first steps in the pre–processing pipeline to finally achieve this aim is to remove any tokens, which could potentially confuse the text classification process, such as HTML tags and every kind of punctuation. This can be done with default PHP functions like `strip_tags` and `preg_replace`. Details can be seen in the following code snippet:

```
$dataContent = strip_tags($dataContent);
$dataContent = trim(preg_replace("/[^0-9a-z ]+/i", "", $dataContent));
```

Subsequently, it is necessary to convert the letters into lower case to reduce the feature variety on the one hand and to prepare the words for stemming on the other hand. For this process the `normalize` function of the `English` class from the NlpTools library is used. In case of sentiment analysis for example, it can be useful to keep upper case words to detect special weightings for certain word sequences. Thus, to give them more importance because of the notation, but in news classification upper case letter most commonly refer to the beginning of a sentence.

Up to this point the article texts are processed in one large string, which gets edited through the different functions. The next task is the Tokenization process, which splits the string into single words. There are several approaches for splitting strings such as splitting on white space or splitting on punctuation. However, in the project's case the string becomes split on white space with the `WhitespaceTokenizer` of the NlpTools.

After the Tokenization, the texts are present in form of arrays containing all words, which can be filtered now to reduce the size of the feature set further. Removing frequently used words, the stop words, like prepositions (“after”) or verbs (“do“), is the sec-

Program 4.1: First part of pre-processing procedure including Normalization, Tokenization and Stop Word Reduction.

```
1  $norm = new English();
2  $string = $norm->normalize($string);
3
4  $wtok = new WhitespaceTokenizer();
5  $array = $wtok->tokenize($string);
6
7  $stop = new StopWords($this->stopwordsENG);
8  foreach ($array as $key => $value) {
9    $array[$key] = $stop->transform($value);
10 }
11
12 $array = array_filter($array);
13 $array = array_values($array)
```

ond stage of the pre–processing pipeline. The reduction can be done with the `StopWords` class of the NlpTools library. When initializing the `StopWords` object, a stop words list can be delivered, which is then used to filter the arrays of words. Iterating over the words, the function sets or unsets array values and thus removes terms, which are contained in the stop word list. Afterwards, the empty array spots have to be removed and the keys of the remaining values refreshed. For this part the default PHP functions `array_filter` and `array_values` are used. The process can be seen in program 4.1.

At this point, the filtered and normalized terms are saved in the database referencing the article with the respective category. However, the next step includes the removal of digits, done by the PHP function `preg_replace`. In general, numbers can also be kept in the text, but evaluations on data with removed digits achieved significantly better results, so they are dropped.

Another important task of the pre-processing pipeline is stemming, which is done by the `PorterStemmer` class. The general idea of stemming is to reduce all words with the same root to one common form to decrease the complexity of the feature set. This is usually done by removing the derivational and inflectional suffixes from each word. In the project it is taken on by the `transform` function of the `PorterStemmer`. Due to the transformation, some extremely short terms (less than three characters) can occur in the array. These words have no remarkable information value, so they can be dropped. The same applies for extraordinary long words, which can come up through editing the text (more than 20 characters). The respective code snippet can be seen in program 4.2.

Finally, when the features are selected, filtered and edited, the last part of the pipeline initiates. It concerns the transformation from words to respective numbers, the weighting. Here again, NlpTools takes on the main functionality. In general, for the calculation of the *tf-idf* value, the library needs an object, named `TrainingSet`, which contains multiple `TokensDocument` objects containing the word vectors. This implies, after initializing the `train-Set`, each document of the training data, gets converted into a bag of words array, delivered to a `TokensDocument` to finally get added to the `trainSet`. The notation can be seen in program 4.3.

Program 4.2: Second part of pre-processing procedure including digit removal and Stemming.

```
$content = preg_replace('/[0-9]+/', '', $content);
$stem = new PorterStemmer();
foreach ($array as $key => $value) {
  $array[$key] = $stem->transform($value);
}
foreach($array as $k => $v){
  if(strlen($v) <3 || strlen($v) > 20 ){
      unset($array[$k]);
  }
}
```

Program 4.3: Creation of the `TrainingSet` for the weighting process.

```
$trainSet = new TrainingSet();
foreach($this->trainingsData as $document){
  $content = $document->getTerms();
  $array = $this->prepareData($content);
  $trainSet->addDocument("", new TokensDocument($array));
}
```

Program 4.4: Computation for the *idf* value.

```
foreach($tokens as $token=>$v){
  if (isset($this->idf[$token])){
      $this->idf[$token]++;
  }else{
      $this->idf[$token] = 1;
  }
}
$D = count($tset);
foreach ($this->idf as &$v){
  $v = log($D/$v);
}
```

Thereafter, the training set is passed to the `Idf` class, whose constructor calculates the *idf* value for each token. Counting the occurrences of terms, inverting it by dividing it through the total number of tokens and taking the logarithm as it can be seen in program 4.4. For detailed information, have a look inside the class `Idf` of the NlpTools.

Afterwards, the *idf* values are submitted further to the `TfIdfFeature Factory`. This class is not included in the library, but provided in the library's documentation[1]. It gets the term frequencies from its parent class `FunctionFeatures`, and multiplies this value with the respective *idf* value of the term. For reasons of normalization in the thesis project, the *tf* value gets additionally divided by the total number

[1] For the documentation see http://php-nlp-tools.com

of terms in the document to avoid a distortion of the results from long articles. The procedure can be seen in the code snippet below:

```
$frequencies = parent::getFeatureArray($class, $doc);
$numbFeatures = array_sum($frequencies);
foreach ($frequencies as $term=>&$value) {
  $value = ($value/$numbFeatures)*($this->idf[$term]);
}
```

The process of calculation gets triggered by the `getFeatureArray` function, applied to each document in the `trainSet`. Thereafter, the array of training vectors is filled with each token and its corresponding weight. The notation for the entire computation process is as follows:

```
$idf = new Idf($trainSet);
$featureFactory = new TfIdfFeatureFactory(
  $idf,
  array(function ($c, $d) {return $d->getDocumentData();})
);
foreach($this->trainingsData as $key => $d){
  $trainVector[$key] = $featureFactory->getFeatureArray("", $trainSet[$i]);
}
```

4.2 Algorithms

The subsequent section describes in detail the implementation of the algorithms used for classifying the news articles. After pre–processing, the data is present in form of word vectors for each article with the respective weight, which can be processed further in the algorithms kNN, Naive Bayes and Semantic Fingerprinting.

4.2.1 k-nearest-neighbor

The success of the k-nearest-neighbor is mainly based on a similarity function to detect the k most similar documents in the training set to finally determine the class of the unlabeled text. Just before starting to calculate the similarities, the whole corpus has to be split into a training data set and a testing set. The relation of the sizes can be chosen variable. For the thesis project multiple ratios from 67 to 33 up to 90 to ten have been tested. Finally the decision is made in favour of 80% train to 20% test data, because of better end results. The splitting is done by the default PHP function `array_slice` (see code snippet below).

```
$this->trainingsData = array_slice($this->dataTerms, 0,$trainingNumb,true);
$this->testData = array_slice($this->dataTerms, $trainingNumb,$testingNumb,true);
```

Thereafter, each document in the test data can be compared with the training data to get the similarities. The function `cosineSim` starts the comparison by iterating over the training data and delivering the test terms and the training terms to the `CosineSimilarity` object. The `similarity` function of it returns the respective similarity values, which get collected and stored. See the `cosineSim` function in program 4.5.

The `similarity` function, contained in the NlpTools, iterates over the terms of the first document, checking if the term is contained in the second document and if this is the

Program 4.5: Function for calculating all similarities between the test document and the training data.

```
function cosineSim($testTerms,$sim){
  $distances = [];
  foreach($this->trainingsData as $key => $value){
     $distances[$key] = $sim->similarity($testTerms,$this->trainingsData[$key]);
  }
  return $distances;
}
```

Program 4.6: Function for calculating the cosine similarity between two arrays.

```
public function similarity($v1, $v2){
 $prod = 0.0;
 $v1_norm = 0.0;
 foreach ($v1 as $i=>$xi) {
   if (isset($v2[$i])) {
       $prod += $xi*$v2[$i];
   }
   $v1_norm += $xi*$xi;
 }
 $v1_norm = sqrt($v1_norm);
 $v2_norm = 0.0;
 foreach ($v2 as $i=>$xi) {
         $v2_norm += $xi * $xi;
 }
 $v2_norm = sqrt($v2_norm);
 return $prod/($v1_norm*$v2_norm);
}
```

case, the weights are multiplied and summed up to the inner product. Simultaneously, the sum for the magnitude of the word vector is computed. After the iteration the square root has to be extracted from the sum to get the length of the first vector. The same procedure of calculating the vector length has to be done for the second document. Finally, the inner product can be divided with the product of the two vector lengths, resulting in a value describing the angle between the two word vectors. The closer the documents, the bigger is the value in a range from minus one to one. Notation of the function can be seen in program 4.6.

Another possibility to define similarities, is to use the `Euclidean` class of the Nlp-Tools. In this case the `dist` function calculates the distances between the documents and determines the k closest documents on this way. The computation is started by the `euclidSim` function in the respective kNN class.

The computation itself consists of recording all words contained in both articles and if a word occurs in both, the weights are subtracted. At the end of the function, all values of the words get squared and summed up. Finally, the square root of the sum results in the distance between the documents, because of vector calculus as explained in section 2.3. When using this method, it implies that the smaller the value, the closer

Program 4.7: Function for calculating the euclidean distance between two arrays.

```
1 public function dist($v1, $v2)
2   $r = array();
3   foreach ($v1 as $k=>$v) {
4     $r[$k] = $v;
5   }
6   foreach ($v2 as $k=>$v) {
7     if (isset($r[$k])){
8         $r[$k] -= $v;
9     }else{
10        $r[$k] = $v;
11    }
12  }
13  return sqrt(array_sum(array_map(function ($x) {return $x*$x;},$r)));
14 }
```

Program 4.8: Testing function and final prediction of the class for each test document.

```
1  foreach($testData as $key => $test){
2     $data = $knn->cosineSim($test,$sim);
3     arsort($data);
4     $topK = array_slice($data,0,$k,true);
5
6     foreach ($topK as $c =>$value) {
7       $a=trim(strtolower(strstr($dataTerms[$c]->getArticleID()->getCategory(), ' '))
      );
8     }
9
10    $countCat = array_count_values($categories);
11    arsort($countCat);
12    $predictedCat = current(array_keys($countCat));
13  }
```

and more similar are the documents. See the implementation contained in the NLpTools library in program 4.7.

When assembling the single parts of the algorithm, a similar procedure and implementation as the following can be the result. During the iteration of the different test documents, the similarities get determined, sorted from big to small and sliced to get the k most similar documents. Thereafter, the categories of the top k articles have to be figured out, to decide which category should be assigned to the test document. In general there are multiple ways of determining which category wins. One possibility would be to decide in favour of the most votes, which can be seen in program 4.8. This means the class which most frequently occurs in the top k documents, will be kept as the predicted category. To prevent a tie, an odd number has to be taken for the actual k value.

Another approach can be to decide in favour of the highest similarity per class. In the thesis project both versions have been tested and the highest similarity approach

generated better results. This implies, after summing up the similarities per category, the ranking has to be sorted again from big to low and the first one, the category with the highest similarity rate, is kept as prediction for the test document. The notation of the process can be seen below:

```
foreach($weights as $i => $val){
  $weightingRank[$categories[$i]] = $weightingRank[$categories[$i]]+($weights[$i])
  ;
}
arsort($weightingRank);
$predictedCat = current(array_keys($weightingRank));
```

This implementation process describes the testing phase of the kNN algorithm with the aim to detect the accuracy and the success rate of it. In a case where only one new document has to be classified, the corpus does not have to be split into training data and testing data. In such particular application the whole corpus will be taken as reference and training set for the assigning process.

4.2.2 Naive Bayes

The basic idea of the Naive Bayes algorithm is to use conditional probability to assign a class to a new input document. The method takes a close look on the words contained in the unknown article and computes for each word the probability that the document has a certain class when it is given that the word is comprised in the text. It determines the conditional probability for each word, summing it up to a joint probability and calculates this value for each category. The class with the highest probability is finally assigned to the document.

The overall implementation of the Naive Bayes is inspired by the following resources [47, 52, 57] and starts the same as the kNN. It splits the data corpus into a training and a testing set at the ratio of 80 to 20. Afterwards, the classifier has to be trained, which means the method has to analyse the training data, it has to work with. The `trainClassifier` function is consequently the first important part of the Naive Bayes algorithm.

Training the classifier beforehand, saves time and performance effort, especially for the prediction part. The method records each article and fills four different arrays: the `terms` array, the `classes` array, the `documents` array and the `data` array. Starting from the coarsest, the `documents` array stores the total number of articles per class, the `classes` array contains the count of words per class and the `terms` array contains all words contained in the training corpus with the corresponding amount of occurrences in the whole data set. However, the `data` array saves only the words and its quantity per class. After passing through the articles and storing the numbers, a major and important task has been ensured. The notation can be seen in program 4.9.

Compared to the kNN implementation, the Naive Bayes prediction phase, which is shown below, seems extremely short and simplified.

```
foreach($testData as $key => $value){
  $probabilities = $naive->classifyDocument($testData[$key][1]);
  $predictedCat = current(array_keys($probabilities));
}
```

Program 4.9: Training the Naive Bayes Classifier beforehand.

```
protected function trainClassifier($class, $termArray){
  if (!isset($this->classes[$class])) {
      $this->classes[$class] = 0;
      $this->data[$class] = [];
      $this->documents[$class] = 0;
  }
  foreach ($termArray as $term) {
      if (!isset($this->terms[$term])) {
          $this->terms[$term] = 0;
      }
      if (!isset($this->data[$class][$term])) {
          $this->data[$class][$term] = 0;
      }
         $this->classes[$class]++;
         $this->terms[$term]++;
         $this->data[$class][$term]++;
   }
  $this->documents[$class]++;
}
```

The function iterates over the test documents, submits their word arrays to the `classifyDocument` method, which returns a sorted probability breakdown for each class. The class with the highest probability gets assigned to the test document.

The core functionality of the Naive Bayes and thus its second significant part is included in the `classifyDocument` function, which returns a ranking of the classes contained in the training corpus. The underlying approach of the function is to look separately on each class and the terms in the documents contained in these classes.

Leading an example back to the Bayes rule it might look like the following and can be summarized in the question: what is the probability that the latest article is an article about sport, given that it contains the word game. The equation is

$$p(sport|game) = \frac{p(game|sport)p(sport)}{p(game)}. \tag{4.1}$$

At this point, with this formula only one word of the news article has been taken into account. To get the joint probability the calculation has to be applied to each word. The computation has to be repeated for the remaining classes as well to finally choose the class with the highest probability value.

This implies, back in the implementation, that the process starts with the iteration over the `classes` array, the array which stores the different categories with the corresponding overall number of words. Simultaneously it iterates over the terms of the test document submitted to the function and checks if the terms are contained in the training corpus, thus in the `terms` array. If a term exists in the training data, the number of appearances in it is saved in the `totalTokenNumb` variable. This number does not consider in which classes the term occurs, it only shows how often the term is included in the training corpus. Prior to that, the total amount of documents in the corpus as well as the number of documents in the class and outside of it, are stored for the subsequent

calculations. The notation of the beginning of the prediction process is shown below:

```
1 public function classifyDocument($testDocument){
2   $totalDocCount = array_sum($this->documents);
3
4   foreach ($this->classes as $class => $classCount) {
5     $log = 0;
6     $docCount = $this->documents[$class];
7     $inversedDocCount = $totalDocCount - $docCount;
8
9     foreach ($testDocument as $term) {
10      $totalTokenNumb = 0;
11      if(isset($this->terms[$term])){
12        $totalTokenNumb = $this->terms[$term];
13      }
```

Thereafter, it continues with the class specific analysis. This section checks if the term is contained in any document in the specific class by browsing through the `data` array and saves the total count of it in the `tokenNumbInClass` variable. After storing these two values, the computation part begins. The next snippet of the prediction process is written as follows:

```
14   if ($totalTokenNumb === 0) {
15     continue;
16   }else{
17     $tokenNumbInClass= 0;
18     if(isset($this->data[$class][$term])){
19         $tokenNumbInClass = $this->data[$class][$term];
20     }
```

At this stage, there are two different implementations of the computation of the occurrences of a term in a class. A common way is to count the documents in the class in which the term is contained. This implies, that the terms in the documents have to be unique and it is only decisive if the term occurs in the text or not. This number is then divided by the total amount of documents in the class, to get the word probability. The same procedure is kept also for calculating the inverse word probability.

However, another approach is to have a look on the amount of appearances of a term in a specific class, without deleting the duplicates of it. This value is then divided by the total number of words in the class, resulting again in the word probability. When we are given the numbers, how often a term occurs in all classes and how often a term occurs in a specific class, the number, how often a term occurs outside the specific class can be easily calculated by subtraction. This value is needed to determine the inverse word probability. In conclusion, the level of the documents, has been skipped in this approach.

The word probability value now shows the probability, that a term occurs in this category, which is the first part of the Bayes rule. The second part of it, concerns the already mentioned inverse probability, which is the probability that the term occurs in any remaining class. It can be computed by dividing the outside token count of the term with the total number of words outside the one certain class.

```
21   $inversedTokenNumb = $totalTokenNumb - $tokenNumbInClass;
22   $outsideClassCount = array_sum($this->classes)-$classCount;
23   $wordProbability = $tokenNumbInClass / $classCount;
24   $inversedWordProbability = $inversedTokenNumb / $outsideClassCount;
```

If the first approach has been used the following two lines change, under the assumption that the words in the documents are unique.

```
23   $wordProbability = $tokenNumbInClass / $docCount;
24   $inversedWordProbability = $inversedTokenNumb / $inversedDocCount;
```

Afterwards, the conditional probability can be defined. This value consists of the word probability, the probability that the word occurs in this class, or in the documents of the class, divided by the probability that the word occurs in any class or document at all. This summarized probability is calculated by adding the probability that the term occurs in this class, or in the documents of the class, and the probability that the term occurs outside this class, the inverse probability. All in all, the calculation results in the probability that the document has the certain class, when it is given that this certain word is part of it.

```
25   $probability = $wordProbability / ($wordProbability + $inversedWordProbability);
```

After receiving the first probability with the first term, the process is repeated for each following term in the test document. To get the joint probability for the whole test document, the single probabilities are usually combined via multiplication of each, divided by the multiplication of each inverse probabilities, which indicates the likelihoods that the searched document has some other class. These multiplications often result in extremely small values, which can not be utilized further. Thus, the logarithms should be applied to both parts of the computation. Thus, the formula is then

$$\ln \frac{p(c|w)}{p(\neg c|w)} = \ln \frac{p(c)}{p(\neg c)} + \sum_{i=1}^{n} \ln \frac{p(w_i|c)}{p(w_i|\neg c)}. \tag{4.2}$$

In this kind of classification task, it is not important to know the exact values for each class. The decisive part is to detect the most probable class. According to the rules of logarithm, the fraction can be transformed into a subtraction of nominator and denominator. With the `exp` function of PHP, the logarithm can be undone afterwards and the value restored. The most probable class is finally assigned to the document, even if the number is not the exact probability value.

```
27   $log += (log($probability) - log(1 - $probability));
28   $log += log(($docCount/$totalDocCount)/($inversedDocCount/$totalDocCount));
29   $probabilities[$class] = exp($log);
30   arsort($probabilities, SORT_NUMERIC);
31   return $probabilities;
32 }
```

The two approaches of the implementation have been both evaluated. Overall, the one which takes the exact word number into account performed slightly better than the one which only monitors in how many documents the word is contained. The remaining test cases and results of the Naive Bayes algorithm are explained in detail in chapter 5.

4.2.3 Semantic Fingerprinting Method

The following reimplementation is based on the theory of Semantic Folding and Semantic Fingerprinting developed by cortical.io. The overall implementation can be separated again into two parts, the preparation procedure of the classifier and the classification

part itself. In this kind of algorithm, the preparation part is a lot more sophisticated and time expensive than the classification part. The explanation of the implementation starts with a rough description of the preparation procedure and goes into detail afterwards.

The very first step of preparing the classifier is to split the data set into training data and testing data. Thereafter, the words of the training data get weighted with *tf-idf*, which is needed to create the context map in an ordered way. Subsequently, the creation of the map initiates as well as the storing of the labelled context map, which is needed for later calculations.

During the following iteration over the train vectors, the classes and the respective words get collected and recorded. Finally, this resulting array is used for the construction of class fingerprints, by going through each class and each word, and summarizing the binary maps to one fingerprint per category. The notation of the starting process can be seen below:

```
$this->tfidf();
$this->contextLabelMap = $this->createContextMap();
foreach($this->trainVector as $key => $doc){
   $cat = $this->dataTerms[$key]->getArticleID()->getCategory();
   $this->prepareClassifier($cat,array_keys($this->trainVector[$key]));
}
foreach($this->data as $class => $words){
    $this->createCategoryFingerprints($class,$words);
}
```

After the weighting process, the context map gets created in the `createContextMap`-function. The general task of this function can be roughly summarized, in placing training documents in an array according to their similarities to each other. The result then should be an array which contains all IDs of training documents, where similar documents are close to each other and dissimilar ones are far apart. This object is later used to encode words to their fingerprints, so it is a underlying grid template for the fingerprints.

In particular, the function iterates over all training vectors, which represent the documents consisting of words and their weights, and collects all similarities between the new document and each document already existing in the context map. If the new document is not the first one to place, it extracts the ID of the most similar document on the context map and pastes the ID of the new document behind it in the array and thus, on the context map. The notation for this procedure can be seen in program 4.10.

During the computation of the context map, the respective labels of the document IDs have been recorded as well and stored in the `contextLabelMap` variable. This variable is used in the classification task.

The next step is to collect all terms for each class in the training set and saving them in the `data` array with the according number of occurrences in the category. This procedure is similar to the preparation of the Naive Bayes classifier. Afterwards, the encoding of category fingerprints can be started.

The creation of category fingerprints is the main part of the preparation procedure. These fingerprints represent each class in form of binary vectors and are finally used to compare the new test document fingerprint with each of them. In general, to get these fingerprints, the function iterates over each word in a class and each document on the context map once, and checks if the word exists in these documents. If this is the case,

Program 4.10: Function creates the context map using the similarity value as decisive factor.

```
protected function createContextMap(){
 foreach ($this->trainVector as $textID => $val) {
    $indexSim = [];
    $sim = new CosineSimilarity();
    $cat = trim(strtolower(strstr($this->dataTerms[$textID]->getArticleID()->
  getCategory(), ' ')));

    for ($i = 0; $i < count($this->contextMap); $i++) {
        $indexSim[$i] = $sim->similarity($this->trainVector[$textID], $this->
  trainVector[$this->contextMap[$i]]);
    }

    if (count($indexSim) == 0) {
        $this->contextMap[0] = $textID;
    } else {
        arsort($indexSim);
        $pastBehind = current(array_keys($indexSim)) + 1;
        array_splice($this->contextMap, $pastBehind, 0, $textID);
    }
  }
}
```

a one is put into this certain place in the fingerprint array of the respective word. In the opposite case, a zero is added to it. After converting each word of the class into its fingerprint, they are stacked over each other, summed up and sorted from high to low. Then the `x` highest stacks are cut out, determined by a `threshold` value, and the keys of them are set to one in the final fingerprint. The remaining places in the fingerprint stay zero and get filtered out at the end to save memory. The resulting array consists of IDs of training documents and ones, describing the category and its identification as fingerprint. The function can be seen in program 4.11.

To get the whole stack of all word fingerprints, the helper function `getSumSDRs` sums them up immediately, which saves processing time. At this point, it should be mentioned that the `numb` parameter holds the actual term count inside the class and to improve the choosing of the right stacks, this value is here additionally included in the computation. Afterwards, the `getClassSDR` function only has to check if each stack is contained in the x highest stacks and sets the certain value in the class fingerprint to one or zero. The two helper functions are shown in program 4.12. With the construction of the semantic fingerprints for each of the categories existing in the training corpus, the preparation of the classifier is completed.

A different approach of creating category fingerprints would be to take one document of each class and convert it into a text SDR, which is then used for the classification. For this method, the crucial factor is that the example text document should definitely be a prime example for the category to have a good pattern template for the comparison with the test document fingerprint. Otherwise it would have adverse effects on the results of the classification.

Program 4.11: Function creates semantic fingerprints for all categories.

```
protected function createCategoryFingerprints($class,$words){
  $this->categoryFingerprints[$class] = [];
  $categoryFP = array_fill(0,count($this->trainVector),0);
  foreach($words as $word => $numb){
      $categoryFP = $this->getSumSDRs($word,$categoryFP,$numb);
  }

  arsort(categoryFP);
  $topX = array_slice($categoryFP,0,$this->threshold,true);

  $this->categoryFingerprints[$class] = $this->getClassSDR($categoryFP,$topX);
  ksort($this->categoryFingerprints[$class]);
  $this->categoryFingerprints[$class] = array_filter($this->categoryFingerprints[
    $class]);
}
```

Program 4.12: Helper functions getSumSDRs() and getClassSDR().

```
protected function getSumSDRs($word,$categoryFP,$numb){
  foreach($this->contextMap as $key => $index) {
      if(array_key_exists($word,$this->trainVector[$index])){
          $categoryFP[$key] +=  1*$numb;
      }else{
          $categoryFP[$key] += 0;
      }
  }
  return $categoryFP;
}
protected function getClassSDR($stackedFP,$threshold){
  $array = null;
  foreach($stackedFP as $id => $value){
      if(array_key_exists($id,$threshold)){
          $array[$id] = 1;
      }else{
          $array[$id] = 0;
      }
  }
  return $array;
}
```

Going further, to the classification part itself, the testing function is kept very similar to the one for the Naive Bayes, which iterates over each single test document, classifies it and gets the similarities of each class back. To determine the similarity value between the different class fingerprints and the test document fingerprint, the cosine similarity has been used. The class with the highest similarity is then assigned to the document. This kind of decision criterion is called the semantic closeness or binary overlap.

A different approach for deciding which category will be chosen is to have a close look on the positions of ones in the fingerprint. For this method, it is not necessary

to compare fingerprints. By detecting the category of the underlying document in the context map and summing up the ones per category, can also be a possible way to get most overlaps between the test document and the categories. This method uses the shared context between the fingerprints as decision criterion. In the thesis project both forms of decision factors have been tested, but the first approach worked slightly better overall, even if in single test cases the second approach delivered better results. The notation of the testing method can be seen below with both possibilities.

```
foreach($testData as $key => $value){
  $pack = $fingerprinting->classify($testData[$key],false);

  $similarities = $pack['prob'];
  $overlaps = $pack['over'];

  $predictedCat01 = current(array_keys($similarities));
  $predictedCat02 = current(array_keys($overlaps));
}
```

The classification process itself is carried out in the `classify` function in the Semantic Fingerprinting class. This function retrieves the test documents and starts with the preparation of the test data, including the removal of numbers and the splitting into a bag of words object. Afterwards, duplicate values in the document get removed to simplify the word vector, which is then forwarded to compute its semantic fingerprint in the respective function. The function will be explained subsequently. It is the second important part of the classification process.

```
1 public function classify($testDoc){
2   $testTerms = $this->prepareData($testDoc->getTerms());
3   $testTerms = array_unique($testTerms);
4   $package = $this->createTestDataFingerprint($testTerms);
5   $fingerprint = $package['fp'];
```

After the construction of the test document fingerprint, the comparison phase can be started. During an iteration over the category fingerprints, each class gets compared to the test fingerprint and by computing the cosine similarities between them, a ranking can be computed, which is finally used for assigning a class to the document.

```
6   $sim = new CosineSimilarity();
7   foreach($this->categoryFingerprints as $cat => $fp){
8       $probabilities[$cat] = 0;
9       $probabilities[$cat] =  $sim->similarity($fp,$fingerprint);
10  }
```

By this, the result can be defined, but to have the opportunity to test the different approaches of decision criterion, a second iteration starts. In this case, the test fingerprint itself is disassembled, by determining the class of the document under each one in the fingerprint. For this process the labeled context map is used to get the underlying classes. The values are summed up for each class and the scores sorted.

```
12  $overlaps = [];
13  foreach($fingerprint as $i => $k){
14    if($k > 0){
15      $overlaps[$this->contextLabelMap[$this->contextMap[$i]]]++;
16    }
17  }
18  arsort($overlaps);
```

Program 4.13: Creation of test data fingerprint.

```
protected function createTestDataFingerprint($testTerms){
    $testFP = array_fill(0,count($this->contextMap),0);
    foreach($testTerms as $k => $word){
        $testFP = $this->getStackOfWordSDRs($word,$testFP,1);
    }
    arsort($testFP);
    $topX = array_slice($testFP,0,$this->customThreshold,true);
    $fpPackage['fp'] = $this->getClassSDR($testFP,$topX);
    $fpPackage['fp'] = array_filter($fpPackage['fp']);
    ksort($fpPackage['fp']);
    $presentClass = [];
    foreach($topX as $id => $stack){
        $presentClass[$this->contextLabelMap[$this->contextMap[$id]]]++;
    }
    arsort($presentClass);
    $fpPackage['frequentClass'] =  current(array_keys($presentClass));
    return $fpPackage;
}
```

Before returning the two results, one last check has been added to the `classify` function. In case that the two top classes have the same value, it would be random which class of the two will be chosen. To avoid this, the two top classes get checked, and if it is indeed the circumstance of two similar values, the class which occurs most frequently in the top x stacks, gets a boost. After another sorting process, the values get returned to the prediction function.

```
  arsort($probabilities);
  $class = $package['frequentClass'];
  $check = array_values(array_slice($probabilities,0,2,false));
  if($check[0] == $check[1]){
      $probabilities[$class] += 0.0001;
  }
  arsort($probabilities);
  $pack['prob'] = $probabilities;
  $pack['over'] = $overlaps;
  return $pack;
}
```

The construction of the test data fingerprint works the same as the creation of the category fingerprints. Only the number of stacks, which are taken into account is defined by a custom threshold, which can be the same as the class threshold, but can also differ. Additionally, at the end of the function, the detection of the most common class in the top x stacks follows. Therefore, the classes of the IDs in the context map get collected and sorted. Finally, the highest value in the present classes gets returned, together with the final test data fingerprint. The function for creating the test data fingerprint is shown in program 4.13.

With this implementation, a stable and solid test area is created. The environment is used to detect the characteristics and criteria for a text classification task with Semantic Fingerprinting and the test results will be explained in detail in chapter 5.

Chapter 5

Evaluation

The following chapter provides a detailed insight into the results of the k-nearest-neighbor, Naive Bayes and Semantic Fingerprinting classifiers, an overview of the development process and a final comparison of the performances. During the evaluation, two different data sets are used. The first set is the result of the first data gathering and is smaller than the second one. Unfortunately, it is additionally unequally distributed on class level. First tests revealed that more data is needed to create a balanced data set to have meaningful evaluation results. Finally, all classifiers have been tested on both data sets.

The initial setup of the evaluation phase contained 731 different articles from *The Guardian* split into twelve rough categories, which are the general sections on their website. The exact breakdown of numbers can be seen in figure 5.1. As it is shown, the numbers of articles per category are not even. Thus, the distribution is not homogeneous. Through these disparate amounts of articles, the results tend to prefer large classes and miss small ones. Therefore, only the five largest categories (`World-News`, `Sport`, `Football`, `Fashion`, `UK-News`) have been taken into account during the testing procedure to establish stable and balanced conditions.

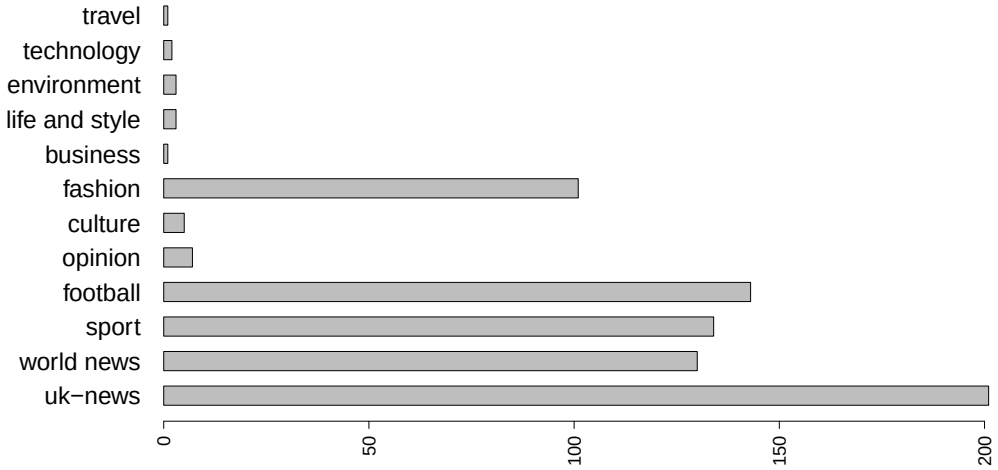


Figure 5.1: Number of articles per categories in initial data corpus.

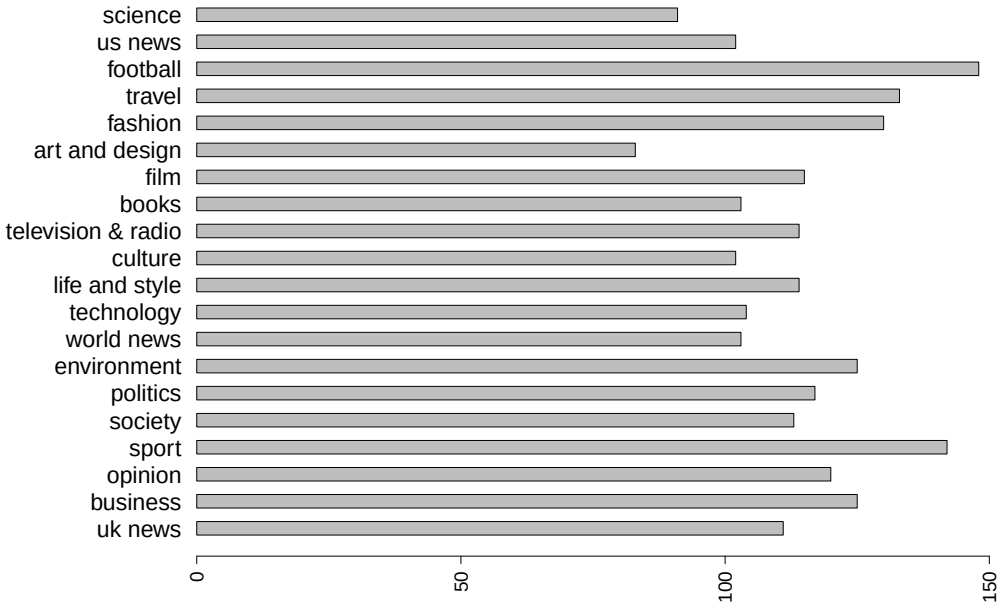


Figure 5.2: Number of articles per categories in final data corpus.

After some developing time with firm, but not persuasive results, the need for an even corpus with preferably precise classes was growing. Thus, the data set got increased and new articles and new classes added. Finally, the data corpus holds 20 categories with about 80 to 150 articles per class. With this setup the intense testing phase initiates. The total breakdown of the numbers is shown in figure 5.2. In this constitution, the two classes with the most articles are `Football` and `Sport` with 148 and 142 articles, whereas the two smallest classes are `Art/Design` and `Science` with 83 and 91 articles. Although the numbers of the smallest and largest classes are a bit apart, for the majority of the categories the difference is less significant. In the following sections the accuracy performance metric is often used to evaluate the algorithms. This number is calculated by dividing the correct classified articles by the total amount of test articles in the test set. In the figures, the values are shown in a range from zero to one, but can be interpreted as a percentage values, which range from zero to 100%.

5.1 k-nearest-neighbor

This algorithm has three adjustable parameters to change the performance and the resulting accuracy. Since the test data is so called unseen data, which implies that it is completely new and the algorithm does not know which kind of data comes next, other variables, like these parameters, have to be changed to improve the classifiers performance. The three main parameters in case of kNN is on the first place the k value, which decides how many documents are included in the valuation, on the second the similarity function or distance function, which detects the most similar or closest documents, and on the third place, the decision criterion for the assignment of the class to the document. Thus, the testing is focused on these variable parameters.

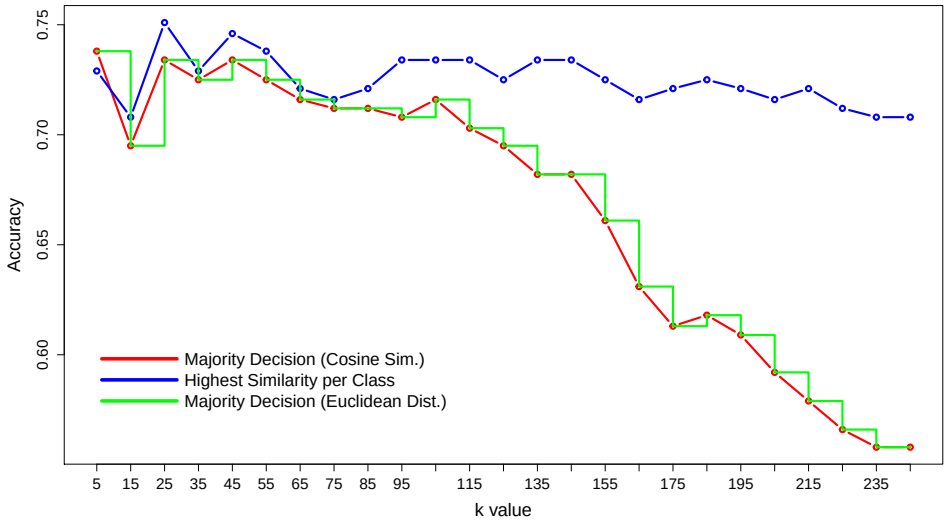


Figure 5.3: Testing k parameters of the kNN with the initial data corpus in three different settings.

5.1.1 Initial Findings

In the initial trails, the tested data amount accounts 233 articles on basis of the training corpus, which consists of 476 files and five different classes `World-News`, `Sport`, `Football`, `Fashion` and `UK-News`. The resulting accuracy value is 73.82% in case of the kNN classifier with a k value of five. This number is the maximum accuracy in this setup constellation, computed with the cosine similarity and the decision criterion of majority.

Testing different values of k up to 245 provides no better results, whereas switching the decisive factor to the sum of the similarities for each class, a new maximum accuracy of 75.10% is yielded by the k values of 25. Now changing the k value, results in continuously higher values than before. The definite numbers of the couple of test cases in the initial phase can be seen in figure 5.3. In the figure the red curve shows the development of accuracy using the majority decision with the cosine similarity, while the blue one presents the course when using the summed up similarity and the cosine similarity. Yet, when changing the similarity function to the euclidean distance and using the majority decision, the same values are provided as before with the cosine similarity and the majority decision. The euclidean distance curve is shown in the figure as green line and due to the exact same results, the green line has a different representation mode, to get both lines visible in the figure. As it is shown, the results of the summed up similarities per category remain quite stable when changing the k value, whereas the majority decision steadily decreases.

When adding the remaining smaller classes to the five large ones, the values decrease. With twelve classes, the maximum accuracy is about 72.2% with a k value of 15, the cosine similarity and the highest similarity as decisive factor. Using the majority decision the maximum result, drops off to 70.12% and switching the distance measurement delivers the same result. This decline might be caused through the uneven distribu-

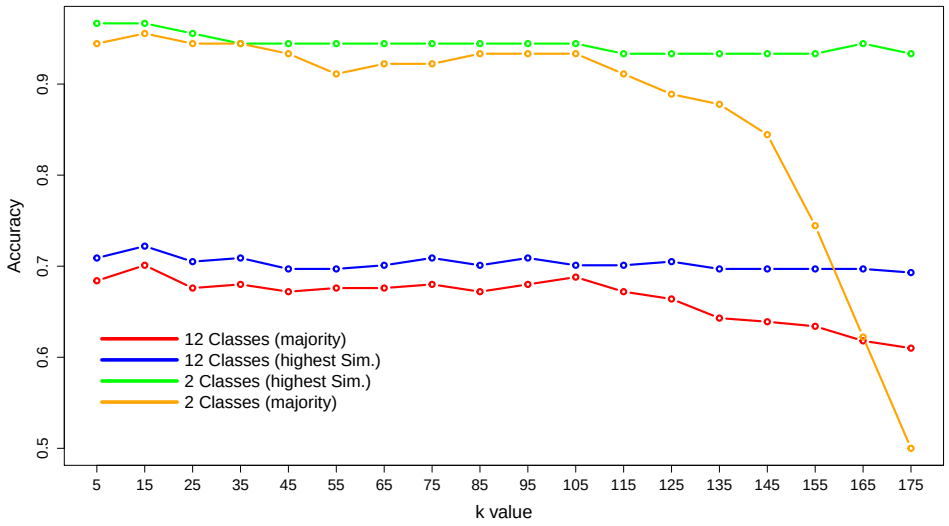


Figure 5.4: Testing extreme cases with variable k value and the two different decision criterion, majority decision and highest similarity.

tion and through the small data amount and the rough categories. This affirmation is supported by the test cases concerning two classes, `Football` and `World-News`, which nearly have the same amount of articles and are supposed to be dissimilar considering the topic. For the class `Football`, it is highly probable that it contains only articles about football, whereas `World-News` consists of multiple subjects, such as politics, technology or business and only a small part of it may contain content about sports or in particular about football. Thus, there are fewer intersections between them. In such constellation, the differentiation between the categories is easier, because of the clearer separation of the subjects. Testing the two classes with the cosine similarity, the highest similarity decision criterion and a k value of 15, yields an accuracy of 96.67%, which is quite strong. At this point, it should be mentioned that the general data amount is in this case a lot smaller than before with a test data set of 90 articles and training basis of 183 articles, which also has an impact on the percentage value.

The development of the performance of these two opposite cases, two and twelve classes, are visible in figure 5.4. The striking drop of the orange line and the starting decrease of the red line, which represent classifications in combination with the majority decision, show that high values for the k value are not meaningful to use in this setup, because of its small number of training data. If the k most similar documents are nearly all articles of the training data, the majority decision decides in most cases in favour of the largest category. The highest similarity decision does not have this difficulty, because it takes the actual similarity values into account. In general, it is shown again, that the results are higher when using the summarization of similarities as decision criterion. Thus, it is reasonable to use.

In all the cases mentioned so far, a ratio of 67 to 33 is used for splitting the data corpus into training and test data. Although, there are multiple possibilities for improvement at this stage of the kNN, this ratio is the first variable factor to have a closer look at.

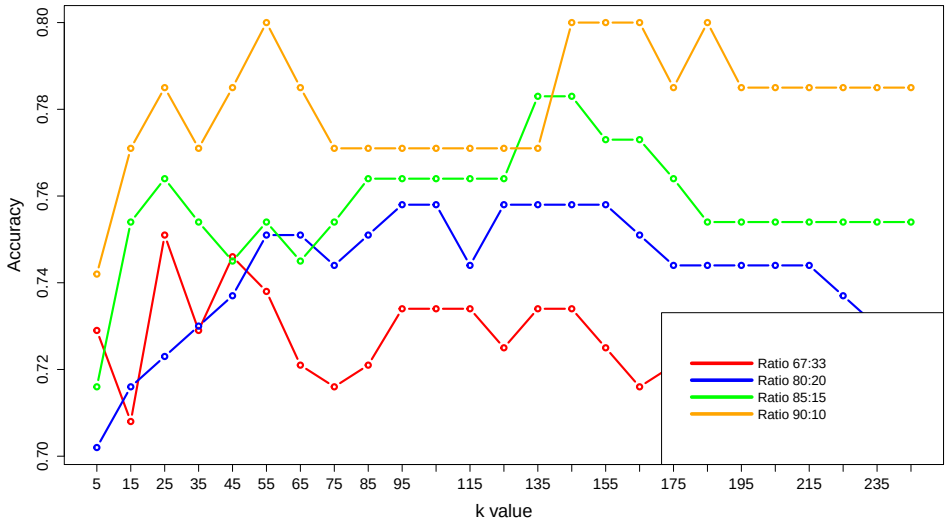


Figure 5.5: Testing different train–test data ratios with variable k and initial data corpus.

5.1.2 Improvements

After the first test results, some parameters have been fixed. For the distance measurement the cosine similarity is used together with the decision criterion of the highest similarity per class, whereas the k value can not be set, because it seems to have several dependencies. By this, an improvement phase initiates after these specifications.

Ratio of Training Data to Test Data

At the first stage a ratio of 67 to 33 is used, but after some researching and testing time, a larger training corpus becomes more and more recommended. Thus, the proportion switches to 80 to 20, 85 to 15 and 90 to 10. The test case with a ratio of 67 to 33, a test data amount of 233 and five classes results in an accuracy of 75.10% by a k value of 25, when using the highest similarity as decision criterion. When switching the ratio to 80 to 20, but keeping the other settings, the accuracy decreases to 72.34%. Though, when rising the k value step by step to 95, the result ascends again to 75.88%.

Going further to a ratio of 85 to 15, the result goes farther up to 78.30% by a k value of again 135 and 145. And finally by the ratio of 90 to 10 even further to 80% accuracy by a k value of 55, 145, 155, 165 and 185. An overview of the course of the results can be seen in figure 5.5.

As it is shown in the graph, the bigger the training data proportion the higher are the accuracy values. However, at this stage, it has to be pointed out that the test data amount is simultaneously shrinking if the training data is rising. This implies, that the accuracy is based on a smaller amount of test articles, which leads to a smaller denominator and this is distorting the percentage values. Thus, to get more balanced and realistic results, the data corpus has to be expanded and tested with a moderate ratio like 80 to 20.

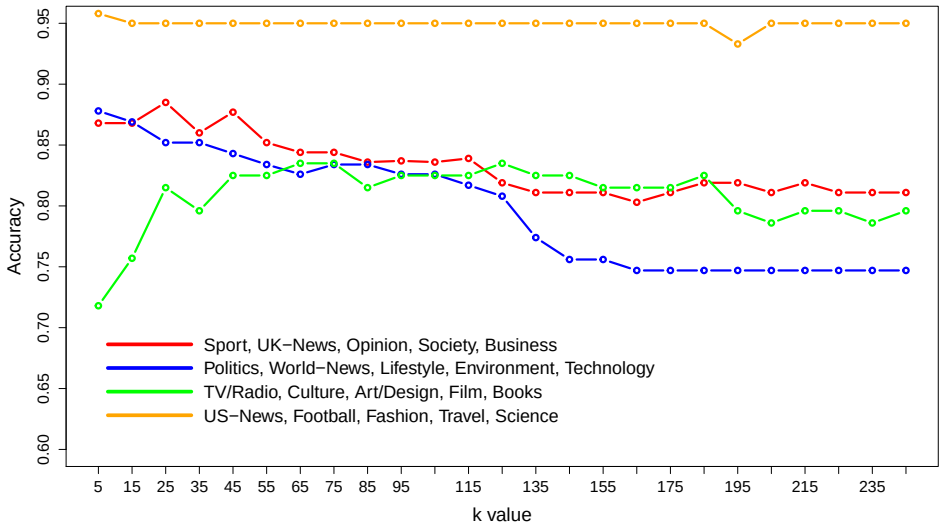


Figure 5.6: Testing the new data corpus in four different configurations and a variable k value.

Data Increase

After the data growth, the data corpus contained about 2315 news articles with 20 different classes. The main test cases at this stage are testing a data corpus containing 20 categories, ten categories and five categories. Some binary samples have been tested as well, so a data set with only two classes.

But before starting the comparison, the k value has to be analyzed with the new topics and constellations of data sets. This is done by testing four different compositions of data sets containing different topics. The first data sets consist of five classes, about 100 and 120 test articles and are based on 400 to 500 training samples. The variable k value and its impact on the results is shown in figure 5.6.

As it can be seen in this figure with the new data corpus, the perfect k value differs from data set to data set and concluding from its composition. However, some analogies can be witnessed. In this constellation of five classes, the front third of the area, ranging from 5 to 65, includes the highest results, apart from some exceptions beneath 15. This implies, a kNN with a small k value seems to be sharp and strict and this can cause positive or negative effects on the accuracy. Whereas, a kNN with higher values outside the first third seem to be softer but also more imprecise and spongier, because the accuracy values decrease more and more the higher the k values becomes.

A similar observation can be made in the data corpus, which consists of ten classes (see figure 5.7). In this construction of data set, the amount of training data is naturally larger than with five classes. This means the k value can go higher than before and this is also necessary. Again the first third of the field provides better results than beyond, but in these cases the preferring range passes from 45 up to 325.

However, to get comparable results and focus on the accuracies in the different data set constellations, the k parameter has been fixed to 15 in the subsequent test cases. As it is shown in table 5.1, testing two classes results in quite strong values above

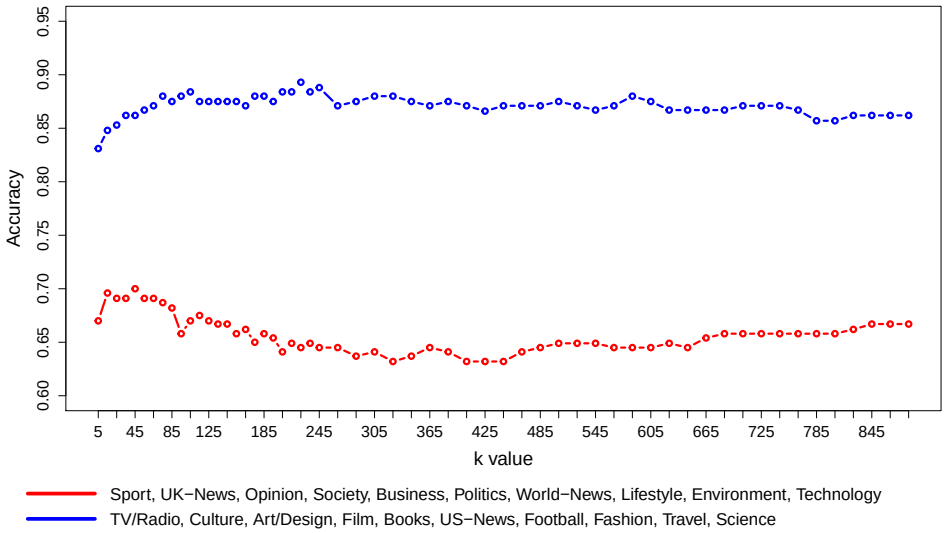


Figure 5.7: Testing the new data corpus with data sets of ten classes and a variable k value.

95% accuracy, whereas test data sets with more classes lead to a decrease. The data sets of two classes contain each about 50 test articles and the classification is based on approximately 200 training examples in each test run. When switching to a data corpus of five classes, the accuracy varies much more than before with two classes. It depends on the classes contained in the data set, but the values range from about 75% up to 95%. The exact compilation of classes are contained in table 5.1. Combining now two data sets with each five classes to unite them to a ten classes corpus, results in slightly worse accuracy values. These tests are performed on approximately 200 test articles and with the basis of about 900 training samples. The two constellations lead to 69.62% and 84.88% accuracy. Especially the second test run is surprisingly good, because it contains the five classes which performed worst in the previous test run and the classes which performed best. The second half of the ten classes seems to compensate the deficiencies of the first half. Finally, all 20 classes united in one large data corpus provides an accuracy of 68.68% by a test data amount of 463 and a training corpus of 1852 articles.

To summarize, the more classes contained in the data corpus, the lower are the accuracy values. But yet, with the optimal k value for a particular data set, the numbers can be boosted. Nevertheless, the overall accuracy for 20 categories is still low and to discover the reason why, the next step's focuses on the data itself, the words contained in the articles and the constellation of the articles.

Noise Reduction

Dealing with a real-life data corpus as foundation for the classification task, involves articles and words or phrases in articles, which contain no useful information for the primary goal of determining the topic in the unseen article. This data is called noise and affects the classification accuracy adversely. For this purpose the noise should be

Table 5.1: Testing various data corpus constellation containing different topics with a k value of 15.

Categories in Data Corpus	**Accuracy**
World-News, Football	96.00
Fashion, Technology	97.91
Sport, UK-News, Opinion, Society, Business	86.88
Politics, World-News, Lifestyle, Environment, Technology	86.95
TV/Radio, Culture, Art/Design, Film, Books	75.72
US-News, Football, Fashion, Travel, Science	95.04
Sport, UK-News, Opinion, Society, Business, Politics, World-News, Lifestyle, Environment, Technology	69.62
TV/Radio, Culture, Art/Design, Film, Books, US-News, Football, Fashion, Travel, Science	84.88
Sport, UK-News, Opinion, Society, Business, Politics, World-News, Lifestyle, Environment, Technology, TV/Radio, Culture, Art/Design, Film, Books, US-News, Football, Fashion, Travel, Science	68.68

reduced as much as possible.

Concerning the thesis project, the noise reduction took place in several steps. After the first test results with classification accuracies, which were still not as promising as hoped for, it is necessary to look over the actual articles and it soon becomes apparent that an additional stop word list is needed. The general stop word reduction in the pre-processing part does not include short forms of words like “we’ll” or “couldn’t”. Therefore another stop word list is created for eliminating these words.

Throughout the inspection of the articles, some phrases are especially noticeable because of their multiple repetition in different articles belonging to various topics. These sentences pertain to an advertisement for a chat forum, where user can register with their email and post their comments in the discussion, thus they do not possess topic related information and should be removed. These noises lead to a misinterpretation and impureness of the topic related similarity checks, if these lines occur in multiple classes multiple times.

Finally, another conspicuousness is detected by analyzing the articles data. In some articles, entity names for HTML escape characters have been found, such as “&”, “ ” or “>”, which occur various times for representing special characters and goes along during the automatic read-out. These values also have to be deleted from the articles.

At the end, the length of the articles is reviewed. During this examination, some very short articles with only one or two sentences have been observed. In general, these articles are hard to get classified because of the small amount of words, which may have useful information for the classification. Hence, these extremely short articles get erased from the data corpus to not interfere the classification.

After these corrections, the data corpus consists of 2295 articles and has been tested

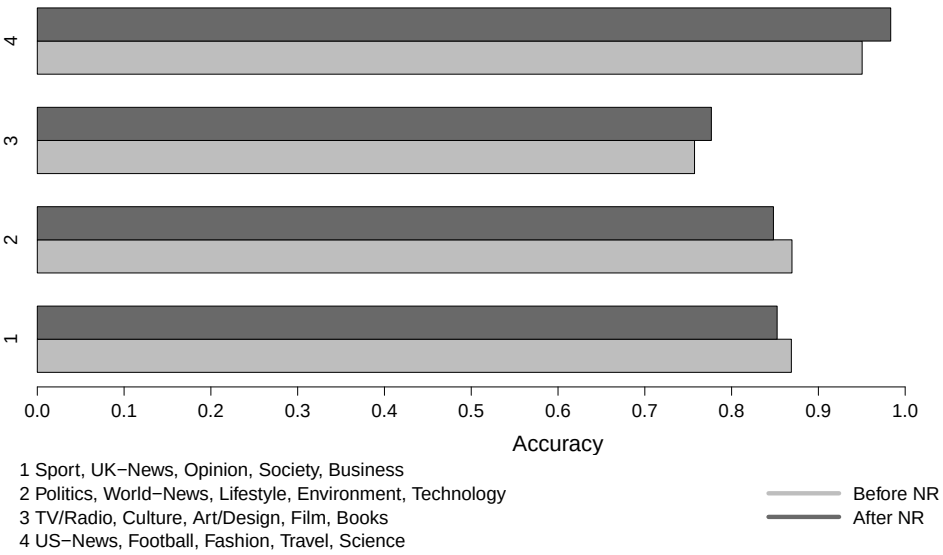


Figure 5.8: Comparing data sets with five classes before and after noise reduction (NR) with kNN.

again with the same constellations of topics as before. Starting with five classes in four different compositions. As it is shown in the respective figure 5.8, the noise reduction affects the first two test cases adversely. The accuracy values of them decline from 86.88% and 86.95% to 85.24% and 84.82%. Those effects might be caused through the removal of some short documents in the data corpus, which have been classified correctly before the improvement step. On the contrary, the last two data sets get a slight boost through the adjustments. The numbers go up from 75.72% and 95.04% to 77.67% and 98.33% accuracy.

Combining two of these data sets and creating one corpus with ten classes, produces similar performances. It is certainly interesting that joining the first two test data sets, with the slight fall of accuracy, results in a minor increase of the overall accuracy. It rises from 69.62% to 72.34%, whereas the second combination of data sets drops from 84.88% to 83.03%, even though classifying the single groups of five both leads to an improvement. Finally, summarizing these two data sets as well, yields an accuracy of 69.93%, a small increase.

A possible reason for this behavior, might be simply the similarity between the documents and of course, the selection of the documents. If more articles are available, different articles might become the most similar ones and thus change the results. Additionally, the more articles are contained, more articles can have similar similarity values, which results in a higher density of nearer documents. This can be visualized in multidimensional scaling graphs and will be shown in the next section.

Another aspect is that these values are measured at a k value of 15. It is a snapshot of the classification accuracy and not the highest value which can be reached. The rough development of the performance when changing k is shown in figure 5.10. One can well see that the higher accuracies are again located in the front section of the figure for both data set constellations. The first one which has 72.34% by a k of 15, has the maximum

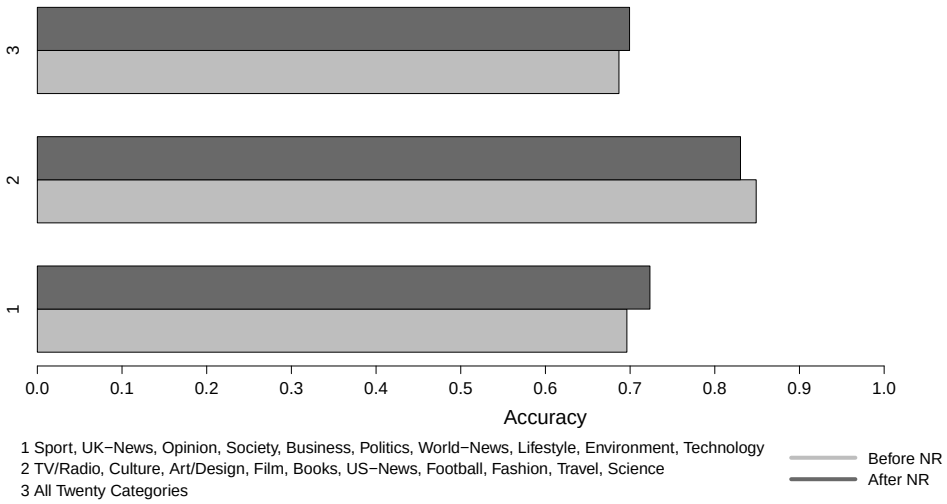


Figure 5.9: Comparing data sets with ten and 20 classes before and after noise reduction (NR).

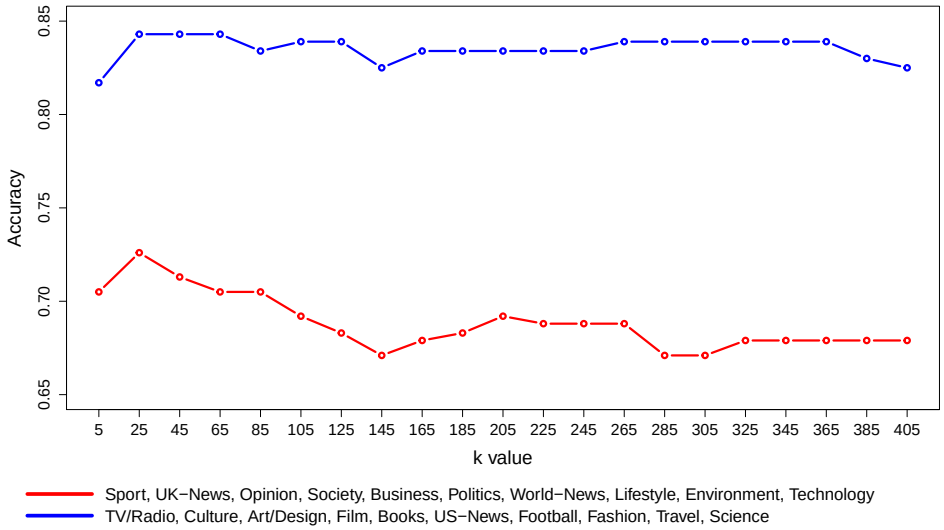
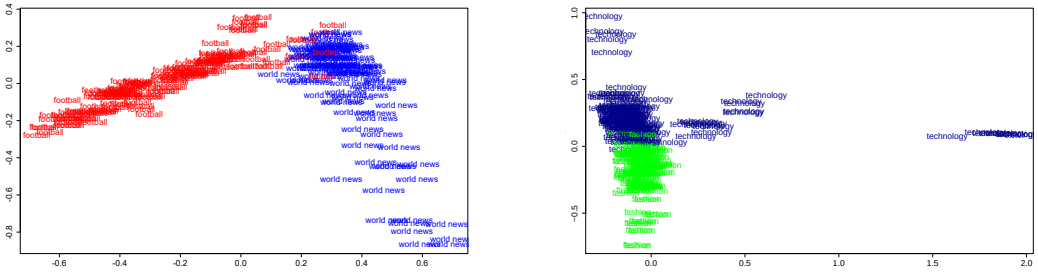


Figure 5.10: Development of performance of data sets with ten classes after the noise reduction with kNN.

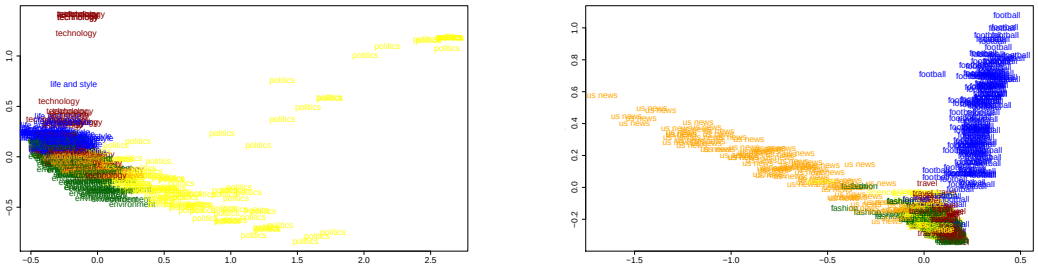
at 25 with 72.6% and the second one also by a k of 25, with 84.3%. This implies by adjusting the k value the classification performance of the kNN can still be improved.

To conclude, with the decision in favour of the cosine similarity in combination with the decision criterion of highest similarity per class, can achieve solid and strong accuracy values, always depending on the data and the data related k value.



(a) (b)

Figure 5.11: Multidimensional Scaling graph with two data sets: (a) `World-News` and `Football` and (b) `Fashion` and `Technology`.



(a) (b)

Figure 5.12: Multidimensional Scaling graph with two data sets: (a) `Politics`, `World-News`, `Lifestyle`, `Environment`, `Technology` and (b) `US-News`, `Football`, `Fashion`, `Travel`, `Science`.

5.1.3 Analysing Similarity

Throughout the test phase, it occurs occasionally that the accuracy numbers are surprising and different than expected. To determine the reason for this behavior, it is often necessary to have a look on the data the algorithm works with. In case of the kNN, the decisive factor is the similarity value. Thus, during the analyzing procedure, the primary concern is on this factor. For this purpose, a multidimensional scaling graph is reasonable to use. To create the graph, all documents in the data corpus get compared with each other and the respective similarity values are stored in an array. By means of *RStudio*, this matrix can be visualized and enables comprehending the significance for the algorithm.

As it is shown in figure 5.11, the separation between the categories `World-News` and `Football` works well. The same conclusions can be seen with the topics `Fashion` and `Technology`, and a number of other binary data sets. The clear division is as well reflected in the accuracy values.

Visualizing some more classes the separation becomes blurry and a dense intersection happens between many documents. Already in case of data sets with five classes, it is visible why the accuracy value decreases (see figure 5.12). The documents are moving

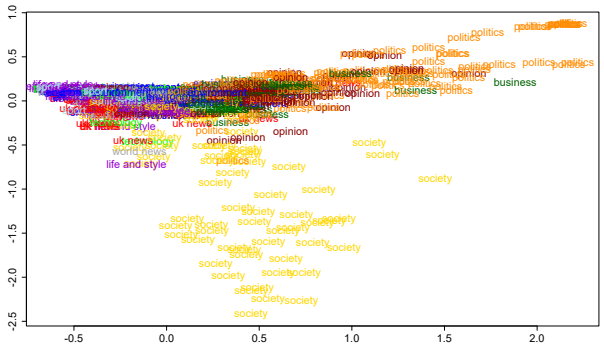


Figure 5.13: Multidimensional Scaling graph with `Sport`, `UK-News`, `Opinion`, `Society`, `Business`, `Politics`, `World-News`, `Lifestyle`, `Environment`, `Technology`.

close together and through these circumstances it becomes difficult to detect the correct class on basis of similarity. The main areas are crowded, which means lots of documents are similar to each other, despite the class, which adversely affects the compositions of the k nearest neighbors and thus the assignment of the new document. When analyzing a data set with ten classes, it gets even worse clustered, as it is shown in figure 5.13. These arrangements and the positioning of documents are finally visible in the accuracy values and lead to a decrease of performance.

5.2 Naive Bayes

The main idea of the Naive Bayes is to use conditional probability to classify a new document. For this it focuses on the words in the new document as well as in the training documents and computes the various likelihoods. Adjustable parameters are rare in case of Naive Bayes, but two different approaches of calculating the probabilities are possible. These two concepts differ in the aspect of taking the exact word number into account or not and are tested at the end of this section. Before that, the initial as well as the final findings are analyzed and the development of the performance during the improvement phase.

5.2.1 Initial Findings

In the initial test case, the amount of data which gets classified is 233 articles, containing the general five classes: `UK-News`, `World-News`, `Sport`, `Football` and `Fashion`. The accuracy value is slightly higher than in the first test case of the kNN with 75.10% accuracy. Adding the remaining smaller classes to the data corpus of five, the value decreases to 69.71% for the Naive Bayes; a similar behavior to the above analyzed kNN.

Furthermore, testing `World-News` and `Football` as a two classes data set, results in strong 97.77% accuracy and also four classes, `Football`, `World-News`, `Sport` and `Fashion`, have a significant value of 83.23%. An interesting fact can be shown here, when adding the last of the five main classes, `UK-News`, to the data set of four, the values sharply decrease about ten percent to 75.10% accuracy as already mentioned. A

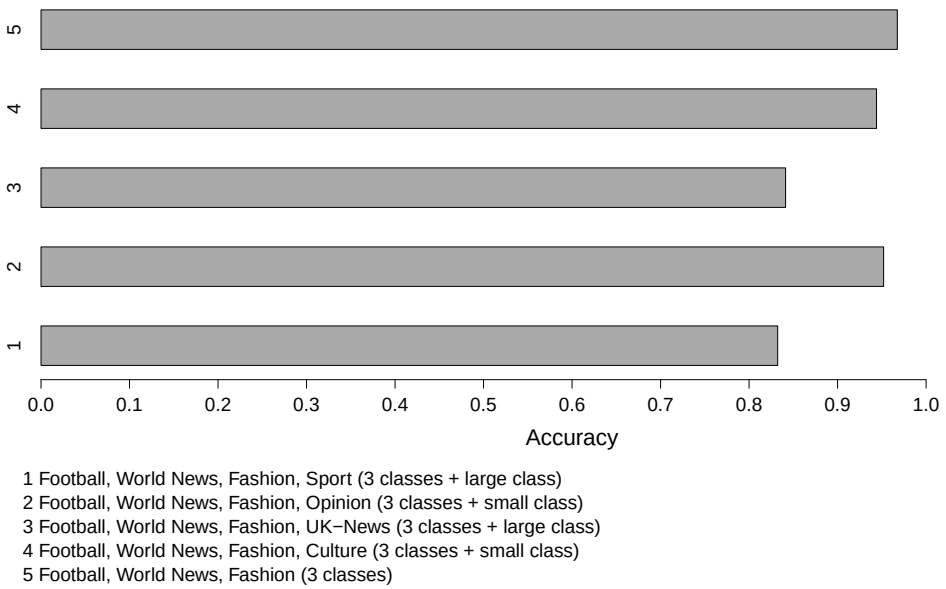


Figure 5.14: Testing different compositions of data sets with Naive Bayes.

possible reason for this development may be the similarity of `World-News` and `UK-News`, which affects the separation between texts belonging to these classes.

Switching to another combination of four classes the value can get even higher. With `Football`, `World-News`, `Fashion` and `Opinion` the accuracy rises to 95.20% and adding again `UK-News`, results in an drop of over ten percent to 82.29% accuracy. Another reason for this performance, might be the different sizes of the classes. `Opinion` is one of the smallest classes, while `UK-News` is the class with the most articles. In the constellation of four classes including `Opinion`, it is not decisive because it has few texts. This statement is supported by the test case, which shows that testing only three classes, `Fashion`, `World-News` and `Football`, results in nearly the same accuracy value, 96.74%. This implies, that adding `Opinion` does not have major consequences. The same outcome can be witnessed when `Culture` is included in the data set, in place of `Opinion`. Then the accuracy value is also quite strong with 94.40%. Adding `UK-News` instead, shows a decrease to 84.12% accuracy. Thus adding a large class, provides more test and training samples and this influences the amount of data which get classified correctly. The breakdown and the development of the results of the test cases with four classes, can be seen in figure 5.14.

5.2.2 Improvements

The next step, after the initial testing phase, is the improvement set up as mentioned in the chapter before. In case of the Naive Bayes, there are few possible parameters which can be adjusted to change the resulting outcome. Thus, the only thing which can be adapted is the data and the amount of training data and test data. The last concern is the first one, which gets analyzed during the improvement stage.

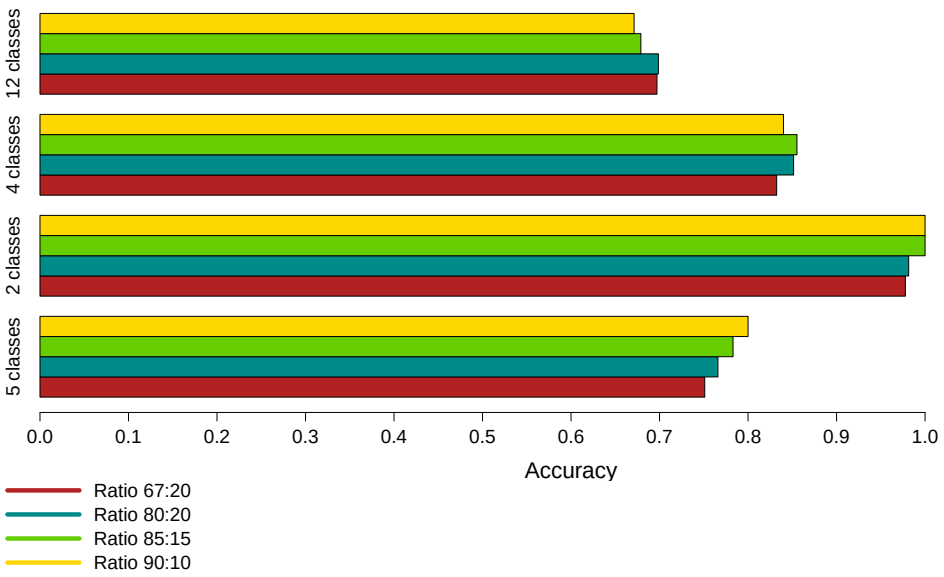


Figure 5.15: Testing different ratios of splitting the data sets in training data and test data with Naive Bayes.

Ratio of Training Data to Test Data

The initial test cases have been generated by a ratio of 67 to 33 for training data and test data. As follows, this relation gets shifted to the next step of 80 to 20. With a data corpus of the major classes, `UK-News`, `World-News`, `Sport`, `Football` and `Fashion`, the accuracy values gets a small boost to 76.59% from 75.10%. However, when the ratio changes further to 85 to 15 and 90 to 10, the boost grows to 78.30% and even 80%. The same behavior can be witnessed with a binary test case, including `Football` and `World-News`. Their accuracy rises from 97.77% to 98.14% and significant 100% at a ratio of 85 to 15 and 90 to 10. Here must be noted, that the test data amount with 90 to 10 and 85 to 15 comprises only about 40 and 27 articles. Thus implies, that the computation of the accuracy is based on a smaller denominator and this distorts the percentage value. At a ratio of 67 to 33, the test data amount is about 90 articles large.

Switching to a data corpus of twelve classes, the accuracy value rises from 67.71% to 69.86%, but then decreases with a data ratio of 85 to 15, to 67.88% accuracy. Shifting the ratio further to 90 to 10, the figure drops off to 67.12%. Another test case with four classes, `World-News`, `Sport`, `Football` and `Fashion`, yields a similar performance development. The accuracy value grows up to 85.52% at a ratio of 85 to 15, and then declines a little to 84%. The overview of the developments with the different training and test data ratios can be seen in figure 5.15.

As already observed during the testing phase of the kNN, a training and test data ratio of 80 to 20 produces the most promising results in the majority of the different data corpus constellations. Thus, the decision is made again in favour of this ratio for the following test cases.

Table 5.2: Testing various data corpus constellation containing different topics with Naive Bayes.

Categories in Data Corpus	**Accuracy**
World-News, Football	96.00
Fashion, Technology	100.00
Sport, UK-News, Opinion, Society, Business	84.42
Politics, World-News, Lifestyle, Environment, Technology	87.82
TV/Radio, Culture, Art/Design, Film, Books	79.61
US-News, Football, Fashion, Travel, Science	95.04
Sport, UK-News, Opinion, Society, Business, Politics, World-News, Lifestyle, Environment, Technology	72.57
TV/Radio, Culture, Art/Design, Film, Books, US-News, Football, Fashion, Travel, Science	87.11
Sport, UK-News, Opinion, Society, Business, Politics, World-News, Lifestyle, Environment, Technology, TV/Radio, Culture, Art/Design, Film, Books, US-News, Football, Fashion, Travel, Science	75.59

Data Increase

After the determination of the training and test data ratio, the data corpus gets increased with new articles and new categories. Finally, it contains 20 different classes with a summarized amount of 2315 articles. The first tests consists of data sets with only two classes, which works quite well with similar results as before with the kNN algorithm. Some combinations of topics are working better, and some worse. For instance, `Fashion` and `Technology` yield an accuracy of 100%, but when classifying `Fashion` and `Lifestyle`, the accuracy drops off to 91.83%, whereas kNN classifies all test articles correctly in this specific case.

Testing a data corpus with five classes, generates mostly higher accuracy results compared to the kNN. One exception is the first combination of classes with `Sport`, `UK-News`, `Opinion`, `Society` and `Business`, with an value of 84.42%. The remaining combinations result in slightly higher values and one significant outstanding result with the five difficult classes, including `TV/Radio`, `Culture`, `Art/Design`, `Film` and `Books`. Because of their similarity, the classification and the separation are complicated, and when using kNN, this test case is the one which yields the lowest accuracy compared to the other test cases with five classes. Using Naive Bayes this value gets improved to nearly 80% accuracy, a growth of about four percent.

Finally, combining each two of the data sets with five classes, creates two data sets with ten classes and this setup yields in both cases three percent higher accuracy values than using kNN. With 72.57% for the first and 87.11% for the second data corpus, the results up to now, can be exceeded. And classifying a data corpus with all classes together, increases the accuracy value further to 75.59% with a boost of seven percent compared to kNN.

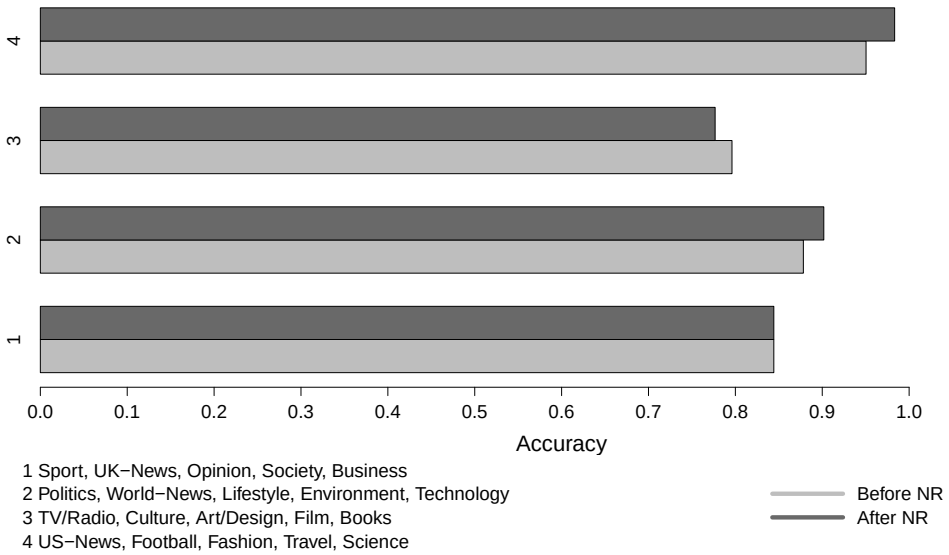


Figure 5.16: Comparing data sets with five classes before and after noise reduction with Naive Bayes.

The overall values are significant and their breakdown is shown in table 5.2. After the data increase, the second noise reduction initiates, including an additional stop word reduction and the removal of distracting phrases.

Noise Reduction

In case of real life data, noise is one of the main factors, which influences the classification accuracy. After removing this data, in some data set constellations a recession in the accuracy value can be witnessed. Concerning data compositions with five classes the decline is less striking, only the third combination of classes, `TV/Radio`, `Culture`, `Art/Design`, `Film` and `Books`, results in a slight drop. The remaining constellations performed better or equally. Compared to the kNN results, this development of the values differ. The only agreement is the last combination of five categories, the other combinations performed even in opposing directions. See figure 5.16 and figure 5.8 for a comparison.

When testing the data sets with ten classes, similar results to those of kNN are achieved. For the first combination of classes the noise reduction increases the accuracy value for about three percent, whereas the second data set gets a small decline of one percent. Focusing on all 20 categories the noise reduction has a slight adversely affect on the accuracy value compared to the results of the kNN, where the value rises. The actual development of the numbers can be seen in figure 5.17.

One surprising fact should be highlighted at this point. The noise reduction concerning Naive Bayes, has a different impact on the outcome, compared to the kNN. One reason for the occasional decrease of accuracy, might be caused by the removal of documents, which distracted the accuracy value beforehand. The Naive Bayes is depending on the words in the articles in the diverse categories, and if some of the texts

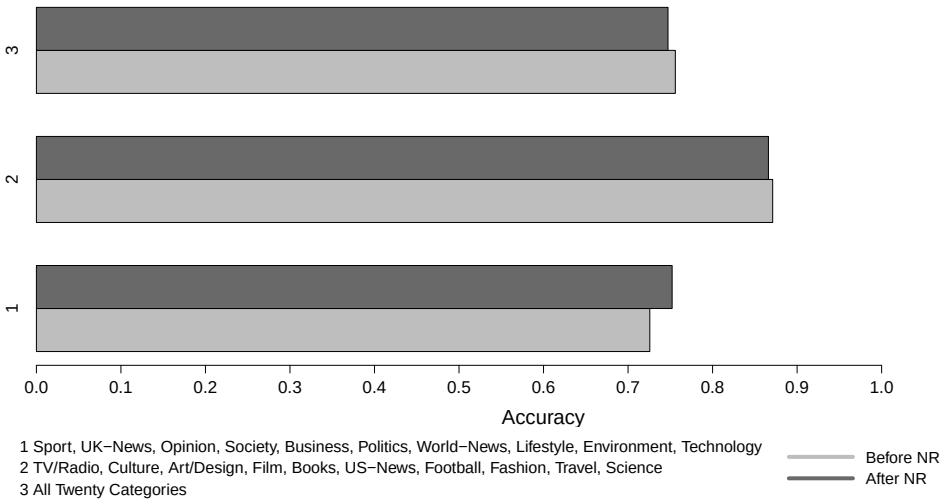


Figure 5.17: Comparing data sets with ten and 20 classes before and after noise reduction with Naive Bayes.

get erased, the overall probability values for each document change. For the kNN, the removal of documents is less impressive, because it focuses on the similarity between the new article and each single article in the training set, despite the whole class. Thus, the values for the single documents do not change, but for the Naive Bayes these values get modified.

In conclusion, the accuracy values for the Naive Bayes are satisfactory and show a similar behavior like the algorithm tested before. The more classes contained in the data set, the more difficult it gets for the algorithm to separate. Overall, this algorithm performs better with multiple classes in the data corpus than the first algorithm. With nearly 75% accuracy for all 20 categories, it beats the result of the kNN.

5.2.3 Comparison of Two Approaches

Another key point to remember concerns the two different approaches of the Naive Bayes. The one used for the test cases above, takes the actual word frequency in the classes into account, whereas the second approach only checks in how many document the term occurs, despite how often. During the test phase, both variations of the Naive Bayes have been tested and the results can be seen in figure 5.18. It shows that the first approach reaches overall higher or equal results compared to the second one, with different sizes and compositions of data sets. In case of two classes the first approach results in an strong increase, with `Fashion` and `Lifestyle`, or equal values, with `Fashion` and `Technology`, and `World-News` and `Football`. These classes are represented in the figure by the number one to three. The following four numbers belong to the usual data sets with five classes, where as well a slight growth is visible, especially at the last spot, number seven. The numbers eight and nine, are related to the data sets with ten classes and behave the same way with the first approach. Similarly, the last test case with twenty classes yields higher results with the first concept. Thus, the decision was made in favour of the first variation of the Naive Bayes and against the second.

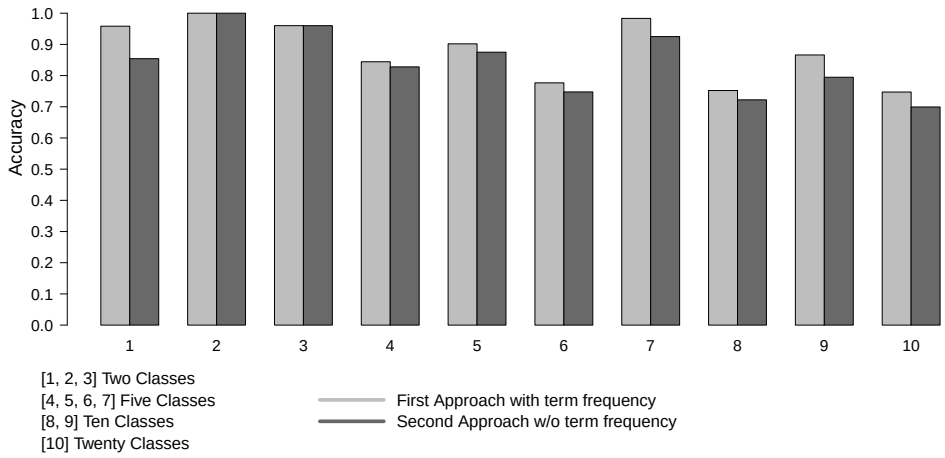


Figure 5.18: Comparing different data sets and their results of two variations of Naive Bayes.

5.3 Semantic Fingerprinting

This classification method has multiple parameters, which can be adjusted. The first parameter is comparable with the k value of kNN. This number called threshold value determines, which stacks will be kept in the final fingerprint of the category fingerprint or the fingerprint of the test document. Thus, it defines how much data to include in the decision process. The next adjustable factor is the filling of the context map, hence the arrangement of the training documents in the array for the context map. This map defines the pattern of the subsequent fingerprints and is therewith, decisive for the classification process. Another point to consider, is the creation of the category fingerprints. These fingerprints can be composed from all words contained in a class or by using an example article, which represents the entire class. The second approach depends heavily on this single document and for this project with articles from a news agency, it can not be decided in favour of one of the articles for each class, because the articles usually cover only a small aspect of the whole field. The method has been evaluated at the very beginning with unfortunately low accuracy values. Therefore, the first approach has been chosen for this procedure. The last variable parameter is the decision criterion, similar to the kNN. For comparing fingerprints, one possibility is to focus on the similarity of the fingerprints by using for instance the cosine similarity, which is then called the semantic closeness or binary overlap. Another possible method is to determine the positions of ones in the fingerprint and predicting the class from the majority class in the underlying context map. Thus, this approach uses the shared context between the fingerprints for deciding. These are all parameters, which are adjustable to finally enhance the classification accuracy.

5.3.1 Initial Findings

The evaluation phase focuses on data sets with five different classes. The first parameter, which gets adjusted is the threshold value. When using the semantic closeness as decision

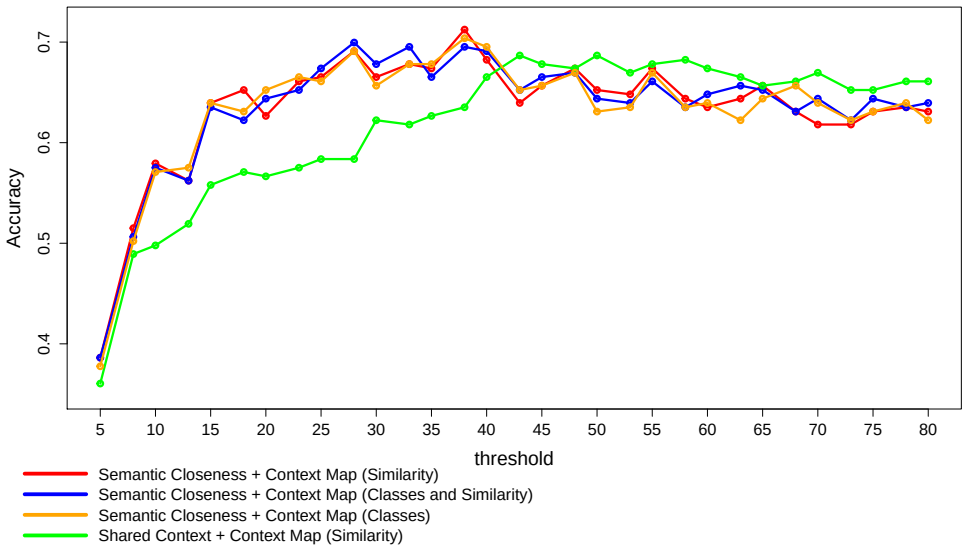


Figure 5.19: Comparing different test settings and their results of the Semantic Fingerprinting Method.

criterion and a context map, which uses the overall similarity between each document for its creation, a maximum accuracy value of 71.24% can be yielded at a threshold of 38. This value is the highest level of the development curve, shown in figure 5.19 represented with the red line. It is visible in figure that small threshold values as well as higher ones reduce the accuracy, similar to the behavior of the k value in the kNN.

Between thresholds of 15 and 40 the accuracy values are rising in case of this test setting. But also when switching the creation mode of the context map, this area contains the highest performance values. The orange line shows the development when using semantic closeness as decisive factor again, but for the creation of the context map, the actual classes of the labelled training data have been used to determine the order of the map. The respective curve is developing in a similar way to the red one, which was mentioned before. It also has its climax at a threshold of 38 with a value of 70.38%.

The blue one differs slightly from these two. Its peak is located at a threshold of 28 with a value of 69.95% and the context map is in this case a combination of the similarity factor and the ordering per classes. However, the overall course is nearly the same. Low accuracy at the very beginning, rising up to a threshold of 40 and then constantly dropping.

Changing now the decision criterion to the shared context factor, which is looking at the ones in the fingerprint and counting up for each class, results in a different curve. This test setting is represented with the green line in the graph. Its maximum accuracy is 68.67% at a threshold of 43, thus the curve is ascending up to this point and than slowly decreasing again.

To conclude, the best results can be achieved with the combination of semantic closeness and a context map ordered by similarity, and this set up has been kept for the subsequent test cases. These accuracy values are similar to the two classical algorithms, the k-nearest-neighbor and the Naive Bayes algorithm, analyzed in the previous sections.

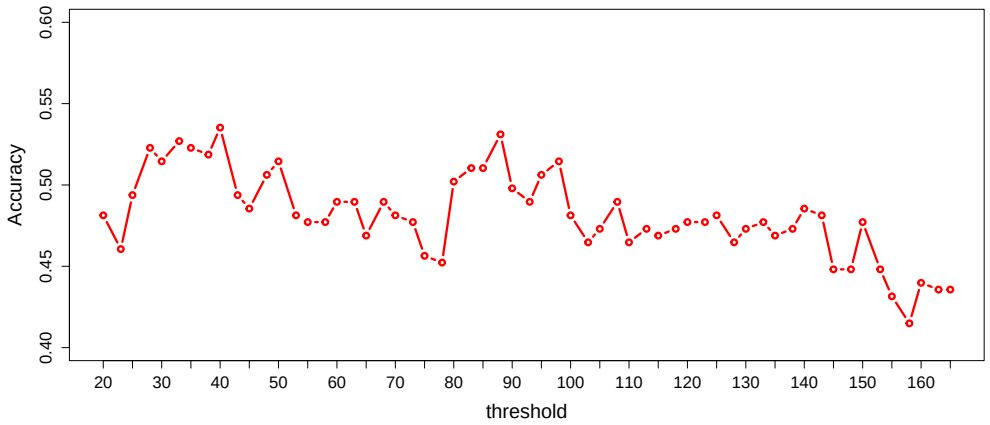


Figure 5.20: Results of the Semantic Fingerprinting Method with twelve classes.

The maximum accuracy of this data set with five classes is only four percent lower than the result of these methods.

Unfortunately, when adding the remaining smaller classes to the data set, the highest accuracy value decreases to 53.52% at the threshold 40. As shown in figure 5.20, the area between 28 and 40 contains the best accuracy values for this data set, similar to the curves analyzed before. Another summit seems to be located between 80 and 90. However, the maximum value is worse than the results from the classical algorithms. Already at this stage, it seems that this method is more vulnerable, either for the uneven distribution of articles per class or for the higher number of categories contained in the data corpus.

Switching to a data set with only two classes, increases the accuracy value again to a maximum number of 94.44% at a threshold of 40 and 43. Overall, this is a good value, but also slightly lower compared to the remaining algorithms at this stage, with 97.77% and 96.67% accuracy.

It becomes apparent that the semantic fingerprinting method tries to keep up with the other algorithms, but barely misses equal results. The more classes contained in the data set, the more difficult gets the classification for this method. The accuracy values become lower and the gap between this method and the other algorithms gets larger. The following challenge focuses on the analysis of this behavior as well on the improvement of the performance of this classification algorithm. To facilitate the subsequent comparison of different data set settings, the threshold value is fixed to the distinct number, 35, contained in the area, which seems to yields the highest accuracy values.

5.3.2 Improvements

After the initial test cases, some settings are fixed. The context map is created ordered by similarity, the decision criterion is set to semantic closeness and the threshold to 35. The remaining adjustable factors are the amount of data for training and testing, the overall data amount in the corpus and the noise filtering, thus the enhancement of the feature selection. These key factors have been tested during the subsequent improvement phase.

Ratio of Training Data to Test Data

The first variable factor, which gets tested is the ratio of training and test data. The initial testing phase uses a ratio of 67 to 33 and when shifting this division to 80 to 20 and starting with a data corpus of two classes, the accuracy value increases from 93.33% to 96.29% at a threshold of 35. Going further to a ratio of 85 to 15, the value rises even more to 97.5% accuracy. Unfortunately, at a ratio of 90 to 10 the number drops sharply back to 92.59%. The actual development of this data set constellation can be seen at the bottom of figure 5.21.

Testing a data corpus with the five major classes, shows a similar behavior of the results to them of the Naive Bayes (see figure 5.15 for comparison). The higher the training data amount, the better become the accuracy values. The numbers are rising from the initial 67.81% to 70.75% at a ratio of 85 to 15 and finally to 71.42% accuracy with 90 to 10. An interesting development can be witnessed, when adding the remaining classes to the data set at this stage. The expansion of the data set with only 22 articles, and seven new and extremely small classes, causes a accuracy drop of 20%. With a ratio of 90 to 10, a data set of finally twelve classes and a threshold of 35, the accuracy value is about 49.31%. This shows, compared to the 71.42% of the five classes test case at this ratio, that the data corpus definitely should be balanced. The highest value, 52.74%, with the twelve classes can be reached at a ratio of 80 to 20. The performance breakdown is shown in figure 5.21.

In conclusion, the results are again lower than the numbers of the remaining algorithms. The most drastic decline occurs with twelve classes. The gap between Naive Bayes and this method is between 10% and 15%. However, the values at this point are measured at a distinct threshold number, thus they are not the maximum accuracy values, but a snapshot of the performance. A detailed analysis of the impact of the threshold will be covered at the end of this chapter with the final balanced data set of articles.

After this session of testing, the ratio of training data and test data has been set to a moderate value of 80 to 20. The subsequent trail runs use this ratio. The next step after determining an appropriate ratio is to increase the amount of data and especially to balance the distribution of articles per class.

Data Increase

To make the distribution of the amount of articles per class even, new articles from overall 20 classes have been collected and stored. After that, each category includes about 80 to 150 articles, thus the data corpus is quite balanced. Subsequently, the testing phase starts again and the values seems now definitely promising (see table 5.3 for the exact results). The first test cases concern a data corpus with only two classes. With the categories `World-News` and `Football` an accuracy value of 94% is yielded, with `Fashion` and `Technology` the number drops slightly to 89.58%, but for `Film` and `Politics` the algorithm classifies all test documents correctly. For this section, a threshold of 35 is used once more for all test runs.

For the next step, five classes get combined to one data set in four different variations. The first constellation results in 74.59% accuracy, the second in 66.95%, the third one even lower with 59.22% and the fourth one exceeds them all with 90.08% accuracy.

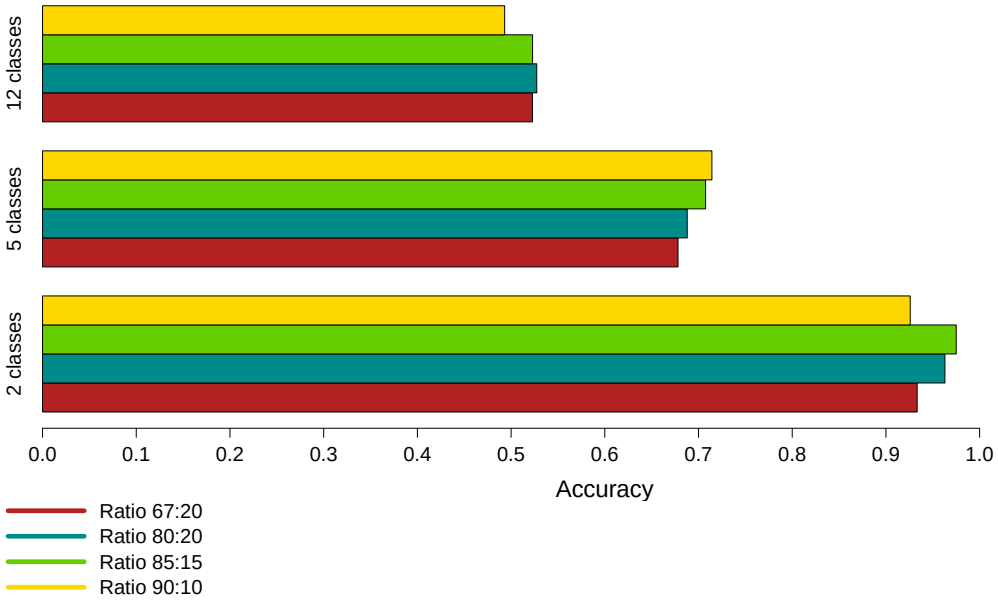


Figure 5.21: Testing different ratios of training and test data amount with Semantic Fingerprinting.

One significant case focuses on a combination of four classes including `Film`, `Business`, `Football`, `Travel` and `Science`. This data set constellation reaches an accuracy value of 93.49%, which is significantly higher compared to the other values of five classes. On the contrary, when using different four classes, `World-News`, `Football`, `Fashion` and `Technology`, the accuracy drops to 81.63%. Even at this early stage, the reliance on the class composition is highly apparent for this method.

Combining two of the data sets with five categories to two data sets with ten classes, diminishes the accuracy drastically. An accuracy of 47.67% can be yielded for the first composition, which is surprisingly low. The second combination can outpace this with a 17% higher value.

However, classifying all articles together, the accuracy is only 2% lower compared to the first set. The accuracy for the 20 classes is thus 45.57%. But in summary, the values are lower than desired. The evaluations, carried out on the described data set, suggest that this method is much more depended on multiple factors, such as the composition of the data corpus, the similarity between the classes or the height of the threshold.

Noise Reduction

Up to this point, no remarkable enhancement of the accuracy values can be reported. However, after an additional noise reduction the results of the method improved significantly compared to the classical algorithms. The noise reduction itself focuses on stop words, distorting short articles and distracting phrases inside of documents as already explained in the kNN section. After this filtering procedure, the previously performed test cases are repeated and documented again.

The results from the data sets with two classes are raised from 94% to 96% accuracy

Table 5.3: Testing various data corpus compositions with Semantic Fingerprinting and a threshold of 35.

Categories in Data Corpus	**Accuracy**
World-News, Football	94.00
Fashion, Technology	89.58
Sport, UK-News, Opinion, Society, Business	74.59
Politics, World-News, Lifestyle, Environment, Technology	66.95
TV/Radio, Culture, Art/Design, Film, Books	59.22
US-News, Football, Fashion, Travel, Science	90.08
Sport, UK-News, Opinion, Society, Business, Politics, World-News, Lifestyle, Environment, Technology	47.67
TV/Radio, Culture, Art/Design, Film, Books, US-News, Football, Fashion, Travel, Science	64.44
Sport, UK-News, Opinion, Society, Business, Politics, World-News, Lifestyle, Environment, Technology, TV/Radio, Culture, Art/Design, Film, Books, US-News, Football, Fashion, Travel, Science	45.57

in case of `World-News` and `Football` as classes, and from 89.58% to 95.65% accuracy for the fields `Fashion` and `Technology`. This shows, that the second data corpus contains a lot more noise, what adversely affects the classification result.

In case of the data sets with five classes, the growth is not as representative. The results of the first data set decline about three percent, in contrary to the second data combination, which grew from 66.95% to 68.75% accuracy. The remaining combinations of classes change both barely in the negative direction. The development of the numbers is shown in figure 5.22.

The most interesting behavior can be discovered on testing the data sets with ten classes and all 20 classes. The data set with the first ten categories gets a boost of four percent, whereas the result of the second composition declines by five percent. The surprising development concerns the classification of all classes. The additional noise reduction improves the accuracy value by ten percent and this value is now four percent higher than the result of classifying only ten classes. To summarize, classifying 20 classes performs much better, than classifying only these ten classes. Especially, what should be kept in mind, more classes implies more articles, thus more test data and a higher denominator for calculating the accuracy. To reach the same or higher percentage of classifying ten classes, much more texts have to be classified correctly. The actual data is visible in figure 5.23.

To conclude, the filtering of noise, which can be detected by examining the actual data in detail, is extremely important for this classification method. This algorithm seems more vulnerable for noise, because it strongly depends on the words contained in the texts, irrespective of the actual class of the text. This makes it even more prone to noise than the other algorithms. Thus, reducing noisy data can boost the accuracy values significantly. Unfortunately, the exact values can not keep up with the results of

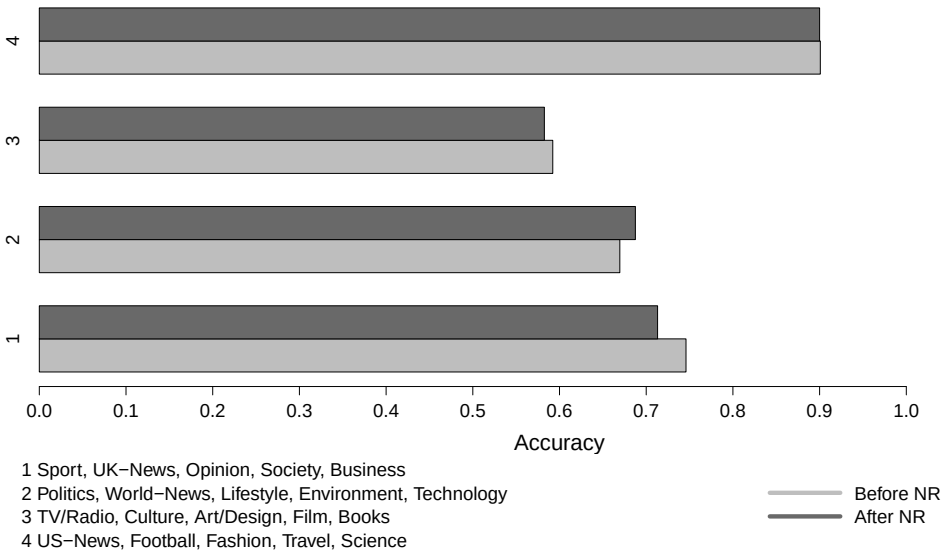


Figure 5.22: Comparing data sets with five classes before and after noise reduction with Semantic Fingerprinting.

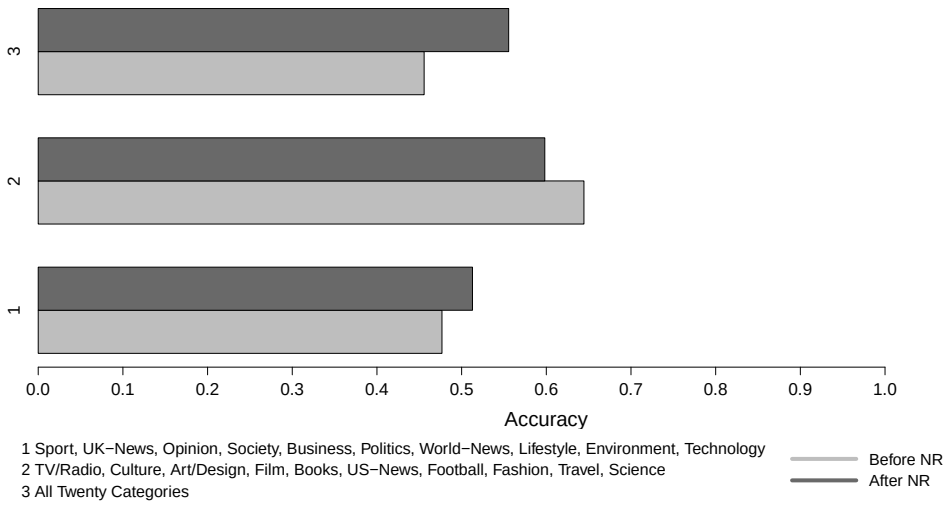


Figure 5.23: Comparing data sets with ten and twenty classes before and after noise reduction with Semantic Fingerprinting.

the classical methods. By this, the next goal is to analyze the components of the method and to figure out why it is working as it does.

5.3.3 Comparison of Two Approaches

Similar to the Naive Bayes algorithm, a different approach can be tested concerning the including of term frequency into the classification process. The test cases up to this point have not taken the actual word frequency in the test documents into account. This implies, all duplicates of words are removed to finally simplify the word vector

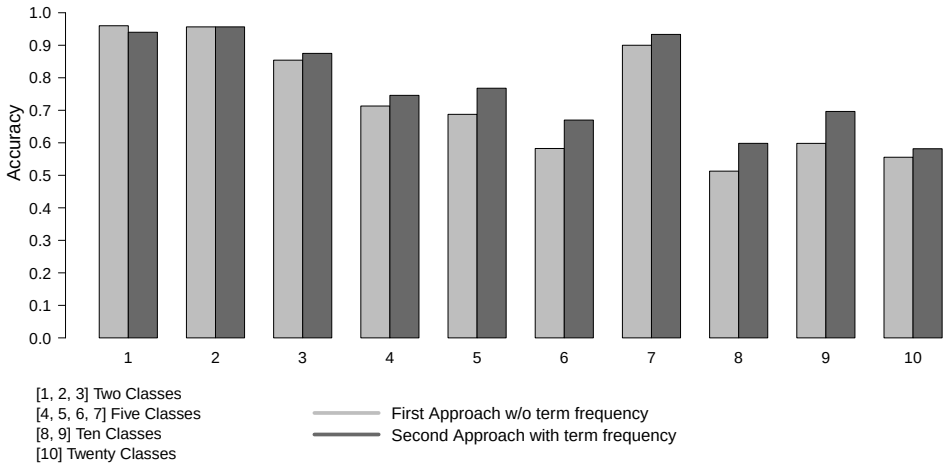


Figure 5.24: Comparing different data sets and their results of two variations of Semantic Fingerprinting.

and to create the test document fingerprint from the unique words. However, with the inspiration of the Naive Bayes, this small modification of the Semantic Fingerprinting method has been integrated and tested additionally.

The removal of word repetitions in the test documents accelerates the computation time, because it minimizes the word vector length. But for this algorithm it might be more essential to improve the selection of the right stacks for the fingerprint to enhance the correct labelling. After changing this setting for term frequency, the tests are repeated.

As shown in figure 5.24, the removal of the duplicates improves the majority of the accuracy values. For data sets with two classes, it is not as high as for the data sets with five, ten and 20 classes. All values, despite the first two data sets, are increasing by at least three percent and even up to ten percent. A data corpus with 20 classes has now an accuracy value of 58.17% and with ten classes, a value of 59.83% and 69.64%. Even if these values are still not as high as the results of the classical algorithms, the gap between them is reduced considerably with this extension approach.

5.3.4 Data Corpus Composition

One of the conspicuous characteristics of this method is the dependency on the data set composition. As witnessed multiple times during the testing phase, the classification seems to perform better or worse, according to the classes contained in the data corpus. Thus, it implies something puts a damper on the results when particular classes are contained. After this perception, another in-depth testing period is initiated. The aim of it is to find the classes which diminish the accuracy value of the classification. However, the outcome took a slightly different direction as expected. The results are depending on the combination of classes in the data corpus, not on particular classes which always reduce the accuracy value.

In conclusion, the only factor, which can affect such behavior is the similarity value between classes. This assumption is also supported by the overall workflow of Seman-

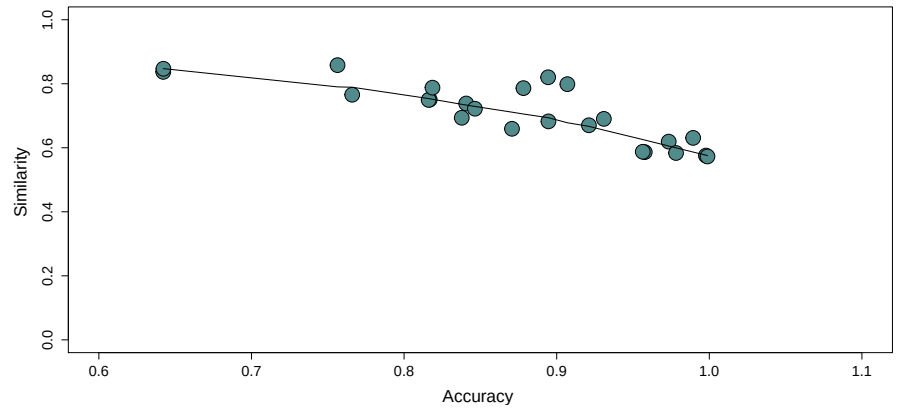


Figure 5.25: Relation between similarity and average accuracy by means of data sets with two classes.

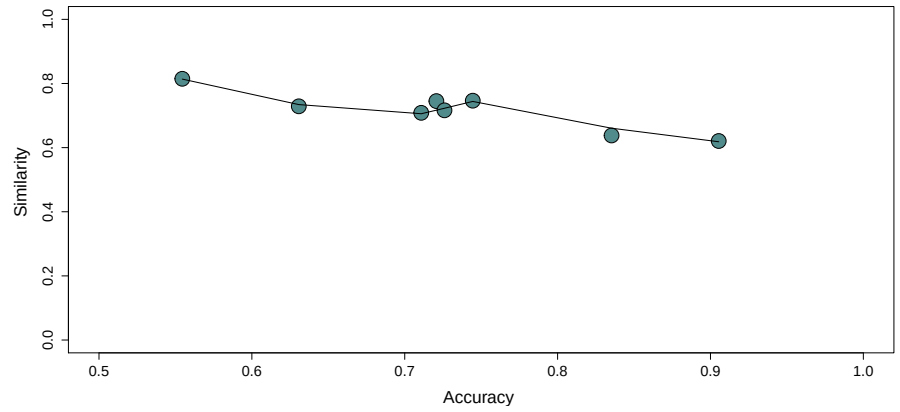


Figure 5.26: Relation between average similarity and average accuracy by means of data sets with five classes.

tic Fingerprinting. If classes are similar, they consist of similar articles and contain in particular similar words. This results in similar fingerprints, which then complicates the assignment of a respective fingerprint and further, a respective class. To obtain assurance, the similarity and average accuracy get measured using 24 different combinations of binary[1] data sets and are visualized in figure 5.25. As it is shown in the figure with the black line, the obvious tendency is falling. The lower the similarity, the higher is the accuracy performance. The same inclination is contained when data sets with five classes are tested. As it is shown in figure 5.26, the average similarity and accuracy are used.

With these circumstances in mind, some distinct trial runs can be performed to visualize the impact of the similarity value. Starting with a binary classification and adding iteratively new classes, shows the effect of each newly added category very well. The first classification deals with articles from the classes `World-News` and `Football`. To

[1] binary = data set with two classes

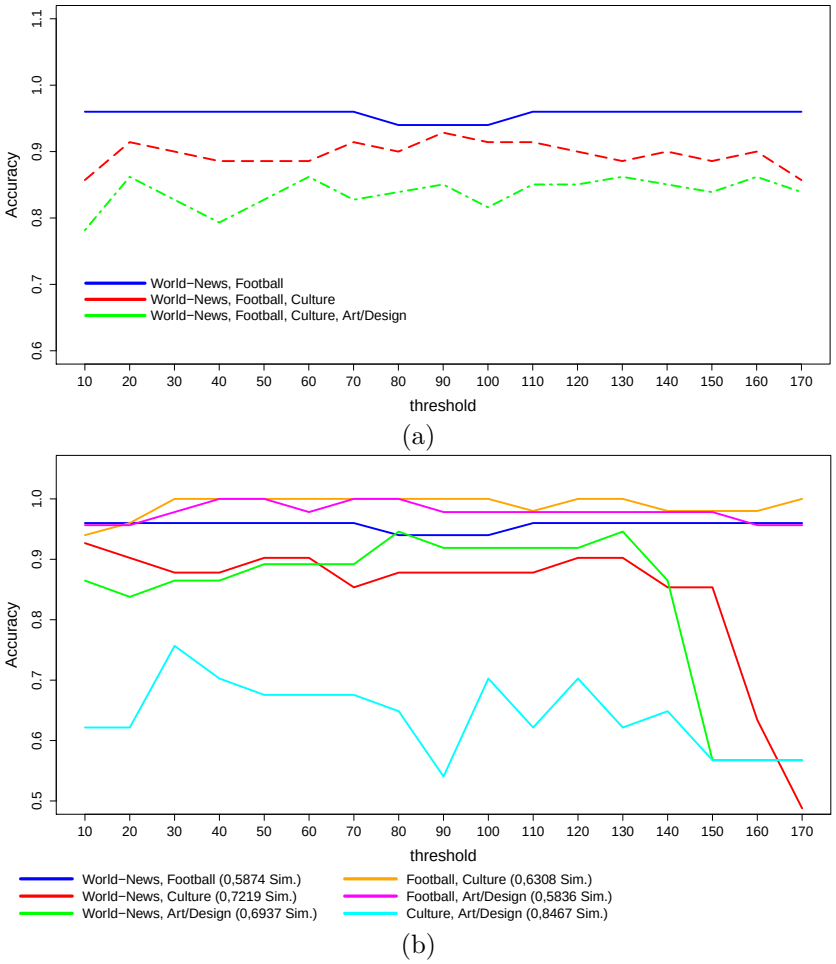


Figure 5.27: Impact of single classes when adding to the data corpus (a) and break down of six binary classifications in relation to their similarity (b).

get an overview of the development of the accuracy for this data set, multiple test cases with different threshold values have been executed and recorded. The resulting curve is shown in figure 5.27 (a) as a blue line and the average accuracy is about 95.64%, which is quite strong. Adding now the next class, in this case `Culture`, the mean accuracy drops off to 89.57% and the performance curve as well (shown as red line in figure 5.27 (a)). Finally, the green line in the figure represents the last adjustment, thus adding the class `Art/Design` to the already measured data set. Now the average accuracy is about 83.76% and the curve goes down further.

To find out the reason for this decline, the single components of the data sets have to be broken down and analyzed. Combining each component of the corpus with each other and classifying these binary combinations, shows that `World-News` and `Football`, `Football` and `Culture`, and `Football` and `Art/Design` works quite well, whereas `World-News` and `Culture`, and `World-News` and `Art/Design` causes problems. Though, the worst combination is `Culture` and `Art/Design`. Looking now on the similarity values, provides insight into the behavior of the single classifications. The classes, which

Table 5.4: Different data set compositions and their maximum accuracy (five classes).

Categories in Data Corpus	Accuracy
Sport, UK-News, Opinion, Society, Business	80.32
Politics, World-News, Lifestyle, Environment, Technology	80.35
TV/Radio, Culture, Art/Design, Film, Books	67.96
US-News, Football, Fashion, Travel, Science	95.00
US-News, Technology, Science, Sport, Opinion	78.37
World-News, Football, Politics, Fashion, TV/Radio	90.16
Culture, Environment, Art/Design, Lifestyle, Travel	72.97
Books, UK-News, Business, Film, Society	70.79

work well in combination have the three lowest similarities, while the classes of the worst average accuracy have the highest similarity. This confirms again the theory of the dependency on the similarity between classes in the data corpus. The lower the similarity between classes, the higher the accuracy value can get. This implies, that the red development curve in figure 5.27 (a) goes down because of the difficulty between `World-News` and `Culture`, and the green curve because of the problems between `World-News` and `Art/Design`, and `Art/Design` and `Culture`. The break down of the binary classifications is shown in figure 5.27 (b).

5.3.5 Threshold Analysis

The second striking characteristic concerns the influence of the threshold. As already shown in the graphs above, the threshold value changes the resulting accuracy value. With the appropriate value, the performance can be boosted significantly. The question is now, is there an area, where the accuracy values are slightly higher than in the remaining sections, similar to the behavior of the kNN. To find an answer to this issue, data sets in ten different combinations have been tested and their results are finally recorded on a graph.

As shown in figure 5.28, the accuracy values are fluctuating continuously in a small area, but on the whole, the tendency is again towards the first third or the center of the graph. Small and high values seem not expediently. However, the fluctuation is the major issue, because it is not predictable. The actual breakdown of the maximum values of the different compositions are recorded in table 5.4.

What is evident on these definite numbers, is that some data sets are located in the central field with moderate numbers between 75% and 80%, other can reach a climax with over 90% and still others are falling to a low beneath 75%. Few particular class combinations seem to boost or on the contrary lower the resulting accuracy. `Football` and `Fashion` seems to be a booster, while `Culture` and `Art/Design` or `Film` and `Books` are rather a damper. In closing it can be stated that these enhancing or weakening factors are summing up and influence the final result.

To return to the impact of the threshold, when testing four different data sets with

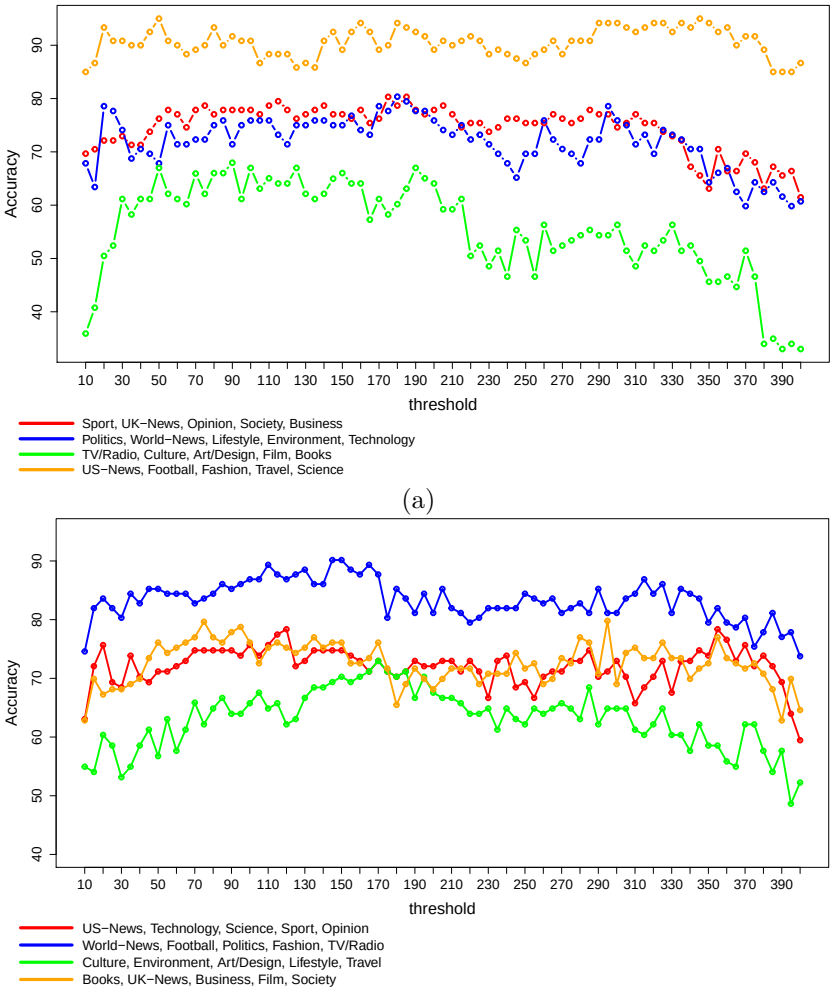


Figure 5.28: Testing different combinations of data sets (five classes) with changing threshold.

ten classes, a similar behavior can be witnessed. The accuracy value is again decreasing with the height of the threshold and the best results are located in the central area. The performance curves are shown in figure 5.29.

The exact numbers of the maximum accuracy are recorded in table 5.5. Once again, the numbers show that the highest accuracy value for a data set with ten classes is the second one with 75%. However, this data set consists again on the one side of the best data set with five classes and on the other side of the worst composition of five classes. But in the end, this consolidation of weak class combination and strong ones, seems to restore the balance. This observation has already been witnessed in case of the kNN. It also applies for the Naive Bayes, but less significant.

The last note to mention, concerns the general procedure of the Semantic Fingerprinting method. To make the comparison of articles possible, the texts have to be converted into their semantic fingerprints as well as the whole training data of the class

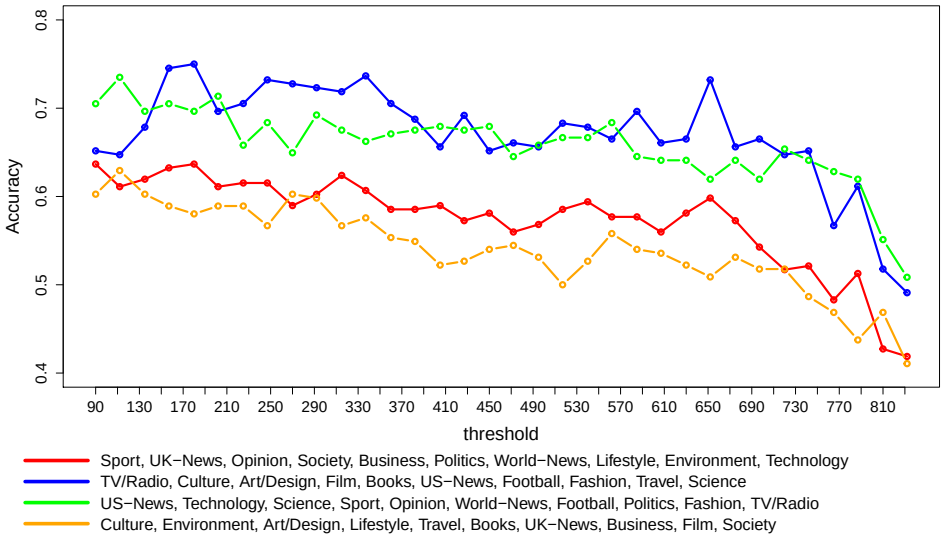


Figure 5.29: Testing different combinations of data sets (ten classes) with changing threshold.

Table 5.5: Different data set compositions and their maximum accuracy (ten classes).

Categories in Data Corpus	**Accuracy**
Sport, UK-News, Opinion, Society, Business, Politics, World-News, Lifestyle, Environment, Technology	63.67
TV/Radio, Culture, Art/Design, Film, Books, US-News, Football, Fashion, Travel, Science	75.00
US-News, Technology, Science, Sport, Opinion, Politics, Football, World-News, Fashion, TV/Radio	73.50
Culture, Environment, Business, Lifestyle, Art/Design, Travel, Books, UK-News, Film, Society	62.94

to combine them to the category fingerprint. For both computations, a threshold is needed. All the test cases before used the same threshold for the test document and the category. After a small investigation it turned out that the tendency is the following: with a slightly lower test document threshold than the threshold for the class fingerprint a small improvement can be caused in some circumstances. Unfortunately, the actual value is again variable and differs from data set to data set, which makes it difficult to distinguish. Nevertheless, it should be kept in mind that there are two possibilities to change the threshold and to affect the outcome.

After determining the maximum accuracy values for the different data sets, a final adjustment gets tested, the combination of the second approach, which includes the term frequency of the words inside the test document into the classification process, and the threshold which maximizes the accuracy. This change seems to effect the accuracy values again slightly, in a positive direction or negative one. In three of four data set

combinations, mentioned in the table before, including ten classes, it increases the value by one or one and a half percent and in one case it decreases by this value.

To conclude, Semantic Fingerprinting has many parameters, which can be adjusted and in the perfect combination, the resulting accuracy values can be boosted and nearly keep up with the results from the classical algorithms. A more detailed comparison can be found in the following section.

5.4 Comparison of Results

The main goal of this thesis and project was to compare the neuroscientific algorithm (Semantic Fingerprinting) with two classical algorithms in a small and manageable test environment, where parameters can be changed easily and the effects recorded. After the reimplementation of the algorithms, the major task was an in-depth analysis. The following section summarizes all different assets and drawbacks of the respective methods.

The first analyzed method was the k-nearest-neighbor algorithm. This method classifies new, unseen data on basis of the similarity value between the unlabeled article and each article in the training data set. The class of the article with the highest similarity value gets assigned to the new data. This algorithm performs very well, but the computation time is rather long, because of the various comparisons and calculations of similarity values. The decisive parameters for this method are the k value, the decision criterion and the similarity function. In figure 5.30, this algorithm is represented as red bars and as it is shown, for five classes the results range from about 77.66% up to 98.33% accuracy, for ten classes it yields an accuracy between 72.6% and 84.3% and for 20 classes, 71.67% accuracy. The overall tendency is, the more classes contained, the more difficult seems the separation.

The next algorithm, the Naive Bayes, is four times faster, because it computes a major part of the used numbers beforehand. It is basically based on conditional probability to assign a class to a new input document. By computing for each word in the article the probability that the document has a certain class when it is given that the word is included and summing them up to a joint probability, results in an overall probability for each category. The class with the highest probability is finally assigned to the document. This algorithm yields, with a few exceptions, equal or in many cases about several percent higher results than the kNN, which are shown in figure 5.30 with blue bars. For five classes the results range again from about 77.66% up to 98.33% accuracy, for ten classes from 75.21% to 86.60% and for 20 classes it yields 74.72% accuracy. This shows that this algorithm is the best and fastest working algorithm in this comparison set up.

Finally, the juxtaposition is concluded with the Semantic Fingerprinting method, which is in general based on the conversion of words, texts and categories into semantic fingerprints as well as on the comparison of them by similarity. Certainly, it has the longest computation time in this kind of reimplementation, because of the context map calculation, which has to be sorted by a similarity measure, and the various fingerprints. As it is shown in figure 5.30 with yellow bars, this algorithm provides the lowest results in this test implementation set up. The accuracy values are constantly a few percent lower than the previously recorded values from the other methods, with a maximal

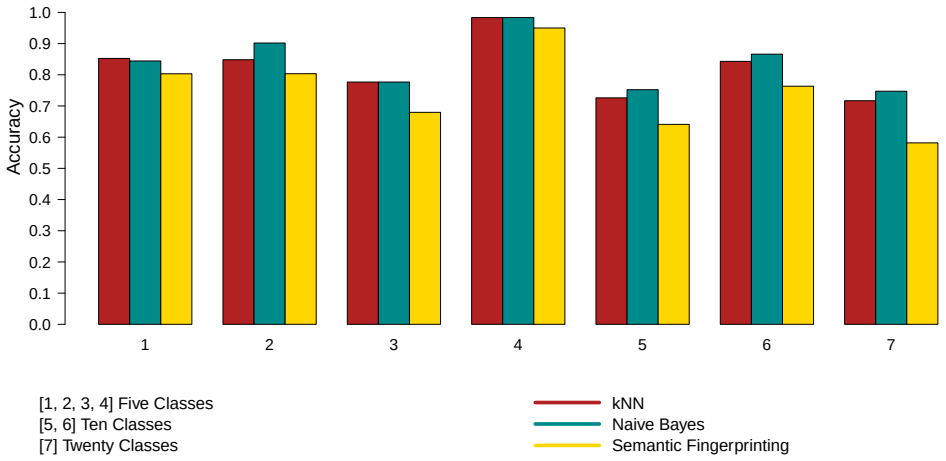


Figure 5.30: Comparison of result from all reimplemented algorithms including five, ten and twenty classes.

difference of 15%. It yields for five classes accuracy values between 67.96% and 95%, for ten classes the results range form 64.10% to 76.34% and for 20 classes, the accuracy is about 58.17%. Even if the values seem rather underwhelming at the first glance, with progressive examination, it becomes apparent that the values are definitely justifiable and in general, it performs similar to the classical methods and this is satisfactory. The algorithm holds great potential with its approach, but it has multiple sensitive aspects.

The main influencing factors, which got extracted during the analysis are the threshold value, the similarity between classes inside the data corpus and the kind of construction of the context map. These three key parameters can change the accuracy values significantly. Whereas, the underlying difficulty for this method is something different. It is, on the one hand that all words are equally important, so there is no weighting, and on the other hand, that it merely examines in which texts of the context map are the words included, irrespective of the actual class of this training data.

The first aspect shows the relevance of the underlying training data and the importance of filtering uninformative documents or words from the texts, because each word and each text gets converted into its fingerprint and subsequently included in the classification process. The more noise is contained, the more the results are distorted. This applies for the remaining algorithms as well, but in combination with the absent weighting, it has greater impact.

The second key point mentioned above, explains why this algorithm has more difficulties to separate the classes than the remaining methods. The reason is in the end, again related to the similarity of classes. During the conversion of words into fingerprints, the method only checks if the word is contained or not in the underlying text on the context map. It does not take the actual class of this text into account. Compared to the Naive Bayes, which even takes care of the relation of the exact term frequency inside a distinct class and outside of it, this option seems relatively restricted. Finally, if two classes consist of similar vocabulary, similar class-intersecting fingerprints are produced, which affects the separation between the classes negatively. Consequently, with these circumstances, this method is more fragile and vulnerable. But in spite of these

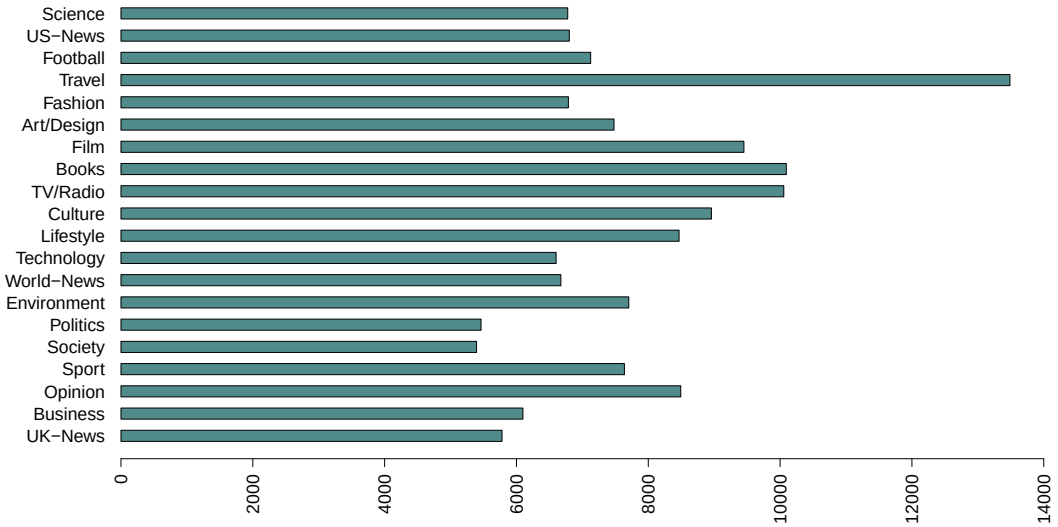


Figure 5.31: Number of unique words (vocabulary) per categories in final test data corpus

weaknesses, the general approach and the underlying mechanism are interesting and the results seems definitely worth for further research.

A general property, which applies for all three classifiers, is the dependency on the data set. During the classifiers' learning process, only the training samples are known and the whole classification is based on them. The classified data is not taken care of. Another property, which has been witnessed in case of all methods, is that a binary classification yields the best results and the more classes are added to the data corpus, the lower gets the accuracy value. Naive Bayes comes off best, followed by kNN and finally Semantic Fingerprinting.

5.5 Text Data Properties

During the intense algorithm analysis, also some zero results have been attained. For instance, the accuracy results of the algorithms are not related to the total amount of vocabulary inside the data corpus, it is as well not depending upon the length of the articles in the test data and also the total amount of words is irrelevant. However, these examinations of the text data are useful, because it provides an insight into the data and reveals some interesting properties.

In figure 5.31 the total number of unique words per class are shown. These values contained in the graph, describe the whole vocabulary of the categories and some of them are striking, when comparing the numbers with the total amount of articles per category, shown in figure 5.2. For instance, the class `Travel` has by far the largest vocabulary, but the number of articles is only the third largest. This means, this class contains the most various words in a smaller amount of articles. To the contrary, the class `Football`, which includes the most articles of all categories, has a conspicuous small vocabulary. Thus, it has no great diversity of words. A similar behavior can be witnessed for the

classes `Sport`, `Fashion`, `Business` and `Society`. The classes with a wide vocabulary originate largely from the entertainment sector like `TV/Radio`, `Books`, `Film`, `Culture` and `Lifestyle`. Interestingly, these classes are also those, where the classification algorithms have the most difficulties. The class `Opinion` also has many different words which is quite comprehensible, because this class includes multiple topics for discussions. Categories with objective subjects like `Politics`, `World-News`, `Technology` and `Science` have a more restricted vocabulary.

Correlated to these numbers is the length of the articles contained in the category and thus, the total number of words in the classes. `Travel`, `Opinion`, `Books` and `TV/Radio` are the highest four categories, which include the largest number of words in their fields and also hold the longest articles compared to the remaining categories and which finally results in a larger vocabulary.

To conclude, the examination of the data enables and facilitates the understanding of correlations between algorithms, the processed text data and the results at the end. Nevertheless, it is always important to capture the entire network and see the big picture of the working system.

Chapter 6

Closing Remarks

The following chapter summarizes the issue and results of the thesis and the related project, describes the challenges and difficulties during the development process and concludes with hints for future improvements of the developed work.

6.1 Summary

With the growth of the World Wide Web, the amount of digital free text data increases as well, including online news, blogs and social media communication. This data is 80% unstructured data and to categorize it, text classification algorithms are useful. This thesis focuses on online news articles and their classification process comparing three different algorithms, the k-nearest-neighbor classifier, the Naive Bayes classifier and the Semantic Fingerprinting method. As an instance for this kind of data, articles from *The Guardian* are used. The kNN algorithm classifies new text by comparing it with each training document and determining the highest similarity value. The Naive Bayes classifier dismembers the unlabeled document and computes with the single word probabilities, a joint probability value for each class. The last algorithm, the Semantic Fingerprinting method is the main part of this thesis. It uses the Hierarchical Temporal Memory theory of Jeff Hawkins to convert words of categories and texts into Sparse Distributed Representations, which can than be compared to the new unclassified document to finally detect the most similar topic. The SDRs are named Semantic Fingerprints and consist of binary vectors. For the creation of these vectors, a context map is needed, which consists of the articles in the training data. It constitutes the semantic environment the algorithm is able to work in. This method is novel and controversial, because it uses no statistical approach compared to many other natural language processing algorithms. It is based on the HTM theory, which represents a theoretical approach of how the neocortex works. Thus, it is at the research stage and under development. But the company called Cortical.io, already implemented an API with this idea as central element and won several awards. In the thesis related project, a reimplementation is developed, with the aim to compare the results of it with the reimplementation of the two classical algorithms, k-nearest-neighbor and Naive Bayes. The general task, the algorithms are tested in, is the categorization of unseen news articles based on a training data set. This can be useful for online news services to sort or label online news content.

6.2 Conclusion

The results of the testing phase are overall satisfactory. The methods are compared on basis of the accuracy value. This number indicates how many articles have been classified correctly compared to the total amount of tested data. The algorithm, which performed best with the news articles from *The Guardian* is the Naive Bayes algorithm. It is the fastest of the three tested methods and yields the highest results, with for instance 74.72% accuracy for a data set of 20 classes. The second highest values for the accuracy, generated by the k-nearest-neighbor classifier, whose computation time is namely higher, but the overall results are solid and in some cases even higher than the results of the Naive Bayes. It classifies the 20 classes data set with 71.67% accuracy. The last method is the Semantic Fingerprinting method, but essentially it performs similar to the classical ones. Its accuracy value is 58.17%, about 15% lower, when classifying a data corpus with 20 classes. The less classes and the more diverse classes in the data set, the higher gets the accuracy value. However, its main influencing parameters are the threshold value and the composition of the context map. One main difference between this method and the remaining is that it does not take care, during the creation of the fingerprints, what class the data has beneath the context map, which are subsequently used for the comparison to detect the searched label. It only checks if the word is contained or not, irrespective of the exact class. As a result, the borders between the class fingerprints become blurred by partly matching the same bits in the array. The general tendency of all three methods is the dependency on the contained data and the number of classes in it. If various categories are contained in the data corpus, the accuracy decreases. For binary classification all the algorithms achieved in most cases above 90% accuracy.

In general, the major challenge of this project was to get used to the algorithms, especially for the Semantic Fingerprinting method. All three methods have different priorities and to re-implement them ensures a detailed inside view of machine learning and natural language processing. Another big difficulty was to get an insight into the data the algorithms process. An overall text analysis before starting to work with the algorithms is recommended, because it prevents unpleasant surprises during the testing phase and it might help to explain the final results. For the analysis it is important to use a visual representation to make the data and the properties of it easier to understand. Finally, the examination of the articles and the exact words in them, was definitely reasonable. Due to that a number of noise could be detected and deleted.

On the whole, the articles all algorithms worked with are real life samples and it was an interesting experience to visualize them and see their relation to each other. The k-nearest-neighbor classifier is a recommendable method to start with machine learning and to get used to the principles of predicting labels. Whereas Naive Bayes is one step further towards dismembering the data and counting words. It produced the best results with less computation time and this was fascinating to observe. The gripping key point of the Semantic Fingerprinting method was its different approach of converting text data into binary representations and to apply machine learning functions to them. Despite the gap between the classical algorithms and the neuroscientific one, the general approach and the mechanism of it is highly interesting and the results seem definitely promising and worth for further investigation.

6.3 Future Work

An interesting point in case of Semantic Fingerprinting, which has not been examined in detail and which may hide some additional properties of the algorithm is the construction of the context map. In case of this implementation the whole trainings articles have been used as contexts and compared to the functionality of the Retina API and its production of the fingerprints, this seems rather rough. Smaller parts and finer subdivisions may be preferable to make the difference between classes more evident. Another point to consider would be the representation of the context map. For the practical part of this thesis a one-dimensional array has been used to create the context map. It would be interesting if a two-dimensional arrangement like a grid, would make a difference. The last note to mention, concerns the class fingerprints. They are created by converting all words contained in the texts belonging to the same category into semantic fingerprints and stacking these layers on top of each other to get the final binary array. Another possibility would be to add a step between, to create text SDRs from the stack of word fingerprints in the first place, and than stacking these layers again to get the class fingerprint. With this approach the differentiation between the classes could be more precise.

It might be stated that the overall work flow consists, according to the points above, of adapting parameters, changing some computation parts and testing the algorithms repeatedly. Without extensive testing, it will not be possible to state which algorithm works the best for this application and it always depends on the data and the parameters the algorithm uses.

Appendix A

DVD Contents

Format: DVD-ROM, Single Layer, DVD-R-Format

A.1 Master Thesis (PDF)

Path: /

Master_Thesis.pdf . . .	Comparison of Text Classification Techniques supporting Journalistic Data Organization

A.2 Images

Path: /Images

.pdf,.png	Used Images in Master Thesis
MA_Graphics.ai	Adobe Illustrator-File of graphics

A.3 Evaluation

Path: /Results

Graphs_SFP.xlsx	Detailed graphs about Semantic Fingerprinting results
Results.xlsx	General results of the project

Path: /Results/Diagrams

.jpeg,.png,*.pdf . . .	Graphs of general results

Path: /Results/Application

*.png	Screenshots of User Application

Path: /Results/WordClouds

*.png	Word clouds of different categories

A.4 News Classification Module

Path: /NewsClassificationModule

NCM.zip	TYPO3 Project "News Classification Module“
ReadMe.txt	General introduction of the project files

A.5 Online Sources

Path: /OnlineSources

Adaboost_About.pdf .	Short Description of Adaboost
Cortical_io_About.pdf	Information about Cortical.io
Cosine_Similarity_About.pdf	Description of Cosine Similarity in the Vector Space
Morphology_About.pdf	Description of the term Morphology
Naive_Bayes_Approach01.pdf	Naive Bayes general approach
Naive_Bayes_Approach02.pdf	Naive Bayes theoretical background
Naive_Bayes_Approach03.pdf	Implementation of Naive Bayes in JavaScript
NlpTools_About.pdf . .	Introduction to PHP NlpTools
Structured_Unstructured_Data_Jeremy_Ronk.pdf	Definition of structured and unstructured data
Structured_Unstructured_Data_Sherpa_Software.pdf	Definition of structured and unstructured data
Text_Classification_ Applications_Use_Cases.pdf	Introduction to text classification and related applications
TYPO3_7_LTS.pdf . .	Introduction of TYPO3 7 LTS
TYPO3_About.pdf . .	General introduction of TYPO3
TYPO3_CMS_7_LTS.pdf	New features of TYPO3 7 LTS
TYPO3_Extbase.pdf .	Introduction to Extbase
TYPO3_Extension_Workshop.pdf	Practical demonstration of extension development
TYPO3_Extensions_Development.pdf	Introduction of extension development
TYPO3_OOP.pdf . . .	Object-oriented programming approach of TYPO3
uClassify_About.pdf . .	Short Description of uClassify
Unstructured_Data_80_percent_rule.pdf	General approach of splitting training and testing data

References

Literature

[1] Charu C. Aggarwal and ChengXiang Zhai. *Mining Text Data*. Springer Science and Business Media, 2012 (cit. on pp. 1, 4, 5, 7, 15–17).

[2] Subutai Ahmad and Jeff Hawkins. "Properties of sparse distributed representations and their application to hierarchical temporal memory". *ArXiv e-prints* (2015), pp. 1–18 (cit. on p. 17).

[3] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern information retrieval*. ACM Press, 1999 (cit. on p. 6).

[4] Li Baoli, Lu Qin, and Yu Shiwen. "An adaptive k-nearest neighbor text categorization strategy". *Journal of ACM Transactions on Asian Language Information Processing* 3 (2004), pp. 215–226 (cit. on p. 15).

[5] Wojciech Basalaj. *Proximity visualisation of abstract data*. Tech. rep. University of Cambridge, 2001 (cit. on pp. 23, 24).

[6] Ingwer Borg and Patrick JF. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science and Business Media, 2005 (cit. on pp. 23, 24).

[7] S. Brindha, K. Prabha, and S. Sukumaran. "An survey on classification techniques for text mining". In: *Proceedings of the 3rd International Conference on Advanced Computing and Communication Systems*. IEEE, 2016, pp. 1–5 (cit. on pp. 2, 3, 5, 14).

[8] Heide Brücher, Gerhard Knolmayer, and Marc-Andre Mittermayer. *Document classification methods for organizing explicit knowledge*. Tech. rep. University of Bern, 2002 (cit. on pp. 2, 5, 14, 15).

[9] Andreas Buja et al. "Data visualization with multidimensional scaling". *Journal of Computational and Graphical Statistics* 17 (2008), pp. 444–472 (cit. on pp. 23, 24).

[10] Chee-Hong Chan, Aixin Sun, and Ee Peng Lim. "Automated online news classification with personalization". In: *Proceedings of the 4th International Conference on Asian Digital Libraries*. Nanyang Technological University. 2001, pp. 320–329 (cit. on p. 5).

[11] Yoav Freund and Robert Schapire. "A short introduction to boosting". *Journal of Japanese Society for Artificial Intelligence* 14 (1999), pp. 771–780 (cit. on p. 6).

[12] Eui-Hong Sam Han, George Karypis, and Vipin Kumar. "Text categorization using weight adjusted k-nearest neighbor classification". In: *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2001, pp. 53–65 (cit. on p. 15).

[13] Anna Huang. "Similarity measures for text document clustering". In: *Proceedings of the 6th New Zealand Computer Science Research Student Conference*. The University of Waikato. 2008, pp. 49–56 (cit. on pp. 12, 13).

[14] David A. Hull. "Stemming algorithms: A case study for detailed evaluation". *Journal of the Association for Information Science and Technology* 47 (1996), pp. 70–84 (cit. on p. 10).

[15] Nitin Indurkhya and Fred J. Damerau. *Handbook of natural language processing*. Chapman and Hall/CRC, 2010 (cit. on pp. 7–10).

[16] Peter Jackson and Isabelle Moulinier. *Natural language processing for online applications. Text retrieval, extraction and categorization*. John Benjamins Publishing, 2007 (cit. on p. 4).

[17] Thorsten Joachims. "Text categorization with support vector machines: Learning with many relevant features". In: *Proceedings of the European Conference on Machine Learning*. Springer. 1998, pp. 137–142 (cit. on pp. 5, 7).

[18] Aurangzeb Kahn et al. "A Review of Machine Learning Algorithms for Text-Documents Classification". *Journal of Advances in Information Technology* 1 (2010), pp. 4–20 (cit. on pp. 1–5, 8, 13, 14, 17).

[19] Joseph B. Kruskal and Myron Wish. *Multidimensional scaling*. SAGE Publications, 1978 (cit. on pp. 22–24).

[20] Ken Lang. "NewsWeeder: Learning to Filter Netnews". In: *Proceedings of the 12th International Machine Learning Conference*. Carnegie Mellon University. 1995, pp. 331–339 (cit. on p. 6).

[21] Martin Lennon et al. "An evaluation of some conflation algorithms for information retrieval". *Journal of Information Science* 3 (1981), pp. 177–183 (cit. on p. 10).

[22] David D. Lewis. "Naive (Bayes) at forty: The independence assumption in information retrieval". In: *Proceedings of the European Conference on Machine Learning*. Springer. 1998, pp. 4–15 (cit. on p. 17).

[23] Baoli Li, Shiwen Yu, and Qin Lu. "An improved k-nearest neighbor algorithm for text categorization". *ArXiv e-prints* (2003), pp. 1–7 (cit. on pp. 14, 15).

[24] Julie B. Lovins. "Development of a stemming algorithm". *Journal of Mechanical Translation and Computational Linguistics* 11 (1968), pp. 22–31 (cit. on p. 10).

[25] Christopher D. Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999 (cit. on p. 4).

[26] Miriam Martinez-Arroyo and Luis Enrique Sucar. "Learning an optimal naive bayes classifier". In: *Proceedings of the 18th International Conference on Pattern Recognition*. IEEE. 2006, pp. 1236–1239 (cit. on p. 17).

[27] Andrew McCallum and Kamal Nigam. “A comparison of event models for naive bayes text classification”. In: *Proceedings of the 1998 AAAI Workshop on Learning for Text Categorization*. Citeseer. 1998, pp. 41–48 (cit. on pp. 15, 16).

[28] Marie Rochery et al. “BoosTexter for text categorization in spoken language dialogue”. *Unpublished manuscript* 1 (2001), pp. 1–4 (cit. on p. 6).

[29] Stuart Russell and Peter Norvig. *Artificial Intelligence: A modern approach*. Prentice-Hall Inc, 1995 (cit. on p. 16).

[30] Gerard Salton and Christopher Buckley. “Term-weighting approaches in automatic text retrieval”. *Journal of Information Processing and Management* 24 (1988), pp. 513–523 (cit. on p. 11).

[31] Gerard Salton, Anita Wong, and Chung-Shu Yang. “A vector space model for automatic indexing”. *Communications of the ACM* 18 (1975), pp. 613–620 (cit. on pp. 11–13).

[32] Robert E. Schapire and Yoram Singer. “BoosTexter: A boosting-based system for text categorization”. *Machine Learning Journal* 39 (2000), pp. 135–168 (cit. on p. 6).

[33] Fabrizio Sebastiani. “Machine learning in automated text categorization”. *Journal of ACM Computing Surveys* 34 (2002), pp. 1–47 (cit. on pp. 1, 4, 10).

[34] Catarina Silva and Bernardete Ribeiro. “The importance of stop word removal on recall values in text categorization”. In: *Proceedings of the International Joint Conference on Neural Networks*. IEEE. 2003, pp. 1661–1666 (cit. on p. 10).

[35] N. Suguna and K. Thanushkodi. “An improved k-nearest neighbor classification using genetic algorithm”. *International Journal of Computer Science Issues* 7 (2010), pp. 18–21 (cit. on p. 15).

[36] Ah-hwee Tan. “Text Mining: The state of the art and the challenges”. In: *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining 1999: Workshop on Knowledge Discovery from Advanced Databases*. Kent Ridge Digital Labs. 1999, pp. 65–70 (cit. on pp. 1, 4).

[37] Vladimir N. Vapnik and Vlamimir Vapnik. *Statistical learning theory*. Wiley New York, 1998 (cit. on p. 16).

[38] Francisco E. De Sousa Webber. *Semantic Folding Theroy*. Tech. rep. Cortical.io, 2015 (cit. on pp. 2, 5, 17–22).

[39] Jonathan J. Webster and Chunyu Kit. “Tokenization as the initial phase in NLP”. In: *Proceedings of the 14th Conference on Computational Linguistics*. Association for Computational Linguistics. 1992, pp. 1106–1110 (cit. on p. 8).

[40] Kilian Q. Weinberger, John Blitzer, and Lawrence Saul. “Distance metric learning for large margin nearest neighbor classification”. *Journal of Machine Learning Research* 10 (2006), pp. 207–244 (cit. on p. 14).

[41] W. John Wilbur and Karl Sirotkin. “The automatic identification of stop words”. *Journal of Information Science* 18 (1992), pp. 45–55 (cit. on p. 9).

[42] Peter Willett. “The Porter stemming algorithm: then and now”. *Program: Electronic Library and Information Systems* 40 (2006), pp. 219–223 (cit. on p. 10).

[43] Yiming Yang. “An Evaluation of Statistical Approaches to Text Categorization”. *Information Retrieval Journal* 1 (1999), pp. 69–90 (cit. on pp. 2, 5, 14, 15).

[44] Yiming Yang. “Noise reduction in a statistical approach to text categorization”. In: *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 1995, pp. 256–263 (cit. on pp. 5, 9).

Online sources

[45] Stephen R. Anderson. *Morphology*. 2003. URL: https://cowgill.ling.yale.edu/sra/morphology_ecs.htm (visited on 07/03/2017) (cit. on p. 10).

[46] Florian Brinkmann. *TYPO3 CMS 7 LTS: Das kann die neue Version des Enterprise-CMS*. 2015. URL: http://t3n.de/news/typo3-cms-7-lts-neue-version-654824/ (visited on 10/04/2017) (cit. on p. 25).

[47] Yannick De Lange. *Machine Learning: Naive Bayes*. 2016. URL: https://stovepipe.systems/post/machine-learning-naive-bayes (visited on 07/05/2017) (cit. on p. 41).

[48] Lars Ebert. *Einstieg in Extbase – ein Typo3-Plugin ohne Models*. 2013. URL: https://advitum.de/2013/05/einstieg-in-extbase-ein-plugin-ohne-models/ (visited on 12/04/2017) (cit. on p. 31).

[49] Seth Grimes. *Unstructured Data and the 80 Percent Rule*. 2008. URL: https://breakthroughanalysis.com/2008/08/01/unstructured-data-and-the-80-percent-rule/ (visited on 27/02/2017) (cit. on p. 3).

[50] Sam Hocevar. *PHP NlpTools*. 2004. URL: http://php-nlp-tools.com/blog/ (visited on 11/04/2017) (cit. on p. 27).

[51] Jon Kågström, Roger Karlsson, and Emil Ingridsson. *uClassify*. 2008. URL: https://uclassify.com/ (visited on 02/03/2017) (cit. on p. 6).

[52] Burak Kanber. *Machine Learning: Naive Bayes Document Classification Algorithm in Javascript*. 2013. URL: http://burakkanber.com/blog/machine-learning-naive-bayes-1/ (visited on 07/05/2017) (cit. on p. 41).

[53] Sebastian Kurfürst and Jochen Rau. *Developing TYPO3 Extensions with Extbase and Fluid*. 2009. URL: https://docs.typo3.org/typo3cms/ExtbaseFluidBook/Index.html (visited on 12/04/2017) (cit. on pp. 29–32).

[54] Sebastian Kurfürst and Jochen Rau. *Object-oriented programming in PHP*. 2009. URL: https://docs.typo3.org/typo3cms/ExtbaseFluidBook/2-BasicPrinciples/1-Object-oriented%20Programming-in-PHP.html (visited on 10/07/2017) (cit. on p. 29).

[55] Patrick Lobacher. *Praxis-Workshop für Einsteiger: Extension-Entwicklung mit Extbase und Fluid*. 2010. URL: http://t3n.de/magazin/praxis-workshop-einsteiger-extension-entwicklung-extbase-223999/ (visited on 10/04/2017) (cit. on p. 26).

[56] Benni Mack. *Announcing TYPO3 CMS 7 LTS*. 2015. URL: https://typo3.org/news/article/announcing-typo3-cms-7-lts/ (visited on 10/04/2017) (cit. on p. 25).

[57] Alexandru Nedelcu. *How To Build a Naive Bayes Classifier*. 2012. URL: https://alexn.org/blog/2012/02/09/howto-build-naive-bayes-classifier.html (visited on 07/05/2017) (cit. on p. 41).

[58] Christian Perone. *Machine Learning :: Cosine Similarity for Vector Space Models (Part III)*. 2013. URL: http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/ (visited on 04/05/2017) (cit. on pp. 11, 12).

[59] Jeremy Ronk. *Structured, semi structured and unstructured data*. 2014. URL: https://jeremyronk.wordpress.com/2014/09/01/structured-semi-structured-and-unstructured-data/ (visited on 27/02/2017) (cit. on pp. 1, 3).

[60] RStudio. *Why RStudio?* 2016. URL: https://rstudio.com/about/ (visited on 11/04/2017) (cit. on p. 27).

[61] Robert Schapire. *Explaining AdaBoost*. 2012. URL: http://www.research.att.com/talks_and_events/2012_distinguished_speakers/r_schapire_explaining_adaboost/2012_DSS_schapire_explaining_adaboost?fbid=dTrH_7A8y3u (visited on 03/03/2017) (cit. on p. 6).

[62] Doc Searls. *Keeping Linux Safe Since 1994*. 2008. URL: https://linuxjournal.com/content/keeping-linux-safe-1994-0 (visited on 03/03/2017) (cit. on p. 6).

[63] Parth Shrivastava. *Text Classification: Applications and Use Cases*. 2017. URL: https://blog.paralleldots.com/text-analytics/text-classification-applications-use-cases/ (visited on 05/07/2017) (cit. on p. 5).

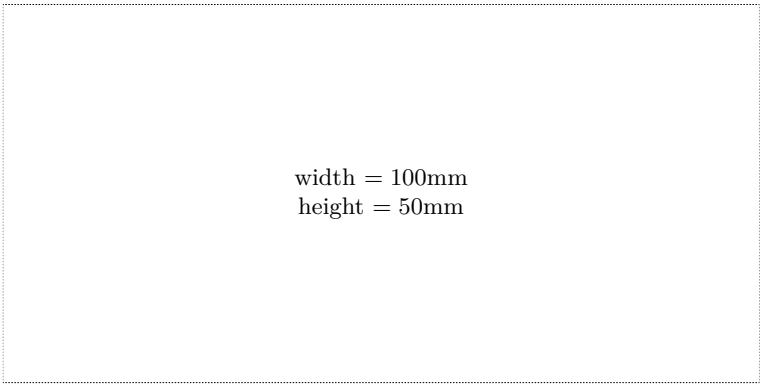
[64] Sherpa Software. *Structured and Unstructured Data: What is It?* 2016. URL: http://sherpasoftware.com/blog/structured-and-unstructured-data-what-is-it/ (visited on 27/02/2017) (cit. on pp. 1, 3).

[65] typo3.org. *The TYPO3 Universe*. URL: https://typo3.org/about/ (visited on 10/04/2017) (cit. on pp. 25, 26, 28).

[66] Francisco E. De Sousa Webber et al. *On a mission to enable high performance language intelligence*. 2011. URL: http://cortical.io/company.html# (visited on 04/03/2017) (cit. on p. 7).

Check Final Print Size

— Check final print size! —



— Remove this page after printing! —