Technology
Arts Sciences
TH Köln

TH Köln – University of Applied Sciences

Faculty of Information Science and Communication Studies

The Institute of Information Science

Degree program: Market and Media Research

Master's Thesis to obtain the degree Master of Science:

"Analysing the systematics of search engine autocompletion functions by means of data mining methods"

First examiner: Prof. Dr. Gernot Heisenberg

Second examiner: Prof. Dr. Philipp Schaer

Semester: Summer semester 2017

Submitted on:

28.08.2017

By the candidate:

Anastasiia Samokhina

Abstract

In the internet era, the information that can be found about politicians online can influence events such as the results of elections. Research has shown that biased search rankings can shift the voting preferences of undecided voters. This shows the importance of studying online search behaviour, especially in the pre-elections phase, when search results can have a particular influence on the future political scene of a country.

This master thesis aimed to study the behaviour of online search engines in a period before the German federal election in 2017. The aim was to ascertain if there is any pattern to be found in the auto-suggestions for searches related to politicians.

In order to gather data for this experiment, a crawler browsed search engine web pages, input a name and a surname of a politician, and saved that together with all autosuggestions from the search engine. The autosuggestions were prepared for the analysis and divided into semantic groups with the help of clustering algorithms.

Different statistical methods, such as correlation analysis, regression analysis, and clustering were used to identify patterns in the data. The research showed that there are no particularly strong patterns in the autosuggestions for searches related to politician's names. Only moderate dependence was found between gender and personal topics, and showed that a higher amount of personal information autosuggestions correspond more to female politicians.

Keywords: data mining, text categorisation, autosuggestions, search engines, politics, lemmatisation

Declaration of Academic Integrity

I hereby confirm that the present thesis is solely my own work and that if any text passages or diagrams from books, papers, the Web or other sources have been copied or in any other way used, all references – including those found in electronic media – have been acknowledged and fully cited.

Date, Place

Signature

Acknowledgements

I am sincerely grateful for the support given to me by my adviser Prof. Dr. Gernot Heisenberg during this thesis. He consistently allowed this master thesis to be my own work, but steered me in the right direction whenever I needed it.

I would also like to acknowledge Prof. Dr. Philipp Schaer as the second adviser of this thesis, and I am gratefully indebted for his very valuable input on this thesis.

I am forever thankful to my beloved partner and friend Isaac True, who was there for me and experienced all of the ups and downs of my research. His knowledge, support and tremendous patience continually amazed me during the work on this thesis.

Finally, I thank my parents Irina Samokhina and Evgeny Samokhin for encouraging me throughout all my studies. Their support has been unconditional for all these years and I could never ask for better friends than them.

Table of Contents

List of Figures

List of Tables

Listings

1. Introduction

1.1. Research questions and motivation

Freedom of information has become an increasingly important resource in the digital era. The growth of information in society has influenced it in many ways, one of which is that it changed the way people search for information and the tools they use for doing so. The fact that the internet became the most important source of information for many people and is used for making day-to-day decisions on the basis of found information, motivated scientists to research different areas which could be affected by it.

One such aspect of social organisation is politics. The internet changed politics in many different ways; the movement of political activity in the internet is already generating massive amounts of data, such as individuals' political conversations, donations, online formats of news and politics, political blogs, online public speeches and openly available information about political activities and involved individuals. And as all public figures, politicians do not always become the object of interest only because of their main work, but also because of other reasons such as information about their private life, health, or appearance.

The information that can be found about politicians online can influence events, such as the results of elections, especially if search engines manipulate the way information is queried. Research has shown that biased search rankings can shift the voting preferences of undecided voters by 20% or more, and such rankings can be masked so that people show no awareness of the manipulation (Epstein and Robertson 2015). This shows the importance of studying online search behaviour, especially in the pre-elections phase, when search results can have a particular influence on the future political scene of a country.

This master thesis is aimed to study online search behaviour in a period before the German federal election in 2017. Search behaviour on the internet is influenced by many factors, such as advertisements and ranking of results in a search engine. Some engines also try to predict search terms and therefore suggest possible keywords when user types search terms. The auto-suggestion can possibly influence a user and change their idea of what they are searching for. Search engine suggestions will be used during the research process, as they are supposed to be based on how often other users have searched for a term, and show the range and variety of information in the internet (Google 2017).

This master thesis raises the question of whether there is any pattern in the auto-suggestions for searches related to politicians' names; if so, are there any differences in the patterns depending on available attributes (social-demographic information, times of search and others)? The master thesis also raises the question of whether the patterns can be explained from a political point of view, though giving a political interpretation is not a main focus of this thesis. The thesis explores data in a hypothesis free way and searches for pattern and particular relations in attributes.

The outset of the research is as follows: a crawler browses search engine web pages, types in a name and a surname of a politician, and saves that together with all auto-suggestions. The data is saved in SQL database which includes information about the politician (socio-demographic information and party), search results, time of search and which browser was used.

A particular method of data analysis will be used to conduct the research: machine learning. It is a combination of several disciplines, such as statistics, information theory, and functional analysis, that allows people to optimise machines by learning from available data (Dietterich 1998). This is important, because nowadays research quite often involves working with unknown condition mapping and overly complicated data which can't be easily manipulated in code.

Machine learning is used for different types of problems, ranging from DNA sequence classification, to stock market analytics. One of the most challenging areas of machine learning is text mining: the process of deriving high-quality information from text. It is particular challenging due to the way data is structured in text. Although text is understandable for people, for the performance of any machine learning algorithms, it must be structured and preprocessed prior to the analysis.

1.2. Thesis structure

The master thesis will consist of five chapters.

Firstly, this introduction positions the paper in relation to existing research in the area, performed through an overview of existing literature. It comprises of different traditional methods of addressing the issue and introduce the context of the research. This section will also elaborate on the main research question and show its importance.

The second chapter of the thesis will provide an overview of the research area and build up a theoretical basis for the thesis. With the help of the literature research it will be shown what the most important terms in machine learning and particularly text mining are, and how the lifecycle of a project is supposed to be carried out in this area. It will define why the main problem of the work lies in the area of both supervised and unsupervised learning.

The third chapter describes the experimental set up the thesis. Firstly, the mechanism of the data crawling program used to collect the data are described together with the data storage. Further, the data is prepared for future analysis. The chapter describes the preparation process: cleaning of irrelevant data and whole process of text categorisation of autosuggestions.

The fourth chapter describes the implementation and results of the analysis. It also presents the interpretation of results.

The last chapter of the thesis will offer a conclusion and position the results in a larger context and offer suggestions for future research in the area.

1.3. Literature overview

In last years data mining area has become widely developed and the term itself became a buzzword both in the scientific and business areas (Delen 2014), resulting in a wide choice of literature available for this topic.

Some works were especially important for providing theoretical background and experimental results. “Data Clustering: Algorithms and Applications” by Charu C. Aggarwal provided the important background knowledge on the problem of clustering analysis (Aggarwal and Reddy 2016). Another important books for this research is “Texts in Computer Science” edited by David Gries, Orit Hazzan and Fred Schneider. The book covers foundational and theoretical material about text mining, main challenges and methods in the area (Gries 2005). Also important for the background knowledge building of the thesis was the article “Text categorization with Support Vector Machines: Learning with many relevant features” by Thorsten Joachims (Joachims 1998) that explores the use of Support Vector Machines algorithm for text categorisation problems.

Data mining methods have become well-known in political science during the last decade and were often used for election analysis. The text mining, in terms of data science, is still relatively young, (the first researches in this area were performed in the late 1990s (Grimes 2007)) has already also been widely used for the political research. Examples such as (O’Connor et al. 2010), (Conover et al. 2011), or (Bae, Son and Song 2013) prove this. But it is important to mention that the majority of work in this area focuses on social media; the area of search behaviour remains rather uncovered. The position of this thesis relative to existing research in text mining area is that it covers particular cases that haven’t been widely researched.

1.4. Methods of data collection and analysis

The following methods will be used in the master thesis:

1. Data generation. This involves the development of a crawler program to generate data from Bing and DuckDuckGo search engines, based on an existing crawler program that generates data from Google Search.

2. Data storage. Obtained data is stored in SQL database that contains information about politicians (their socio-demographic data and party), auto-suggestions, time of search and browser.

3. Data preparation. This requires an identification of possible errors in data and irrelevant for the analysis data points and also prepossessing that involves text categorisation. The goal of text categorisation is the classification of documents into a fixed number of predefined categories. This will be performed following standard procedures necessary for text mining: lemmatisation (the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form (Dictionary 2017)) and vectorisation. The dataset is prepared for analysis according to the requirements of the algorithms. This means that categories which are supposed to cover all search terms are first developed with the help of a clustering algorithm, after which the actual data is categorised with the help of the Support Vector Machines algorithm, which is proven to be very well suited for text categorisation (Joachims 1998). The k-means algorithm is used for the categorisation process.

4. Data analysis. In order to find patterns in data different types of analysis will be performed, from basic descriptive statistics to cluster analysis. The main aim of this method is to find a pattern in the data and to see if there is a difference in these patterns, according to certain attributes such as search terms, socio-demographic data, and party. The descriptive analysis are used to summarise features of data: describe frequencies, central tendency, and dispersion. The inductive part will deduce properties of an underlying distribution by analysis of data. This analysis is performed with the means of correlation analysis, regression analysis and clustering, the results are evaluated to choose the best outcome. This part answers questions such as if there are patterns in auto-suggestion ranking, and if it is possible to build a groups of politicians based on their socio-demographic information and results of search. Centroid-based clustering for a mixed datasets will be used to answer a question about group-building.

The Python programming language is the main software tool used in this thesis. Various libraries and modules such as numpy, scikit, matplotlib, and pattern were used to solve tasks such as the development of the crawler, data preparation, text classification, clustering and production of charts. The full list of used models is indicated in each programming coded listed in the end of the thesis.To perform correlation and regression analysis SPSS is used. The SQL database is used to store the data. The Thesis is written with the help of LaTeXsystem.

2. Data mining overview

2.1. Overview

The chapter has the main goal of describing the field and characteristics of data mining and analysis. An overview of main terms, common procedures, and problems of the field will be provided.

The Cross Industry Standard Process for Data Mining process model and main terms will be described in the first section. Further sections describe analysis methods and algorithms, starting with multivariate analysis methods such as different forms of correlation analysis and regressions and continues with classification and clustering methods.

2.2. Main terms and cycle of analysis

Data volumes have grown significantly during recent years, and manual data analysis has becoming completely impractical in many fields; data analysis in the modern world is performed with the help of data mining.

Data mining "is the computing process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems" (Chakrabarti 2017). Data mining is an interdisciplinary subfield of computer science (Clifton 2009) that evolved with intersection of different research areas such as machine learning, databases, statistics, data visualization, and high-performance computing (Fayyad, Piatetsky-Shapiro and Smyth 1996).

The abstract goal of data mining is making sense of data. This involves mapping big volumes of low-level data into other forms that are more easy to understand for humans, which means that they are more compact, more abstract or more useful (for example, predictive models). To sum up, "the core of the process is the application of specific data-mining methods for pattern discovery and extraction." (Fayyad, Piatetsky-Shapiro and Smyth 1996)

The most widely applied applications of data mining are classification, identification and prediction (Gries 2005). This means that data mining is used to identify patterns in data, divide data into classes, and determine how certain variables will behave.

The data mining process is commonly defined with the following stages:

1. Business Understanding: this stage focuses on understanding the project goals and

requirements from a perspective of a practical implementation, then converting this understanding into a data mining problem definition (Azevedo and Santos 2008).

2. Data Understanding: this stage includes data collection and identification of data quality problems, discovering first insights into the data (Association 2012).

3. Data Preparation: this stage includes all activities to construct the final dataset used for the analysis. This stage may include multiple steps and repetitions (Wirth and Hipp 2000).

4. Modeling: "various modeling techniques are selected and applied, and their parameters are calibrated to optimal values." (Wirth and Hipp 2000).

5. Evaluation: obtained results are evaluated and the models that best suit requirements are chosen (Azevedo and Santos 2008).

6. Deployment: the knowledge gained in the previous stages is organised and presented in a way that the end user can use it (Wirth and Hipp 2000).

This data mining process model is named the Cross Industry Standard Process for Data Mining, commonly known by its acronym CRISP-DM (Shearer 2000). This model is the leading methodology used by the professional community (Piatetsky-Shapiro 2014).

CRISP-DM model was used to perform the analyses in this thesis. The research was performed using the following steps using the model:

1. Business Understanding: determine research objectives, main research field and background, data mining goals and produce project plan.

2. Data Understanding: write a web crawler program [1] and collect data, store this data in an SQL database [2], explore data and verify its quality, and identify main quality problems.

3. Data Preparation: identify and clean irrelevant items from the dataset [3], perform text categorisation [4], and build a final dataset.

[1] A Web crawler is an Internet bot that systematically browses the Internet, typically for the purpose of indexing the information (ICTEA 2017).

[2] A database a usually large collection of data organized especially for rapid search and retrieval. The data is typically organised to to support processes requiring information (Merriam-Webster 2017).

[3] A dataset is a collection of data. Most commonly a data set corresponds to the single statistical data matrix, where every column of the table represents a variable, and each row corresponds to a given member of the dataset (Snijders, Matzat and Reips 2012).

[4] Text mining refers to the process of extracting interesting and non-trivial patterns or knowledge from text documents (Tan et al. 1999). Text mining is a part of data science and it shares main applications with data mining as if classification and prediction. The overarching goal is, essentially, to turn text into data for analysis, via application of natural language processing and analytical methods (Ojeda et al. 2014).

4. Modeling: select methods to identify new knowledge in the dataset. Select modeling techniques and perform multivariate analysis and clustering.
5. Evaluation: evaluate results and review the process.
6. Deployment: produce the final text of the thesis.

The sequence of the phases in CRISP-DM is not strict, and moving back and forth between different phases is expected (Wirth and Hipp 2000). For example, during the Modeling stage stepping back to the data preparation phase was often needed. Therefore, while the CRISP-DM was chosen as a basis for work, it is not suitable as a basis for the final report, since it would be difficult to follow.

The particular challenge of the data preparation process in this research was a work with unstructured data in the form of text. Text mining methods were used to transform this type of data in the suitable for the analysis numeric form.

The significant difference between data mining and text mining is, that data mining mostly operates with numbers, while input for a text mining is an unstructured text data (Gries 2005).

Text mining process typically includes following subtasks (Vidhya. K. 2010):

1. Information retrieval: this is a preparatory step; collecting or identifying a set of textual materials relevant for the analysis.
2. Text Preprocessing: preparation of the text for the analysis, often including tokenising, stop words elimination, stemming or lemmatisation (Vijayarani, Ilamathi and Nithya 2015).
3. Text Transformation: text transformation refers to convertign text into a vector space model, which can be used for further effective analysis (Kumar and Karthika, n.d.).
4. Text Mining Techniques: on this stage different text mining methods can be introduced to the research. For example, clustering, text categorisation or semantics analysis (Kumar and Karthika, n.d.).
5. Evaluation. Evaluation and deployment of obtained results.

The traditional text mining deals with a corpus of texts, but in this research it is used to separate individual words in the semantic groups. Therefore, not all methods and steps of text mining process were reasonable and possible to use for the research purposes. The text mining routine of this thesis is as follows:

1. Information retrieval: obtain a list of unique words from the database.
2. Text Preprocessing: clean irrelevant terms, solve encoding problems, and perform lemmatisation.

3. Text Transformation: transform words into vectors.
4. Text Mining Techniques: cluster the training set of words and perform text categorisation on the rest of the dataset.
5. Evaluation: evaluate results and prepare data for the further analysis.

An important component of Text Preprocessing is bringing a word into its canonical form that will be recognised by further algorithms. There are two ways to solve this problem: stemming and lemmatisation. The goal of both algorithms is to "reduce inflectional forms and sometimes derivationally related forms of a word to a common base form." (Manning, Raghavan and Schütze 2008)

Stemming is "the process of reducing inflected (or sometimes derived) words to their word stem, base or root form" (Lovins 1968). For example the words "argue", "argued", "argues", "arguing", and "argus" reduce to the stem "argu". Stemming algorithms work by cutting off the derivational affixes of the word while looking for the root. The most common German stemming algorithm is Snowball. This includes characters with umlaut marks and the grapheme ß. The algorithm searches for the suffixes of cases (such as -em, -ern, -er), form (for example -en, -er, -est for adjectives) and part of speech (for example -keit, -ung, -isch) and removes them to find the root (Porter and Boulton 2017).

The stemming algorithm is often used in the text mining. For example, it is a common element in query systems such as Web search engines (Deyasi et al. 2016). However, it is not useful for this research, since the main aim of Text Preprocessing stage in this thesis is to prepare words for their further transformation into the vectors, which will be impossible to perform on stems that are only parts of words; vector transformation requires complete words.

More suitable for this purpose was the lemmatisation algorithm. "Lemmatisation is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form" (Dictionary 2017). For example, the word "better" has "good" as its lemma. Such link would be missed by stemming but correctly identified by lemmatisation algorithm.

The lemmatisation system has two components: an algorithm and a lexicon (Perera and Witte 2005). The first one lemmatises the words depending on morphological classes. The lexicon includes words and their part-of-speech tag, along with rules for unknown words based on word suffix (Linguistics and Psycholinguistics 2017a). The lemmatisation algorithms are much more complex, they also require a part of speech taggers and chunking algorithms to perform. The complexity of the lemmatisation algorithms might be a reason, why the choice of them is narrow, especially for languages rather than English. This thesis uses the pattern.de Python module which includes all needed algorithms and has an overall lemmatisation accuracy of about 87%. (Linguistics and Psycholinguistics 2017b).

Returning words into a dictionary form is a required step for the vector transformation. It takes a corpus of text as its input and builds a vector space of several hundred dimensions. Each unique word in the input is assigned a corresponding vector in the space (Mikolov 2013). The reason why this step is important is that the vector, as a array of floating point numbers, has a usable coordinates as it relates to other words, and can be used to perform calculations. So while text itself can't be used directly to perform clustering, the vectors can be. This is becuase the distances between them can be used as a measure of similarity.

The Python module word2vec, created by a team of researchers led by Tomas Mikolov at Google, is used to perform a vectorisation in this research. But to produce German word embeddings, the model must first be trained. This means that the model will build a vector space first and then the input words will be located in this vector space. The GermanWordEmbeddings toolkit was used as a trained model in this research. Its corpus includes 651,219,519 words, and was trained with word2vec with the German Wikipedia and news articles written in German (Mueller 2015). The model is trained using skip-gram as training algorithm, where "the model uses the current word to predict the surrounding window of context words. The skip-gram architecture weighs nearby context words more heavily than more distant context words" (Mikolov 2013). This approach, according to the author of word2vec, is slower, but performs better in the datasets with infrequent words (Mikolov 2013).

Embedding vectors created using the word2vec algorithm has advantages compared to algorithms like Latent Semantic Analysis, such as speed and accuracy. Research shows that models trained on large enough corpora perform better than other approaches (Mikolov 2013).

The vectorising of text allows one to perform further analysis, such as, in this case, text categorisation. Text categorisation (or text classification) is the task of assigning predefined categories to free-text documents (Ko and Seo 2000) The main goal of text categorisation is "the classification of documents into a fixed number of predefined categories." (Joachims 1998).

Automatic text classification can be performed with the help of supervised classification algorithms, but some external mechanism should provide information on the correct classification for items. Which means, that the categories should be defined beforehand. For this research the definition of categories was part of the task. Which means that text classification task in the research had two sub-tasks: define categories with the help of unsupervised learning methods, and use defined categories to perform supervised text classification. Both methods will be discussed in the following sections.

2.3. Unsupervised learning for clustering

"Unsupervised learning is the machine learning task of inferring a function to identify hidden structures in "unlabeled" data" (Hsu, Chang and Hsu 2017). There is no particular corresponding target output associated with each input in the unsupervised learning, therefore unsupervised learning tends to be more subjective than supervised learning.

The most common approach of unsupervised learning is clustering. Clustering is the task of grouping objects in sets, such that objects within a cluster are "as similar as possible, whereas objects from different clusters are as dissimilar as possible" (Oliveira and Pedrycz 2007).

Since there is no exact definition on how to measure similarity between objects, the various clustering algorithms co-exist and the results of clustering by different algorithms may vary significantly. While there is no objectively "good" clustering algorithm, an algorithm that is designed for one kind of model or particular type of data will generally fail on a dataset that contains a radically different data (Estivill-Castro 2002).

There are two particular requirements for the clustering algorithm in this research: +

1. The ability to perform in high dimensional space.
2. The ability to perform clustering on mixed numeric and categorical data.

Clustering of high dimensional data means that clustering is performed with more than three features, which causes a particular problems. First of all, it is impossible to visualise results and make any particular observations on the quality of clustering or probable pattern in the data (Steinbach, Ertöz and Kumar 2004).

But one of the biggest problems of high dimentional clustering is a scaling problem. The distances between values of variables vary; for example the effective "distance" between the values 20 and 80 for variable Age is much higher than for the variable Salary in Euros. This means, that before working with the data, standardisation should be performed. Data standardisation is the critical process of bringing data into a common format (Sciences and Informatics 2017). Standardisation is a common requirement for many machine learning estimators that are not designed to scale data on their own. Therefore, standardisation is an important task for working with high dimensional dataset.

In terms of the second requirement, working with the mixed and categorical data, clustering categorical data needs a special approach. One can not just use the same algorithms as for numerical data, because while distances between numerical data can be defined and interpreted, there are no obvious way to define distances between categories, such as the distance between "male" and "female."

Clustering of categorical data, however, is widely researched and tested in statistics, and scientists have established the following common methods to approach this problem:

1. Establishing dummy variables for each categorical feature of the dataset and compute inter-subject distances for each category, followed by the use any of linkage procedures on these distances. There two problems in this approach. The first is that the scientist needs to define distances between categories manually. This can sometimes be impossible due to the lack of explanation on how exactly distances are measured. The second problem of the approach is an insufficient usage of memory, if the dataset is rather big (for example, more than 4000 cases of many-hundred-dimensional data) (Chaturvedi, Green and Caroll 2001).

2. Latent class procedures. Latent class techniques do show good results, but "become computationally intensive when the number of variables becomes larger." (Chaturvedi, Green and Caroll 2001)

3. The k-modes algorithm. This is a non-parametric approach for defining groups in a dataset with categorical features (Chaturvedi, Green and Caroll 2001). The experiment carried out by Dr. Anil Chaturvedi shows that k-modes performs as well as latent class procedures in clustering, but outperforms them in speed. K-modes also includes the k-prototype algorithm that can be used to cluster mixed data.

Taking into consideration the requirements, ease of use, and well-supported and documented Python modules, the k-means algorithm and its extensions (k-prototypes) were chosen to perform clustering in this research.

k-means clustering is a popular method for cluster analysis. The procedure follows a procedure to divide a dataset into a certain number of fixed clusters. At the beginning, cluster centroids will be defined, and then each data point is associated to the nearest centroid. When no point are left, the first step is completed. This procedure will be repeated to achive the best result where an objective function, in this case a squared error function, reaches its minimum (Matteucci 2017).

The main disadvantage of k-means is the sensitivity of the final clusters to the selection of the number of centroids and the fact that the algorithm can produce empty clusters, while other algorithms such as DBSCAN do not require the specification of this parameter (Lab 2013). Therefore, a careful choice of k[5] is required.

Another disadvantage is that algorithm is based on spherical clusters that are separable in a way such that "the mean value converges towards the cluster center" (Eckroth 2017). Which brings not always good results for the clustering with a different cluster shape.

[5] k refers to the number of clusters used in the execution of the algorithm.

K-means is a simple and efficient algorithm that can be used for a large variety of data types despite its disadvantages.

2.4. Supervised learning for classification problems

”Supervised learning is the machine learning task of inferring a function from labelled training data” (Mohri, Rostamizadeh and Talwalkar 2012). This means that supervised learning have an expected output, called the supervisory signal. The output value, together with the input object, forms the training data (a set of data used to discover potentially predictive relationships) (Reviews 2016a). A supervised learning algorithm analyses the training data and produces a function that can be used for mapping an output value to new inputs (Lu et al. 2017).

After the categorisation is performed, the results are evaluated with the help of the test set. This is used to assess the strength and utility of a set of predictions, and is built by manually assigning correct output values to a set of inputs. The obvious problem of this approach is a need to sacrifice the data that could otherwise be used to train the model for the evaluation. Therefore, this research used cross-validation technique, where the training set is split into x smaller sets, and then a model is trained using $x-1$ of the sets for training. The results are then validated using the remaining part of the data (Schneider and Moore 2000). This approach helps to use maximum data for the training and still perform the evaluation.

There is a wide choice of supervised learning algorithms available. Support Vector Machines (SVM) are considered as a good approach for text mining problems, as Thorsten Joachims showed in the paper “Text Categorization with Support Vector Machines: Learning with Many Relevant Features”, “the experimental results show that SVM consistently achieve good performance on text categorisation tasks, outperforming existing methods substantially and significantly” (Joachims 1998).

Support Vector Machines are based on the Structural Risk Minimisation principle. This means that the algorithm searches for a hypothesis for which the lowest true error can be guaranteed (Osuna, Freund and Girosi 1997). True error is a type of error that describes the probability that the hypothesis will cause an error on randomly selected test example. The main aim of SVM is to find the hypothesis which minimises this bound on the true error by efficiently controlling the the capacity of a hypothesis space (Joachims 1998).

Given a set of training data, in which each object is marked as belonging to one or the other of two categories, a Support Vector Machines training algorithm calculates a model that can assign new objects to one or the other category. Such algorithm makes SVM a non-probabilistic binary linear classifier. An SVM model represents the object as points in a space, “mapped so that all object of the separate category are divided by a gap that is made as wide as possible” (Ordóñez-Blanco et al. 2010). New objects are mapped into the same space and predicted to become part of a category based on which side of the gap they are mapped (Zaman et al. 2013).

One property of Support Vector Machines that makes it especially useful for this research is that their ability to learn can be independent of the dimensionality of the feature space. This means that SVM will bring good results even in datasets with many features, if the data is separable using functions from the hypothesis space (Joachims 1998).

SVM is a commonly supervised learning model that performs well in the high dimensional space and therefore suits the purpose of the research.

2.5. Multivariate analysis

Statistical techniques, such as correlation analysis and regression analysis, were also used alongside machine learning techniques in this research.

“Correlation is a broad class of statistical relationships involving dependence” (Reviews 2016b). Correlation is used to study the relationships among variables and can be used to make predictions.

The Pearson’s χ^2 test can be used to study a relationship between two categorical variables (most of variables in this analysis are categorical). The test evaluates the likelihood of any observed differences between the occurring by chance (Gosall and Gosall 2012). The Pearson’s χ^2 test “compares the observed data to a model that distributes the data according to the expectation that the variables are independent” (Department of Linguistics 2008). It is important to emphasise that Pearson’s χ^2 test can only be used to determine if there is a significant correlation between variables or not; it does not return any information about the strength of the correlation.

Apart from significance of Pearson’s χ^2 test itself, there are two tests for the strength of association that can be used to describe the goodness of the dependence: Phi and Cramer’s V.

“The phi coefficient is a measure of association for two binary variables” (Guilford 1954). The range of possible values lies between -1.0 and 1.0. As the value tends towards ± 1.0, the correlation increases; the sign of the value shows either the negative or positive direction of the correlation. The high correlation score, either positive or negative, shows the low probability of variables being dependent only by chance.

Cramer’s V test is used as post-test to determine strengths of association after the Pearson's χ^2 test has determined significance. If the Pearson's χ^2 test showed the presence of correlation, the Cramer's V test can give additional information about how strong the correlation is. Values of the test may vary from 0.0 (no association between the variables) to 1.0 (complete association), but and can reach 1.0 only when the analysed variables are equal to each other (Dumas 2016).

Another way to explore the relationships among variables is regression analysis.

Logistic regression is a regression model where the dependent variable is categorical (Backhaus et al. 2015). In case of logistic regression with binary dependent variables, it can take only two values, 0 and 1. Cases where the dependent variable has more than two outcome categories should be analysed by multinomial logistic regression (Walker and Duncan 1967). The binary logistic model is used to estimate the probability of a response based on one or more independent variables (Backhaus et al. 2015). It allows one to state "if the presence of an independent variable increases the probability of a given outcome by a specific percentage" (Chen and Kang 2017). Multinomial logistic regression is suitable for the cases in which the outcome can have three or more possible types that, what is important, are not ordered, for example: Group A, Group B, and Group C.

The logistic regression can be understood simply as finding the beta parameters that best fit the equation (where ε is an error distributed by the standard logistic distribution)

$$y = \begin{cases} 1 & \beta_0 + \beta_1 x + \varepsilon > 0 \\ 0 & \text{otherwise} \end{cases}$$

An important part of the work with logistics regression is the evaluation of the goodness of fit. Three common ways of doing it are omnibus test of model, Pseudo-R^2 statistics and classification tables.

The omnibus test for the logistic regression is a likelihood-ratio test based on the maximum likelihood method. The logistic regression uses this method to estimate the coefficients that are able to maximise the likelihood of the regression coefficients given the predictors. Lower values of the likelihood ratio mean that the result is less likely to occur under the null hypothesis as compared to the alternative, therefore the null hypothesis can be rejected (Burns and Burns 2008).

Next way of the evaluation is Pseudo-R^2 statistics. In linear regression the squared multiple correlation, R^2 is used to assess goodness of fit. There is no direct analogue to evaluate the goodness of the logistics regression. There are several Pseudo-R^2 statistics with limitations. Cox and Snell's R^2 is based on the log likelihood for the final model compared to the log likelihood for a model with no variables in it (Cox and Snell 1989). The problem with Cox and Snell's R^2 is that its theoretical maximum value can never reach the value 1, therefore the results can't be interpret as straightforward as Nagelkerke's R^2. Nagelkerke's R^2 s an adjusted version of the Cox and Snell's R^2, such that maximum value can reach the value 1, which makes this statistics suitable for explanation(Nagelkerke 1991)

The classification table is another method to evaluate the accuracy of predictions for the logistic regression model. In the classification table the observed values for the dependent variable and the predicted values are cross-classified.

There are different ways to evaluate the coefficients in the regression – one of the most common is the odds ratio. "An odds ratio (OR) is a measure of association between an exposure and an outcome. The odds ratio represents the odds that an outcome will occur given a particular exposure, compared to the odds of the outcome occurring in the absence of that exposure" (Szumilas 2010). In other words, it is a way to quantify how strongly the presence or absence of a variable A is associated with the presence or absence of a variable B in given data. If the odds ratio is higher than 1, the exposure is associated with higher odds of outcome and otherwise. If the odds ratio is lower than 1, the exposure is associated with lower odds of outcome.

To summarise, this chapter introduced the background knowledge for the further experimental set up and analysis beginning from the main terms and cycle of work, and finishing with an explanation of chosen methods.

3. Experimental methods

3.1. Overview

This chapter describes the first stage of work on the project: data collection, storage, and preparation. It describes how the publicly available information about politicians is stored in the SQL database together with the daily updated information about suggested search terms from different search engines. Next it describes clustering using the k-Means algorithm, and the results of the evaluation with the help of the Elbow method, and the Silhouette and Calinski-Harabaz scores. The end of the chapter describes the task of assigning predefined categories to free-text documents with the help of Support Vector Machines.

3.2. Data storage and collection

3.2.1. Data collection

Socio-demographic data about each politician, such as their date of birth, home town, and party affiliation were collected from publicly available sources.

To get the information about search terms, a web crawler, or an Internet bot, was developed. This crawler systematically browses Internet pages and collects specific data. The crawling programs were developed for three search engines: Google Search, Bing, and DuckDuckGo. The crawler for Google Search was developed by Professor Dr. Philipp Schaer.

The crawler reads a CSV file containing the full names of politicians. For each entry in that file it sends an HTTP request containing the name and surname to the auto-complete API, returning the raw data generated by the API [6] of each search engine. The auto-suggestions are then saved together with the metadata (such as time, language, and browser user agent) in a chosen database. The script parses the raw request data as JSON and extracts the suggestions; the data returned by the search engine is transformed so that it is readable for humans. The script also logs the information about any errors in the search and database.

The web crawler tasks are performed with the help of the built-in Python module concurrent.futures.ThreadPoolExecutor. This module provides classes for working with multiple

[6] An API (Application Programming Interface) provides an interface for other applications to interact and exchange with a given application.

tasks simultaneously. The usage of this module increases the speed of script, because instead of browsing each result and saving it, it utilizes them as a thread pool, spawning a fixed number of threads and executing several simultaneously from a queue. This speeds up the execution of the Python script considerably, as the slowest part of almost any similar script is connecting to a remote server over the Internet.

The script iterates through each politician, and uses extra search parameters depending on the search engine capabilities. When gathering information from Google Search, it pretends to be the following browsers using the User Agent string: Google Chrome, Mozilla Firefox and Opera. Changing the browser User Agent string when searching Bing and DuckDuckGo doesn't provide different results. Searches are made using the German language setting, apart from with DuckDuckGo, which does not provide a language-specific search function. This means all searches made with it are in English, and return English-based results. This is due to the policy of the company, that emphasizes protecting searchers' privacy and avoiding the filter bubble of personalized search results (DuckDuckGo 2017).

There is no officially documented method to get a list of autocomplete suggestions for Google Search. However, there are two unofficial APIs that do not have a documentation. This was also the case for the Bing search engine, who only officially provide a paid autocomplete API. However, they still provide a legacy, undocumented API endpoint for autocomplete. The only search engine in this research that has an officially released free API is DuckDuckGo.

The data from each of the APIs is returned as JSON (JavaScript Object Notation), a language-independent data format that uses human-readable text to transmit array data types. For example, data returned from the Bing API is shown in Listing 3.1.

Listing 3.1: Example return JSON array obtained from the Bing API

```
[
  "Angel Merkel",
  [
    "angela merkel",
    "angel merkels abgekaute fingernagel",
    "angela merkel wiki",
    "angel merkel falten",
    "angela merkels mann",
    "angel merkel humor"
  ]
]
```

The script is run twice a day for each search engine, and the results are appended to the database with each execution.

In case of failures, the script will execute the search up to five times before alerting the user to an error. This mitigates transient server-side errors that would otherwise interrupt the search and cause missed data points, however it does not avoid it completely. A problem with missing data occurred several times due to unexpected errors when querying Google Search. Missing data points were written in a database as performed attempts but with empty cells for the query result. There are various approaches in statistics to deal with missing data, such as imputation, which involves replacing that missing data with substituted values (using mean data or data predicted with the help of regression values). The problem of this approach for this particular case is the difficulty of using it for categorical data such as text. Considering this and a low number of cases, it was decided to simply delete the missing values. The obvious negative side of the deletion is that it discards potentially usable data. But, as Paul D. Allison mentions, "deletion is an 'honest' method for handling missing data, unlike some other conventional methods" (Allison 2002). Therefore, in this work missing points will be excluded from the research.

An example of the data returned by the Google Search API and stored in the database is shown in Listing 3.2. The code used to generate the data is shown in Listing A.1 on page 76.

Listing 3.2: Example raw data obtained from the Google Search Autocomplete API

```
[
  "Katrin Albsteiger",
  [
    "katrin albsteiger mdb",
    "katrin albsteiger schwanger",
    "katrin albsteiger twitter",
    "katrin albsteiger wahlkreis",
    "katrin albsteiger facebook",
    "katrin albsteiger kontakt",
    "katrin albsteiger bundestag",
    "katrin albsteiger instagram",
    "katrin albsteiger hochzeit",
    "katrin albsteiger handball",
    "katrin albsteiger homepage",
    "katrin albsteiger youtube",
    "katrin albsteiger swu",
    "katrin albsteiger brille",
    "katrin albsteiger spiegel",
    "katrin albsteiger wowereit",
    "katrin albsteiger anne will",
    "katrin albsteiger mann"
  ],
  [
    "","","","","","","","","",
    "","","","","","","","",""
  ],
  [],
  {
    "google:clientdata": {"bpc":false,"tlw":false},
    "google:suggestrelevance":
    [601,600,565,564,563,562,561,560,559,558,557,556,555,554,553,552,551,550],
    "google:suggesttype": [
      "QUERY","QUERY","QUERY","QUERY","QUERY","QUERY","QUERY","QUERY","QUERY",
      "QUERY","QUERY","QUERY","QUERY","QUERY","QUERY","QUERY","QUERY","QUERY"
    ],
    "google:verbatimrelevance": 1300
  }
]
```

3.2.2. Data storage

The data is stored in a MySQL Database that can be accessed only within the network of Technical University of Cologne.

The database consists of two tables for each search engine: “Suggestions” and “Terms”.

“Suggestions” keeps the “raw” information returned by query. It consists of seven columns: ID (primary key), queryterm (Name and Surname of politician), URL (for Google Search), date, client (User Agent), language, and raw_data (the raw text returned by search engine).

The second table, “Terms,” keeps the formatted data, here search engine suggestions are saved together with the position (autosuggestions of each search engine come up in an order that might be not uninteresting for a future analysis). The column “Search ID” is a reference to the first table, it indicates the row in which particular term is stored.

The databases were created using the SQL statements shown in Listings 3.3, 3.4, and 3.5.

Listing 3.3: SQL database generation script for the Google Search database

```
CREATE TABLE `wse_suggest`.`suggestions` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `queryterm` VARCHAR(255) NOT NULL,
  `date` DATETIME NOT NULL,
  `client` VARCHAR(45) NULL,
  `lang` VARCHAR(45) NULL,
  `url` VARCHAR(255) NOT NULL,
  `raw_data` TEXT NOT NULL,
  PRIMARY KEY (`id`)
);

CREATE TABLE `wse_suggest`.`terms` (
  `id` BIGINT NOT NULL AUTO_INCREMENT,
  `suggest_id` INT NOT NULL,
  `suggestterm` VARCHAR(255) NOT NULL,
  `position` INT NOT NULL,
  `score` INT NULL
  PRIMARY KEY (`id`)
);
```

Listing 3.4: SQL database generation scripts for the Bing database

```
CREATE TABLE `wse_suggest`.`suggestions_bing` (
  `id` BIGINT NOT NULL AUTO_INCREMENT,
  `queryterm` VARCHAR(255) NOT NULL,
  `date` DATETIME NOT NULL,
  `lang` CHAR(5) NULL,
  `raw_data` TEXT NOT NULL,
  PRIMARY KEY (`id`)
);

CREATE TABLE `wse_suggest`.`terms_bing` (
  `id` BIGINT NOT NULL AUTO_INCREMENT,
  `suggest_id` INT NOT NULL,
  `suggestterm` VARCHAR(255) NOT NULL,
  `position` INT NOT NULL,
  `score` INT NULL,
  PRIMARY KEY (`id`)
);
```

Listing 3.5: SQL database generation script for the DuckDuckGo database

```
CREATE TABLE `wse_suggest`.`suggestions_ddg` (
  `id` BIGINT NOT NULL AUTO_INCREMENT,
  `queryterm` VARCHAR(255) NOT NULL,
  `date` DATETIME NOT NULL,
  `raw_data` TEXT NOT NULL,
  PRIMARY KEY (`id`)
);

CREATE TABLE `wse_suggest`.`terms_ddg` (
  `id` BIGINT NOT NULL AUTO_INCREMENT,
  `suggest_id` INT NOT NULL,
  `suggestterm` VARCHAR(255) NOT NULL,
  `position` INT NOT NULL,
  `score` INT NULL,
  PRIMARY KEY (`id`)
);
```

3.3. Data preparation

The data used for analyses was aggregated using all three of the search engine database using the SQL query shown in Listing 3.6. This data was then exported to a CSV file for use in the analytics.

Listing 3.6: Query used for aggregating autocomplete results from the database

```
(
  SELECT DISTINCT suggestterm FROM terms AS term
  LEFT JOIN suggestions AS suggestion
    ON suggestion.id = term.suggest_id
    WHERE suggestion.lang = 'de'
)
UNION
(
  SELECT DISTINCT suggestterm FROM terms_bing AS term
  LEFT JOIN suggestions_bing AS suggestion
    ON suggestion.id = term.suggest_id
    WHERE suggestion.lang = 'de'
)
UNION
(
  SELECT DISTINCT suggestterm FROM terms_ddg as term
  LEFT JOIN suggestions_ddg AS suggestion
    ON suggestion.id = term.suggest_id
);
```

This script allowed us to save all unique auto-complete results from the three different search engines. The output file contained 10666 unique terms, which will be prepared for the further analysis.

Data preprocessing is an important step that insures that the data is fit for future analysis. In the case of this work, the preparation process faced two main problem areas. The first area is irrelevant search terms. The second is a more complicated text analysis problem: auto-complete suggestions are not always presented in a correct word form that can be used for text mining. For example, adjectives can be used in comparative form, such as "schlanker", which will not be matched correctly by the text mining algorithms as it is not the root word "schlank." Contractions are also present in the auto-complete data, with words such as "op" for "Operation." Performing text mining algorithms without preparing and cleaning the data would significantly reduce the quality of results.

The first data preparation task was to clean data from irrelevant search terms. The most common irrelevancy was a suggestion for a search for other people with the same name. For example, searches for the CSU politician Albert Rupprecht contained such suggestions as "albert ruprecht schauspieler" [7]. Suggestions for Alexander Ulrich of the party Die Linke contained, for example, "alexander ullrich architekt" [8]. In order to combat this, terms that identify occupation that the politicians do not have, such as actor, dentist or lawyer, were cleaned from the dataset due to being irrelevant to the analysis. This type of irrelevancy appeared only for the less popular politicians with more common names, and was not found in the data for parties leaders.

Another similar problem was encountered with the suggestions for people with similar surnames. For example, the search suggestions for Andreas Lenz included: "andreas lenzhofer", "andreas lenzing", "andreas lenzinger". Since these suggestions are irrelevant, they were also deleted from the dataset.

These tasks can be only approached manually due to the text mining algorithms having no context or ability to identify such irrelevant data automatically.

After cleaning the dataset of irrelevant terms its size decreased by 37% to 6705 entries.

The next problem is a usage of different word forms. This needed a more linguistic-based approach that can remove inflectional endings and return the dictionary form of a word, such as lemmatisation. Lemmatisation is the task of grouping together word forms that belong to the same inflectional morphological paradigm and assigning to each paradigm

[7] Schauspieler is the German word for an actor.
[8] Architekt is the German term for an architect.

its corresponding canonical form, called a lemma (Gesmundo and Samardžić 2012). For verbs, this base form is identified with the infinitive, and for most other words with a form without inflectional affixes. The noun “tables”, for instance, is paired with “table”, and the adjective “better” with “good”.

Normal transformation from a word to a lemma requires four following steps:

1. Remove a suffix of length $N1$
2. Add a new lemma suffix, $L1$
3. Remove a prefix of length $N2$
4. Add a new lemma prefix, $L2$

The tuple $t = (N1, L1, N2, L2)$ then defines this lemma transformation. Each tuple is represented with a label that includes these 4 parameters. For example, the transformation of the word “redoing” into its lemma can be encoded by the label (3, 0, 2, 0). This label can be used to describe transformation of earch word, that needs a removal of three-letter suffix, two letter prefix and no additional adding of suffix or prefix. The same label applies to any other transformation which requires only removing the first two and last three characters of the word string (Gesmundo and Samardžić 2012).

Lemmatisation is one of two methods that can be used to reduce inflectional forms to a common base form. The second is stemming, which is much more common in text mining than lemmatisation. Stemming is “a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes” (Manning, Raghavan and Schütze 2008). As example, for the word “saw” stemming will return just “s”, while lemmatisation will return “see”. Lemmatisation is a more complex method, because it requires determining the part of speech of a word, and applying different normalization rules for each part of speech.

Stemming is commonly used in information retrieval systems as a rudimentary device to overcome the vocabulary mismatch problem. For example, a text mentioning “puppy” is probably closely related to a text often mentioning “puppies”. It is therefore more useful for comparing the similarity of large bodies of text. It would not be useful for this particular research work, as the main aim is to group stand-alone words into semantic groups that are not yet determined.

The number of existing lemmatisation implementations is quite limited, especially for any language other than English. The Python text analysis module `pattern.de` was chosen for this project. `pattern.de` is maintained by CLiPS (Computational Linguistics and Psycholinguistics), a research center associated with the Linguistics department of the Faculty of Arts at the University of Antwerp (Linguistics and Psycholinguistics 2017a).

pattern.de contains a built in parsing function, that among other functions, provides lemmatisation. This function annotates each word with its base form (Schneider and Volk 1998). The parser is built on Gerold Schneider and Martin Volk's German language model (Schneider and Volk 1998).

Lemmatisation is an important task, since it improves the results of the following steps of text processing. The code used to perform lemmatisation is shown in Listing A.3 on page 81.

3.4. Text categorisation

3.4.1. Vector transformation

Clustering algorithms can not interpret words directly and divide them into groups according to their meanings without previous preparations and a method of interpreting their respective and relative meanings. The k-Means algorithm measures the distances between cluster centroids, and so a way to transform words into vectors was chosen. This allowed the coordinates, and therefore "distances" between clusters, to be calculated.

The algorithm for this is called word2vec. Word2vec was created by a team of researchers led by Tomas Mikolov at Google. This tool provides an efficient computing of vector representations of words. These representations can be subsequently used in many natural language processing applications. The model implementation was originally written in C and made available under an open-source license. Gensim provides a Python reimplementation of word2vec, which was used in this research.

The algorithm takes a text corpus as an input, constructs a vocabulary from the training text data, and then learns vector representation of words. For each unique word in the text, the word representation vector is collected from the Gensim Python library (Mikolov et al. 2013). Each word is represented by an 300-dimensional vector. 300 dimensions were chosen due to having the best accuracy, according to the research of Tomas Mikolov (Mikolov 2013). The output of algorithm is a numpy array.

For example, the vector representation of the word "Familie" would look like the array of text shown in Listing 3.7.

Listing 3.7: Vector representation of the word "Familie" using word2vec

```
[
    0.094698265194893;  0.033901035785675;
    0.166599497199059;  0.007252652663738;
   -0.259661853313446;  0.02049957588315;
   -0.12736763060929;  -0.04443240165710 5;
   -0.123259238898754;  0.111400775611401;
    ...
```

]

In the 300-dimensional vector space linear algebra can be used to exploit the encoded similarity. For example, using the last name of the current prime minister of Canada, and substituting “Canada” for “Germany”, it can output the name of German chancellor by inference using the meanings of each word:

$$Trudeau - Canada + Germany = Merkel$$

These operations are only possible after training using a model first. In this work, a German word embedding model was used. This model was trained on a corpus from Wikipedia using skip-gram as the training algorithm with hierarchical softmax, and ignoring all words with total frequency lower than 50. The size of the resulting model was 651,219,519 words, which resulted in a model size of 720 MB.

Without lemmatisation the vector model can interpret 3053 words. After lemmatisation this grew to 3240, bringing a 6% improvement. The code used to transform the data is shown in Listing A.5 on page 83. Most of words that were not recognised by the vector model were abbreviations and names of companies. The approach of these problematic cases will be discussed later in the chapter.

3.4.2. Choice of categories

As there were no predefined categories for this research, they needed to be chosen. The first exploratory method to identify the number and items of categories was k-Means clustering.

Determining the number of clusters is a task for the user with the k-means algorithms. The correct choice of the number of clusters is often ambiguous; the interpretations depend on the scale of the distribution of points and the possibility of interpretation. It is important to mention, that increasing the number of k without penalty will always reduce the amount of error in the results of clustering. In the extreme case the clustering will perform with zero error if each case is considered its own cluster (Omatu et al. 2015). If an appropriate value of k is not apparent from prior knowledge of the properties of the data set, it must be chosen somehow.

There is no absolute scheme with which to measure clusterings, but there are a variety of evaluation measures from diverse areas such as theoretical statistics, machine vision and web-page clustering, which are applicable. The evaluation result should define a (numerical) measure indicating the value of the clustering. The resulting value should either be easy to interpret or otherwise be illustrated with respect to its range and effects, in order to facilitate the evaluation interpretation. The chosen method should be defined without a bias towards a specific number and size of clusters and perform well in a high dimensional space.

One method to validate the number of clusters is the elbow method. The basic idea behind partitioning methods, such as k-means clustering, is to define clusters so that the total within-cluster variation (or total within-cluster sum of square) is minimized. The elbow method looks at the percentage of variance explained as a function of the number of clusters. Algorithm computes clustering for different number of k (by varying k from 1 to 20), for each k, it calculates the total within-cluster variation and plots the curve. he location of a bend in the plot is generally considered as an indicator of the appropriate number of clusters.

The elbow method was used to generate two graphs: the average within-cluster sum of squares, and the percentage of variance. These graphs are shown in Figures 3.1 and 3.2 on page 37, respectively.

After analysing the an angle in the graphs, it is apparent that the best number of clusters is between 2 and 3. The code used to generate the graphs is shown in Listing A.6 on page 83

Since elbow method cannot be unambiguously identified, two additional different scoring methods were chosen to confirm the chosen the number of clusters: the Silhouette and Calinski-Harabaz scores.

The Silhouette score is a measure of how similar a data point is to its own cluster, when comparing to other clusters. The silhouette score ranges from -1.0 to 1.0; a high value shows that the object is well classified to its own cluster and poorly matched to neighbouring clusters (Rousseeuw 1987). The problem of the score is that it is generally higher for convex clusters than other cluster concepts, such as density based clusters like DB-SCAN (Ojeda et al. 2014). That is why, for k-means clustering, the expectation of the score should be not that high.

Calinski-Harabaz index evaluates the cluster validity based on the average between- and within-cluster sum of squares (Liu et al. 2010). It is an internal clustering criterion, which means that the proper way to use it is to compare clustering solutions obtained using the same data. It is a dimensionless metric; there is no acceptable cut-off value. The higher the score, the better the separation between clusters. The results of the score generation for these values are shown in Table 3.1.

Both scores indicate that best number of clusters is three, which correspond to results of previous analysis. The clusters, however, need to be checked for their ability to be interpreted. The number of clusters might need to be changed to a variant with a less separation power, if the interpretation is more meaningful.

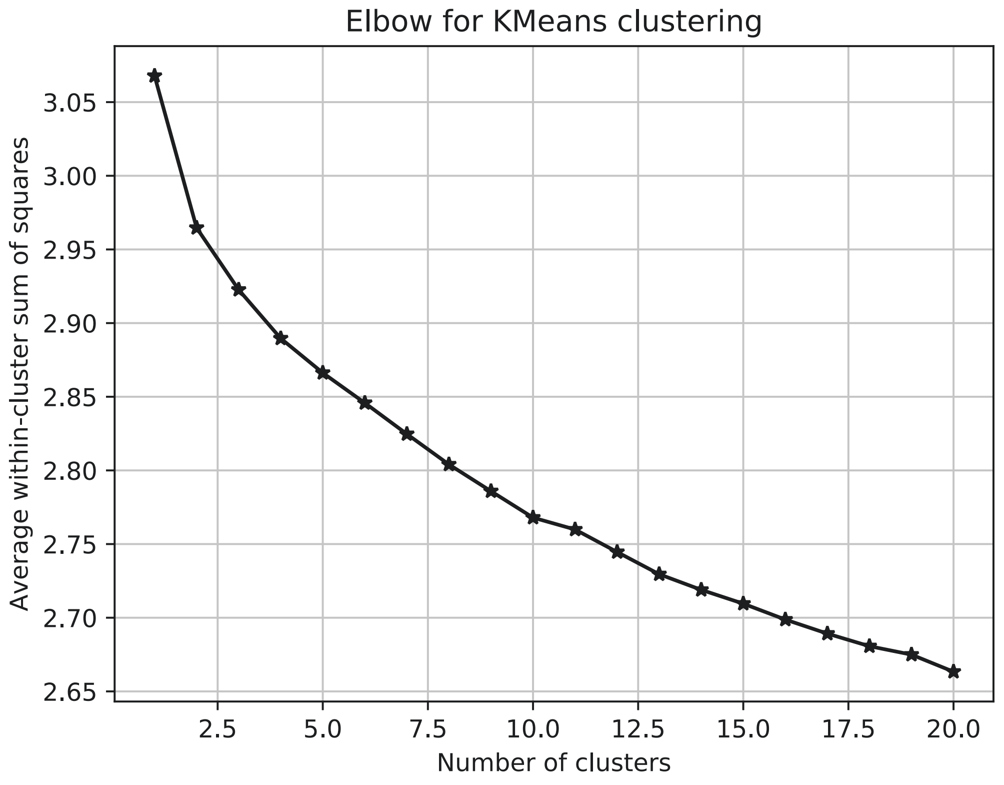

Figure 3.1.: Average within-cluster sum of squares

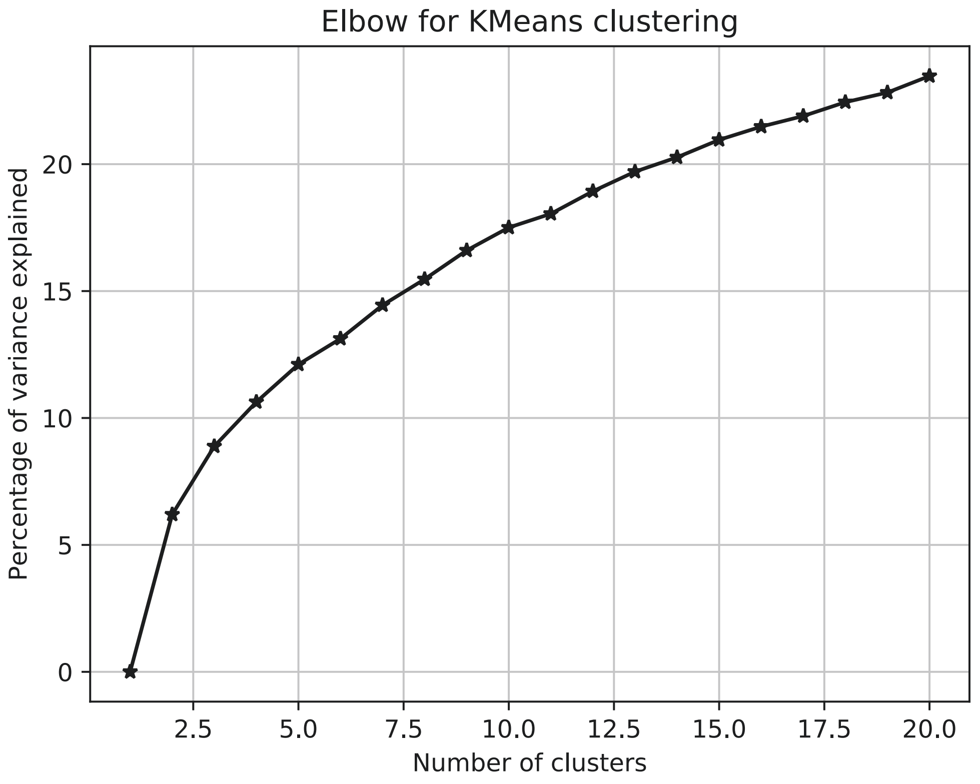

Figure 3.2.: Percentage of variance explained

Number of clusters	**Silhouette score**	**Calinski-Harabaz score**
2	0.05413	157.7990
3	0.0541	157.8038
4	0.0293	128.3042
5	0.0332	111.4364
6	0.0278	99.4350

Table 3.1.: Clustering quality evaluation using Silhouette and Calinski-Harabaz scores

It appears that using three clusters, as suggested by the k-means algorithm and the analytics, separates the search terms into the following groups:

1. Political and economical terms
2. Geographical locations in Germany
3. Personal information about politicians and their family and various search terms

An excerpt of data assigned to the three clusters is shown in Table 3.2.

Cluster number	**Example search terms**		
0	Gladbeck	Offenburg	Norddeich
1	Gesundheitsausschuss	Bundestagswahl	Integrationsbeauftragte
2	Kostueme	Lustig	Wife

Table 3.2.: Example data separated into three clusters

A four cluster solution seemed to not improve the effectiveness, as it splits the group with geographical locations in Germany into two groups. Example data for four clusters is shown in Table 3.3.

Cluster number	**Example search terms**		
0	Wust	Foto	Schmuck
1	Umweltbundesamt	Tourismus	Altenpflege
2	Heuchlingen	Girkhausen	Heiligenstadt
3	Wetzlar	Ennigerloh	Waltrop

Table 3.3.: Example data separated into three clusters

No logical grouping was found in with a five cluster solution. Example data for five clusters is shown in Table 3.4.

The code used to perform clustering is shown in Listing A.8 on page 87.

Cluster number	Example search terms		
0	Friseur	Bauernregel	Beruf
1	Aurich	Werdohl	Iserlohn
2	Fotograf	Romania	Lebenslauf
3	Tattoos	Mittelberg	Egesheim
4	Kanzleramt	Menschenrechte	Steuern

Table 3.4.: Example data separated into three clusters

The three cluster solution seems to be the best and most easily explainable, though the group with German geographical positions doesn't seem to be helpful in explanation what exactly users are searching for. Searches for geographical position inside Germany could be for the place of birth of the politician, their electoral district, or connection or reaction of the politician to a particular event that happened in the location. Since this cluster itself doesn't bring new information but influences the clustering a lot, the decision to exclude geographical locations from the dataset for clustering was made.

After excluding geographical locations in Germany, the word list was tested again to identify if the three-cluster solution is still preferable, or if excluding the group of data influenced the set and caused a different number of clusters to be superior. The results of this test are shown in Table 3.5.

Number of clusters	Silhouette score	Calinski-Harabaz score
2	0.0475	88.2812
3	0.0374	69.5076
4	0.0113	60.7197

Table 3.5.: Clustering quality evaluation using Silhouette and Calinski-Harabaz scores after removing one group

The two cluster solution scored the best after excluding the geographical data. The lower values of the Silhouette and Calinski-Harabaz scores (compared with the results of previous test) indicate that groups of words with geographical positions were well-separated from the two other groups, and the exclusion of it brought particular bias into clustering.

The categories of the three cluster solution after geographical data exclusion are hard to explain. While the two cluster solution brings the same explainable results as in the previous test (i.e. the first cluster describes political and economical activity, while the second cluster describes personal information), the three cluster solution groups don't seem to be explainable, and words in the same cluster appear to be unrelated. An excerpt of data from the three clusters is shown in Table 3.6.

Cluster number	**Example search terms**		
0	Transfermarkt	Odessa	Pictures
1	Abgeordnete	Haushalt	Fraktion
2	Villa	Aufgaben	Marathon

Table 3.6.: Example data separated into three clusters without German geographical locations

To summarise the results of exploratory search for the best number of autosuggestions' groups, the best tests score had clustering solution with three groups. One of them, geographical locations inside Germany, was hard to interpret. Because of this, the best solution with the greatest score results and clearest grouping is a two-cluster solution with this data removed. This solution will be used in further analysis.

The final clustering groups are:

1. Political and economical terms
2. Personal information about politicians and their family and various search terms

It is important to note that no matter how search terms are divided into clusters, groups will never be separated from each other perfectly. Any division of data is still a simplification that will bring a bias.

These groups were later used as predefined categories for supervised learning.

3.4.3. Other considered approaches

Clustering was not the only way considered to obtain word categories: various methods were tested prior that were not considered possible to use or didn't bring satisfactory results. The general approach that was considered was a usage of pre-existing categories for words. This approach needed a basis category structure. Two sources were considered: Wikipedia, and BabelNet.[9]

The Wikipedia categories system at first sight can be considered a perfect solution, especially the MediaWiki action API. This is web service from Wikipedia that provides convenient access to Wikipedia features, data, and metadata over HTTP. The major problem with the API was that Wikipedia assigns many of different categories to each page. For example, the page "Donald Trump" (a search term extracted from the dataset) is assigned

[9] BabelNet is a multilingual lexicalised semantic network and ontology developed at the Linguistic Computing Laboratory in the Department of Computer Science of the Sapienza University of Rome (Navigli and Ponzetto 2012)

to more than 30 different categories, some of which are: “1946 births”, “Living people”, “20th-century American businesspeople”, “21st-century American businesspeople”, and “21st-century American politicians” (Wikipedia 2017). Most of them are irrelevant for this research, but there are no exact rules in which order categories come up on the page. Which means, category, that would be most important for this research (in case of Trump such category would be “President”) can appear at any position of the list of assigned categories. It can lead to the situation where the words “Trump” and “Putin”, which should to be for the aim of research in the same category, will be divided. This is because one may be assigned to the category “1946 births” first, and another to the category “20th century politicians”. There is no algorithm or process which could properly identify the correct categories relevant to this research, aside from manually selecting the most appropriate category for each politician, costing an inordinate amount of time.

Another problem with Wikipedia is its inability to categorise any terms other than objects, places, and events; there are no pages or categories for many words, as it is not a dictionary.

The issue with Babel is similar: there is no way to predict which categories will be relevant to the research. For example, the list of categories for the word “zwillinge”[10] includes the following: “Astrologie”,[11] “Tierkreiszeichen”,[12] “Zoologie”,[13] “Genetik”,[14] “Pränatalmedizin”.[15] The first two outputs seem to not be relevant to this research, but it is impossible to predict which of the defined categories will be relevant.

3.4.4. Approach for problematic cases

Not all words were identified by lemmatisation and the vector model. All words identified as unused by the model algorithm were analysed to fix the problem and add them to either the dataset prior to preparation activities or directly into the final clustered groups.

One challenging part of data preparation is the categorisation of combinations of words, for example business names such as “Aargauische Kantonalbank” or “Süddeutsche Zeitung”. Combinations can not be identified by the lemmatisation algorithm, nor by the vector model. To fit them in the vector model, a word that includes the main meaning of the combination was manually chosen. For example: “erste Ehefrau” $\rightarrow$ “Ehefrau,”[16] “AFD

[10] Zwillinge is the German word for twins.
[11] Astrologie is the German term for astrology.
[12] Tierkreiszeichen is the German term for astrological sign.
[13] Zoologie is the German term for Zoology.
[14] Genetik is the German term for genetic.
[15] Pränatalmedizin is the German term for prenatal medicine.
[16] Ehefrau is the German term for wife. “Erste Ehefrau” means “first wife.”

Hessen" → "AFD,"[17] "Büro Bonn" → "büro,"[18] and "Fasching 2017" → "Fasching." [19] However this approach didn't significantly change the overall number of words in the dataset, since most of these words were already in the model as standalone terms.

Another group of words that can be not directly identified by the vector model were abbreviations. Some are commonly used, such as "ZDF," while others are colloquialisms, like "sz" for the Süddetsche Zeitung. For these cases a similar approach was used: reduce the combination of the words in an abbreviation to the main word, such as "sz" → "Süddetsche Zeitung" → "Zeitung."[20]

The last of the problematic groups contains names of companies, for example "Rfr", "Tesla", "Dvgw." There 522 names of companies in the dataset that were not identified by the algorithm. These cases were assign to their respective groups manually after performing text categorisation.

3.4.5. Text classification and evaluation of results

Previously clustered data now will be used as a training set to perform a classification of the rest of the data set. For this purpose Support vector machines (SVMs) will be used.

As with other classifiers, SVM take two arrays as input: the first is an array of size includes the training samples, while the second one is an array of class labels (clusters). The training data can be used to predict new values after being fitted.

Evaluation will be performed with the help of cross-validation metrics. Normally, part of the clustered data is kept separate from the training set to be used for evaluation, as a so-called "validation set." Training proceeds with the training set, after which the evaluation is done on the validation set. When the experiment appears to be successful, the final evaluation can be done on the test set. This is done to avoid overfitting of the model. The problem with this approach is that the number of samples which can be used for validating the model would drastically reduce the amount of data points with which it can learn. Cross-validation metrics are important to use to avoid this. The basic approach of cross validation, the so called k-fold CV, is as follows: the training set is split into k smaller sets, a model is trained using $k-1$ of the folds as training data. The model is validated on the remaining part of the data, and then the procedure is repeated for each k in the model. This approach allows to use the whole training set for both learning and validation.

[17] AFD is a German political party.
[18] Büro is the German term for office.
[19] Fasching is a German regional festival.
[20] Zeitung is the German term for newspaper

The Python code used to perform text classification and evaluation is shown in Listing A.7 on page 85.

The SVM algorithm predicted the categories of 2240 objects. To calculate an accuracy of this output the mean of the cross-validation scores for each object is taken. To quantify the amount of variation two standard deviations are taken. The accuracy worked out to be $91.0 \pm 4\%$. This means that the accuracy of prediction lies between 87% and 95%.

3.5. Method summary

To sum up the results of text categorisation, an overview of the results is presented. The original dataset included 10666 cases. After deleting irrelevant data, it was down to 6803 entries, from which 3762 were unique. 3240 words were identified by the vector-transformation algorithm. 30% of the data was clustered into two groups. This data was then used as a training set for the classification algorithm, which predicted group relationship from rest of object in the dataset with a $91 \pm 4\%$ accuracy. After this, 537 names of companies and abbreviations were added manually to the dataset. The final text categorisation model includes 3546 cases.

This model will be used to identify clusters of autosuggestion terms in the dataset. The Python code used to calculate the number of autosuggestions in each group for each politician is shown in Listing A.9 on page 88.

4. Analysis

4.1. Overview

This chapter describes the analyses that were performed on the dataset. Firstly, the sociodemographic characteristics will be described, followed by a description of the differences between the search engines. Multivariate analyses will then be introduced, such as correlation and regression analysis. Then, the categorical cluster analysis will be described. Lastly, an interpretation of the results is presented.

4.2. Descriptive statistics

In this section the basic features of the data will be described. The following subsections provide summaries about the sample. Together with the graphical analysis, they form the basis of the quantitative analysis of the data. The following graphics were created with the Matplotlib plotting library. The correlation and regression analysis were performed in SPSS.

4.2.1. Describing the dataset

The dataset for the analysis consisted of following information: the name of each politician, their year of birth, age (in 2017), gender, state, party, and term score for each search category defined in a previous chapter. The term score shows how many unique autosuggest terms belong to each. Only the unique terms were chosen to avoid the bias of time; if each term was added to the dataset the results would be dependent on time, and this would need to be taken into account.

There are 398 male and 232 female politicians in the Bundestag in 2017. This corresponds to an approximate gender ratio of 7 : 4, male to female. Figure 4.1 on page 45 illustrates the gender proportions.

Members of the Bundestag have to be minimum 18 years old, as required by German law (Parlamentarischer Rat 1949). In the dataset, the range of ages lies between 28 and 82 years. The average age is 54 years, and the median age is 55 years. Figure 4.2 on page 45 represents the distribution of age.

Half of the Members of the Bundestag are elected from the parties' candidate Members in each state in such a way as to achieve proportional representation for the total Bundestag. This means all sixteen federal states are represented in the dataset. Most members of the

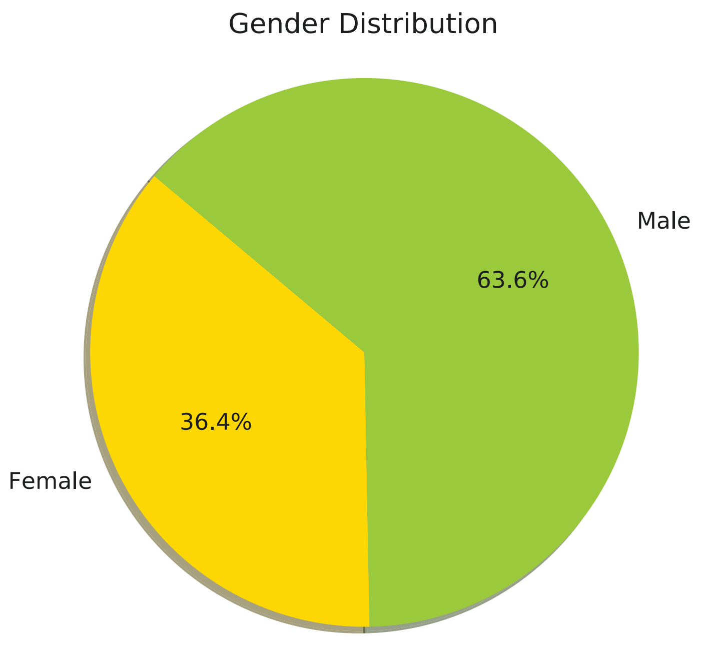

Figure 4.1.: Gender distribution in the dataset

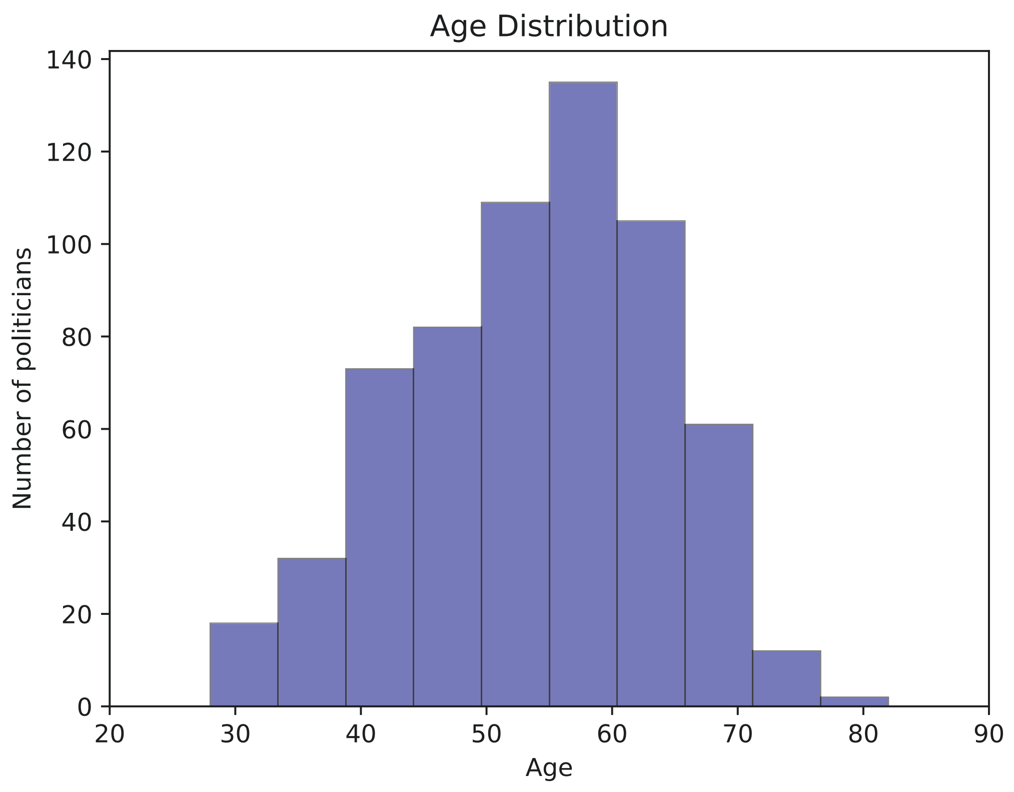

Figure 4.2.: Age distribution in the dataset

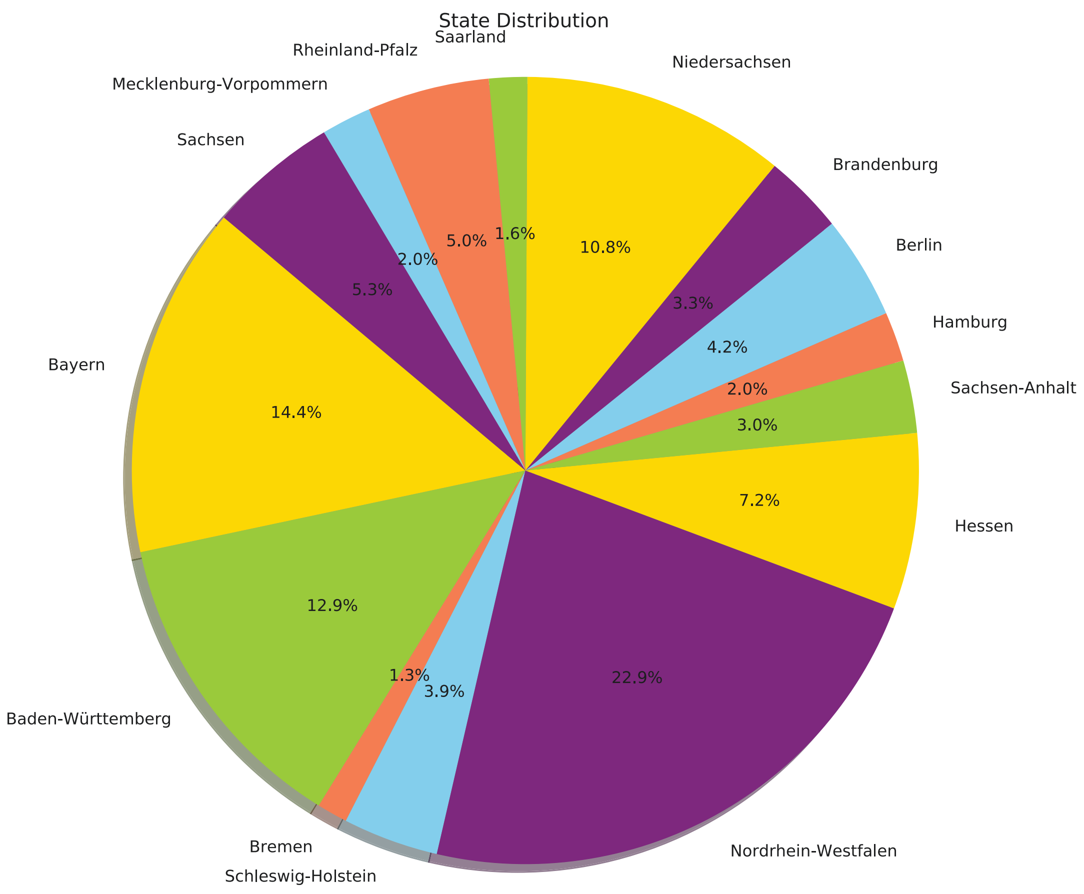

Figure 4.3.: State distribution in the dataset

Bundestag originate from North Rhine-Westphalia (almost 23%), while the least amount of members of the Bundestag originate from Bremen (1.3%). Figure 4.3 on page 46 represents the distribution of states.

The distribution of seats in Bundestag between parties is well-known information. The Christian Democratic Union holds 40% of seats, which makes it the most represented party in the Bundestag. The least represented party in the Bundestag is the Christian Social Union in Bavaria with almost 9%.

Figure 4.4 on page 47 shows the party distribution in the dataset.

There are two text categories in the dataset: personal information, and political and economical terms. 605 of the 630 politicians have at least one search term in both categories. Personal information is less common than the political and economic. Figure 4.5

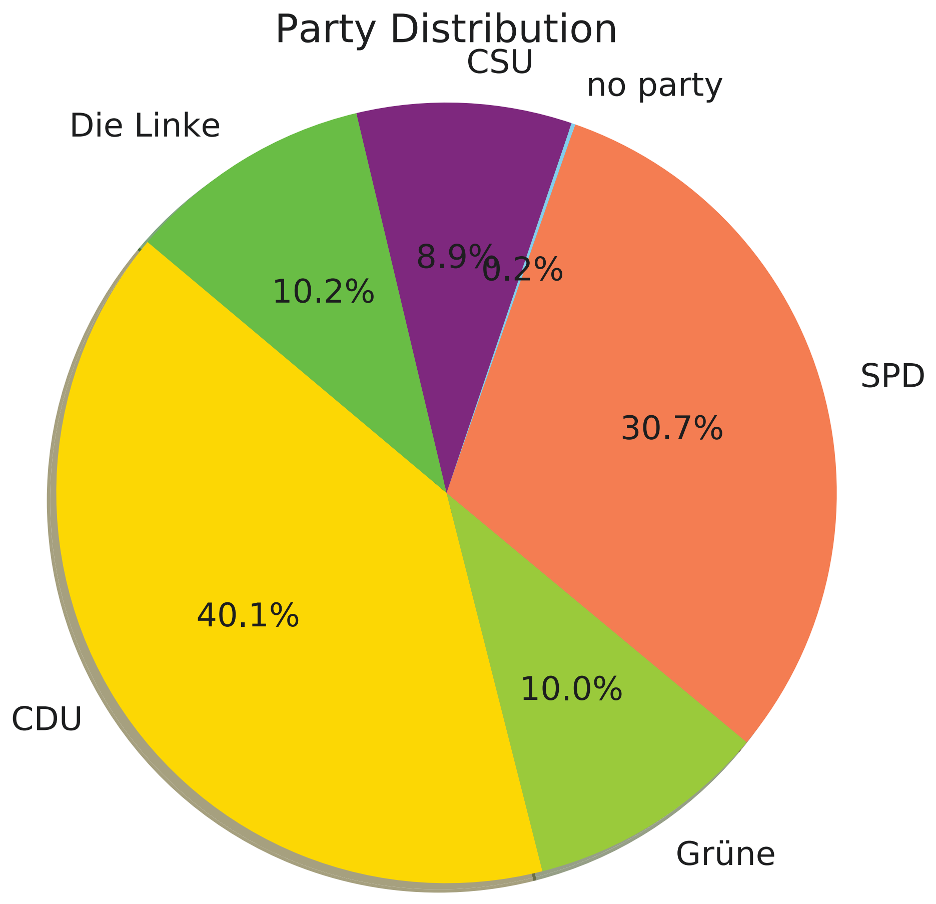

Figure 4.4.: Party distribution in the dataset

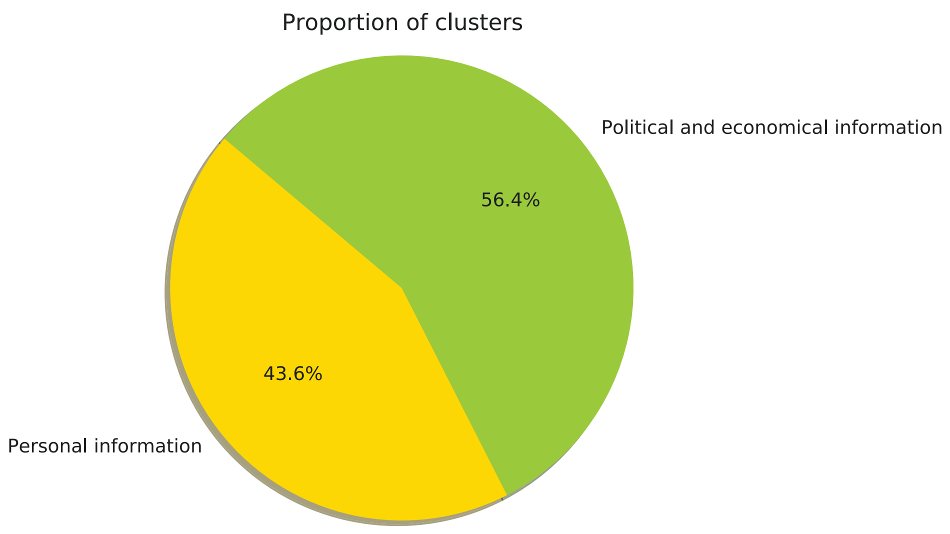

Figure 4.5.: Proportion of clusters

on page 48 illustrates the numerical proportion of categories. The average number for political and economical terms is 9, and for personal information is 7.

The number of recognised unique search terms per politician varies between 1, for example for politicians Pia-Beate Zimmermann, Bettina Bähr-Losse and Klaus-Peter Flosbach, and 54, for example for Erika Steinbach. For Mark André Helfrich, Mathias Edwin Höschel, Hans-Ulrich Krüger, Elisabeth Charlotte Motschmann, Alois Georg Josef Rainer, Ursula Schauws, Volker Michael Ullrich, and Pia-Beate Zimmermann, all autosuggest terms were eliminated during the preparation phase.

The average number of unique search terms is 16, and the median number of unique search terms is 14. Figure 4.6 on page 49 represents the number of unique search terms per politician.

To summarise, the dataset includes information about 630 members of the Bundestag. The gender ratio is 3 : 2 male to female, the average age is 54 years, and the age range is from 28 to 82 years. All states are represented in the Bundestag, but most members are from North Rhine-Westphalia. 96% of politicians have at least one word in both search terms categories, and the average number of terms is 16.

4.2.2. Describing search engines

The search terms in the dataset were collected from three different search engines. It is important to note the differences in the autosuggestion results from each of them.

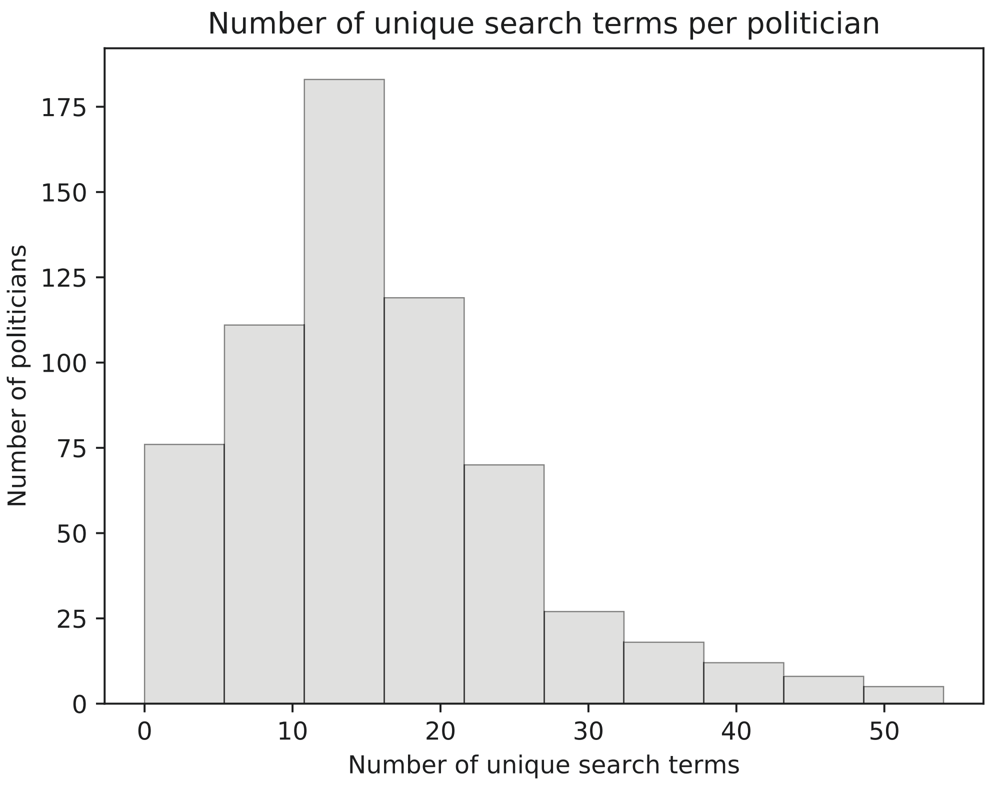

Figure 4.6.: Proportion of unique search terms per politician

The first difference is the number of unique search terms, as shown in Table 4.1 on page 50.

Search Engine	Number of unique terms
Google Search	8649
Bing	2882
DuckDuckGo	2264

Table 4.1.: Number of unique search terms from each search engine

Not all of the autosuggestions are unique to each search engine, and some appear in results for at least one other. Table 4.2 on page 50 shows the amount of search engine's autosuggested terms that overlap. Table 4.3 on page 50 shows the proportion of overlapping results for each search engine, compared to the number of unique search terms of the other engines. The code used to perform calculations in Listing A.10 on page 90.

	Google Search	Bing	DuckDuckGo
Google Search	-	1223	153
Bing	1223	-	137
DuckDuckGo	153	137	-

Table 4.2.: Overlapping search terms

	Relative to the overlap with:		
	Google Search	**Bing**	**DuckDuckGo**
Google Search	-	42.44%	6.76%
Bing	14.14%	-	6.054%
DuckDuckGo	1.77%	4.75%	-

Table 4.3.: Proportion of overlapping search terms

Google Search and Bing overlap the most with 42.44%, meaning that 42.44% of Google Search autosuggestion terms were also shown in Bing. However, only 14.14% of Bing autosuggestion terms were also shown in Google Search. But as even the biggest overlap is still less than 50%, showing that using three different search engines and their autosuggestions allows to get bigger dataset with more unique search terms.

4.3. Correlation and dependence

Correlation analysis is used to quantify the degree of relationship between two variables (between an independent and a dependent variable or between two independent variables). Regression analysis is a related technique to estimate the relationships among variables. This helps to understand how the value of the dependent variable changes when one of the independent variables is varied, while the others are not.

To perform logistics regression on all variables, the variable “Age” was transformed into the categorical variable “Age group”. The following groups were introduced in this variable: 20–40 years, 40–50 years, 50–60 years, 60–70 years, and 70–90 years. The first and last group are not following pattern of 10 years gap, as in a group of 20–30 years there would be only 4 members, and in a group of 80–90 years would be only 1 member. The division into very small groups can damage results for correlation analysis and is inappropriate for logistics regression, since it violates the assumption that each group will have at least 25 members (Backhaus et al. 2015). Therefore, these groups are joined with their neighbouring group.

Since there is only one member in the group Fraktionlos [21] group in the “Party” category, the group was excluded from both correlation analysis and logistics regression, for the same reasons.

4.3.1. Correlation analysis

The search for strong correlations in the dataset helps to not only describe and understand underlying relationships between variables, but also helps to avoid mistakes in the cluster analysis. Highly correlated variables (> 0.8) tends to be overrepresented in the clustering solution, and are therefore problematic (Mooi and Sarstedt 2011).

To perform correlation analysis for categorical variables Pearson’s χ^2 test was used. This is also called the χ^2 test of association. This test shows if there is any statistically significant association between variables. Phi coefficient and Cramer’s V, known together as Symmetric Measures, are both used to measure the strength of association, if one is found.

The script used to perform correlation analysis in SPSS is shown in Listing 4.1.

Listing 4.1: SPSS script performing correlation analysis

```
CROSSTABS
/TABLES=AgeGroup BY Party Bundesland Gender
/FORMAT=AVALUE TABLES
/STATISTICS=CHISQ CC PHI LAMBDA
```

[21] Fraktionlos is the German term for no party.

```
/CELLS=COUNT
/COUNT ROUND CELL.

CROSSTABS
/TABLES=Gender BY Party Bundesland Gender
/FORMAT=AVALUE TABLES
/STATISTICS=CHISQ CC PHI LAMBDA
/CELLS=COUNT
/COUNT ROUND CELL.

CROSSTABS
/TABLES=Gender BY State Bundesland Gender
/FORMAT=AVALUE TABLES
/STATISTICS=CHISQ CC PHI LAMBDA
/CELLS=COUNT
/COUNT ROUND CELL.

/TABLES=Gender BY Age Bundesland Gender
/FORMAT=AVALUE TABLES
/STATISTICS=CHISQ CC PHI LAMBDA
/CELLS=COUNT
/COUNT ROUND CELL.
```

Table 4.4 on page 52 shows the χ^2 Tests and Symmetric Measures for the pair of Age Group and Party.

A value for Pearson $\chi^2 = 60.39$, $p = 0.000$ shows that there is a statistically significant association between Age Group and Party. However, Phi and Cramer's V both show that the strength of association between the variables is very weak. Weak correlations don't have any high impact on clustering and show a lower likelihood of there being a relationship between variables.

Test	**Value**	**Sig.**
Pearson χ^2	60.39	0.000
Phi	0.31	-
Cramer's V	0.15	-

Table 4.4.: Correlation analysis for Age Group and Party

Table 4.5 and on page 53 shows the Chi-Square Tests and Symmetric Measures for the pair of Gender and Party.

The value for Pearson $\chi^2 = 41.45$, $p = 0.000$ means that there is a statistically significant association between Gender and Party. However, Phi and Cramer's V both show that the strength of association between these variables is also weak.

Table 4.6 on page 53 shows the Chi-Square Tests and Symmetric Measures for the pair of State and Party.

The value for Pearson $\chi^2 = 442.58$, $p = 0.000$ shows that there is a statistically significant

Test	Value	Sig.
Pearson χ^2	41.45	0.000
Phi	0.26	-
Cramer's V	0.26	-

Table 4.5.: Correlation analysis, Gender and Party

association between the two variables and, as Phi score shows, the correlation is high. Cramer's V shows a medium correlation score. It can therefore be determined here that there is a significant medium-strong correlation (Interior 2017).

Strong correlation of the variables Party and State can be explained, for example, with all members of CSU party being from Bavaria, as the CSU only operates in Bavaria.

Test	Value	Sig.
Pearson χ^2	442.58	0.000
Phi	0.84	-
Cramer's V	0.38	-

Table 4.6.: Correlation analysis, State and Party

Table 4.7 on page 53 shows the Chi-Square Tests and Symmetric Measures for the pair of State and Gender.

The value for Pearson $\chi^2 = 17.57$, $p = 0.288$ shows that there is no statistically significant association between the two variables.

Test	Value	Sig.
Pearson χ^2	17.57	0.288
Phi	0.17	-
Cramer's V	0.17	-

Table 4.7.: Correlation analysis, State and Gender

Table 4.8 on page 54 shows the Chi-Square Tests and Symmetric Measures for the pair of Age and Gender.

The value for Pearson $\chi^2 = 14.87$, $p = 0.005$ shows that there is a statistically significant association between the variables. However, as both tests of of the strength of association show, the correlation is very low.

Tables 4.9, 4.10, 4.11 and 4.12 show the results of correlation analysis with all variables paired with both autosuggestion text categories. There is no significant correla-

Test	Value	Sig.
Pearson χ^2	14.87	0.005
Phi	0.15	-
Cramer's V	0.15	-

Table 4.8.: Correlation analysis, Age and Gender

Cluster	Political and economical terms		Personal information	
Statistics	**Value**	**Sig.**	**Value**	**Sig.**
Pearson χ^2	35.32	0.406	16.89	0.660
Phi	0.23	-	0.16	-
Cramer's V	0.23	-	0.16	-

Table 4.9.: Correlation analysis, Autosuggestion categories and Gender

tion between both categories and gender and state, or between personal category and age. There is a significant but weak correlation between political category and age and party.

To summarise, there was only one pair of strongly correlated variables in the dataset: State and Party. One of them has to be excluded from clustering to avoid overrepresentation. Since the Party variable is more valuable for the interpretation of clustering, the State variable will be excluded.

All other variables in the dataset don't show strong correlation between each other, and can be safely used in further analyses.

4.3.2. Regression analysis

Regression analysis will be used to estimate the relationships among variables. Logistics regression was chosen for binary categorical variables (i.e. Gender), and multinomial logistic regression for non-binary categorical variables (i.e. Party, State, and Age Group).

The purpose of this analyses is to calculate if there is a strong relationship between socio-demographic information about politicians and the search term categories. To answer this question, several logistics regression models were built. Each model was built with one socio-demographic parameter (gender, age, state, party) as a dependent variable, and the search term categories as independent variables. Such a set up will show which socio-demographic parameters have strong relationships with which search term category (if

Cluster	Political and economical terms		Personal information	
Test	**Value**	**Sig.**	**Value**	**Sig.**
Pearson χ^2	200.45	0.000	96.10	0.106
Phi	0.56	-	0.39	-
Cramer's V	0.28	-	0.20	-

Table 4.10.: Correlation analysis, Autosuggestion categories and Age group

Cluster	Political and economical terms		Personal information	
Test	**Value**	**Sig.**	**Value**	**Sig.**
Pearson χ^2	490.34	0.727	331.23	0.104
Phi	0.88	-	0.73	-
Cramer's V	0.23	-	0.19	-

Table 4.11.: Correlation analysis, Autosuggestion categories and State

any).

Before performing logistics regression, it is important to check if the data can actually be analysed using this method. The following assumptions should not be violated in each model: (Backhaus et al. 2015)

1. The dependent variable should be measured in discrete categories. The dependent variables Gender, State and Party satisfy this (e.g. in Gender, "male" and "female" are discrete categories), however the variable Age is an ordinal variable on a continuous scale. Therefore, in order to not violate this assumption, the Age variable was transformed to the Age Group variable, as described previously.

2. There should be no multicollinearity. As the previous section proves, there is no independent variables that are highly correlated with each other that break this assumption.

3. The number of cases should be at least 25 per group of dependent variable. This assumption is violated for two variables: the variable Party includes the group "Fraktionslos" with only one case in it, and the variable State includes seven groups that violate assumption. In the case of "Fraktionslos," the value was removed from the analysis.

The assumption of dataset size was violated, so the results of the logistics regression with the dependent variable State should be treated with caution.

After checking the assumptions for logistics regression, the code shown in Listing 4.2 was

Cluster	Political and economical terms		Personal information	
Test	**Value**	**Sig.**	**Value**	**Sig.**
Pearson χ^2	183.47	0.004	65.80	0.792
Phi	0.54	-	0.32	-
Cramer's V	0.27	-	0.16	-

Table 4.12.: Correlation analysis, Autosuggestion categories and Party

used to perform the analysis.

Listing 4.2: SPSS script performing regression analysis

```
LOGISTIC REGRESSION VARIABLES Gender
/METHOD=ENTER Cluster0 Cluster2
/PRINT=GOODFIT
/CRITERIA=PIN(0.05) POUT(0.10) ITERATE(20) CUT(0.5).
```

A logistic regression was performed to ascertain the relationship between gender and search terms categories. The logistic regression model was not statistically significant, with a result of $p = 0.082$. The model explained 1% (using Nagelkerke R^2) of the variance and correctly classified 63.5% of cases. The results are shown in Tables 4.13, 4.14, and 4.15. Only the cluster describing personal information is significant, and so only this cluster was further tested.

The results of the analyses using only this cluster are shown in Tables 4.16, 4.17, and 4.18. The logistic regression model was not statistically significant, with a result of $p = 0.082$. The cluster variable was also not statistically significant, with a result of $p = 0.082$. This leads to the conclusion that the effect of both text categories should be studied simultaneously.

To perform this, an interaction model was built. An interaction model is a model where the interpretation of the effect of one independent variable depends on the value of another independent variable and vice versa (Fisher 1992) The interactions analysis includes the exploration of differences in differences. “If the differences are not different then there is no interaction” (Digital Research and Education 2011). This means that the interaction model explores the relationship between dependent and independent variable with the expected effect of another independent variable.

The results of the interaction model are shown in Tables 4.19, 4.20, and 4.21. The logistic regression model was statistically significant, with a result of $p = 0.009$, as were two variable in the equation: Cluster Personal and Cluster Politics by Cluster Personal, with results of $p = 0.014$ and $p = 0.003$, respectively. The model explained 20% (using Nagelkerke R^2) of the variance and correctly classified 63.8% of cases. This shows that new model brings slightly improved results. The analysis of Odds Ratio (represented in

	χ^2	df	Sig.
Step	4.998	2	0.082
Block	4.998	2	0.082
Model	4.998	2	0.082

Table 4.13.: Omnibus Tests of Model Coefficients

Step	-2 Log likelihood	Cox and Snell R^2	Nagelkerke R^2
1	819.903	0.008	0.011

Table 4.14.: Model summary

Table 4.21 in the "Exp(B)" column) shows that the chance of successfully predicting the gender "Female" increases with the growth of the numbers in the personal information cluster, but the female gender is less likely to be successfully predicted with the increase of the politics cluster.

Multinomial logistic regression was performed for all other variables. The script used in SPSS for the process can be seen in Listing 4.3.

Listing 4.3: SPSS script performing multinomial regression analysis

```
NOMREG Party (BASE=LAST ORDER=ASCENDING) BY Cluster0 Cluster2
/CRITERIA CIN(95) DELTA(0) MXITER(100) MXSTEP(5) CHKSEP(20) LCONVERGE(0)
    PCONVERGE(0.000001)
SINGULAR(0.00000001)
/MODEL
/STEPWISE=PIN(.05) POUT(0.1) MINEFFECT(0) RULE(SINGLE) ENTRYMETHOD(LR) REMOVALMETHOD(LR)
/INTERCEPT=INCLUDE
/PRINT=CELLPROB CLASSTABLE PARAMETER SUMMARY LRT CPS STEP MFI.

NOMREG Bundesland (BASE=LAST ORDER=ASCENDING) BY Cluster0 Cluster2
/CRITERIA CIN(95) DELTA(0) MXITER(100) MXSTEP(5) CHKSEP(20) LCONVERGE(0)
    PCONVERGE(0.000001)
SINGULAR(0.00000001)
/MODEL
/STEPWISE=PIN(.05) POUT(0.1) MINEFFECT(0) RULE(SINGLE) ENTRYMETHOD(LR) REMOVALMETHOD(LR)
/INTERCEPT=INCLUDE
/PRINT=CLASSTABLE PARAMETER SUMMARY LRT CPS STEP MFI.
```

	B	S.E.	Wald	df	Sig.	Exp(B)
Cluster Politics	-0.026	0.014	3.356	1	0.067	0.974
Cluster Personal	0.074	0.029	6.284	1	0.012	1.077
Constant	0.267	0.180	2.182	1	0.140	1.305

Table 4.15.: Variables in the Equation

	Chi-square	df	Sig.
Step	4.998	2	0.082
Block	4.998	2	0.082
Model	4.998	2	0.082

Table 4.16.: Omnibus Tests of Model Coefficients - Cluster Personal

Step	-2 Log likelihood	Cox and Snell R^2	Nagelkerke R^2
1	819.903	0.008	0.011

Table 4.17.: Model summary - Cluster Personal

	B	S.E.	Wald	df	Sig.	Exp(B)
Cluster Politics	-0.026	0.014	3.356	1	0.067	0.974
Cluster Personal	0.074	0.029	6.284	1	0.012	1.077
Constant	0.267	0.180	2.182	1	0.140	1.305

Table 4.18.: Variables in the Equation - Cluster Personal

	χ^2	df	Sig.
Step	9.228	1	0.002
Block	9.228	1	0.002
Model	9.332	2	0.009

Table 4.19.: Omnibus Tests of Model Coefficients - Interaction model

Step	-2 Log likelihood	Cox and Snell R^2	Nagelkerke R^2
1	819.773	0.015	0.020

Table 4.20.: Model summary - Interaction model

	B	S.E.	Wald	df	Sig.	Exp(B)
Cluster Politics with Cluster Personal	-0.003	0.001	6.068	1	0.014	0.997
Cluster Personal	0.109	0.036	9.035	1	0.003	1.115
Constant	0.054	0.196	0.076	1	0.783	1.055

Table 4.21.: Variables in the Equation - Interaction model

Model	-2 Log Likelihood	χ^2	df	Sig.
Final	947.252	192.754	270	1.000

Table 4.22.: Model Fitting Information - Party

Test	Result
Cox and Snell R^2	0.264
Nagelkerke R^2	0.280
McFadden	0.108

Table 4.23.: Pseudo R-Square - Party

```
NOMREG AgeGap (BASE=LAST ORDER=ASCENDING) BY Cluster0 Cluster2
/CRITERIA CIN(95) DELTA(0) MXITER(100) MXSTEP(5) CHKSEP(20) LCONVERGE(0)
    PCONVERGE(0.000001)
SINGULAR(0.00000001)
/MODEL
/STEPWISE=PIN(.05) POUT(0.1) MINEFFECT(0) RULE(SINGLE) ENTRYMETHOD(LR) REMOVALMETHOD(LR)
/INTERCEPT=INCLUDE
/PRINT=CLASSTABLE PARAMETER SUMMARY LRT STEP MFI.
```

The multinomial logistic regression model to ascertain the relationship between party and search terms categories was not statistically significant, with a result of $p = 1.000$. The model was able to explain 28% (using Nagelkerke R^2) of the variance and 47.2% of cases were correctly classified. Tables 4.22, 4.23, and 4.24 show the data from the tests.

The model with the dependent variable Age Group has a result of $p = 0.744$, showing that the model as a whole does not fit significantly better than an empty model (a model with no predictors). The model explained 29% (using Nagelkerke R^2) of the variance, and 45% of cases were correctly classified. Tables 4.25, 4.26, and 4.27 show the the result of the analysis.

Party	Correctly classified cases
CDU	59.0%
CSU	3.8%
SPD	26.6%
Die Linke	4.9%
Die Grünen	5.6%
Overall	47.2%

Table 4.24.: Classification Table - Party

Model	-2 Log Likelihood	χ^2	df	Sig.
Final	923.918	202.019	216	0.744

Table 4.25.: Model Fitting Information - Age Group

Test	Result
Cox and Snell R^2	0.275
Nagelkerke R^2	0.293
McFadden	0.115

Table 4.26.: Pseudo R^2 - Age Group

Age Group	Correctly classified cases
20 – 39 years	7.9%
40 – 49 years	16.7%
50 – 59 years	48.0%
60 – 69 years	23.8%
70 – 90 years	3.5%
Overall	45.0%

Table 4.27.: Classification Table - Age Group

Model	**-2 Log Likelihood**	χ^2	**df**	**Sig.**
Final	1937.329	261.645	810	1.000

Table 4.28.: Model Fitting Information - State

Test	**Result**
Cox and Snell R^2	0.340
Nagelkerke R^2	0.343
McFadden	0.086

Table 4.29.: Pseudo R-Square - State

The model with the dependent variable State was not statistically significant, with a result of $p = 1.000$. The proportion of variance that can be explained by the model is 34% (Nagelkerke R^2). 27% of cases were correctly classified. Tables 4.28, 4.29, and 4.30 show the output of the analysis.

To summarise the above results, the two clusters can't be used to predict the age group, state, or party of politicians in the Bundestag, but a significant model to predict the gender can be built. It is important to note that such a model only brings moderately good predictive results.

4.4. Exploratory data mining

This section aims to explore the dataset and to try to identify similarities in objects. The main purpose of the following analyses is to identify if groups of politicians can be built on a base of socio-demographic information and categories of search terms, and what can be interpreted from these groups.

To build these groups, cluster analysis was performed. Since the data in the dataset is mixed (some variable are categorical and some are numeric), it is impossible to use most clustering algorithms. To perform analysis the k-prototypes algorithm was used, since it has proven to be a good method for recovering underlying structure in the data (Huang 1998). Clusters will be derived from the data about the politicians, and the politicians will then be assigned to each cluster.

The following variables were used for the clustering: Name, Gender, Party, Age, Cluster Politics and Economy, and Cluster Personal information. As in the previous sections, the State variable was excluded due to strong correlation with the Party variable. The values for the cluster variables are the number of search terms for each politician that were in

State	**Correctly classified cases**
Baden-Wurttemberg	6.2%
Bayern	13.14%
Berlin	4.0%
Brandenburg	1.6%
Bremen	6.8%
Hamburg	7.5%
Hessen	3.3%
Mecklenburg-Vorpommern	5.4%
Niedersachsen	6.5%
Nordrhein-Westfalen	21.1%
Rheinland-Pfalz	1.4%
Saarland	6.4%
Sachsen	7.6%
Sachsen-Anhalt	0.5%
Schleswig-Holstein	6.8%
Thuringen	1.4%
Overall	27.5%

Table 4.30.: Classification Table - State

Clusters	Score
2	0.3760
3	0.3554
4	0.3080
5	0.2262
6	0.1812
7	0.2023
8	0.1459

Table 4.31.: Clustering Politicians - Silhouette score

each respective cluster.

As in the previous uses of clustering methods, the first step is to identify optimal number of clusters using the silhouette score. This method can be used for both categorical and numeric data. The Silhouette scores and elbow curve for the range of 2 to 8 clusters can be seen in Table 4.31 and Figure 4.7, respectively. The Python code used to generate the scores is shown in Listing A.12 on page 93.

The silhouette score shows that best number of clusters is again two. The Python code used to assign the clusters to the data is shown in Listing A.11 on page 91. Tables 4.32, 4.33 and 4.34 describe the population of the clusters.

The gender and party ratio in both clusters seems to be similar to ratio in the whole dataset. In the first cluster the average age is however lower than in the whole data, while the number of terms in the text category with political and economical terms is higher. The number of terms in the personal information category is almost equal to average.

In the second cluster the age is slightly higher than the average, and number of terms in the both categories is lower than the average of the dataset.

To explore the differences between two clusters statistically, the Analysis of variance (ANOVA) is performed. The results of ANOVA and Effect size are shown in Table 4.35. All variables, apart from Gender, contribute significantly to the division between clusters, but the influence of Party is very low. The biggest contributor, as shown by η^2 are from the text category Political and economical terms. The other text category contributes slightly less, while the Age variable contributes moderately.

The first cluster tends to represent mostly younger members of the Bundestag whose political and economical activity is more interesting for internet users as their personal life. The second cluster tends to represent the older politicians who have, on average, fewer terms, but are almost equally represented. Table 4.36 shows an excerpt of politicians from each final cluster.

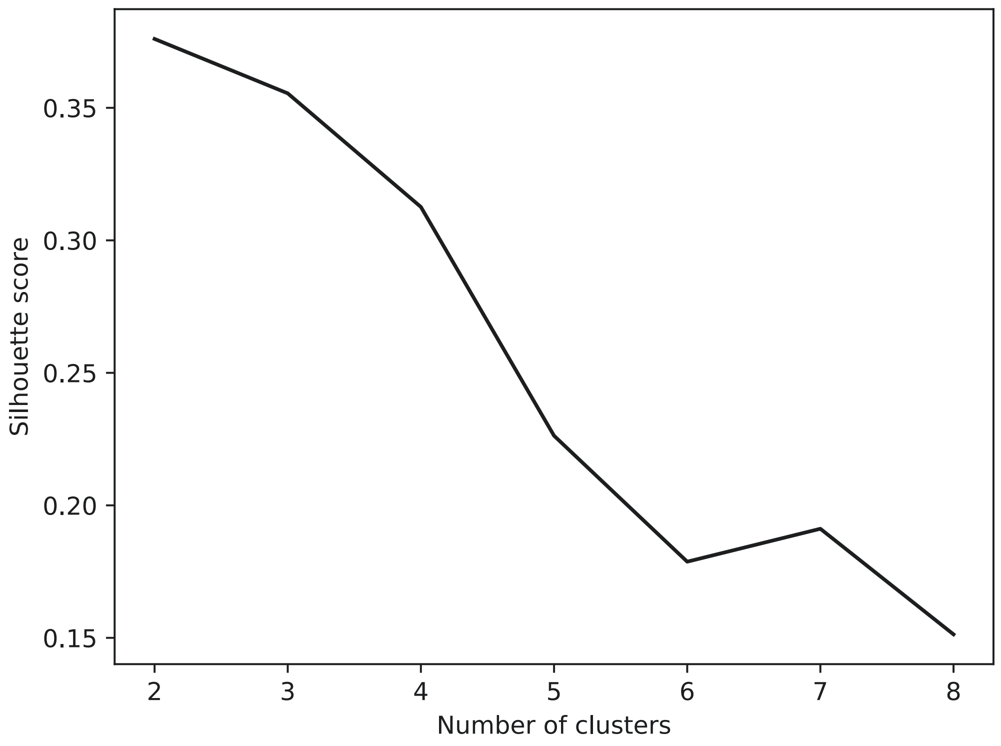

Figure 4.7.: Clustering Politicians - Silhouette score

Cluster	**Age**	**Politics**	**Personal**
0	49.6	14.7	9.8
1	56.9	4.8	4.9
All data	53.8	9.0	7.0

Table 4.32.: Mean comparison

Cluster	Party	Frequency	Percent
0	CDU	99	37.6%
	CSU	30	11.4%
	SPD	79	30.0%
	Die Linke	19	7.2%
	Die Grünen	35	13.3%
	No party	1	0.4%
1	CDU	154	42.0%
	CSU	26	7.1%
	SPD	114	31.1%
	Die Linke	45	12.3%
	Die Grünen	28	7.6%

Table 4.33.: Frequencies party comparison

Cluster	Gender	Frequency	Percent
0	Male	171	65.0%
	Female	92	35.0%
1	Male	227	61.9%
	Female	140	38.1%

Table 4.34.: Frequencies gender comparison

Variable	F	Sig.	η^2
Party	2.867	0.014	0.022
Gender	0.659	0.417	0.001
Age	3.081	0.000	0.207
Cluster Political	25.655	0.000	0.594
Cluster Personal	34.487	0.000	0.531

Table 4.35.: ANOVA report

Cluster	Excerpt
0	Angela Merkel, Gregor Gysi, Sigmar Gabriel, Peter Tauber
1	Stephan Albani, Ulrike Bahr, Valerie Wilms, Waltraud Wolff

Table 4.36.: Example of clustering results

4.5. Interpretation of results

The analysed dataset describes the politicians of the Bundestag, their socio-demographic information, such as age, gender, home state and party, and also the number of recognised autosuggestions that belong to two categories: political and economical information and personal information.

The gender ratio in the dataset is 7 : 4 male to female, and the most common age group is 50 – 60 years old. All German states are represented in the dataset, but the most common are North Rhein-Westphalia and Bavaria. The biggest parties are CDU and SPD. The proportion of clusters in the dataset is not equal; the cluster political and economical information is slightly bigger than the other. For most politicians, at least one word was assigned to each cluster, and for most of them 10–20 autosuggestion terms have been recognised and assigned to a cluster.

The autosuggestions of different search engines vary a lot one from another. It was obvious that DuckDuckGo would overlap very little with two other search engines, because DuckDuckGo returns terms only in English. The most overlapping terms are names of other significant people, names of companies and parties, and some English terms. The overlap of Bing and Google Search is significantly higher, with almost half of Google Search terms also being found in Bing terms. A possible explanation is that both search engines tend to suggest themes and terms that were popularly searched for a particular search term, and represent the trending topics. Both search engines declare on their websites that suggestions are based on how often other users have searched for a term. Although, the Google Search User Guide implies that autosuggestions also show the range and variety of information on the internet (Google 2017).

The exact autosuggestion search terms suggested by each search engine are not able to be predicted due to a lack of publicly available information on how the process of choosing these terms is carried out. It is however important to mention that webcrawler used for Google Search was active for a longer period of time, which could partly explain the higher amount of unique search terms for Google Search.

The correlation analysis found only one pair of strongly correlated variables in the dataset: Party and State. This correlation can be explained as the CSU party is only active in one state, Bavaria, and other parties tend to have a home state where they perform better. Because of this strong correlation the variable State was not included in further clustering.

No correlation was found between autosuggestion categories and any socio-demographic variables. This means that there are no obvious tendencies in the distribution of autosuggestion categories between gender, age, state, or party.

To prove the results of correlation analysis and see if autosuggestion categories can be used at to predict the variables, multinominal logistics regression was performed. The results show that only gender can be predicted with the help of text categories. The personal information corresponds more to female politicians. The possible interpretation can be to the tradition view of the female role in society. Various researches in political science describe stereotypical view of female politicians as being family-orientated, or show that people are more interested in information about spouses of women rather than men (Schneider and Bos 2014). The pattern found in this research may highlight one of the traditional problems of gender studies in political science: stereotypes about female politicians are still strongly held on to by the public (Huddy and Terkildsen 1993).

Finally, clustering analysis was performed to see if it is possible to build groups of politicians and find similarities in data. Two groups of politicians were found: one mostly represented the younger politicians with significantly higher amount of items in the text category political and economical terms.

4.6. Further research

The research carried out has shown possible further areas of interest for future work. Further analysis of the data can include time series analysis to obtain the information about stability of autosuggestions and attempt to draw the conclusions based on which terms and why are most stable. Furthermore, it could show the influence of different events, for example, the G20 Summit in Hamburg, or the North Rhine-Westphalia state election, to a particular politician's list of autosuggestions. The deeper psychological analysis of users behaviour and how it affects the search terms suggested could also be a further area of interest.

5. Conclusion

This master thesis aimed to study the behaviour of online search engines in a period before the German federal election in 2017. To answer the research question of the thesis, methods of data mining and statistics were used. The work was based on the standard CRISP-DM process. To perform the analysis, autosuggestion search terms for the members of the Bundestag were collected from three different search engines (Google Search, Bing, and DuckDuckGo) and stored. The data preparation stage included not only the standard cleaning of irrelevant data, but also the text categorisation procedure. This required lemmatisation, vectorising, development of text categories, and applying text classification algorithms.

The lemmatisation method was used to derive the dictionary form of the autosuggestions and prepare them for the following vectorisation, which was performed with the help of word2vec algorithm. The vectors were divided into groups by a simple and efficient k-means clustering algorithm. Then, grouped data was used to train an SVM categorisation algorithm.

The prepared data was used together with the information about politicians (their gender, age, party, and home state) for different types of analyses: description of the dataset, correlation analysis performed through Pearson's χ^2 test, logistic regression analysis, and clustering to define groups of politicians. The text analysis has determined two semantic groups in the list of unique autosuggest terms: one group describing the personal life of politicians and their family, and the other including political and economical terms and the names of the companies and organisations.

The main goal was to identify any patterns in the autosuggestion terms for searches related to politicians' names, and, if there are any patterns, to identify if there any differences in them depending on available attributes. The researched showed that there is indeed a pattern in these terms, which relates to politicians and gender. There is a higher probability of the politician being female if the number of autosuggestions describing personal information about the person increases.

The subsequent clustering analysis defined two groups of politicians: one of the groups included younger politicians with significantly higher number of items in the text category political and economical terms, while the other tended to represent mostly older politicians.

The research shows that there are no particularly strong patterns in the autosuggestions for searches related to politician's names. Only moderate dependence was found between gender and personal topics. Otherwise there is no difference in the patterns depending on available attributes.

6. Bibliography

Aggarwal, C.C., and C.K. Reddy. 2016. *Data Clustering: Algorithms and Applications.* Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. CRC Press. ISBN: 9781498785778. https://books.google.de/books?id=p8b1CwAAQBAJ.

Allison, Paul D. 2002. 'Missing data: Quantitative applications in the social sciences'. *British Journal of Mathematical and Statistical Psychology* 55 (1): 193–196.

Association, Information Resources Management. 2012. *Data Mining: Concepts, Methodologies, Tools, and Applications.* Contemporary research in information science and technology. Information Science Reference. https://books.google.de/books?id=4xNGngEACAAJ.

Azevedo, Ana Isabel Rojão Lourenço, and Manuel Filipe Santos. 2008. 'KDD, SEMMA and CRISP-DM: a parallel overview'. *IADS-DM.*

Backhaus, Klaus, Bernd Erichson, Wulff Plinke and Rolf Weiber. 2015. *Multivariate analysemethoden: eine anwendungsorientierte einführung.* Springer-Verlag.

Bae, Jung-Hwan, Ji-Eun Son and Min Song. 2013. 'Analysis of twitter for 2012 South Korea presidential election by text mining techniques'. *Journal of Intelligence and Information Systems* 19 (3): 141–156.

Burns, Robert P, and Richard Burns. 2008. *Business research methods and statistics using SPSS.* Sage.

Chakrabarti, Soumen. 2017. 'Data mining curriculum: A proposal'. Accessed 3rd August 2017. http://www.kdd.org/curriculum/index.html.

Chaturvedi, Anil, Paul E Green and J Douglas Caroll. 2001. 'K-modes clustering'. *Journal of Classification* 18 (1): 35–55.

Chen, X., and Z. Kang. 2017. *Stripe Rust.* Springer Netherlands. ISBN: 9789402411119. https://books.google.de/books?id=CFAsDwAAQBAJ.

Clifton, Christopher. 2009. 'Encyclopaedia Britannica, data mining'. Accessed 3rd August 2017. https://www.britannica.com/technology/data-mining.

Conover, Michael D, Bruno Gonçalves, Jacob Ratkiewicz, Alessandro Flammini and Filippo Menczer. 2011. 'Predicting the political alignment of twitter users'. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on,* 192–199. IEEE.

Cox, D.R., and E.J. Snell. 1989. *Analysis of Binary Data, Second Edition.* Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis. ISBN: 9780412306204. https://books.google.de/books?id=QBebLwsuiSUC.

Delen, D. 2014. *Real-World Data Mining: Applied Business Analytics and Decision Making.* FT Press Analytics. Pearson Education. ISBN: 9780133551112. https://books.google.de/books?id=O%5C_nbBQAAQBAJ.

Department of Linguistics, University of Pennsylvania. 2008. 'Tutorial: Pearson's Chi-square Test for Independence'. Accessed 5th July 2017. http://www.ling.upenn.edu/~clight/chisquared.htm.

Deyasi, A., S. Mukherjee, P. Debnath and A.K. Bhattacharjee. 2016. *Computational Science and Engineering: Proceedings of the International Conference on Computational Science and Engineering (Beliaghata, Kolkata, India, 4-6 October 2016).* CRC Press. https://books.google.de/books?id=3ZO%5C_DQAAQBAJ.

Dictionary, Collins. 2017. 'Collins English Dictionary, entry for "lemmatise"'. Accessed 4th July 2017. https://www.collinsdictionary.com/dictionary/english/lemmatize.

Dietterich, Thomas G., ed. 1998. 'Special issue on applications of machine learning and the knowledge discovery process'. *Mach. Learn.* (Hingham, MA, USA) 30 (2-3). ISSN: 0885-6125.

Digital Research, Institute for, and Education. 2011. 'Deciphering interactions in logistic regression'. https://stats.idre.ucla.edu/stata/seminars/deciphering-interactions-in-logistic-regression/.

DuckDuckGo. 2017. 'We don't collect or share personal information. That's our privacy policy in a nutshell.' Accessed 4th July 2017. https://duckduckgo.com/privacy.

Dumas, J. 2016. 'Example of Cramer's V Calculation in R'. Accessed 5th August 2017. https://jasdumas.github.io/tech-short-papers/Example_of_CramersV_Calculation.html.

Eckroth, J. 2017. 'k-means clustering'. Accessed 4th August 2017. http://cse630.artifice.cc/k-means.html.

Epstein, Robert, and Ronald E Robertson. 2015. 'The search engine manipulation effect (SEME) and its possible impact on the outcomes of elections'. *Proceedings of the National Academy of Sciences* 112 (33): E4512–E4521.

Estivill-Castro, Vladimir. 2002. 'Why so many clustering algorithms: a position paper'. *ACM SIGKDD explorations newsletter* 4 (1): 65–75.

Fayyad, Usama, Gregory Piatetsky-Shapiro and Padhraic Smyth. 1996. 'From data mining to knowledge discovery in databases'. *AI magazine* 17 (3): 37.

Fisher, Ronald A. 1992. 'The arrangement of field experiments'. In *Breakthroughs in statistics,* 82–91. Springer.

Gesmundo, Andrea, and Tanja Samardžić. 2012. 'Lemmatisation as a tagging task'. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2,* 368–372. Association for Computational Linguistics.

Google. 2017. 'Search using autocomplete'. Accessed 3rd May 2017. `https://support.google.com/websearch/answer/106230?hl=en`.

Gosall, N.K., and G.S. Gosall. 2012. *The Doctor's Guide to Critical Appraisal.* PasTest. ISBN: 9781905635818. `https://books.google.de/books?id=STKSmhyemN8C`.

Gries, David. 2005. 'Texts in Computer Science'.

Grimes, S. 2007. 'A Brief History of Text Analytics'. Accessed 5th August 2017. `http://www.b-eye-network.com/view/6311`.

Guilford, Joy Paul. 1954. 'Psychometric methods'.

Hsu, H.H., C.Y. Chang and C.H. Hsu. 2017. *Big Data Analytics for Sensor-Network Collected Intelligence.* Intelligent Data-Centric Systems: Sensor Collected Intelligence. Elsevier Science. `https://books.google.de/books?id=%5C_KI9DQAAQBAJ`.

Huang, Zhexue. 1998. 'Extensions to the k-means algorithm for clustering large data sets with categorical values'. *Data mining and knowledge discovery:* 283–304.

Huddy, Leonie, and Nayda Terkildsen. 1993. 'Gender stereotypes and the perception of male and female candidates'. *American Journal of Political Science:* 119–147.

ICTEA. 2017. 'What is a web crawler or web spider?' Accessed 3rd August 2017. `http://www.ictea.com/cs/knowledgebase.php?action=displayarticle&id=2097&language=english`.

Interior, U.S. Department of the. 2017. 'Statistical Interpretation'. Accessed 5th August 2017. `https://www.fort.usgs.gov/sites/landsat-imagery-unique-resource/statistical-interpretation`.

Joachims, Thorsten. 1998. 'Text categorization with Support Vector Machines: Learning with many relevant features'. In *Machine Learning: ECML-98: 10th European Conference on Machine Learning Chemnitz, Germany, April 21–23, 1998 Proceedings,* edited by Claire Nédellec and Céline Rouveirol, 137–142. Berlin, Heidelberg: Springer Berlin Heidelberg. `http://dx.doi.org/10.1007/BFb0026683`.

Ko, Youngjoong, and Jungyun Seo. 2000. ‘Automatic text categorization by unsupervised learning’. In *Proceedings of the 18th conference on Computational linguistics-Volume 1,* 453–459. Association for Computational Linguistics.

Kumar, Sathees, and R Karthika. n.d. ‘A Survey on Text Mining Process and Techniques’.

Lab, The Data Science. 2013. ‘Finding the K in K-Means Clustering’. Accessed 4th August 2017. http://cse630.artifice.cc/k-means.html.

Linguistics, Computational, and Psycholinguistics. 2017. ‘CLiPS’. Accessed 4th July 2017. http://www.clips.ua.ac.be/.

Linguistics, Computational, and Psycholinguistics. 2017. ‘pattern.de - Verb conjugation’. Accessed 4th July 2017. http://www.clips.ua.ac.be/pages/pattern-de.

Liu, Yanchi, Zhongmou Li, Hui Xiong, Xuedong Gao and Junjie Wu. 2010. ‘Understanding of internal clustering validation measures’. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on,* 911–916. IEEE.

Lovins, Julie Beth. 1968. ‘Development of a stemming algorithm’. *Mech. Translat. & Comp. Linguistics* 11 (1-2): 22–31.

Lu, L., Y. Zheng, G. Carneiro and L. Yang. 2017. *Deep Learning and Convolutional Neural Networks for Medical Image Computing: Precision Medicine, High Performance and Large-Scale Datasets.* Advances in Computer Vision and Pattern Recognition. Springer International Publishing. ISBN: 9783319429991. https://books.google.de/books?id=M64sDwAAQBAJ.

Manning, Christopher D., Prabhakar Raghavan and Hinrich Schütze. 2008. *Introduction to Information Retrieval.* New York, NY, USA: Cambridge University Press.

Matteucci, M. 2017. ‘A Tutorial on Clustering Algorithms’. Accessed 4th August 2017. http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html.

Merriam-Webster. 2017. ‘Database – Definition of database by Merriam-Webster’. Accessed 3rd August 2017. https://www.merriam-webster.com/dictionary/database.

Mikolov, Tomas. 2013. ‘word2vec’. Accessed 1st July 2017. https://code.google.com/archive/p/word2vec/.

Mikolov, Tomas, Kai Chen, Greg Corrado and Jeffrey Dean. 2013. ‘Efficient estimation of word representations in vector space’. *arXiv preprint arXiv:1301.3781.*

Mohri, Mehryar, Afshin Rostamizadeh and Ameet Talwalkar. 2012. *Foundations of machine learning.* MIT press.

Mooi, Erik, and Marko Sarstedt. 2011. *A concise guide to market research: the process, data, and methods using IBM SPSS statistics.*

Mueller, Andreas. 2015. 'GermanWordEmbeddings'. Accessed 4th August 2017. http://devmount.github.io/GermanWordEmbeddings/.

Nagelkerke, N. J. D. 1991. 'A note on a general definition of the coefficient of determination'. *Biometrika* 78 (3): 691–692. doi:10.1093/biomet/78.3.691. eprint: /oup/backfile/content_public/journal/biomet/78/3/10.1093/biomet/78.3.691/2/78-3-691.pdf. +%20http://dx.doi.org/10.1093/biomet/78.3.691.

Navigli, Roberto, and Simone Paolo Ponzetto. 2012. 'BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network'. *Artificial Intelligence* 193:217–250.

O'Connor, Brendan, Ramnath Balasubramanyan, Bryan R Routledge and Noah A Smith. 2010. 'From tweets to polls: Linking text sentiment to public opinion time series.' *ICWSM* 11 (122-129): 1–2.

Ojeda, Tony, Sean Patrick Murphy, Benjamin Bengfort and Abhijit Dasgupta. 2014. *Practical Data Science Cookbook.* Packt Publishing.

Oliveira, J.V. de, and W. Pedrycz. 2007. *Advances in Fuzzy Clustering and its Applications.* Wiley. ISBN: 9780470061183. https://books.google.de/books?id=Pn0e1xm4YBgC.

Omatu, Sigeru, Qutaibah M. Malluhi, Sara Rodriguez-Gonzalez, Grzegorz Bocewicz, Edgardo Bucciarelli, Gianfranco Giulioni and Farkhund Iqba, eds. 2015. *Distributed Computing and Artificial Intelligence, 12th International Conference, DCAI 2015, Salamanca, Spain, June 3-5, 2015.* Vol. 373. Advances in Intelligent Systems and Computing. Springer. doi:10.1007/978-3-319-19638-1. https://doi.org/10.1007/978-3-319-19638-1.

Ordóñez-Blanco, D, B Arcay, C Dafonte, M Manteiga and A Ulla. 2010. 'Object classification and outliers analysis in the forthcoming Gaia mission'. *Lecture Notes and Essays in Astrophysics* 4:97–102.

Osuna, Edgar, Robert Freund and Federico Girosi. 1997. 'Support vector machines: Training and applications'.

Parlamentarischer Rat. 1949. *Grundgesetz für die Bundesrepublik Deutschland.* Textausgabe der Bundeszentrale fur politische Bildung.

Perera, Praharshana, and René Witte. 2005. 'A self-learning context-aware lemmatizer for German'. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing,* 636–643. Association for Computational Linguistics.

Piatetsky-Shapiro, Gregory. 2014. 'What main methodology are you using for your analytics, data mining, or data science projects? Poll'. Accessed 3rd August 2017. `http://www.kdnuggets.com/polls/2014/analytics-data-mining-data-science-methodology.html`.

Porter, M., and R. Boulton. 2017. 'Snowball - German stemming algorithm'. Accessed 4th July 2017. `http://snowballstem.org/algorithms/german/stemmer.html`.

Reviews, CTI. 2016. *Data Mining for Business Intelligence, Concepts, Techniques, and Applications in Microsoft Office Excel: Business, Business.* Cram101. ISBN: 9781467230995. `https://books.google.de/books?id=ISKqKCeJwn4C`.

Reviews, CTI. 2016. *Research Methods and Statistics.* Cram101. ISBN: 9781497097735. `https://books.google.de/books?id=4KnDDAAAQBAJ`.

Rousseeuw, Peter J. 1987. 'Silhouettes: a graphical aid to the interpretation and validation of cluster analysis'. *Journal of computational and applied mathematics* 20:53–65.

Schneider, Gerold, and Martin Volk. 1998. 'Adding manual constraints and lexical look-up to a Brill-tagger for German'. In *Proceedings of the ESSLLI-98 Workshop on Recent Advances in Corpus Annotation, Saarbrücken.*

Schneider, Jeff, and Andrew W Moore. 2000. *A locally weighted learning tutorial using vizier 1.0.* Carnegie Mellon University, the Robotics Institute.

Schneider, Monica C, and Angela L Bos. 2014. 'Measuring stereotypes of female politicians'. *Political Psychology* 35 (2): 245–266.

Sciences, The Observational Health Data, and Informatics. 2017. 'Data Standardization'. Accessed 4th August 2017. `https://www.ohdsi.org/data-standardization/`.

Shearer, Colin. 2000. 'The CRISP-DM model: the new blueprint for data mining'. *Journal of data warehousing* 5 (4): 13–22.

Snijders, Chris, Uwe Matzat and Ulf-Dietrich Reips. 2012. '" Big Data": big gaps of knowledge in the field of internet science'. *International Journal of Internet Science* 7 (1): 1–5.

Steinbach, Michael, Levent Ertöz and Vipin Kumar. 2004. 'The challenges of clustering high dimensional data'. In *New directions in statistical physics,* 273–309. Springer.

Szumilas, M. 2010. 'Explaining Odds Ratios'. Accessed 5th August 2017. `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2938757/`.

Tan, Ah-Hwee, et al. 1999. ‘Text mining: The state of the art and the challenges’. In *Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases,* 8:65–70. sn.

Vidhya. K., G., Aghila. 2010. ‘Text Mining Process, Techniques and Tools: an Overview’. *International Journal of Information Technology and Knowledge Management.*

Vijayarani, S, Ms J Ilamathi and Ms Nithya. 2015. ‘Preprocessing techniques for text mining-an overview’. *International Journal of Computer Science & Communication Networks* 5 (1): 7–16.

Walker, Strother H., and David B. Duncan. 1967. ‘Estimation of the probability of an event as a function of several independent variables’. *Biometrika* 54 (1-2): 167–179. eprint: /oup/backfile/content_public/journal/biomet/54/1-2/10.1093/biomet/54.1-2.167/2/54-1-2-167.pdf. http://dx.doi.org/10.1093/biomet/54.1-2.167.

Wikipedia. 2017. ‘Donald Trump - Wikipedia, The Free Encyclopedia’. Accessed 4th July 2017. https://en.wikipedia.org/wiki/Donald_Trump.

Wirth, Rüdiger, and Jochen Hipp. 2000. ‘CRISP-DM: Towards a standard process model for data mining’. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining,* 29–39.

Zaman, H.B., P. Robinson, P. Olivier, T.K. Shih and S. Velastin. 2013. *Advances in Visual Informatics: Third International Visual Informatics Conference, IVIC 2013, Selangor, Malaysia, November 13-15, 2013, Proceedings.* Lecture Notes in Computer Science. Springer International Publishing. ISBN: 9783319029580. https://books.google.de/books?id=xD-7BQAAQBAJ.

A. Source code

Listing A.1: bing.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Crawls the Bing API for autocomplete terms
"""

from time import sleep
import random
import datetime
from timeit import default_timer as timer
import re
import sys
import argparse
import threading
import pymysql.cursors
import requests
from crawl_utils import *

# global variables
HEADERS = {'User-agent':'Mozilla/5.0'}
URL = 'http://api.bing.net/osjson.aspx'
LANGUAGES = ['de-DE'] # ['de-DE', 'en-GB']
MAX_RETRIES = range(5)
TIMEOUT = 1

LOGFILE, LOG = init_logging('crawler_bing.log')

def log_error(message):
    """
    Outputs a log entry to both the stderr logger and the file logger
    """
    LOG.error(message)
    LOGFILE.error(message)

def query_bing_suggest(query, lang):
    """
    Sends an HTTP request to the Bing autocomplete API and returns the raw data
    returend by the API.
    """
    params = {'Market': lang, 'query': query}
    return send_request(LOG, URL, MAX_RETRIES, params, HEADERS, TIMEOUT)

def get_suggestion_terms(request_data, queryterm):
    """
    Parses the raw request_data as JSON and extracts the suggestions.

    Data returned from the Bing API is formatted like so:

    [
        "Angel Merkel",
        [
            "angela merkel",
            "angel merkels abgekaute fingernägel",
            "angela merkel wiki",
            "angel merkel falten",
            "angela merkels mann",
            "angel merkel humor"
        ]
    ]

    Also removes queries if they match the queryterm exactly, and strips the
    queryterm from the beginning of the result.
    """
    raw_data = request_data.json()
    suggestions = [
        suggestion.lower().replace(queryterm.lower(), '').strip()
        for suggestion in raw_data[1]
        if suggestion.lower().strip() != queryterm.lower().strip()
    ]
    return suggestions

def store_in_db(connection, suggestions, queryterm, lang, raw_data):
    """
    Stores the data into the database
    """

    sql_suggestions = """
    INSERT INTO `suggestions_bing`
        (`queryterm`, `date`, `lang`, `raw_data`)
    VALUES
        (%s, %s, %s, %s);"""

    sql_terms = """
    INSERT INTO `terms_bing`
        (`suggest_id`, `suggestterm`, `position`, `score`)
    VALUES
        (%s, %s, %s, %s);"""

    try:
        with connection.cursor() as cursor:
            cursor.execute(sql_suggestions,
                           (queryterm, datetime.datetime.now(), lang, raw_data.content))
            suggest_id = cursor.lastrowid
            for position, suggestterm in enumerate(suggestions):
                cursor.execute(sql_terms,
                               (suggest_id, suggestterm, position, 0))
    except:
        log_error("Unexpected database error: Could not write results to database")
        raise

def do_request(lang, queryterm, connection):
    """
    Iterates through each URL, performing an autocomplete API lookup on each
    using the parameters
    """
    counter = 0
    try:
        request_data = query_bing_suggest(query=queryterm, lang=lang)
        suggestions = get_suggestion_terms(request_data=request_data,
    queryterm=queryterm)
        LOG.debug(suggestions)
        LOG.info('Thread: %s query: %s\tlang: %s', threading.get_ident(), queryterm,
    lang)
        store_in_db(connection=connection, suggestions=suggestions, queryterm=queryterm,
                    lang=lang, raw_data=request_data)
        counter = counter + 1
    except:
        print("Unexpected error:", sys.exc_info())
        log_error(repr(sys.exc_info()))
        log_error(repr(request_data.content))
    return counter

def do_languages(queryterm, inputfile):
    """
    Iterates through each language, calling do_request() on each
    """
    counter = 0
    connection = db_connect()
    for lang in LANGUAGES:
        if ('turks.csv' in inputfile and lang == 'tr') or (lang != 'tr'):
            counter = counter + do_request(lang=lang, queryterm=queryterm,
                                           connection=connection)
            #sleep(random.randint(0, 1)) # sleep between 1 and 2 seconds between each 2
    queries
    connection.commit()
    connection.close()
    return counter

def do_queryterms(queryterms, inputfile):
    """
    Calls do_languages for each queryterm and runs it in its own thread.
    """
    return execute_in_thread_pool(do_languages, queryterms, inputfile)

def main():
    """
    Main function
    """
    inputfile = ''
    args = parse_args()
    queryterms = read_csv(args.inputfile)
    start = timer()
    counter = do_queryterms(queryterms=queryterms, inputfile=inputfile)
    LOG.info('Done after %s seconds. In total we wrote %s results into the database',
             make_timestamp(start), counter)

if __name__ == "__main__":
    main()
```

Listing A.2: duckduckgo.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Crawls the DuckDuckGo API for autocomplete terms
"""

from time import sleep
import csv
import random
import datetime
from timeit import default_timer as timer
import re
import sys
import logging
import argparse
import threading
from concurrent.futures import ThreadPoolExecutor, as_completed
import pymysql.cursors
import requests
from crawl_utils import *

# global variables
HEADERS = {'User-agent':'Mozilla/5.0'}
URL = 'https://duckduckgo.com/ac/'
MAX_RETRIES = range(5)
TIMEOUT = 1

LOGFILE, LOG = init_logging('crawler_ddg.log')

def log_error(message):
    """
    Outputs a log entry to both the stderr logger and the file logger
    """
    LOG.error(message)
    LOGFILE.error(message)

def query_ddg_suggest(query):
    """
    Sends an HTTP request to the Bing autocomplete API and returns the raw data
    returend by the API.
    """
    params = {'q': query}
    return send_request(LOG, URL, MAX_RETRIES, params, HEADERS, TIMEOUT)

def get_suggestion_terms(request_data, queryterm):
    """
    Parses the raw request_data as JSON and extracts the suggestions.

    Data returned from the DuckDuckGo API is formatted like so:

    [
        {
            "phrase": "angela merkel"
        },
        {
            "phrase": "angela merkel biography"
        }
    ]

    Also removes queries if they match the queryterm exactly, and strips the
    queryterm from the beginning of the result.
    """
    raw_data = request_data.json()
    suggestions = [
        suggestion["phrase"].lower().replace(queryterm.lower(), '').strip()
        for suggestion in raw_data
        if suggestion["phrase"].lower().strip() != queryterm.lower().strip()
    ]
    return suggestions

def store_in_db(connection, suggestions, queryterm, raw_data):
    """
    Stores the data into the database
    """

    sql_suggestions = """
    INSERT INTO `suggestions_ddg`
        (`queryterm`, `date`, `raw_data`)
    VALUES
        (%s, %s, %s);"""

    sql_terms = """
    INSERT INTO `terms_ddg`
        (`suggest_id`, `suggestterm`, `position`, `score`)
    VALUES
        (%s, %s, %s, %s);"""

    try:
        with connection.cursor() as cursor:
            cursor.execute(sql_suggestions,
                           (queryterm, datetime.datetime.now(), raw_data.content))
            suggest_id = cursor.lastrowid
            for position, suggestterm in enumerate(suggestions):
                cursor.execute(sql_terms,
                               (suggest_id, suggestterm, position, 0))
    except:
        log_error("Unexpected database error: Could not write results to database")
        raise

def do_request(queryterm, dummy):
    """
    Iterates through each URL, performing an autocomplete API lookup on each
    using the parameters.
    The dummy arg makes it work with the thread pool abstraction.
    """
    connection = db_connect()
    try:
        request_data = query_ddg_suggest(query=queryterm)
        suggestions = get_suggestion_terms(request_data=request_data,
    queryterm=queryterm)
        LOG.debug(suggestions)
        LOG.info('Thread: %s query: %s', threading.get_ident(), queryterm)
        store_in_db(connection=connection, suggestions=suggestions, queryterm=queryterm,
                    raw_data=request_data)
    except:
        print("Unexpected error:", sys.exc_info())
        log_error(repr(sys.exc_info()))
        log_error(repr(request_data.content))
    connection.commit()
    connection.close()
    return 1

def do_queryterms(queryterms):
    """
    Calls do_request for each queryterm and runs it in its own thread.
    """
    return execute_in_thread_pool(do_request, queryterms, None)

def main():
    """
    Main function
    """
    inputfile = ''
    args = parse_args()
    queryterms = read_csv(args.inputfile)
    start = timer()
    counter = do_queryterms(queryterms=queryterms)
    LOG.info('Done after %s seconds. In total we wrote %s results into the database',
             make_timestamp(start), counter)

if __name__ == "__main__":
    main()
```

Listing A.3: parse.py

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

from __future__ import absolute_import, division, print_function
import sys
# sys.setdefaultencoding() does not exist, here!
reload(sys)  # Reload does the trick!
sys.setdefaultencoding('utf-8')
from pattern.web import decode_utf8
from pattern.de import parse
from pattern.de import singularize, pluralize
from pattern.de import conjugate
from pattern.de import INFINITIVE, PRESENT, SG, SUBJUNCTIVE
from pattern.de import attributive, predicative
from pattern.de import MALE, FEMALE, SUBJECT, OBJECT
from pattern.de import parse, split
from pattern.de import tag
import requests
import json
import pprint
import io
import argparse
import string
from utils import clean_umlauts

FILENAME = 'test3007.csv'
OUTPUT_FILENAME = "pattern_output.txt"
OUTPUT_SINGLE_FILENAME = "pattern_single_word_output.txt"

def parse_args():
    parser = argparse.ArgumentParser(description='Normalise words.')
    parser.add_argument("--input", default=FILENAME, type=str, help="Input file to be
    processed")
    parser.add_argument("--output", default=OUTPUT_FILENAME, type=str, help="Output
    file")
    parser.add_argument("--singleoutput", default=OUTPUT_SINGLE_FILENAME, type=str,
    help="Output file (only single word entries)")
    args = parser.parse_args()
    return args

def main():
    args = parse_args()
    words = []
    print("Loading from {}".format(args.input))
    with io.open(args.input,encoding='utf8') as f:
        acceptable_characters = string.letters + string.digits + " äüö"
        for line in f.readlines():
            if line.strip() == "suggestterm":
                continue
            word = filter(lambda c: c in acceptable_characters, line).strip()
            if len(word) > 0 and not any(c.isdigit() for c in word):
                words.append(word)

    print("Parsing {} words".format(len(words)))
    parsed_words = []
    for f in words:
        parsed = parse(f, tags=False, chunks=False, relations=False, lemmata=True)
        parsedlist = u" ".join([word.split("/")[2] for word in parsed.split(" ")])
        parsed_words.append(parsedlist)

    print("Saving {} words to {}".format(len(parsed_words), args.output))
    with open(args.output, "w") as f:
        for word in parsed_words:
            print(clean_umlauts(word), file=f)

    single_word_entries = list({clean_umlauts(f.strip()) for f in parsed_words if " "
    not in f})
    single_word_entries.sort()
    print("Saving {} words to {}".format(len(single_word_entries), args.singleoutput))
    with open(args.singleoutput, "w") as f:
        for word in single_word_entries:
            print(word, file=f)


if __name__ == "__main__":
    main()
```

Listing A.4: utils.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
utils.py

Utility functions
"""
import gensim
import string

FILENAME = 'pattern_output.txt'

def load_word_vectors(filename=FILENAME):
    print("Loading vectors from {}".format(filename))
    model = gensim.models.KeyedVectors.load_word2vec_format("german.model", binary=True)
    words = []
    with open(filename, "r", encoding="utf8") as f:
        words = f.readlines()
    words = [f.strip().title() for f in words]
    word_vectors = []
    used_words = []
    unused_words = []
    for b in words:
        if b in model:
            word_vectors.append(model[b])
            used_words.append(b)
        else:
            unused_words.append(b)
    print("Unused words: {}\nUsed words: {}".format(len(unused_words), len(used_words)))
    return (word_vectors, used_words, unused_words)

def clean_umlauts(input):
    return input.replace("ö", "oe").replace("ü", "ue").replace("ä", "ae")

def filter_characters(input):
    """
    Removes öüä from input, returns None if it contains a number, removes non-ascii
    characters
    """
    if any(c.isdigit() for c in input):
        return None
    acceptable_characters = string.ascii_letters + " "
    cleaned = clean_umlauts(input.lower().strip())
    filtered = "".join(list(filter(lambda c: c in acceptable_characters, cleaned)))
    if len(filtered) == 0:
        return None
    return filtered
```

Listing A.5: word_vectorise.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
word_vectorise.py
"""

from utils import load_word_vectors
import numpy as np
import argparse
import csv

def parse_args():
    parser = argparse.ArgumentParser(description='Vectorises words.')
    parser.add_argument("--input", type=str, required=True, help="Input file to be
    vectorised")
    parser.add_argument("--output", type=str, required=True, help="Vector output CSV
    file")
    args = parser.parse_args()
    return args


def main():
    """
    Open csv
    """
    args = parse_args()
    word_vectors, used_words, unused_words = load_word_vectors(args.input)

    with open(args.output, "w", encoding="utf-8", newline="") as f:
        writer = csv.writer(f, delimiter=',', quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for i, word in enumerate(used_words):
            vectors = word_vectors[i]
            writer.writerow([word,] + vectors.tolist())

if __name__ == "__main__":
    main()
```

Listing A.6: words_elbow.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Get data from file and compare average within-cluster sum of squares to alocate the
    best cluster number
"""

import time
from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist, pdist
from utils import load_word_vectors
import matplotlib.pyplot as plt
import numpy as np
FILENAME = 'pattern_single_word_output.txt'

def plot_elbow_curve(n_clusters, avgWithinSS, bss, tss):
    """
    Plots an elbow curve
    """
    # elbow curve
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(n_clusters, avgWithinSS, 'b*-', color='k')
    #ax.plot(n_clusters[kIdx], avgWithinSS[kIdx], marker='o', markersize=12,
    #markeredgewidth=2, markeredgecolor='r', markerfacecolor='None')
    plt.grid(True)
    plt.xlabel('Number of clusters')
    plt.ylabel('Average within-cluster sum of squares')
    plt.title('Elbow for KMeans clustering')

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(n_clusters, bss/tss*100, 'b*-', color='k')
    plt.grid(True)
    plt.xlabel('Number of clusters')
    plt.ylabel('Percentage of variance explained')
    plt.title('Elbow for KMeans clustering')

    plt.show()

def main():
    """
    Create a range of clusters and compare them
    """
    word_vectors, used_words, unused_words = load_word_vectors(FILENAME)
    start = time.time()
    n_clusters = range(1, 21)
    print("Using cluster sizes from {} to {}".format(min(n_clusters), max(n_clusters)))
    kmeans_clusters = [KMeans(n_clusters=n).fit(word_vectors) for n in n_clusters]
    centroids = [k.cluster_centers_ for k in kmeans_clusters]

    D_k = [cdist(word_vectors, cent, 'euclidean') for cent in centroids]
    cIdx = [np.argmin(D, axis=1) for D in D_k]
    dist = [np.min(D, axis=1) for D in D_k]
    avgWithinSS = [sum(d) / len(word_vectors) for d in dist]

    # Total with-in sum of square
    wcss = [sum(d**2) for d in dist]
    tss = sum(pdist(word_vectors)**2)/len(word_vectors)
    bss = tss-wcss

    stop = time.time()
    print("Time taken for clustering: {} seconds.".format(stop - start))

    print("Plotting elbow curve")
    plot_elbow_curve(n_clusters=n_clusters, avgWithinSS=avgWithinSS, bss=bss, tss=tss)

if __name__ == "__main__":
    main()
```

Listing A.7: words_svm.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Get training data from the file and perform text classification, evaluate the results
"""
from sklearn import svm
import csv
from utils import clean_umlauts
import numpy as np
import argparse
from sklearn.model_selection import cross_val_score
VECTOR_FILE = "word_vectors_t.csv"
CLUSTER_FILE = "word_clusters_t.csv"
OUTPUT_FILE = "svm_output.csv"
INPUT_WORDS_FILE = "classification_input.csv"
class Word:
    def __init__(self, word, vectors=[]):
        self.word = word
        self.vectors = np.asarray(vectors, dtype=np.float64)
        self.cluster = -1
    def __repr__(self):
        return "{} in cluster {}".format(self.word, self.cluster)
    def __eq__(self, other):
        return self.cluster == other.cluster and self.word == other.word

def setup_classifier(words):
    clf = svm.SVC()
    n_features = len(words[0].vectors)
    n_samples = len(words)
    training_set = np.asarray([f.cluster for f in words], dtype=np.int16)
    input_vectors = np.asarray([f.vectors for f in words])
    clf.fit(X=input_vectors, y=training_set)
    scores = cross_val_score(clf, input_vectors, training_set)
    print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
    return clf

def classify(classifier, input):
    return classifier.predict(np.asarray([f.vectors for f in input]))

def load_vectors(filename=VECTOR_FILE):
    print("Loading vectors from {}".format(filename))
    vectors = []
    with open(filename, newline='') as csvfile:
        reader = csv.reader(csvfile, delimiter=',', quotechar='|')
        for row in reader:
            vectors.append(Word(row[0],  row[1:]))
    return vectors

def load_clusters_with_word_array(filename=CLUSTER_FILE, words=[]):
    print("Loading clusters from {}".format(filename))
    with open(filename, newline='') as csvfile:
        reader = csv.reader(csvfile, delimiter=',', quotechar='|')
        for row in reader:
            for word in words:
                if word.word == row[0]:
                    word.cluster = row[1]
                    break


def load_words(vector_file, cluster_file):
    words = load_vectors(filename=vector_file)
    load_clusters_with_word_array(filename=cluster_file, words=words)
    print("Loaded {} words".format(len(words)))
    return words

def check_words(words):
    unclustered = [f for f in words if f.cluster == -1]
    if len(unclustered) > 0:
        raise Exception("Unclustered data found")

def parse_args():
    parser = argparse.ArgumentParser(description='Classify words.')
    parser.add_argument("--input", default=INPUT_WORDS_FILE, type=str, help="Input file to be classified")
    parser.add_argument("--clusters", default=CLUSTER_FILE, type=str, help="Training data containing cluster assignments")
    parser.add_argument("--vectors", default=VECTOR_FILE, type=str, help="Training data containing word vectors")
    parser.add_argument("--output", default=OUTPUT_FILE, type=str, help="Destination file for classified clusters")
    args = parser.parse_args()
    return args

def main():
    """
    Main
    """
    args = parse_args()
    training_words = load_words(cluster_file=args.clusters, vector_file=args.vectors)
    check_words(training_words)
    input_vectors = load_vectors(filename=args.input)
    print("Loaded {} words for input".format(len(input_vectors)))
    classifier = setup_classifier(training_words)
    output = classify(classifier, input_vectors)
    for i, j in enumerate(input_vectors):
        j.cluster = output[i]

    print("Writing classification output to {}".format(args.output))
    with open(args.output, "w", encoding="utf-8") as f:
        writer = csv.writer(f, delimiter=',', quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for word in input_vectors:
            writer.writerow([word.word, word.cluster])

if __name__ == "__main__":
    main()
```

Listing A.8: words2vectest.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
words2vectest.py
"""

import gensim
import time
from sklearn.cluster import KMeans
from sklearn import metrics
from utils import load_word_vectors
from sklearn import svm
import numpy as np
import csv
FILENAME = 'pattern_single_word_output.txt'
VECTOR_OUTPUT_FILE = "word_vectors.csv"
CLUSTER_OUTPUT_FILE = "word_clusters.csv"
def main():
    """
    Open csv
    """

    word_vectors, used_words, unused_words = load_word_vectors(FILENAME)

    start = time.time()
    n_clusters = 3

    print("Clustering")
    kmeans_clustering_predict = KMeans(n_clusters=n_clusters)
    idx = kmeans_clustering_predict.fit_predict(word_vectors)
    clustered_words = {}
    for index, cluster in enumerate(idx):
        key = used_words[index]
        if key not in clustered_words:
            clustered_words[key] = cluster
        else:
            raise Exception("Key {} already exists!".format(key))

    kmeans_clustering = kmeans_clustering_predict.fit(word_vectors)
    labels = kmeans_clustering.labels_
    metrics.silhouette_score(word_vectors, labels, metric='euclidean')
    metrics.calinski_harabaz_score(word_vectors, labels)

    end = time.time()
    elapsed = end - start
    print ("Time taken for clustering: {} seconds.".format(elapsed))
    print ("Silhouette score: {}".format(metrics.silhouette_score(word_vectors, labels,
     metric='euclidean')))
    print ("CH score: {}".format(metrics.calinski_harabaz_score(word_vectors, labels)))

    print("Saving word clusters to {}".format(CLUSTER_OUTPUT_FILE))
    with open(CLUSTER_OUTPUT_FILE, "w", encoding="utf-8") as f:
        writer = csv.writer(f, delimiter=',', quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for clustered_word, cluster in clustered_words.items():
            writer.writerow([clustered_word, cluster])
    with open("wordsNOTvec_output.txt", "w", encoding="utf8") as f:
        for unused_word in unused_words:
            f.write(unused_word + "\n")
    print("Saving word vectors to {}".format(VECTOR_OUTPUT_FILE))

    with open(VECTOR_OUTPUT_FILE, "w", encoding="utf-8", newline="") as f:
        writer = csv.writer(f, delimiter=',', quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for i, word in enumerate(used_words):
            vectors = word_vectors[i]
            writer.writerow([word,] + vectors.tolist())

if __name__ == "__main__":
    main()
```

Listing A.9: allocate_clusters.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Allocates cluster numbers to search terms then outputs the amount of each cluster each
    search term has.

Run me with the following example command line:

python3 allocate_clusters.py  --clusters full_test.csv --searchterms
    search_terms_with_suggestion.csv --output CHANGEME.csv --sourcefile bundestag.csv
"""


import csv
import argparse
from utils import clean_umlauts, filter_characters

class SearchTerm():
    def __init__(self, search_term, result):
        self.search_term = search_term
        self.result = filter_characters(result)
        self.cluster = -1
    def __repr__(self):
        return "{}: {} ({})".format(self.search_term, self.result, self.cluster)
    def to_array(self):
        return [self.search_term, self.result, self.cluster]

class ClusterAllocation():
    def __init__(self, word, cluster):
        self.word = filter_characters(word)
        self.cluster = int(cluster)
    def __repr__(self):
        return "{}: {}".format(self.word, self.cluster)

class Politician():
    def __init__(self, name, born, party, area, gender, age):
        self.name = name
        self.born = born
        self.party = party
        self.area = area
        self.gender = gender
        self.age = age
        self.cluster_dict = {}
    def to_array(self):
        return [self.name, self.born, self.party, self.area, self.gender, self.age]

def load_csv(filename, action, delimiter=',', skip_header=False):
    print("Loading", filename)
    data = []
    with open(filename, newline='', encoding='utf-8') as csvfile:
        reader = csv.reader(csvfile, delimiter=delimiter)
        if skip_header:
            next(reader)
        for row in reader:
            entry = action(row)
            if entry is not None:
                data.append(entry)
    return data

def _load_search_csv_entry(entry):
    return SearchTerm(search_term=entry[0], result=entry[1])

def _load_cluster_csv_entry(entry):
    return ClusterAllocation(word=entry[0], cluster=entry[1])

def _load_source_csv_entry(entry):
    return Politician(name=entry[0], born=entry[1], party=entry[2], area=entry[3], gender=entry[4], age=entry[5])


def load_search_csv(filename):
    return load_csv(filename=filename, action=_load_search_csv_entry)

def load_cluster_csv(filename):
    return load_csv(filename=filename, action=_load_cluster_csv_entry)

def load_source_csv(filename):
    return load_csv(filename=filename, action=_load_source_csv_entry, skip_header=True)

def parse_args():
    parser = argparse.ArgumentParser(description='Allocates cluster numbers to search terms.')
    parser.add_argument("--clusters", type=str, required=True, help="CSV file containing cluster allocations")
    parser.add_argument("--searchterms", type=str, required=True, help="CSV file containing original search terms")
    parser.add_argument("--sourcefile", type=str, required=True, help="Source file containing original search terms and metadata")
    parser.add_argument("--output", type=str, required=True, help="Output CSV file")
    args = parser.parse_args()
    return args


def write_result(filename, results, num_clusters):
    with open(filename, "w", encoding="utf-8", newline="") as f:
        writer = csv.writer(f, delimiter=',', quotechar='|', quoting=csv.QUOTE_MINIMAL)
        writer.writerow(["Name", "Born", "Party", "Bundesland", "Gender", "Age"] + ["Cluster {}".format(f) for f in range(num_clusters)])
        for f in results:
            writer.writerow(f.to_array() + [f.cluster_dict[g] for g in range(0, num_clusters)])

def uniquify_search_terms(search_terms):
    return set([f.search_term for f in search_terms])

def main():
    args = parse_args()

    clusters = load_cluster_csv(filename=args.clusters)
    search_terms = load_search_csv(filename=args.searchterms)
    politicians = load_source_csv(filename=args.sourcefile)
    num_clusters = max([f.cluster for f in clusters]) + 1
    unique_search_terms = uniquify_search_terms(search_terms)

    print("There are", len(clusters), "clustered words.")
    print("There are", len(politicians), "politicians")


    print("There are", len(search_terms), "search term pairs.")
    print("There are", len(unique_search_terms), "unique search terms.")

    politican_names = [g.name for g in politicians]
    filtered_search_terms = [
        f for f in search_terms
        if f.search_term in politican_names
    ]

    unique_filtered_search_terms = uniquify_search_terms(filtered_search_terms)

    print("There are", len(filtered_search_terms), "search term pairs after filtering
    those not found in the source list of politicians.")
    print("There are", len(unique_filtered_search_terms), "unique filtered search
    terms.")

    print("Found", num_clusters, "clusters.")
    clustered_search_terms = []
    for search_term in filtered_search_terms:
        for cluster in clusters:
            if search_term.result == cluster.word:
                search_term.cluster = cluster.cluster
                clustered_search_terms.append(search_term)
                break
    for politician in politicians:
        politician.search_terms = [f for f in clustered_search_terms if f.search_term
    == politician.name]
        #print(politician.name,"has",len(politician.search_terms), "search terms
    associated")
        for i in range(0, num_clusters):
            politician.cluster_dict[i] = len([f for f in politician.search_terms if
    f.cluster == i])
    print("There are", len(clustered_search_terms), "search term pairs with a clustered
    result.")

    write_result(results=politicians, filename=args.output, num_clusters=num_clusters)

if __name__ == "__main__":
    main()
```

Listing A.10: search_crossover.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Calculates search crossover
"""
import sys
sys.path.append('..') # Search parent directory for modules too
from utils import clean_umlauts
```

```
import csv

def read_csv(filename):
    data = []
    with open(filename) as file:
        data = [clean_umlauts(f.lower().strip().replace('"', '')) for f in
    file.readlines()]
    return data

def calc_relative(intersection, total):
    return "{0:.2f}%".format(intersection / total * 100)

def main():
    all_bing = set(read_csv(filename='../all_bing.csv'))
    all_google = set(read_csv(filename='../all_google.csv'))
    all_ddg = set(read_csv(filename='../all_ddg.csv'))

    bing_google = len(all_bing.intersection(all_google))
    bing_ddg = len(all_bing.intersection(all_ddg))
    google_ddg = len(all_google.intersection(all_ddg))

    total_bing = len(all_bing)
    total_ddg = len(all_ddg)
    total_google = len(all_google)

    print("Absolute intersection:")
    print("----------------------")
    print("Bing-Google:", bing_google)
    print("Bing-DDG:", bing_ddg)
    print("Google-DDG:", google_ddg)
    print("")
    print("Relative intersection")
    print("----------------------")

    bing_google_rel_bing = calc_relative(bing_google, total_bing)
    bing_google_rel_google = calc_relative(bing_google, total_google)

    bing_ddg_rel_bing =  calc_relative(bing_ddg, total_bing)
    bing_ddg_rel_ddg =  calc_relative(bing_ddg, total_ddg)

    google_ddg_rel_google =  calc_relative(google_ddg, total_google)
    google_ddg_rel_ddg =  calc_relative(google_ddg, total_ddg)

    print("Bing-Google overlap relative to Bing:\t", bing_google_rel_bing, "-",
    bing_google, "of",  total_bing )
    print("Bing-Google overlap relative to Google:\t", bing_google_rel_google, "-",
    bing_google, "of",  total_google )
    print("Bing-DDG overlap relative to Bing:\t", bing_ddg_rel_bing, "-", bing_ddg,
    "of",  total_bing )
    print("Bing-DDG overlap relative to DDG:\t", bing_ddg_rel_ddg, "-", bing_ddg, "of",
     total_ddg )
    print("Google-DDG overlap relative to Google:\t", google_ddg_rel_google, "-",
    google_ddg, "of",  total_google )
    print("Google-DDG overlap relative to DDG:\t", google_ddg_rel_ddg, "-", google_ddg,
    "of",  total_ddg )

if __name__ == "__main__":
    main()
```

Listing A.11: analysis_cluster.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Get data from file, build a cluster model and calculate Silhouette score
"""

from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn import preprocessing
from scipy.spatial.distance import cdist, pdist
import matplotlib.pyplot as plt
import numpy as np
from sklearn import metrics
from utils import load_word_vectors
from pandas import factorize
from kmodes import kprototypes
import csv
FILENAMENAMES = 'names.txt'
FILENAME = 'analysis_kmodes.csv'
HEADERS=["Name", "Party", "Gender", "Age", "Cluster 0", "Cluster 2"]
CLUSTER_OUTPUT_FILE = "final_clusters2.csv"

class entry:
    def __init__(self, name, party, gender, age, cl0, cl2):
        self.name = name.encode("utf-8")
        self.party = party
        self.gender = gender
        self.age = int(age)
        self.cl0 = int(cl0)
        self.cl2 = int(cl2)
    def __repr__(self):
        return "{} {} {} {} {} {}".format(self.name, self.party, self.gender, self.age,
    self.cl0, self.cl2)
    def to_row(self):
        return [
            self.name.decode("utf-8"),
            self.party,
            self.gender,
            self.age,
            self.cl0,
            self.cl2,
        ]

def get_data(filename):
    data = []
    with open(filename, newline='', encoding='utf-8') as csvfile:
        reader = csv.reader(csvfile, delimiter=',')
        next(reader, None) # Skip header
        for row in reader:
            data.append(entry(name=row[0], party=row[1], gender=row[2], age=row[3],
    cl0=row[4], cl2=row[5]))
    return data



def main():
    """
    Create a cluster model
    """
    print("Clustering")
    raw_data = get_data(FILENAME)
    data = np.empty((len(raw_data),5))
    n_clusters=2
    data[:,0], _ = factorize([f.party for f in raw_data])
    data[:,1], _ = factorize([f.gender for f in raw_data])
    data[:,2] = preprocessing.scale([f.age for f in raw_data])
    data[:,3] = preprocessing.scale([f.cl0 for f in raw_data])
    data[:,4] = preprocessing.scale([f.cl2 for f in raw_data])
    print("Clustering", len(data), "entries")
    kproto = kprototypes.KPrototypes(n_clusters=n_clusters).fit(X=data,
    categorical=[0,1])
    idx = kproto.fit_predict(data, categorical=[0,1])
    labels = kproto.labels_
    silohouette = metrics.silhouette_score(data, labels, metric='cosine')

    print ("Silhouette score: {}".format(silohouette))

    print("Saving word clusters to {}".format(CLUSTER_OUTPUT_FILE))
    with open(CLUSTER_OUTPUT_FILE, "w", encoding="utf-8") as f:
        writer = csv.writer(f, delimiter=',', quotechar='|', quoting=csv.QUOTE_MINIMAL)
        writer.writerow(["Name", "Party", "Gender", "Age", "Cluster 0", "Cluster 2",
    "Assigned Cluster"])
        for i, j in enumerate(raw_data):
            writer.writerow(j.to_row() + [labels[i],])




if __name__ == "__main__":
    main()
```

Listing A.12: analysis_elbow.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Get data from file and compare silhuette scores to alocate the best cluster number
"""

import time
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn import preprocessing
from scipy.spatial.distance import cdist, pdist
import matplotlib.pyplot as plt
import numpy as np
from pandas import factorize
from sklearn import metrics
from kmodes import kprototypes
import csv

FILENAME = 'analysis_kmodes.csv'
HEADERS=["Name", "Party", "Gender", "Group", "Cluster 0", "Cluster 2"]
CLUSTER_OUTPUT_FILE = "final_clusters.csv"

class entry:
    def __init__(self, name, party, gender, age, cl0, cl2):
        self.name = name.encode("utf-8")
```

```python
        self.party = party
        self.gender = gender
        self.age = int(age)
        self.cl0 = int(cl0)
        self.cl2 = int(cl2)
    def __repr__(self):
        return "{} {} {} {} {} {}".format(self.name, self.party, self.gender, self.age,
    self.cl0, self.cl2)

def get_data(filename):
    data = []
    with open(filename, newline='', encoding='utf-8') as csvfile:
        reader = csv.reader(csvfile, delimiter=',')
        next(reader, None) # Skip header
        for row in reader:
            data.append(entry(name=row[0], party=row[1], gender=row[2], age=row[3],
    cl0=row[4], cl2=row[5]))
    return data



def main():
    """
    Create a range of clusters and compare them
    """

    start = time.time()
    n_clusters = range(2, 9)
    print("Using cluster sizes from {} to {}".format(min(n_clusters), max(n_clusters)))
    raw_data = get_data(FILENAME)
    data = np.empty((len(raw_data),5))
    data[:,0], _ = factorize([f.party for f in raw_data])
    data[:,1], _ = factorize([f.gender for f in raw_data])
    data[:,2] = preprocessing.scale([f.age for f in raw_data])
    data[:,3] = preprocessing.scale([f.cl0 for f in raw_data])
    data[:,4] = preprocessing.scale([f.cl2 for f in raw_data])
    print("Clustering", len(data), "entries")

    sil_scores = []
    for n in n_clusters:
        print(n, "clusters")
        kproto = kprototypes.KPrototypes(n_clusters=n).fit(X=data, categorical=[0,1])
        labels = kproto.labels_
        score = metrics.silhouette_score(data, labels, metric='cosine')
        sil_scores.append(np.array([n, score]))
        print ("Silhouette score: {}".format(score))
    sil_scores = np.array(sil_scores)
    plt.plot(sil_scores[:,0], sil_scores[:,1], color='k')
    plt.ylabel('Silhouette score')
    plt.xlabel('Number of clusters')
    plt.show()

if __name__ == "__main__":
    main()
```