Informatics

Analyse einer dynamischen Sammlung von Zeitungsartikeln mit inhaltsbasierten Methoden

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Markus Neumeyer, BSc.
Matrikelnummer 1225172

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Hannes Werthner
Mitwirkung: Univ.Ass. Mag.rer.nat. Dr.techn. Julia Neidhardt
Univ.Ass. Dipl.-Ing. Mete Sertkan, BSc

Wien, 12. Oktober 2020

Markus Neumeyer Hannes Werthner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Informatics

Analysis of a dynamic collection of news articles with content-based methods

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Markus Neumeyer, BSc.
Registration Number 1225172

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Hannes Werthner
Assistance: Univ.Ass. Mag.rer.nat. Dr.techn. Julia Neidhardt
Univ.Ass. Dipl.-Ing. Mete Sertkan, BSc

Vienna, 12th October, 2020

Markus Neumeyer	Hannes Werthner

Erklärung zur Verfassung der Arbeit

Markus Neumeyer, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 12. Oktober 2020

Markus Neumeyer

Danksagung

Ich möchte mich bei all jenen bedanken, die mir während der Erstellung dieser Arbeit unterstützend zur Seite gestanden sind. Allen voran meiner Betreuern Frau Julia Neidhardt und Herrn Mete Sertkan, welche mich bei fachlichen wie auch bei technischen Fragen stets mit Rat unterstützten und mir viele wertvolle Stunden Ihrer Zeit schenkten. Vielen Dank für die Zusammenarbeit.

Ein herzliches Dank gilt auch dem Team von *Der Falter*, welche Ihren Datensatz mit Artikeln bereitstellten, um die Versuche in dieser Arbeit erst zu ermöglichen.

Danken möchte ich aber auch meiner Familie, meinen Freunden, wie auch meinem Arbeitgeber, welche alle Rücksicht auf den erhöhten Zeitaufwand von Studium und Arbeit nahmen und mir immer wieder unter die Arme griffen.

Acknowledgements

I want to thank everyone that was supportive throughout the creation of this thesis. First and foremost my supervisors Julia Neidhardt and Mete Sertkan, who guided me both scientifically as well as technically through the whole work and spent much of their precious time for me. Thank you very much for your cooperation.

My gratitude goes also out for the team of *Der Falter*, who provided the test dataset with articles, that was needed to enable the work done in this thesis.

Finally I want to thank my family and friends as well as my employer, who all showed their consideration for the time consumption of studying and working and who assisted me many times.

Kurzfassung

Die Art und Weise wie man geschriebene Zeitungsartikel liest hat sich über die letzten Jahrzehnte drastisch verändert, durch das Internet ist eine riesige Menge an Artikeln jederzeit verfügbar. Die Leser und Leserinnen können nicht mehr alle Artikel lesen sondern sind auf Empfehlungen angewiesen. Dies hat zur Entwicklung von "News Recommender Systems" geführt.

Die Empfehlungen dieser Systeme basieren auf verschiedensten Methoden. Ein Typ dieser Methoden sind inhaltsbasierte Methoden, welche ausschließlich den textuellen Inhalt eines Artikels, sowie deren Titel, dazu verwenden, ähnliche Artikel zu finden. Im Gegensatz zu "Collaborative Filtering Methods", welche auch demographische Daten und bereits gesammelte Interessen der Leser verwenden.

In dieser Arbeit vergleichen wir state-of-the-art Methoden für text-basierte Empfehlungen von Zeitungsartikeln.
Der Fokus wird dabei auf zwei Punkte gelegt. Einerseits die Fähigkeit einen dynamischen Datensatz zu analysieren. Dies umfasst sowohl die Möglichkeit, neue Artikel in ein bereits berechnetes Modell einzuarbeiten, sowie die Fähigkeit Trends von Schlagworten oder Themen innerhalb der Artikel zu finden. Andererseits die Vielfalt und Serendipität der Empfehlungen. Meist befassen sich Vergleiche von Recommender Systems mit der Genauigkeit der Empfehlungen. Diese Arbeit widmet sich stattdessen der Vielfalt und der Serendipität der Empfehlungen, um diese noch weiter zu verbessern.

Das Fazit des Vergleichs ist, dass jede der verglichenen Methoden Stärken und Schwächen hat. Es konnte keine Methode identifiziert werden, welche den anderen Methoden in allen betrachteten Aspekten überlegen war.

Abstract

The consumption of news changed throughout the last decades, a huge amount of articles is available at any time in the internet. Consumers therefore need help to find articles that might be relevant for them, as they are not able to scan through all offered articles themselves. This led to the emergence of news recommender systems.

The way in which these systems choose articles that might be relevant varies vastly. One kind of methods are the content-based methods, which use only the written content of news articles and build relations between articles for the recommendations based on it. In contrast to collaborative filtering methods, which also use demographic data and previously gathered interests of users.

In this work we analyze and compare current state of the art methods for content-based recommendations of news articles.
The focus of the comparison will be on two main points. On the one hand is the ability to analyze a dynamic corpus. This includes both the possibility to include new articles to an existing model, as well as finding trends within the found topics or keywords of a model. On the other hand comes the diversity and serendipity of recommendations. Most comparisons of recommender systems put the focus on the accuracy of recommendations, instead this thesis will put the focus on diversity and serendipity to further improve the quality of recommendations.

The conclusion of this comparison is that every method has its strengths and weaknesses. No method could be found that exceeds all other methods in all aspects that were considered.

Contents

CHAPTER 1

Introduction

1.1 Motivation & Problem Statement

Throughout the last few years, the way in which news are consumed has changed drastically. The growth of digitalization and the accessibility of the World Wide Web granted access to a wider spectrum of and therefore also a significantly bigger amount of news. This increase in supply has led to a surplus of information for the consumer. That, in combination with the wish to be offered interesting information as well as being surprised by new topics to discover, resulted in the emergence of recommender systems, or more specifically, news recommender systems as a support to prefilter content for the interests of consumers.
The way in which these recommender systems choose potentially relevant information varies widely and is also dependent on the type of data that is processed and that shall be presented. While some methods are looking to find more specific and focused items, others might be able to offer more diverse and serendipitous items.
Another important factor, when talking about news article recommendation, would be time. Consumers of news articles obviously are less interested in older articles than they are in articles which have been written in the last few hours or days. But not only in this sense time is important, changes in common topics or occurrences of linked keywords are also vital in the consideration of what the reader is recommended. When talking about news article recommendation systems, there are multiple levels in which those can be analyzed and compared. Some of those are: document level, topic level, and author level. The document level looks at each document on itself to enable a comparison between them. Topic level on the other hand views sets of articles to find common topics. On the author level, sets of articles are views as well, although the focus does not lie on what the text is about but more on how the text is written to distinguish between authors and their styles [SNW19].
While specific types of methods for data preparation for recommendation systems have

already been analyzed extensively, there are not a lot of comparisons of them. Especially the inclusion of time as a factor for interpretation and comparison has not been considered in great detail in previous research. Also, most of the already existing analyses are carried out with English and American daily newspapers, not with Austrian or more specific Viennese publications/media outlets.

1.2 Research Questions

The lack of the above-mentioned analyses leads to the following research questions

1. *What methods, concepts or algorithms represent suitable mechanisms for the continuous analysis and subsequent classification and clustering of news articles based on general topics or specific keywords?*
   In comparison to previous research, in this thesis we will focus on the analysis of not only a specific point in time of the news article database, but also a continuous investigation of it. This aim also leads to another sub-question: *"Which methods allow to recognize short or long term changes of e.g. key word clusters or topic correlations and which are not really suited to analyze changes at all?"*.

2. *What methods build close relations between news articles yielding results in a more focused way and what methods lead to a broader spectrum of relations generating serendipitous connections?*
   Although it might be easier to build focused connections between news articles which seem reasonable at first sight, one motivation for this thesis is also to draw attention to connections which look arbitrary at first but might be legitimate on further inspection. Thus, the analysis of these data will also reveal the interesting insights on whether these methods can be fused to achieve the advantages of both approaches simultaneously.

1.3 Aim of the Work

The main objective of this work is the comparison and analysis of different content-based methods or algorithms to find similar news articles as a means for publishers to analyze their articles and trends as well as to build a fundament for a news recommender system, the latter one will not be implemented and is out of scope of this thesis. Therefore, the focus lies on the data preprocessing, preparation, analysis and relation building of newspaper articles with content-based methods and the inclusion of the factor time to these methods for the recognition of appearing trends through the time. After a literature research of the current state of the art of these methods for finding similar or relevant news articles with any kind of relation, a set of methods will be chosen to include time-related analysis and to be tested against a given dataset. Hence, multiple pipeline-models consisting of various methods will be implemented, combined and evaluated to compare them.

To finally test and evaluate these methods, a dataset of newspaper articles of a Viennese weekly newspaper of all articles from 1998 until 2019 is available, from which a subset will be drawn. Based on this it is given that conclusions drawn from these experiments will be influenced by the language and would have led to different results with e.g. datasets from English daily newspapers.
The evaluation and comparison of the different methods will be done in a quantitative manner, but not all aspects will be possible to be evaluated quantitatively. Therefore additionally, a qualitative review will be undertaken to further analyze the results in regards to their abilities, where a quantitative measurement would not be suitable.

1.4 Methodological Approach

1. **Literature research**
   To find a variety of current state of the art methods to compare news articles with each other and find keywords and topics, a literature research was done. After this step, not only were the methods to be analyzed and compared be identified, but background knowledge about them, like their strengths and weaknesses or what libraries are available for the implementation was acquired afterwards. This step built the foundation of all further steps and needs thorough attention. As the literature research is only a part of this thesis, a full sophisticated systematic literature review is not required to be able to gather all the needed information, but a few parts of it will be used. Kitchenham et al. [KBB+09] describes in detail, how an extensive systematic literature review in software engineering is done. The steps, which were used for this literature research are:

   - Research Questions
     These had to be defined before starting the research.
   - Inclusion criteria
     No strict date or minimum year, where literature has to be younger than, was set. But as the goal is to identify state of the art methods, literature from recent years was preferred.
   - Documentation of identified papers
     Both literature, that was defined as useful, as well as literature that can not be further used for this thesis, was documented. An example for a more extensive literature review, where the focus was put on the same three steps, can be found in Hall et al. [HBB+11].

2. **Data analysis and preparation**
   As working in data and text mining requires, the data which is used for later experiments and to test the models on had to be thoroughly analyzed and prepared to be useable. This step can be seen as the beginning of the data analysis itself, for which multiple well-developed methodologies have been created. One example would be the CRISP-DM method, which is described by Wirth et al. [WH00]. It

describes the two steps Data Understanding and Data Preparation, which come before the actual model building. Another model, that describes steps of a data analysis process, is the KDD model as described in Fayyad et al. [FPSS96], where the Preprocessing and the Data Transformation come before the actual data mining. A comparison of these methods can be found in Azavedo et al. [AS08], where it is concluded that CRISP-DM is a more detailed kind of implementation of the KDD model, so this thesis mainly relies on the descriptions of CRISP-DM. According to it, for the data understanding a thorough data exploration is needed. The aim is to get familiar with the data, find quality problems within the data and find first insights. The data preparation part was secluded into steps that are important for all methods to be implemented, like tokenization of the articles, part of speech tagging and named-entity recognition, and the method specific preparation that is needed for the algorithm that implements the method.

3. **Design of models to compare**
   Based on the knowledge and identified methods acquired in step 1, different models were designed for the analysis of the articles as well as for the inclusion of the analysis over time. Examples for methods would be key-word-clustering via TF-IDF, dynamic topic modeling or multi-attribute networks. As this thesis was built upon the CRISP-DM framework, this step can be seen as the first half of the modelling phase described in CRSP-DM. It describes the selection of the modeling technique, which in our case was the selection of libraries, frameworks or algorithms to implement the chosen methods, as well as a selection of important parameters, that have to be used for the chosen algorithms.

4. **Implementation of models and execution**
   The models designed in step 3 had to be implemented as well as executed and the results and various statistics of the execution needed to be saved. In this step it is vital to implement models in a way which allows the analysis and comparison not only of the final result, but of intermediate calculations and data manipulations as well. These models were iteratively refined, which lead to an indistinction of steps 3 and 4. This step was the realization of the second half of the modeling phase of CRISP-DM. This part of the phase is described as the building of the model itself, with its concrete implementation, parameter tuning and testing of it. What is also important here is the assessment of the model, if the results are meaningful and realistic, so they can be used for further evaluation and comparison of the models.

5. **Evaluation and comparison of the models**
   Both the final results and the interim results of the developed models were analyzed and compared. On the one hand, this process was carried out in a mathematical manner, i.e. with various measures, on the other hand, this step also involved a qualitative review of the results together with authors of a Viennese newspaper. Measuring variety or surprise of the results has been done with metrices, the quality of each method in regard to their ability to include new articles and find

trends was done qualitatively. As well as steps 2 to 4, this step was the realization of a CRISP-DM phase, namely the evaluation phase. According to CRISP-DM, not only the mathematical evaluation of the models is important, but also their interpretation and if it has meaning and can be used for the actual business domain. The last step of CRISP-DM, the deployment, would describe how to deploy and present to actual customers. The deployment of the implemented model was out of scope for this thesis. The presentation of the implementations of the methods on the other hand was important, as domain experts were also conducted to evaluate the implemented methods in a qualitative way. This was done to compare some specific attributes of the methods, that can not be analyzed in a mathematical way. These would be the following:

- How well the method can find trends within keywords/topics
  Methods that were implemented should not only be able to identify keywords and topics, but also find trends within these keywords. This information is very important for news article recommender systems, as not all topics and keywords represent current trends, may it be in an ascending or descending way.
- Possibility to graphically present trends, keywords or topics
  Not only should the results of each method be used for the recommender system itself to recommend articles, but also the authors and publishers of the newspaper, along with news agencies or platforms, are interested in current trends and want to analyze them.
- Capabilities to include new articles into the existing model
  As a corpus of news articles is dynamic and changes every day, some methods might be better suited for a stable representation of keywords or topics, if not the whole model has to be recomputed for every new article. Another factor here was the time it takes to either include new articles to the existing model or recompute the whole model.

To compare the models regarding these factors, that can not easily be answered in a mathematical quantitative way, a qualitative review was required. This qualitative evaluation was done by the author of this thesis together with domain experts.
Furthermore, the evaluation was not able to be done on the whole dataset, as this would have require too much computational effort and therefore time, which lead to the requirement of a sampling method. The method that was used was Stratified Random Sampling as described in Araya et al. [APSN13]. To differentiate sub-groups, or strata as they are called, the so called ressorts of the news articles were used, which is given as metadata for every article in the chosen dataset.

The rest of the thesis is organized as follows. In chapter 2, the current state of the art of methods, that are used for news article recommender systems will be discussed. Chapter 3 will give a theoretical background of all used methods and describes a few methods that seemed promising but were excluded from this work. In chapter 4 the dataset that is used for this thesis as well a the general pre-processing steps of the given dataset are described. Chapter 5 gives insights in how the methods were implemented. Chapter 6 shows the result of the data analysis, with a comparison of all methods. Chapter 7 finally concludes this thesis with answering the given research questions as well as summarizing the strengths and weaknesses of all implemented methods.

CHAPTER 2

State of the art

Recommender systems in general are already well established in various application areas like e-market websites, social media platforms or as in this case newspaper websites. Various approaches of these systems have already been analyzed in a scientific environment. Especially for news recommender systems, Kirshenbaum et al. [KFD12], for example, compare multiple methods of news article comparison for recommendations as well, but include the popularity of news articles and previous knowledge about the reader over a longer span of time, like what topics the consumer was previously interested in, not only the currently read article. This research includes a live experiment with different methods on a newspaper website to measure click-through rates and concludes that the Bayesian adjusted term frequency in combination with an item-item collaborative filtering led to the highest click-through rates. In contrast, our work will not include further previously read articles, but only the current one, but nonetheless approaches from this work can be taken as a starting point.

A great overview of challenges and their current approaches of overcoming them is elaborated in Karimia et al. [KJJ18] The authors analyzed recently published articles about news recommendation methods, approaches and comparisons and basically summarized the challenges like

- *cold-start-problem* - starting a new recommender system without previous knowledge about the users, only relevant for collaborative filtering methods
- *data sparsity issues* - transforming articles into vectors or matrices, as described in the following chapters, often leads to sparseness, which might influence the mathematical methods used on these
- *diversity* - creating multiple recommendations might tend to offer very similar articles

The authors gathered current approaches to overcome these challenges, which will also be interesting for this work. Afterwards, they discussed ways to test and benchmark recommender systems and suggested metrices for evaluation, some news article specific, some general. Furthermore, open source frameworks to measure these metrices are presented and discussed. Conclusions from their research suggest that hybrid methods of content-based and collaborative filtering are the current go-to method for news recommendation, while information retrieval and click-through-rates are currently the most prevalent factors for measuring the success of recommendations.
In contrast, stylistic analysis is not very well investigated as a tool for article recommendation. Argamon et al. [AWC$^+$07] present insights into the methods of stylistic text analysis, with the mentioning of classification for author identity, gender or nationality, yet its possible usefulness for article recommendation has not been considered. We will try to include this possibility into our work.
Another way to compare different methods of building relations between news articles would be the predictability of the recommendations. While some methods might lead to very foreseeable classifications, others might result in what appears to be a serendipity. Ge et al. [GDBJ10] discuss this issue with methods of how to measure serendipity in multiple ways, but their research does not continue to take the next step to actually compare methods of recommendations in this regard, therefore we are going to include serendipity into our measurements of article recommendations.
Apart from content-based filtering, other methods like collaborative filtering or demographic analysis can also be used to recommend news articles to the reader. The research carried out by Thorat et al. [TGB15] presents an overview and comparison of collaborative filtering methods, demographic filtering methods, content-based filtering methods as well as hybrid methods of these. Hence it is demonstrated that content-based filtering methods easily allow initial classification, provide user-independence and are transparent to the user and the provider in comparison to the other methods. However, these methods might also be regarded as limited because they do not include user feedback and tend to stay very focused on the same topics. Despite the advantages of collaborative filtering and demographic filtering discussed, these methods will not be employed in this thesis, and the sole focus will be on content-based analysis only. Reason for this choice is that this thesis will not only focus on the recommendation of articles, but on the analysis of trends etc. for the publishers of the articles as well.
One specific approach that mainly uses content-based filtering for recommendations is described in Kompan et. al. [KB10] Here a combination of term frequency, occurrence of names and places in articles, keywords and a category for every article are combined to compute the similarity between articles. The research centers on the computation performance, as this work aims primarily for very big online news portals. This is one specific method created with rapidity in mind, which can be used as a comparison for other methods analyzed in this thesis.
A further method which applies content-based methods only is the approach of multi-attribute networks as a means for filtering as described in Son et al. [SK17]. The approach could be summarized by acquiring attributes of items, calculating similarities

between them, building a multi-attribute network of the items and build clusters to classify new articles into. Here recommendations are built by items of the same cluster. The research mentioned above concludes that content-based filtering via multi-attribute networks achieves higher precisions when tested against real user preferences than other methods such as feature weighting or pure content-based filtering and therefore will be highly considered for this thesis.
One specific way for keyword extraction is described in Lee et al. [LK08], which uses the TF-IDF model. TF-IDF is an acronym for term-frequency and inverse document frequency, which combines the frequency of a term in one article with the ratio of articles which include this term in order to compute the relevance of a term for an article in a set of articles. In the mentioned work, six different variants of TF-IDF are defined and tested against real news articles. The extracted key words are evaluated manually in a qualitative way and it is concluded that combined with cross-domain comparison to remove 'meaningless' words (not critical for the domain), this way appears to be highly effective for topic detection and opinion mining, and therefore will be used in this work.
An overview of topic modeling as a means to extract topics from bulks of unstructured data, in this case news articles in natural text form, is provided in Uys et al. [UDPU08]. This research outlines various different ways of topic modeling, such as Latent Dirichlet Allocation [BNJ03], to find underlying topics of document collections. Also, "dynamic topic modeling" [BL06] is discussed as a tool to analyze changes in topics of a dynamic corpus of documents, such as the article archive of a newspaper. This approach results in a hierarchical model of topics of a sequential document collection and therefore it functions as an adequate approach for this thesis.

CHAPTER 3

Methods

In this chapter, all methods that were identified, analyzed and eventually implemented are described.
The identification of current state of the art methods for keyword extraction, topic modelling or other methods that could be useful for news recommender systems was done via a literature review. Starting point for this review was a publication by Karimi et al. from 2018 [MK18], in which an extensive overview of recent developments and achievements in the topic of news recommender systems is listed and analyzed. This summary of current developments combined with an overview of general text mining methods for online news articles from 2019, which, like this thesis, focuses on going beyond accuracy of article recommendations and tries to analyze diversity and serendipity as well [SNW19]. All the sources that are cited in both aforementioned articles served as a helpful starting point for further literature review.
As this thesis is focused solely on content-based methods, other methods that use collaborative filtering techniques, or any kind of required preceding demographic data or preferences of users, were excluded from further examination. Also the computational cost had to be within a realistic scope, as the experiments had to be done on a specific computer and were not allowed to be uploaded to some bigger scope computational system. Another important aspect of the consideration of methods was their feasibility to be implemented with reasonable workload. Which means that methods, that are only theoretically described with no suggested way of implementation or the like were also not included for further investigation.

To summarize this, methods that were chosen for a detailed review had to fulfill the following prerequisites:

1. Content-Based filtering only
2. Computational cost within realistic scope

3. Feasible effort for implementation

After an extensive review of literature on these methods, and a selection of methods and models to be implemented, it became clear that the methods could be separated: methods that solely calculate an importance value for keywords in a given corpus of documents, i.e. keyword based methods, and other methods, that work in different and so to say more complex ways. The following methods were chosen to be implemented and therefore further analyzed:

1. Keyword based models
   1.1. TF-IDF Scores
   1.2. TextRank Algorithm
   1.3. RAKE Algorithm
2. Wordcloud Comparison
3. Stylometry
4. Dynamic Topic Modelling
5. Word Embeddings
   5.1. Pre-Trained Model
   5.2. Self-Trained Model

Further methods and models were identified, that might be interesting for further research in this topic, but did not fit the previously defined prerequisites for analysis within this thesis.
Examples for this would be Dynamic Embedded Topic Model, LDA2vec, Support Vector Machines or Stylochronometry. A more detailed description of these methods and why they were excluded can be found after the listing of the implemented methods.

As already described above, the first three models focus solely on extracting keywords. Keywords, which can be defined as single words, as well as a sequence of words with one or more words, provide a compact and condensed summary of the essential content of a document or a corpus of documents. Keywords are widely used in many different application areas like natural language processing, information retrieval and enrichment of documents with metadata [RECC10].
Methods that commonly find their application in different setups are TF-IDF Scores, the TextRank Algorithm and the RAKE Algorithm. While TF-IDF calculates the importance values, that are used to rank the importance of extracted keywords, for every single word in every article separately, both TextRank and RAKE calculate their importance values not only for single words but also for word groups with a predefined length and for the whole corpus.

3.1 TF-IDF Scores

One of the, if not the most well known method for calculating an importance value for keywords in documents is the so called TF-IDF method, which stands for "term frequency, inverse document frequency". And as the name states, the formula combines the term frequency of a word within one document with its frequency throughout all documents within the corpus to calculate a specific value. Even though TF-IDF is a relatively old and rather simple weighting scheme, it is still very effective in finding keywords, which makes it a popular starting point for keyword extraction [SB88].
Given a document corpus D, a single word w, and an individual document $d \in D$, the TF-IDF score w_d for every word gets calculated via

$$w_d = f_{w,d} \cdot log(|D| f_{w,D}) \tag{3.1}$$

where $f_{w,d}$ is the number of times the word w appears in d, $|D|$ is the number of documents in the corpus D and $f_{w,D}$ is the number of documents where the word w appears in [R+03].
This algorithm calculates an importance value for every occurring word within a document, which means that the same word usually has two different TF-IDF scores for two different documents in the same corpus.
According to this formula, one can see that a word that receives a high score has to have both a high frequency within the document $f_{w,d}$ as well as a rather less common appearance through all documents $f_{w,D}$. This assures that only words that appear frequently within one document but not throughout all documents will receive high scores.
As an illustrative example, the word Jesus will appear very often in the New Testament, while not very often through other books in a library, and therefore will get a very high TF-IDF score for the New Testament. While words like articles or pronouns will not get a high TF-IDF score, as the second part of the formula punishes their frequency through close to all books within a library, even though they have a high term frequency in a specific book [R+03].

The next step, after calculating the TF-IDF scores for all words within the corpus, is to calculate a distance between articles that are close to each other according to their TF-IDF scores. The first step here is to bring the news articles into a vector form. For this, the document-term matrix of all documents and all words within the corpus with their TF-IDF scores is built, where every row is a sparse vector of the TF-IDF values of all words. If a word does not occur in a document, the value 0 is filled into the word column. An example for this can be seen in Table 3.1, where each line represents a document vector.

One popular method to build a similarity measure between texts, after representing them in a vector form, is the cosine similarity of the representative vectors. In this case, the similarity between the articles corresponds to the correlation between the vectors.

$TF-IDF$	ich	Baum	das	Boot	Stadt	klein
Article A	0,24	0,80	0,11	0,00	0,00	0,41
Article B	0,52	0,00	0,19	1,84	0,00	2,11
Article C	0,47	0,00	0,09	0,00	0,66	1,01

Table 3.1: Document-Term Matrix

Given two articles in their vector representation $\vec{t_a}$ and $\vec{t_b}$, their cosine similarity is

$$SIM_C(\vec{t_a}, \vec{t_b}) = \frac{\vec{t_a} \cdot \vec{t_b}}{|\vec{t_a}| \cdot |\vec{t_b}|} \tag{3.2}$$

With this definition, the cosine similarity SIM_C can be between 0 (no similarities at all) and 1 (all words appear in both articles) [Hua08].
This similarity is calculated for every document pair in the corpus and can now be seen as the similarity between two articles, the higher the cosine similarity, the closer or more related the articles are.

As the TF-IDF method basically only gives keywords with their importance factors as a result, finding trends has to be done manually afterwards. For this, multiple possibilities exist.
The most obvious, or maybe most appropriate way to do so would simply be to to look for the occurrences of the keywords throughout the time of publication of the articles. Keywords that appear more often in a specific period of time then would suggest a trend for this keyword in that period. A visualization of this can be created with a density diagram of the occurrences of a word, as it can be seen in Figure 3.1. Here the two names "Häupl" and "Strache" were compared for their frequency throughout the year 2018 and the first half of 2019.

Another option to find trends within the keywords would be to calculate the values of the keywords for different timespans. For example the TF-IDF value for a specific word can be calculated for the last 3 months ("short term"), as well as for the last 12 months ("long term"). Depending on if the value for the specific keyword stays the same or changes, this could be interpreted in various ways:

- *short term value* $\approx$ *long term value*
  In this case, the importance of the keyword basically stays the same. If the value is high, than the keyword is important for both short- and long-term, if it is low, it is rather not an important keyword.

- *short term value* $>$ *long term value*
  The keyword has a high value for short-term, but a rather low value for long term. This could mean that the keyword appears as a trend in the short-term, and was not important in the long-term, but it could also mean that the keyword appeared more

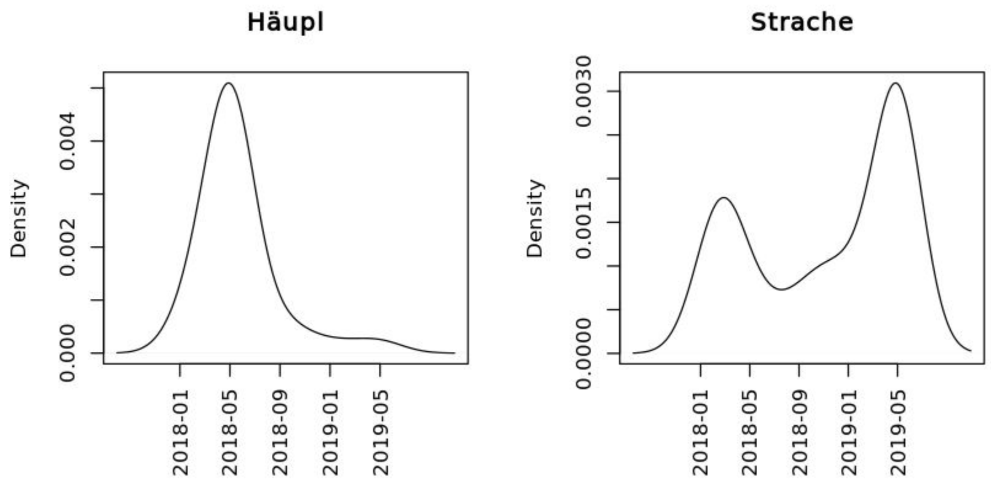


Figure 3.1: Keyword density

regularly throughout all documents in the long-term, and only in a few documents in the short-term.

- *short term value < long term value*
  Here the keywords seems to be very important through the long-term, but not as important in the short-term. This could mean that the keyword appeared more often in the past 12 to 9 months, but not so often anymore in the last 3 months.

When using this method to look out for trends within keywords, it is important to mention that the different cases can be interpreted in multiple ways. Therefore knowledge about the way the keyword values are calculated is very important when using this method, and even than the interpretation needs to be made with considering all possibilities and being very careful about their meaning.

3.2 TextRank Algorithm

The TextRank Algorithm was introduced by Mihalcea and Tarau in 2004 as a graph-based ranking model for text processing, that can be used for keyword and sentence extraction in natural language applications [MT04]. Like the well known PageRank algorithm, that is used by Google for the analysis of the link-structure of the World Wide Web, the TextRank Algorithm, as a graph-based ranking algorithm, is a method to decide on the importance of a vertex within a graph, by taking global information of the graph into account and recursively including it to rank the vertices, instead of only using the local information of the vertex for its importance [BP98].

The basic idea of a graph-based ranking model is to decide on the importance of a vertex via "voting" or "recommendation". One vertex linking to another vortex in this context means a "vote" for the vertex, the higher the number of votes, or rather links from and to a vortex, the higher the importance of the vertex is. Furthermore, the higher the importance of a vertex that gives the vote for another vertex, the more important the vote itself is. So not only the amount of links from and to any vertex, but also the importance of the other vertices that these links lead to, are used for calculating the importance. Let $G = (V, E)$ be a directed graph with the vertices V and the edges E, where E is a subset of $V \times V$. For any given vertex V_i, let $In(V_i)$ be the set of vertices that point to it (called its predecessors), and let $Out(V_i)$ be the set of vertices that the vertex V_i points to (called its successors). The score of a vertex $S(V_i)$ is then defined as [BP98]:

$$S(V_i) = (1 - d) + d \cdot \sum_{j \in In(V_i)} \frac{1}{|Out(V_j)|} S(V_j) \tag{3.3}$$

where d is the damping factor that can be set between 0 and 1, which has the role of integrating the probability of jumping from one vertex to another random vertex within the graph. In the homepage context, this implements the so called "random surfer model", where a human browsing through the World Wide Web clicks on links at random with a probability d, and jumps to a new page with the probability $1 - d$. This factor d is usually set to 0.85 [BP98].
Starting then from arbitrary numbers which get assigned to each vertex in the graph, the values are computed iteratively until convergence below a given threshold is achieved. This final score associated with a vertex represents the "importance" of the vertex within the given graph [MT04].
To finally use this graph-based ranking model for natural language processing, a graph that represents the text and interconnects the words or sentences with meaningful relations has to be built. To do this, the following main steps have to be done [MT04]:

1. Identify units of the text to use as vertices in the graph, e.g. words or sentences.
2. Identify relations between these units and use them as undirected edges between the vertices in the graph, e.g. words written next to each other.
3. Iterate the graph-based ranking algorithm until convergence.
4. Sort the vertices based on their final score to receive the most important ones.

In this thesis, the units described in step 1 were words within the text, which results in a keyword extraction model. For the relation between these units, n-grams with the length of 5 of the text were chosen and the relation were co-occurring words. This means that a co-occurrence of the words is given, when words appear within a text window of maximum n words. Furthermore this was done not with only single words, but with groups of words of length 2 and 3.

To finally find similar articles for recommendations, a distance has to be calculated. As with TF-IDF scores, building a document-term matrix with the previously calculated values of every single word can be used to calculate these distances and find similar articles. A possibility would also be to not only include single words with their score, but n-grams with lengths 2 or 3 of the texts and their scores as columns into the document-term matrix, to consider them with their scores as well.
But this was not done in this thesis, as this requires way more computational resources like memory or processing speed, which were not available.

As this method is keyword based, finding trends can be done in the same way as it is described in the TF-IDF method chapter.

3.3 RAKE Algorithm

The RAKE method was developed by Stuart Rose, Dave Engel, Nick Cramer and Wendy Cowley in 2010, where the authors describe it as an unsupervised, domain-independent and language independent method for keyword extraction from text documents. They also mention that RAKE is computationally more efficient while achieving similar precision and recall scores for keyword extraction as the already described TextRank Algorithm does. The main goal of RAKE is being efficient, operating on individual documents and being easily applied to new domains while working independent from grammar and language [RECC10].

To achieve all this, RAKE functions in multiple steps, that are processed for every document independently [RECC10]:

1. Identify the candidate keywords
   Based on the observation, that keywords rarely contain any stopwords like *and*, *the*, and of, RAKE not only does not include these stopwords from keyword candidates, but it uses theses stopwords, together with punctuational signs, to split the content bearing words into candidate words or groups of words with a predefined maximum length. If a keyword candidate would exceed the predefined maximum length, all possible n-grams, where n being the maximum length, are extracted as candidates. This results in keyword candidates with varying lengths up to the maximum length.

2. Counting co-occurrences
   A graph of co-occurrences of all single words within the extracted keyword candidates is built. Every single word in this graph counts as co-occurring with itself, so if the maximum length for keyword candidates would be zero, this graph only counts the occurrences of the words, otherwise co-occurrences of other words are included as well.

3. Calculating keyword scores
   RAKE calculates three different scores for every single word within the keyword

candidates and afterwords combines them for candidates consisting of more than one word.

- $freq(w)$
  The absolute frequency of this word within the document
- $deg(w)$
  The word degree, which is the frequency added to the co-occurrences of the word with other keyword candidates
- $deg(w)/freq(w)$
  The ratio of the degree to the frequency

$deg(w)$ favors words that occur often in longer keyword candidates, $freq(w)$ obviously favors words that occur frequently with no consideration to the length of longer keywords candidates and $deg(w)/freq(w)$ favors candidates that predominantly occur in longer keyword candidates [RECC10]. The implementation of this method, as described later in this thesis, used the third option.

Again, as described in the TextRank Algorithm chapter, the final step will be to find similar articles, which can be done via building the document-term matrix with the calculated scores and calculating the cosine similarity between every pair of articles.

As this method is keyword based, finding trends can be done in the same way as it is described in the TF-IDF method chapter.

3.4 Wordcloud Comparison

A wordcloud, or tag cloud, is a visual representation of a set of words, typically of key words, in which various attributes of the the text, such as size, weight or color can be used to represent features of the associated terms, examples would be the frequency or the importance of these words [HK07]. Based on this, all three previous methods for keyword extraction can be used to calculate the importance for keywords and afterwards graphically represent them with wordclouds.
An example of such a wordcloud can be seen in Figure 3.2, the textual input for this wordcloud comes from a small sample of the given test data.

But not only can wordclouds be used to visualize extracted keywords from a text corpus, they also give the possibility to compare different texts with each other. This has been done for various text analysis purposes, for example Pyle et al. [PBM13] describe a method to find similar or distinctive wordclouds through building intersections of two word clouds, that represent two different text documents and compare them based on their representations. For this thesis, the representation of each article as a wordcloud and the following comparison would not be fitting, as the number of articles is too big, but still the comparison of wordclouds can be used to compare different time intervals to find keywords that appear more frequently in one period and not so frequently in other

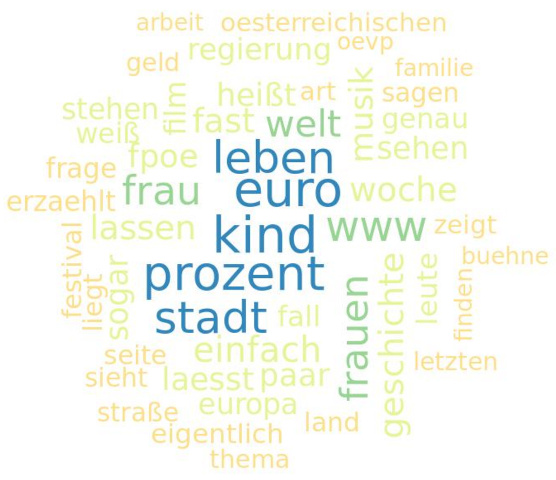


Figure 3.2: Wordcloud example

periods, thus representing a trending keyword.
The R-package *wordcloud* gives the possibility for the representation of wordcloud comparisons. The underlying algorithm compares multiple texts with each other. To achieve our goal of comparing time periods with each other, the documents of one period have to be merged together and given to the algorithm as "one document". The *comparison.cloud* method of the *wordcloud* R-package than calculates the importance, which is represented as the size of a keyword in the cloud, based on its exclusivity in one document, or rather time period in comparison to the others [Fel18]. Formally speaking, let $p_{i,j}$ be the rate at which a word i occurs in document or time period j, and n the number of documents or time periods considered, p_j is the average rate across all documents i:

$$p_j = \sum_i \frac{p_{i,j}}{n} \tag{3.4}$$

The size of each word is than mapped to the maximum deviation of the value $p_{i,j}$ to the average p_j $(max_i(p_{i,j} - p_j))$, and the angular position and color is defined by the document or time period i, to which the value $p_{i,j}$ belongs [Fel18]. An example for this, which is created by using all articles from the years 2015 until 2019 of the given test data set, can be seen in Figure 3.3.

This algorithm, which is used for calculating the size of every word can also be used in a non-graphical way to calculate the importance of every word as well, where the resulting value for every word is just stored, as it is done with TF-IDF, textrank and RAKE, in a document-term matrix with the importance values, which can be built to finally calculate the closeness of articles via the cosine similarity of every article pair.

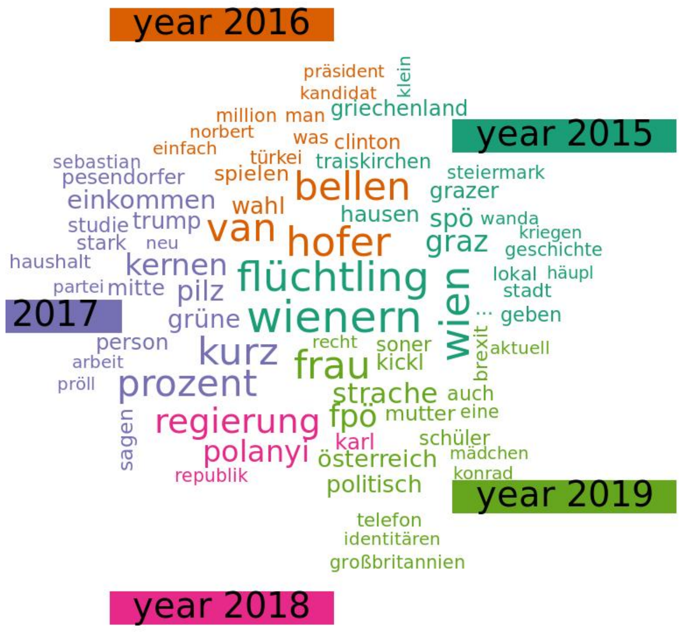


Figure 3.3: Wordcloud Comparison example

3.5 Stylometry

The base idea of stylometry is to analyze the writing style of the author of a text to compare it with other texts. First applications of stylometry used the frequency of word lengths for the suggestion of an authorship of texts and dates back all the way to 1851 to Augustus de Morgan [Hol98]. Through the years more modern and computer assisted ways of stylometry emerged, especially through the emergence of digital technologies, machine learning and other natural language processing methods. Nowadays stylometry is used for various tasks like plagiarism detection, authorship recognition or distance creation between texts based on their style [NSF$^+$17].
Latter one will be the focus for this thesis, to use stylometry for recommendation of news articles based on their similarity in writing style.
For all these tasks, stylometry calculates a number of features for every text, which then will be used for further analysis. The most prominent feature is the MFW-feature, which describes the most frequent words of every article, and is used as the basis for further multidimensional analysis. But it is also useful to consider n-grams of words or characters for every text. N-grams can be seen as a sliding window over the text with the length n. For example the sentence *This is a test* separated into 2-grams of words would be split into *{(This is),(is a),(a test)}* and into the 2-grams of characters *{(Th),(hi),(is),(s),(a),...}* and so on. It has been shown, that using not only the frequent words, but also the n-grams in combination with the frequent word lists, results in a way more robust

analysis when dealing with natural language texts, especially when the written text might include dialect terms [Ede13].
Choosing the size of the n-grams can lead to different results of the stylometric analysis. The ideal size can depend an various factors, most importantly the language, but varies also from corpus to corpus [ER11]. Additionally, other parameters have to be chosen for the analysis of the corpus that may have an influence on the stylistic analysis. This includes the following parameters:

- *Words or characters*
  Using either whole words or only characters, that are chosen for the n-grams.
- *N-gram size*
  The size of the sliding windows over words or characters. Choosing 1 for n results in only considering single words or characters with only their frequency. When using words, naturally a higher number for n can lead to more variety in the n-grams, meaning less repetitiveness, which then leads to poorer statistics due to higher sparseness.
- *Preserve case*
  Preserving lower / upper case for letters, it might be useful to preserve the case, especially in the German language, as the same word written in upper or lower case might have different meanings.
- *Minimum word frequency*
  The minimum amount of the most frequent words or character-groups of the whole corpus that will be included into the relevant attribute list.

Stylometry, or more specific the stylometric analysis implemented in the *stylo* R-package includes multiple unsupervised methods, like principle component analysis of extracted stylistic attributes, like often used words or the length of sentences, or multidimensional scaling, that can be visualized with graphs and will need human interpretation for a detailed analysis of the results [ME19]. An example for these graphs would be the visualization of the principle component analysis of all articles written by three different authors in the year 2018, as can be seen in Figure 3.4. This graphic can be used to compare the writing styles of different authors, which are represented in different colors, with each other or to compare the articles of one author regarding their writing style. Conclusions that could be drawn would be for example if an author has a big variety in their writing style, if some articles are outliers for one author compared to his or her other articles, or if different authors have a comparable writing style.

Another way to analyze the dataset and visualize the results via stylometric analysis is the cluster analysis of the articles as shown in Figure 3.5 with the same articles and authors as in the example for the principle component analysis. This representation gives insights in what articles are similar to each other based on stylistic attributes, what can later be used for the recommendations of articles to a given article.

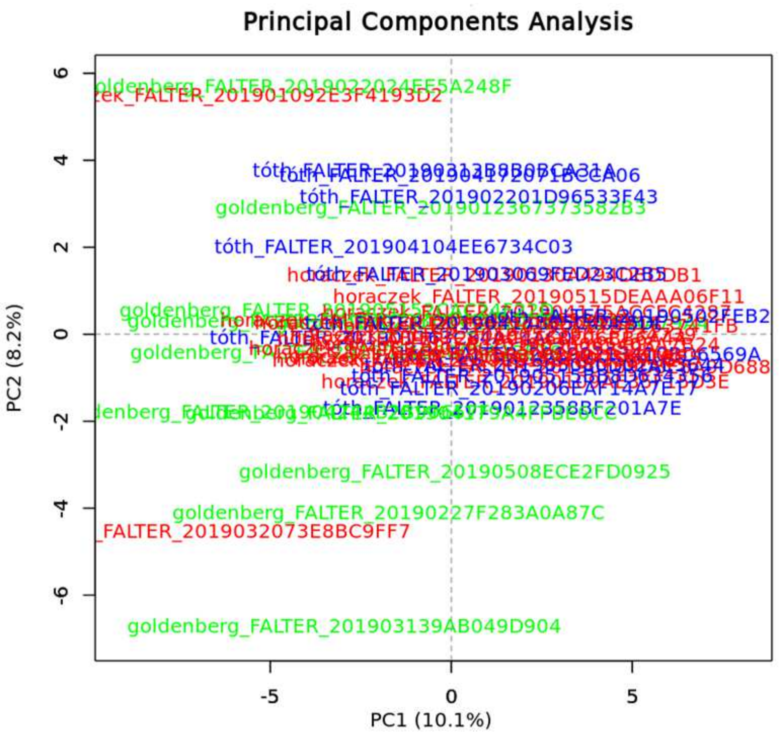


Figure 3.4: Stylometry PCA

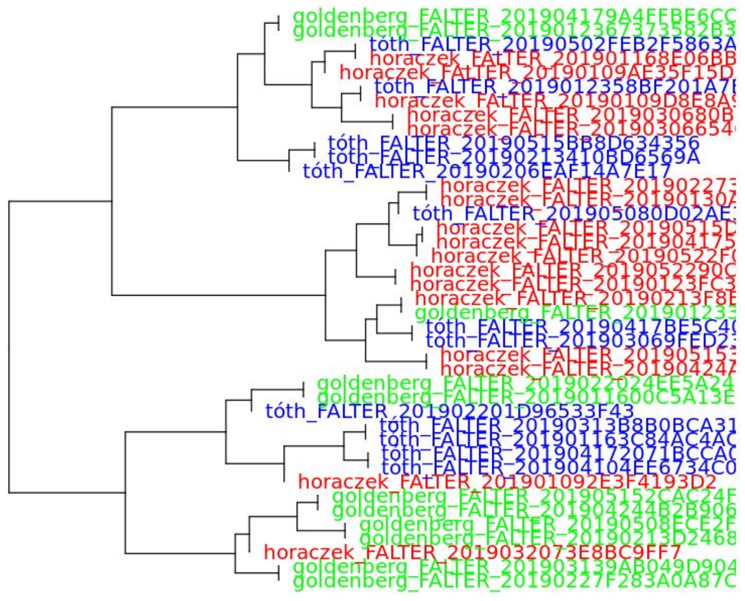


Figure 3.5: Stylometry Cluster-Analysis

3.6 Dynamic Topic Modelling

The term Dynamic Topic Modelling already describes that this method is not only reduced to topic modelling in itself, which focuses on extracting underlying topics within all documents of a corpus, but also includes a dynamic aspect of the topics. Meaning that this method tries to extract topics and correlate these topics with a time component, to find trends within the topics.

To achieve this goal, structural topic modelling (STM) was used in this work to build a dynamic topic model. STM, like many other topic modelling methods, is a model based on generative word counts. Which means that a data generating process is defined first to extract information in the form of words from the documents and then to use this information to find the most likely words for every topic. A topic in this model is defined as a mixture of words, where every word within the topic has a probability of belonging to the topic, and a document can then be described as a mixture of various topics through its containing words and their respective probabilities of belonging to these topics. This means that every document can contain various topics [RST+14].
Unfortunately, STM requires the number of topics within all documents, to be "known" in advance, so this number is needed as a parameter for the model. There are multiple ways to find suggestions for the number of topics depending on different attributes of the corpus. The creators of the *stm* R-package for example write in their manual that there is no right or wrong number of topics, but that a lower numbers leads to more general topics, while a higher number results in more specific topics on the one hand, but on the other hand can lead to less precise estimations of the topics. They also give basic suggestions like 60-100 topics for a corpus with 100 000 documents or more, which would fit our given corpus. But there are also algorithms that try to optimize this number of topics for a specific corpus.
Mimno & Lee [ML14] for example describe an algorithm that can be used to optimize the number of topics by maximizing the semantic coherence of a resulting topic model with k topics. Semantic coherence in this context describes the meaningfulness of the extracted topics. The result of their algorithm again cannot be seen as the *true* number of topics, but it can be used as a starting point for manual optimization, if needed. Figure 3.6 shows an example for the resulting semantic coherence with different numbers of topics k, in this case, k = 20 topics would result in an ideal topic model according to the semantic coherence.

The result of the topic modelling process of STM finally are topics with their associated words. These topics of course are not labelled yet, which means they manually have to be analyzed and given a label for further usage. In Figure 3.7, an example for the resulting topics and their three most prominent words is given. Additionally, further words for every topic can be viewed to help understanding and labelling the topics. Four different types of words are given for every topic:

1. Highest probability words
   Words that have the highest probability to occur within this topic

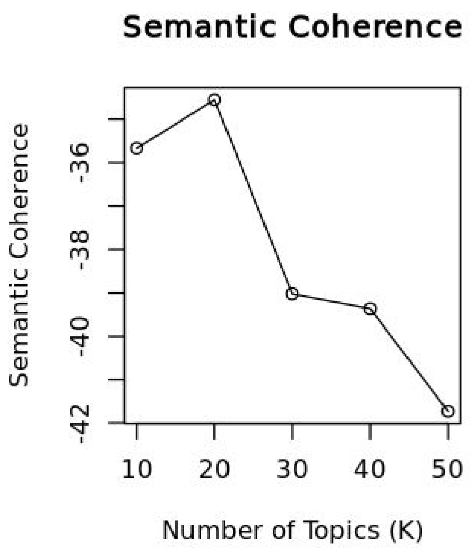


Figure 3.6: Optimizing semantic coherence

2. FREX words
   Words weighted by their overall frequency and their exclusivity to this topic

3. Lift words
   Words with a high frequency in this topic divided by their appearances in other topics

4. Score words
   Similar to Lift, Score takes the logarithm of the word frequency in this topic and divides by the logarithm of their appearances in other topics.

An example for these words can be seen in Figure 3.8 for one specific topic.

The last step, to get from basic topic modelling to dynamic topic modelling, is to include the time component. In STM this can be done with metadata that can be added to each document, more specifically we add the publishing date of each article as metadata. STM then can calculate a correlation between this metadata and each topic and it enables us to visualize this correlation. An example of this correlation between already labelled topics and their appearances through a corpus can be seen in Figure 3.9. It is important to add to this graphic that a topic proportion of less than 0 is not possible, values in the graph that reach a proportion less than 0 are only a result of the smoothing of the graph. This means that only upwards peaks can be interpreted as trends for the respective topic. For example one can see that the topic labeled as “germania liederbuch” was very prevalent in January, while hardly appearing through the rest of 2018.

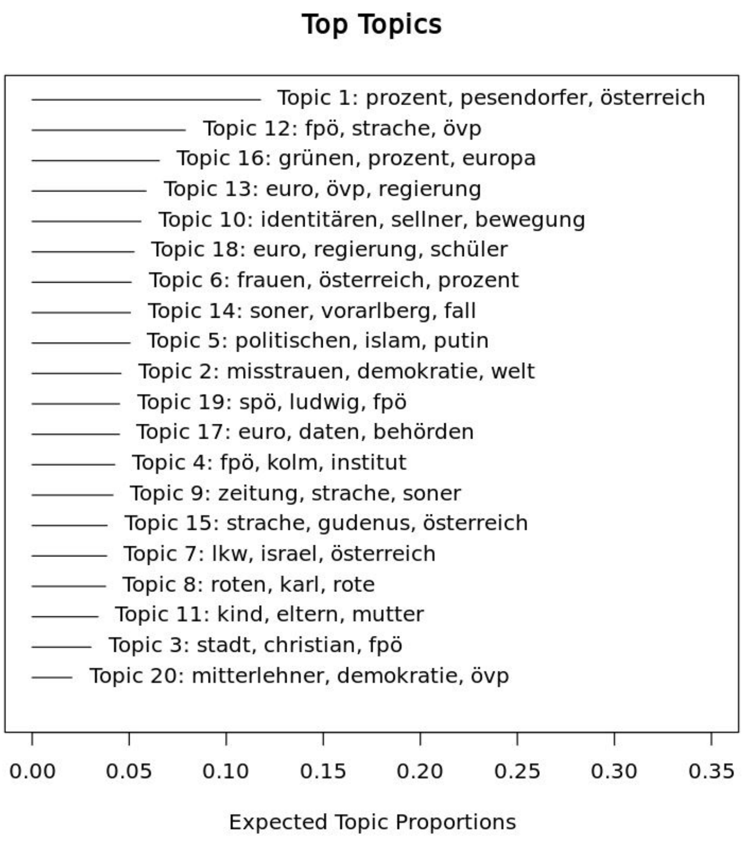
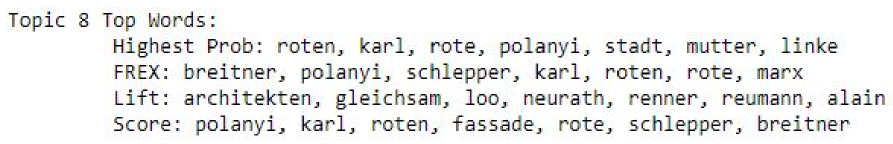


Figure 3.7: Topic proportions



```
Topic 8 Top Words:
         Highest Prob: roten, karl, rote, polanyi, stadt, mutter, linke
         FREX: breitner, polanyi, schlepper, karl, roten, rote, marx
         Lift: architekten, gleichsam, loo, neurath, renner, reumann, alain
         Score: polanyi, karl, roten, fassade, rote, schlepper, breitner
```

Figure 3.8: Topic words

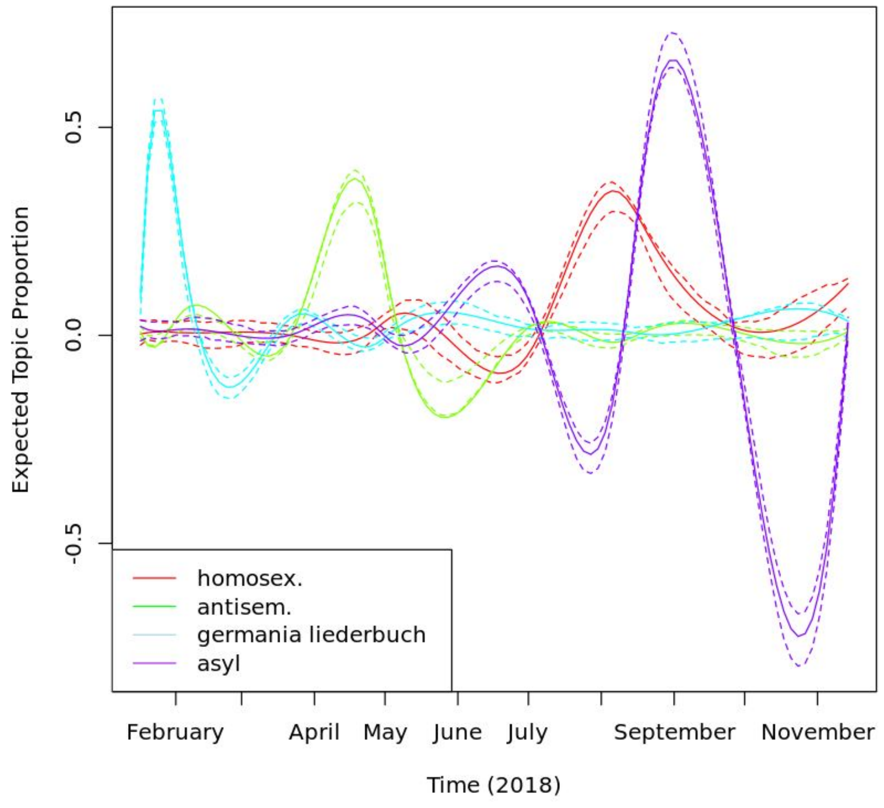


Figure 3.9: Topic trends

The final step is to find articles to recommend to a given article. In STM it is possible to receive the topic or topics that correlate the strongest for every given article. We decided to get the topic with the strongest correlation to the given article and find other articles, that have a strong correlation to the same topic, as the recommended articles.
Another possibility would be to build a correlation matrix for all articles with all topics, and again calculate the cosine similarity for the articles based on this matrix. We did not use this method, mainly because receiving the correlation to every topic for every article is computationally very extensive and takes unreasonably long in the given test environment.

3.7 Word Embeddings

A rather new method, that is used for natural language processing and text recommendations is the usage of word embeddings. A big drawback of most other methods that are described in this thesis is, that they focus on the words themselves instead of the semantic meaning of these words. For example two sentences like *"The dog likes to eat*

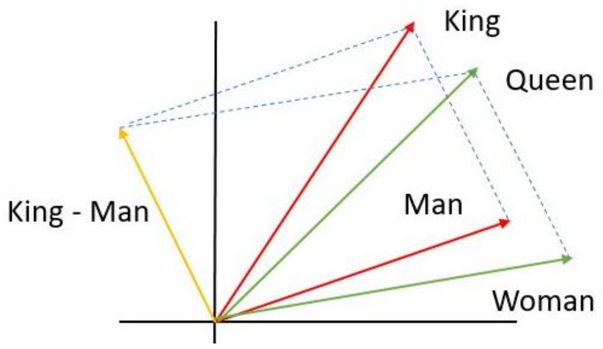


Figure 3.10: Word Embeddings example [wep]

meat" and *"The puppy prefers to chew on sausages"* have no words in common after removing the stopwords like *the* and *to*, but still have a very similar meaning. Methods like TF-IDF, which only use full words, would not find any similarity between these sentences. This means that most other methods don't include the distance or similarity between single words or word pairs of the documents, like *dog* and *puppy*, which, while being different words, still contain a very similar information.

An approach to overcome this problem is the *word2vec* model [MSC$^+$13]. This model creates word embeddings, which means it calculates a representative vector for every word within a corpus of documents. Naturally, the quality of the calculated embeddings scales with the size of the given corpus that is used for creating the word vectors. The authors of this model demonstrate the capabilities of these word embeddings with examples like the following:
When $vec(w)$ describes the vector that represents the word w, than for example $vec(Berlin) - vec(Germany) + vec(France)$ results to a vector that is very close to $vec(Paris)$, so the difference between $vec(Berlin)$ and $vec(Germany)$ could be seen as the representation of $is\ capital\ of$. Another example would be $vec(King) - vec(Man) + vec(Woman)$ is close to $vec(Queen)$, where the difference between $vec(King)$ and $vec(Man)$ or $vec(Queen)$ and $vec(Woman)$ would describe a neutral form of $ruler$. A graphical representation of this can be seen in Figure 3.10 [wep].

Using not the words themselves for further processing but their embeddings would help to overcome the before mentioned problem, and help seeing different words with similar meanings not as completely different, but rather similar words.

Different methods for calculating these word embeddings have been developed since the emergence of word embeddings. For this thesis, two different models of word embeddings have been used for a comparison of these. On the one hand, a pre-trained model, that was calculated via the continuous bag-of-words method with position-weights and that was trained on german Wikipedia articles with 300 dimensional vectors was used [fas] [BGJM17]. On the other hand an own model was trained on the given test-dataset of newspaper articles using the R-package *text2vec* and a GloVe-model to calculate word vectors with 150 dimensions [DS18].

The next step, after creating the word vectors themselves, is to build a model to compare documents with each other. One method that can be found in multiple sources is to build document vectors for every document within the corpus [LS18] [MWHL17]. This method combines the importance of a word within a single document with its vector representation, which is done via multiplying the TF-IDF Score of each word with its word-vector. All the resulting vectors for every word within a document than get summed up to build the document vector. Mathematically speaking, let $tfidf(w, d)$ be the TF-IDF score of word w within document d, $vec(w)$ be the representing vector of word w, the document vector $vec(d)$ for document d can be calculated via [LS18]:

$$vec(d) = \sum_{w\,in\,d} vec(w) \cdot tfidf(w, d) \tag{3.5}$$

The similarity of these document vectors can then be calculated by the cosine similarity. It is described as a final step to afterwards divide the resulting document vector by the sum of all TF-IDF scores of the respective document to normalize the length of the document vector, but this has no impact for this thesis, as we are only interested in the cosine similarity, where the length of each vector has no influence.

To summarize the word embeddings method:

1. *Word vectors*
   Word vectors have to either be trained on the dataset or taken from a pre-trained model

2. *TF-IDF Scores*
   The TF-IDF Scores for every document have to be calculated for weighting the word vectors

3. *Document vectors*
   Combining word vectors with TF-IDF scores and summarize for each document

4. *Similarity*
   Calculate similarities of documents via cosine similarity of document vectors

3.8 Excluded Methods

This chapter summarizes methods and models that were identified during the literature review of news recommendation methods, but did not fit the defined prerequisites for this thesis. The methods, a quick summary of how they work and a description of why they were excluded is given in this chapter.

3.8.1 Support Vector Machines

In machine learning, the term support vector machine describes a concept for supervised machine learning models with the base idea of creating one or more hyperplanes in a space defined by attributes of the data points, to seperate the data points into subsets for classification or regression purposes. How these hyperplanes are calculated depends on the specific model, for example it could be described by having the largest possible distance to its nearest data points [HTF09].
One specific method, to use these models for news article recommendation was described by Gershman et al. [GWFC11]. In this specific method, first some content-based attributes, like TF-IDF Scores with document vectors and their cosine distances, as described in the TF-IDF chapter in this work, were calculated. Then some further attributes that are based on user preferences were not only initially, but also continually added to the data set. The resulting hyperplanes devide the corpus into articles that are relevant for the user and articles that are not relevant to the user.
The main reason why this method was not implemented in this thesis was that we could not find an application of support vector machines for news article, or more generally for text recommendation as well, that only used content-based attributes. While some works included them, the main focus was always put on the user preferences for the machine learning algorithms.

3.8.2 Dynamic Embedded Topic Model

The Dynamic Embedded Topic Model can be seen as somehow similar to the Dynamic Topic Model, that was implemented in this thesis. But with the main difference of not using the words itself, or n-grams of words of the articles, but rather using word embeddings. Dieng et all. [DRB19] described it as a combination of the dynamic topic model that uses the dynamic LDA algorithm, with the embedded topic model, that uses a continuous representation of the words or bags of words. So the dynamic embedded topic model is based on word embeddings, topic models and dynamic topic models.
According to the authors, this combination helps to overcome weak points of the single elements that are combined in this method, e.g. improving the stability of the found underlying topics. It is concluded that the dynamic embedded topic model outperforms the dynamic topic model both in terms of predictive performance and topic quality while requiring significantly less time to fit.
At the time of the creation of this thesis, no R-package or Python script or repository of the authors was available to use. The effort to create this algorithm from scratch would exceed the limits of this thesis. Therefore this model was excluded.

3.8.3 Stylochronometry

The idea of stylochronometry is to find timely changes in styles of a text corpus. Very similar to how the dynamic topic model, that was used and described within this thesis stands to the basic topic model, stylochronometry tries to include the creation date of a

news article via regression of it to the resulting style attributes of the stylometric analysis [KV15]. The cited work shows that stylochronometry can be used to find changes of the style of famous authors like Henry James and Mark Twain, which can be attributed to their changing lifestyles.
While the basic idea of finding changes in the used style of the news articles sounds promising for this thesis, its main purpose and strength lies within finding changes of the style of one specific author through the time. When executing this model on a dataset of articles from various authors, the amount of articles that are written by each author would have more influence on the results than the changing style of single authors. Additionally, finding changes in the style of specific authors was not included in the scope of this thesis, therefore this method was excluded from the used models.

CHAPTER 4

Data

In this chapter the received dataset is described along with some general findings of the articles within it, as well as the general preprocessing steps that were done to prepare the dataset for all further methods and models. Model specific preparation steps of the data are described along with the implementation of the models themselves in the following chapter, here only preprocessing steps for general data sanitation and data understanding are described.

4.1 Raw Dataset

The dataset in use is provided by *Der Falter*. *Der Falter* is a Viennese newspaper that is published weekly. It was founded in 1977, has no political affiliation, and is described as reporting from a left-liberal perspective. The main topics that are covered consist of politics, media, culture and life in Vienna [Tra07].

The given dataset contains a full data extract of the article database for the timespan beginning at 5.8.1998 until 22.5.2019. The full dataset contains 99.286 articles that were published throughout these 21 years. The structure of this dataset can be seen in table 4.1.
Some results from the explorative data analysis include the following:

- The whole corpus contains 99.286 articles including their metadata.
- 7.502 different authors are included.
  This does not mean, that there are 7.502 different real persons. Every combination of authors that co-wrote an article is seen an author as well.
  Furthermore, there are 45.230 articles where no author is existing in the dataset. These articles can for example not be used for author comparison. The distribution

Raw Dataset Structure			
Column	Datatype	Description	Example
article_id	String	Unique ID of Article	FALTER_201905226CF056D225
title	String	Article title	Ein ehrlicher Zirkus, Manege frei!
dachzeile	String	Header line of Article	Menschen und Tiere
subtitle	String	Article subtitle	Bücher, kurz besprochen
date	date-time	Publishing date	2019-05-22T00:00:00.000+00:00
authors	String	Article authors	Nina Horaczek
ressort	String	Department of Article	Politik
text	String	Article Text	Während bereits der Politzirkus um das Ibiza-Video ...
issuenr	Integer	Issue nr. where article was published in	201921

Table 4.1: Structure of the raw dataset

of the authors with the most articles and their number of written articles can be seen in Figure 4.1.

- The dataset contains articles from 239 different ressorts. The ressorts with the most articles are:
  Stadtleben - 18.012 articles
  Lexikon - 16.344 articles
  Politik - 14.303 articles
  Only a total of 972 articles is not labeled with any ressort at all.
- 1.042 different issues are included in the dataset, with an average of 95,28 written articles per issue. But this number changed throughout the years that are included. This change is depicted in Figure 4.2.
- On average, an article consists of 530 words or 3.127 characters. The shortest article has only 2 words or 26 characters, while the longest article contains 82.906 words or 314.915 characters. The article that contained only 2 words was a joking article about a politician.
- The most used words can be found in Table 4.2, on the left side are the most used words in general, on the right side are the most used words after filtering out so called stopwords of the German language.

4.2 Pre-Processing

In this chapter we describe our general data preprocessing steps. Model dependent preprocessing steps are described in upcoming chapters along with the corresponding models.

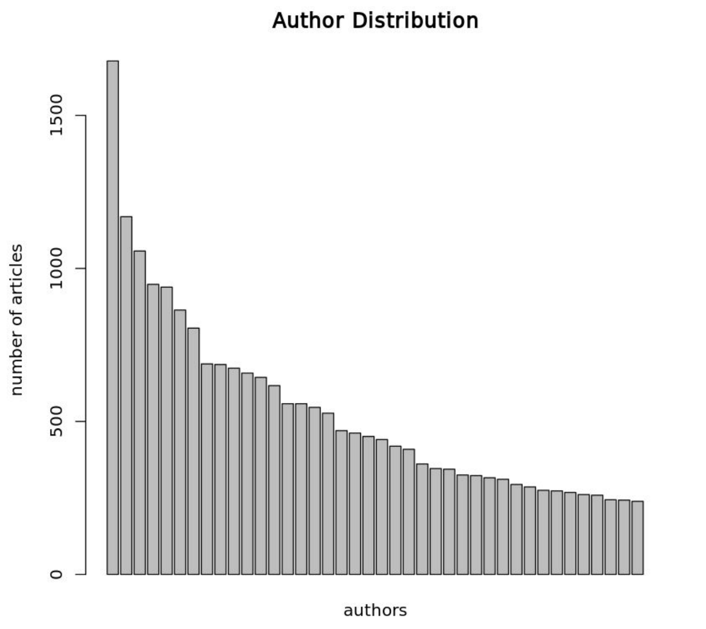


Figure 4.1: Authors with most written number of articles

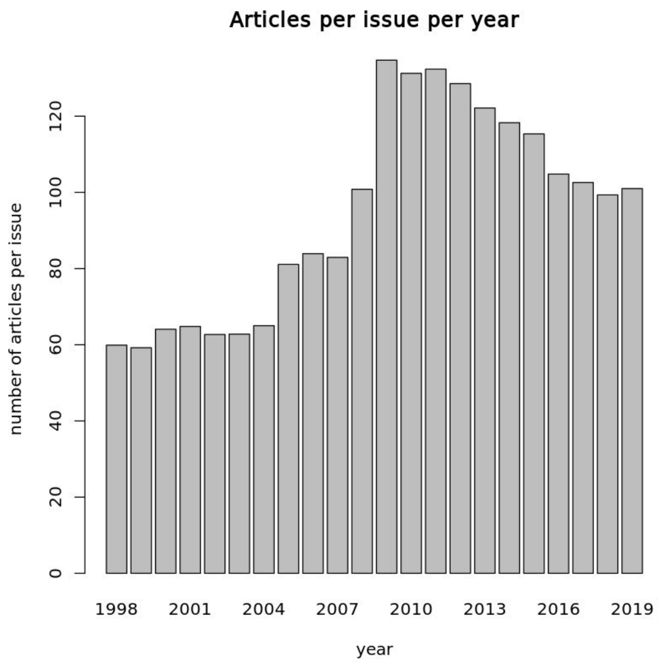


Figure 4.2: Articles per Issue per Year

All words		No Stopwords	
Word	Count	Word	Count
die	1.390.005	leben	31.126
der	1.279.479	www	29.437
und	1.078.593	Euro	28.442
das	590.967	Stadt	26.482
den	452.463	Welt	25.784
ist	445.489	Prozent	23.343
mit	410.719	lassen	22.954
von	381.404	Frau	22.576
nicht	372.477	Musik	21.919
ein	370.513	einfach	21.032
sich	340.866	Woche	19.979
auch	289.647	sehen	19.596
auf	289.547	Kinder	19.488
sie	288.719	Geschichte	19.364
eine	281.684	Letzten	19.278

Table 4.2: Most used words

The dataset was given to us in the .xml format. This format is already very versatile and user friendly, but has the downside of a slow reading and writing speed. Therefore in the very first step, the full dataset was imported from the .xml file via the R-package *XML* [CT20] into a R-Dataframe and stored again as a .csv file, for much quicker read and writing times.

The preprocessing than included multiple steps, which were the following:

1. Separation of text and metadata
   The text data, which is held in the *text* column of the raw dataset, was split from the metadata of each article into its own table, that only contains the *article_id* and the *text* column, therefore the *article_id* column functions as a kind of foreign key.

2. Tokenization of text
   The text was tokenized into single words, so for every single word, one row in the text-table exists now. This was done with the help of the R-package *tidytext* [GDQ20].
   To further ease up the computational effort and memory usage, the tokenized data was also separated into one file for every calendar year, that is included in the data. So i.e. one file exists, that only consists of all words for all articles that were published in the year 2018. By doing this, subsets of the data for smaller

experiments could easily and quickly be accessed.

3. Sanitation of words
   To not separate between words in upper and lower case, all letters in all words were changed to lower case. This helps to not differentiate between the same word once used i.e. at the beginning of a sentence and the same word in the middle of a sentence. Special characters, especially the Umlauts ä, ö & ü, as well as ß, which are used very frequently in the German language, are kept in the general dataset. Noteworthy here is that stopwords, as well as punctuation, which usually are excluded from the dataset very early on in text mining operations, are still kept within the data on purpose. The reason is that some methods, especially *Stylometry*, are not only using, but heavily relying on both stopwords and punctuation. Those methods that excluded stopwords used the *stopwords-iso* list of German stopwords[Dia20]. Which methods did this is mentioned in each methods respective chapter in this work.

4. Extract further metadata for text
   Further metadata to the articles, or rather for every word, was extracted via the R-package *spacyr*, which is a wrapper for the spacy python library [KB19]. Following data was extracted via this R-package:

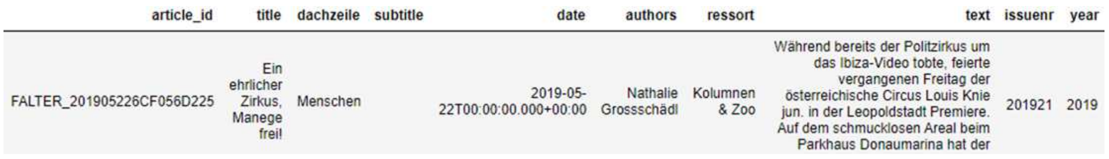
   - Lemmatization - The lemma, which is the base form of every word, was extracted. An example would be the german words *toben, tobte, tobt, getobt*, they all have the same lemma *toben*. For the lemmatization, the *hash_lemmas* list of the R-package *lexicon* was used [ea19].
   - Part of Speech Tagging - For every word, the type of word that this word belongs to, was extracted via the *spacy* R-package. *Politzirkus* for example is a *noun*, or *tobte* is a *verb*.
   - Named Entity Recognition - All words, that are a real world object, or part of the name of a real world object, are marked as a named entity. The *spacy* R-package distinguishes for the german language between locations, persons, organisations and miscellaneous.

A shortform of the used algorithm for the preprocessing can be found in algorithm 4.1.

An example, what the dataset has looked like before the preprocessing can be seen in Figure 4.3. What the dataset after the preprocessing looked like, including the data model, can be seen in 4.4.

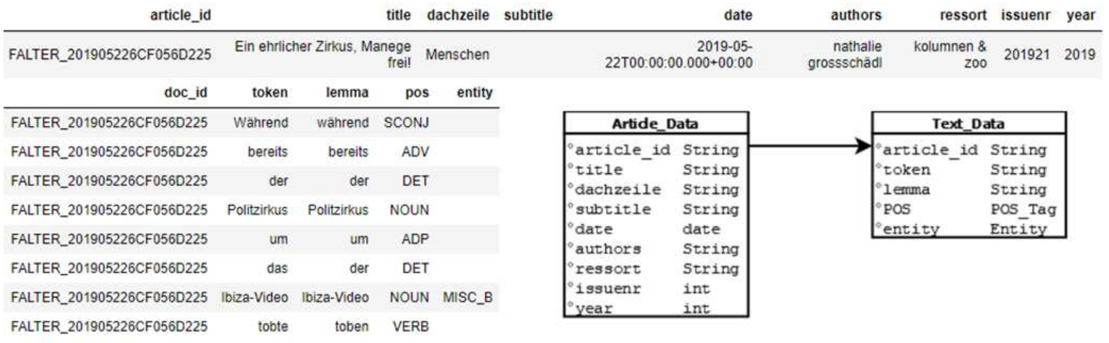
Algorithm 4.1: Preprocessing

```
xmldata <- fread("./falter_data_full.csv")
text_data_full <- select(xmldata,article_id,text)
text_data_full <- unnest_tokens(text_data_full,word,text)
fwrite(text_data_full,
   "./falter_data_full_token.csv",
   row.names = FALSE)
article_data_full <- xmldata[,-8]
fwrite(article_data_full,"./falter_data_full_notext.csv",
   row.names = FALSE)
article_data_full <- fread("./falter_data_full_notext.csv")
text_data_full <- fread("./falter_data_full_token.csv")
for(i in c(1998:2019)){
   article_data <-
       article_data_full[article_data_full$year == i]
   text_data <-
       text_data_full[text_data_full$article_id %in%
           article_data$article_id]
   name <- paste("./falter_data_token_",i,".csv",sep="")
   fwrite(text_data, name, row.names = FALSE)
}
for(i in c(1998:2019)){
   name <- paste("./falter_data_token_",i,".csv",sep="")
   data <- fread(name)
   data$lemma <- lemmatize_strings(data$word)
   data$stem <- text_tokens(data$word,stemmer="de")
   fwrite(data, file = name)
}
```



article_id	title	dachzeile	subtitle	date	authors	ressort	text	issuenr	year
FALTER_201905226CF056D225	Ein ehrlicher Zirkus, Manege frei!	Menschen		2019-05-22T00:00:00.000+00:00	Nathalie Grossschädl	Kolumnen & Zoo	Während bereits der Politzirkus um das Ibiza-Video tobte, feierte vergangenen Freitag der österreichische Circus Louis Knie jun. in der Leopoldstadt Premiere. Auf dem schmucklosen Areal beim Parkhaus Donaumarina hat der	201921	2019

Figure 4.3: Before preprocessing



article_id	title	dachzeile	subtitle	date	authors	ressort	issuenr	year
FALTER_201905226CF056D225	Ein ehrlicher Zirkus, Manege frei!	Menschen		2019-05-22T00:00:00.000+00:00	nathalie grossschädl	kolumnen & zoo	201921	2019

doc_id	token	lemma	pos	entity
FALTER_201905226CF056D225	Während	während	SCONJ	
FALTER_201905226CF056D225	bereits	bereits	ADV	
FALTER_201905226CF056D225	der	der	DET	
FALTER_201905226CF056D225	Politzirkus	Politzirkus	NOUN	
FALTER_201905226CF056D225	um	um	ADP	
FALTER_201905226CF056D225	das	der	DET	
FALTER_201905226CF056D225	Ibiza-Video	Ibiza-Video	NOUN	MISC_B
FALTER_201905226CF056D225	tobte	toben	VERB	

Figure 4.4: After preprocessing and data modell

CHAPTER 5

Data Analysis

In this chapter we show how we employ the methods introduced in chapter 3 onto the news dataset (see section 4.1). Furthermore, we analyze and discuss the results of each method.

5.1 TF-IDF

The TF-IDF method, as it is described in chapter 3, only uses words as tokens and weighs them for their importance within a given corpus. Therefore, we first exclude stopwords using the german iso-stopwords list. The explorative data analysis showed that some words within the corpus of news articles do appear as often as, or sometimes even more often than many stopwords, for example "Wien", "Wiener", "Stadt" or "Prozent". There are no noticeable differences in the results of the TF-IDF analysis with or without the filtering of these words. Which is due to the way TF-IDF works, as words that appear regularly throughout the corpus get punished with a lower score.
After this step, our corpus now consists of only non-stopwords. To calculate the TF-IDF scores of every word, first the occurrences of every word within their article have to be counted

```
article_words <- count(data, article_id, words, sort = TRUE)
```

With data being the dataset of words, and article_id and words being the names of the columns with the article id and the word itself.
Next, the number of occurrences of every word within the whole corpus has to be counted

```
total_words <- article_words %>% group_by(article_id) %>%
                  summarize(total = sum(n))
```

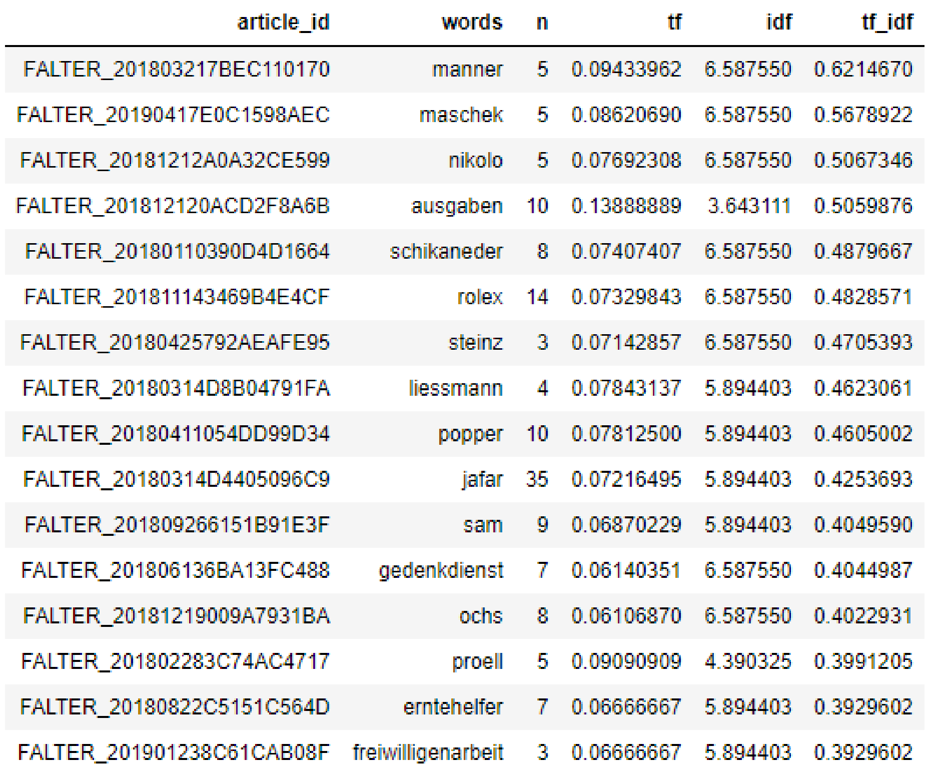
By joining these two results together, finally the TF-IDF scores can be calculated with the help of the *tidytext* R-package [GDQ20]

```
article_words <- left_join(article_words, total_words,
                           by="article_id")
article_words <- article_words %>%
                      bind_tf_idf(words, article_id, n) %>%
                      select(-total) %>% arrange(desc(tf_idf))
```

Finally, one can list top scoring words as following:

```
top_n(article_words,15,tf_idf)
```

and will look like the table seen in Figure 5.1, in this example all words from the corpus of the years 2018 and 2019 have been used.



article_id	words	n	tf	idf	tf_idf
FALTER_201803217BEC110170	manner	5	0.09433962	6.587550	0.6214670
FALTER_20190417E0C1598AEC	maschek	5	0.08620690	6.587550	0.5678922
FALTER_20181212A0A32CE599	nikolo	5	0.07692308	6.587550	0.5067346
FALTER_201812120ACD2F8A6B	ausgaben	10	0.13888889	3.643111	0.5059876
FALTER_20180110390D4D1664	schikaneder	8	0.07407407	6.587550	0.4879667
FALTER_201811143469B4E4CF	rolex	14	0.07329843	6.587550	0.4828571
FALTER_20180425792AEAFE95	steinz	3	0.07142857	6.587550	0.4705393
FALTER_20180314D8B04791FA	liessmann	4	0.07843137	5.894403	0.4623061
FALTER_20180411054DD99D34	popper	10	0.07812500	5.894403	0.4605002
FALTER_20180314D4405096C9	jafar	35	0.07216495	5.894403	0.4253693
FALTER_201809266151B91E3F	sam	9	0.06870229	5.894403	0.4049590
FALTER_201806136BA13FC488	gedenkdienst	7	0.06140351	6.587550	0.4044987
FALTER_20181219009A7931BA	ochs	8	0.06106870	6.587550	0.4022931
FALTER_201802283C74AC4717	proell	5	0.09090909	4.390325	0.3991205
FALTER_20180822C5151C564D	erntehelfer	7	0.06666667	5.894403	0.3929602
FALTER_201901238C61CAB08F	freiwilligenarbeit	3	0.06666667	5.894403	0.3929602

Figure 5.1: TF-IDF example

Note, that the same word can have different TF-IDF scores in different articles. There are different ways to handle this issue, two of these would be the following:

1. Highest Score - Using the highest score of every word within the corpus

2. Average Score - Using the average score of every word within the corpus

In this thesis we chose the second option, as this seemed to be the more appropriate way for our later goals, like finding trends.

With these average TF-IDF scores we can now do some further experiments. One of those would be trying to find out if a word with a high TF-IDF score represents a long- or short-term trend. To do this, we chose to compare the TF-IDF score of words for different subsets of the dataset, selected by their publishing dates. The TF-IDF score for the word “Ungarn” for example is very high, when considering all articles that were published between January and March of 2019, but has a rather low TF-IDF score when looking at the timeframe from May 2018 until May 2019.

To graphically display these short- or long-term trends, we chose to compare these two timeframes with each other:

- Short-term: 01.01.2019 - 22.05.2019

- Long-term: 01.01.2018 - 22.05.2019

With 22.05.2019 being the publishing date of the latest articles contained in the dataset, so this date can be seen as “now”.
We selected the top ten highest scoring words for both the short-term and long-term timeframe. On the left hand side the score for the short-term timeframe is marked, on the right side the score for the long-term timeframe. The result can be seen in Figure 5.2.

Some things that can be interpreted from this graphic are:

1. “Orbán” has by far the highest average TF-IDF score for both short- and long-term

2. Most words appear to have a higher TF-IDF score with the short-term timeframe, most likely due to the TF-IDF algorithm punishing the score with more articles in the corpus

3. “Budapest” is the only word within this list with a higher TF-IDF score for the long-term, so it not only is a trend in the short-term, but it seems to be even more outstanding in the long-term

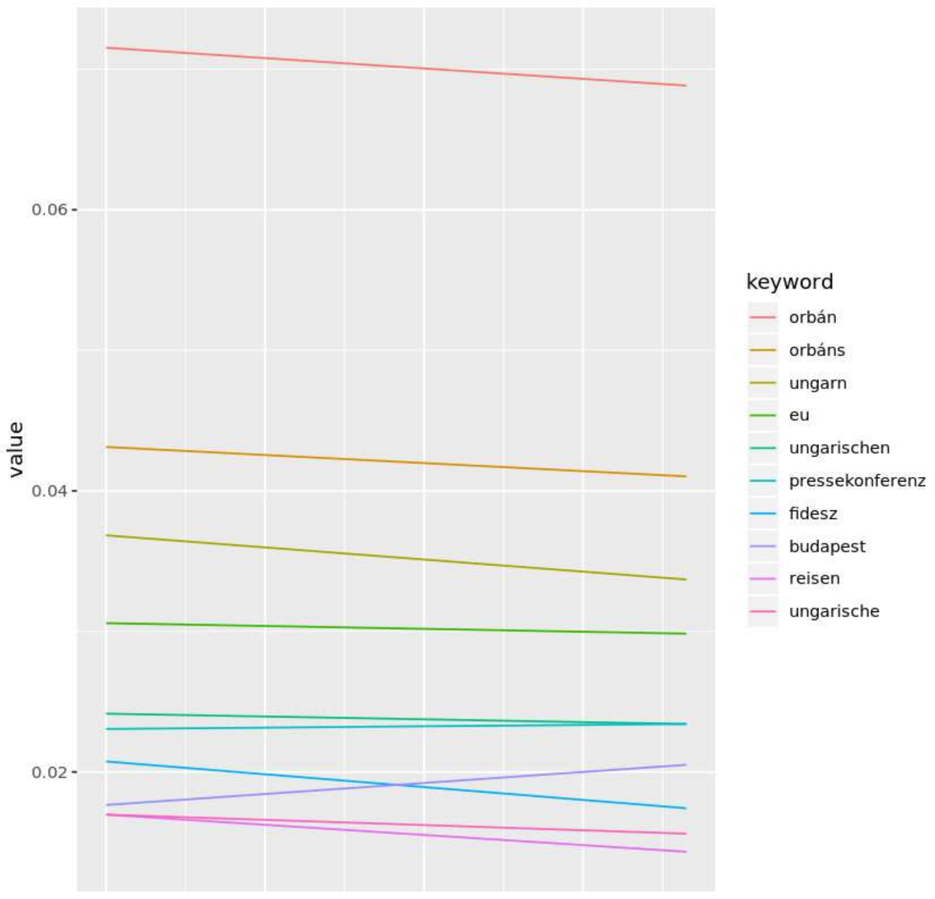


Figure 5.2: TF-IDF trends, short-term on left hand side, long-term on right hand side

Finally, to use this method not only for finding trends but also to recommend articles, the article-similarities between all articles within the considered dataset are calculated. To do this, a document-term matrix with all different words in the corpus and their respective TF-IDF score for the article is built. This results in a vector of TF-IDF scores of all words for all articles within the corpus. This vector form now allows us to calculate the cosine similarity between all articles, resulting in a similarity value between every pair of articles within the corpus.
To do this, first a function to calculate the cosine similarity was defined

```
cos_sim = function(matrix){
    numerator = matrix %*% t(matrix)
    A = sqrt(apply(matrix^2, 1, sum))
    denumerator = A %*% t(A)
    return(numerator / denumerator)
}
```

The creation of the document-term matrix was done with the help of the tidytext R-package

```
dtm = cast_dtm(data=article_words,term=token,
                         document=doc_id,value=tf_idf)
```

Using the self defined cosine similarity function on this document-term matrix results in the similarity matrix. By finding the highest values for every line, we can now build the recommendations based on one article that can be seen as the currently read article.

The full implementation of this method can be seen in algorithm 5.1.

Algorithm 5.1: TF-IDF Scores

```
data <- fread("./falter_data_tok_2019.csv")
stops <- stopwords("de", source = "stopwords-iso")
stops <- append(stops, c("%","␣",""))
'%notin%' <- Negate('%in%')
data <- data[data$token %notin% stops,]
article_data <- fread("./falter_data_full_notext.csv")
article_data <- article_data[article_data$year==2019]
data <- merge(data, article_data[,c("article_id", "date")],
    by.x="doc_id", by.y="article_id")
article_words <- count(data,doc_id, token, sort = TRUE)
#calculating the TF-IDF Scores
total_words <- article_words %>%
                  group_by(doc_id) %>%
                  summarize(total = sum(n))
article_words <- left_join(article_words, total_words,by="doc_id")

#adding TF-IDF Scores to the dataset and sort
article_words <- article_words %>%
                  bind_tf_idf(token, doc_id, n) %>%
                  select(-total) %>%
                  arrange(desc(tf_idf))

#calculating the cosine similarity
dtm = cast_dtm(data=article_words,term=token,document=doc_id,
    value=tf_idf)
dtm_matrix = as.matrix(dtm)
articleSim = cos_sim(dtm_matrix)
```

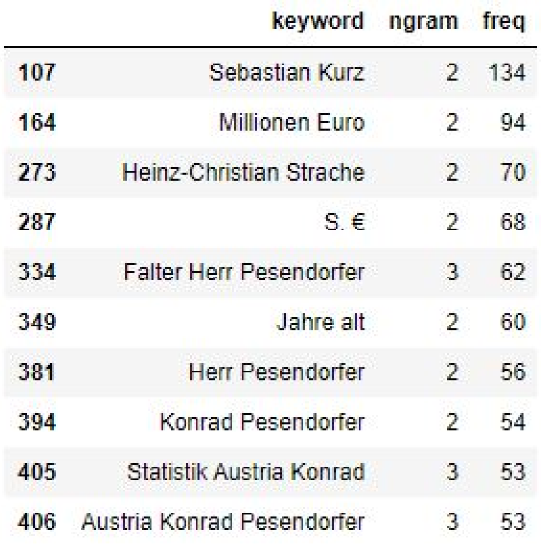
5.2 TextRank Algorithm

Like in TF-IDF, the TextRank algorithm also calculates a score for every word in the corpus. But it is usually used to calculate a score for not only single words, but also for so-called n-grams, which are wordgroups with the length of n words.
The details of how the scores are calculated are described in section 3.2. We used the *textrank* R-package for the calculation [Wij19a].

This method allows to vary the length of the n-grams by simply giving this length to the calculation function of the R-package. An example of this call would be

```
stats <- textrank_keywords(data$token,
        relevant = data$pos %in%
            c("NOUN","ADJ","VERB","PROPN") & data$entity != "",
            ngram_max = 3, sep = " ")
```

The “relevant” variable allows to filter for various aspects. In this example we made use of the metadata for every word by only considering nouns, adjectives, verbs and propositions as relevant, in combination with only including named entities. Furthermore the maximum length of the n-grams was set to 3 and the separator for the words within one n-gram was set to a space, which is only for display purposes.
This call for all articles published in the year 2018 with only considering articles from the political ressort results in the 10 n-grams with the highest score as can be seen in Figure 5.3



	keyword	ngram	freq
107	Sebastian Kurz	2	134
164	Millionen Euro	2	94
273	Heinz-Christian Strache	2	70
287	S. €	2	68
334	Falter Herr Pesendorfer	3	62
349	Jahre alt	2	60
381	Herr Pesendorfer	2	56
394	Konrad Pesendorfer	2	54
405	Statistik Austria Konrad	3	53
406	Austria Konrad Pesendorfer	3	53

Figure 5.3: TextRank results

Noteworthy here is that the filtering via the "relevant" variable, as described, only has to fit for one part of the n-gram. For example in the n-gram "Jahre alt", only the word "Jahre" is a named entity. So common phrases, that only include one named entity are very frequent within the top results.
It can also be seen that the TextRank algorithm usually prefers n-grams that lean more towards the maximum number of words allowed for n-grams. This is also described in the methods chapter, as the benefit of combining multiple words is usually higher than the punishment for doing so.

Finding trends can be done in the same way as in TF-IDF scores, by calculating scores for the same n-grams for different periods of time, to see if they stay the same, improve or worsen over different lengths of time periods or to look at them for not overlapping time periods like for different months of a year.

To finally create article recommendations, we use the cosine similarity. But with TextRank, we have two possibilities. TextRank calculates a score for every n-gram in the corpus, but simultaniously calculates a score for every single word, as these are also n-grams but with the length of one.
We can now decide to either use the score of the single words and use single words for the document-term matrix, or use all n-grams, in the example with 3 being the maximum for n, we would have all occurring 1-, 2- and 3-grams that appear in the corpus. The big downside of the latter option is the vast increase of computational resources needed, as the document-term matrix is far bigger here. Thus, the higher the maximum n (for n-grams) is set, the more resources are needed to represent a document, i.e. an article. Due to limited resources, we only used a higher number of n for a small subset of the dataset, for example for all articles of the political ressort of one month. But eventually we had to use uni-grams for bigger subsets. How this was done can be seen in algorithm 5.2 in the lines 15-21, where the words with their scores have to be extracted as a vector of the TextRank results and merged with the dataset of all words, before finally creating the document-term matrix.

Following the creation of this document-term matrix, we again can calculate the cosine similarity with the same self defined function as seen in the TF-IDF scores to find the most similar articles based on the TextRank scores.

The full implementation of this method using the R-package *textrank* can be found in Algorithm 5.2 [Wij19a].

5.3 RAKE Algorithm

This method is very similar to the TextRank algorithm. Again, scores for words are created based on the algorithm, which is further described in section 3.3.
An implementation of this algorithm can be found in the R-package *udpipe*, where the function keywords_rake calculates the RAKE scores [Wij19b].
As well as in TextRank, a "relevant" parameter exists, with which a filtering of the

Algorithm 5.2: TextRank Algorithm

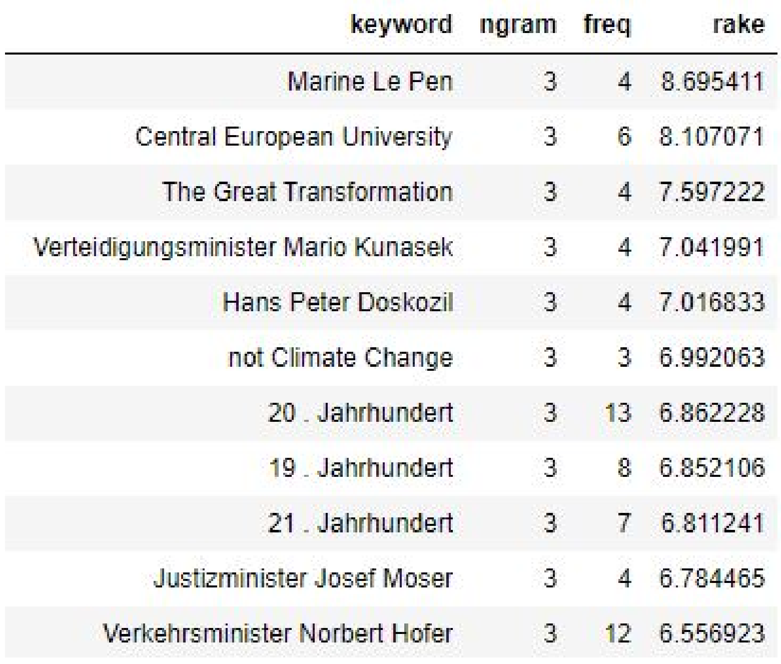
```
data <- fread("./falter_data_tok_2019.csv")
article_data <- fread("./falter_data_full_notext.csv")
article_data <- article_data[article_data$year==2019]
data <- merge(data, article_data[,c("article_id", "date")],
    by.x="doc_id", by.y="article_id", all=FALSE)
'%notin%' <- Negate('%in%')
data2 <- data[data$pos %notin% c("","PUNCT","SPACE","X"),]

#calculating textrank scores with ngram size = 3
#only including nouns, adjectives, verbs and propositions
stats <- textrank_keywords(data2$token,
                           relevant = data2$pos %in%
                           c("NOUN","ADJ","VERB","PROPN"),
                           ngram_max = 3, sep = "␣")
vals <- data.frame(t(data.frame(as.list(stats$pagerank$vector))))
vals$token <- row.names(vals)
names(vals) <- c("textrank","token")
data2 <- merge(data2, vals, by.x="token", by.y="token", all=FALSE)

#calculating the cosine similarity
dtm = cast_dtm(data=data2,term=token,document=doc_id,value=textrank)
dtm_matrix = as.matrix(dtm)
articleSim = cos_sim(dtm_matrix)
```

dataset can be done, to take only those n-grams into consideration that contain at least one word that meets the given requirements. To compare the results of the methods, we used the same requirements as before, so one word has to be either a noun, adjective, verb or proposition and has to be a named entity. The maximum length of the n-grams was chosen with 3 as well.

So in a very similar way to TextRank, the function to calculate the RAKE scores can be called

```
stats <- keywords_rake(x = data,
            term = "lemma", group = c("doc_id"),
            relevant = data$pos %in%
               c("NOUN", "ADJ","VERB","PROPN") & data$entity != "",
            ngram_max = 3)
```

Executing this code on the same test subset as done for the TextRank algorithm, so for all articles published in 2018, the n-grams with the highest score can be seen in Figure 5.4.



keyword	ngram	freq	rake
Marine Le Pen	3	4	8.695411
Central European University	3	6	8.107071
The Great Transformation	3	4	7.597222
Verteidigungsminister Mario Kunasek	3	4	7.041991
Hans Peter Doskozil	3	4	7.016833
not Climate Change	3	3	6.992063
20 . Jahrhundert	3	13	6.862228
19 . Jahrhundert	3	8	6.852106
21 . Jahrhundert	3	7	6.811241
Justizminister Josef Moser	3	4	6.784465
Verkehrsminister Norbert Hofer	3	12	6.556923

Figure 5.4: RAKE results

Like TextRank, the RAKE algorithm also tends to prefer n-grams that lean to the maximum number of words defined for the n-grams. In comparison to the TextRank algorithm the name "Pesendorfer" does not appear at all.
Furthermore, the RAKE algorithm is very resource-friendly. Thus, it was applicable with maximum n (of n-grams) set to 3 to the whole corpus in a reasonable time. The runtime for the same dataset as used for TextRank is only a fraction of the runtime of the TextRank algorithm.

Finding trends and building recommendations both work exactly the same as for TextRank. Which means there is no out of the box option for finding trends, but like with TF-IDF and TextRank, different time periods can be analyzed to find differences.
Calculating close articles can be done by creating a document-term matrix with single words or with all n-grams and their RAKE score. Finally calculating the cosine similarity, which is described in the TF-IDF score section of this chapter, for all pairs of articles results in the recommendations.

The full implementation of this method using the R-package *udpipe* can be found in algorithm 5.3 [Wij19b].

Algorithm 5.3: RAKE Algorithm

```
data <- fread("./falter_data_tok_2019.csv")
article_data <- fread("./falter_data_full_notext.csv")
article_data <- article_data[article_data$year==2019]
data <- merge(data, article_data[,c("article_id", "date")],
    by.x="doc_id", by.y="article_id", all=FALSE)

#calculating rake scores with ngram size = 3
#only including nouns, adjectives, verbs and propositions
stats <- keywords_rake(x = data,
                      term = "lemma", group = c("doc_id"),
                      relevant = data$pos %in% c("NOUN", "ADJ",
                        "VERB","PROPN") & data$lemma != '',
                      ngram_max = 3,n_min=1)
data2 <- merge(data, stats[stats$ngram < 2,], by.x="token",
    by.y="keyword", all=FALSE)

#calculating the cosine similarity
dtm = cast_dtm(data=data2,term=token,document=doc_id,value=rake)
dtm_matrix = as.matrix(dtm)
articleSim = cos_sim(dtm_matrix)
```

5.4 Wordcloud Comparison

Wordclouds are commonly used to find keywords within any kind of text corpora. There are libraries that easily allow the user to create basic wordclouds. But not only finding keywords within the corpus is of relevancy for this thesis, but also finding trends within these keywords. Therefore the first idea was to just create multiple wordclouds for different periods of time and compare them side by side.
For wordclouds especially, removing stopwords from the dataset was important, otherwise the wordclouds would only be filled with these. Additionally to the stopwords from the German stopword-iso list, we removed words that appeared very frequent through the whole dataset with frequencies similar to stopwords, like "wien","wiener","oesterreich" and "falter".
A comparison of wordclouds for the first two quarters of all published articles in 2018 can be seen in Figure 5.5. One can see here that the word "kind", for children, appears frequently in both timespans, while for example "euro" is more common in the first quarter than the second.

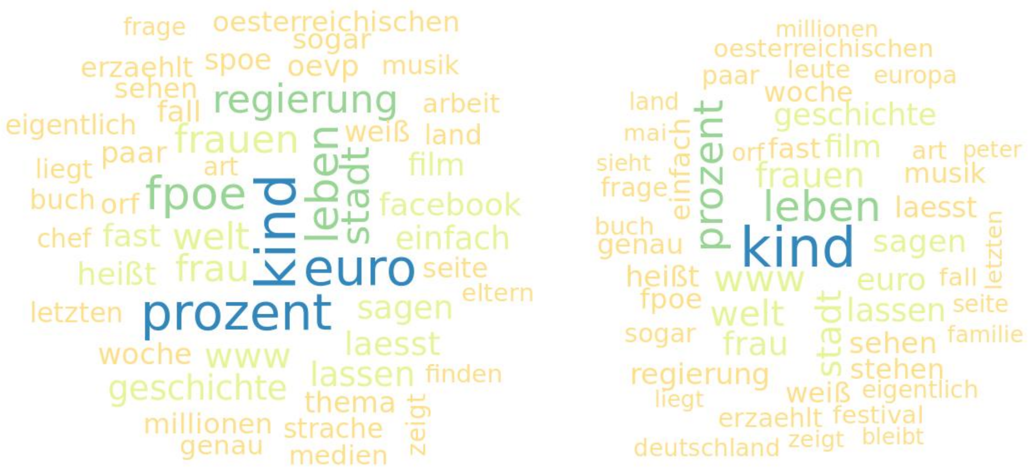


Figure 5.5: Wordclouds for 1. and 2. quarter of 2018

These wordclouds were simply created by splitting the data into the subsets for the 1. and 2. quarter of 2018 and executing the wordcloud function of the R-package *wordcloud*, which can be done like the following example [Fel18]:

```
wordcloud(names(data), data, scale=c(3,.3),
          random.order = FALSE, max.words=50,
          colors=brewer.pal(6, "Spectral"))
```

But finding differences between multiple wordclouds in this way can be very tedious. This is where the so-called wordcloud comparison function has its strengths. This function is usually used to compare different texts, like books, with each other. But in our case we used it to compare different subsets of the corpus for different time periods with each other.
As an example, we split all articles published in the year 2018 into four quarters, so January until March, April until June, July until September and October until December. For the wordcloud comparison function, a frequency matrix has to be built. Every line in it stands for a word in the corpus and every column for one of the subsets that are to be compared. The number of occurrences of the word in the subset is entered into the matrix. After building this frequency matrix, the creation of the comparison wordcloud can be done like this

```
comparison.cloud(as.matrix(freq_mat),max.words=75,
            colors=brewer.pal(8, "Dark2"),title.size=2,
            title.bg.colors=brewer.pal(8, "Dark2"))
```

This example, as mentioned, with the 4 quarters of the year 2018, results in the comparison wordcloud that can be seen in Figure 5.6, where freq1 - freq4 stands for quarter 1 - quarter 4.

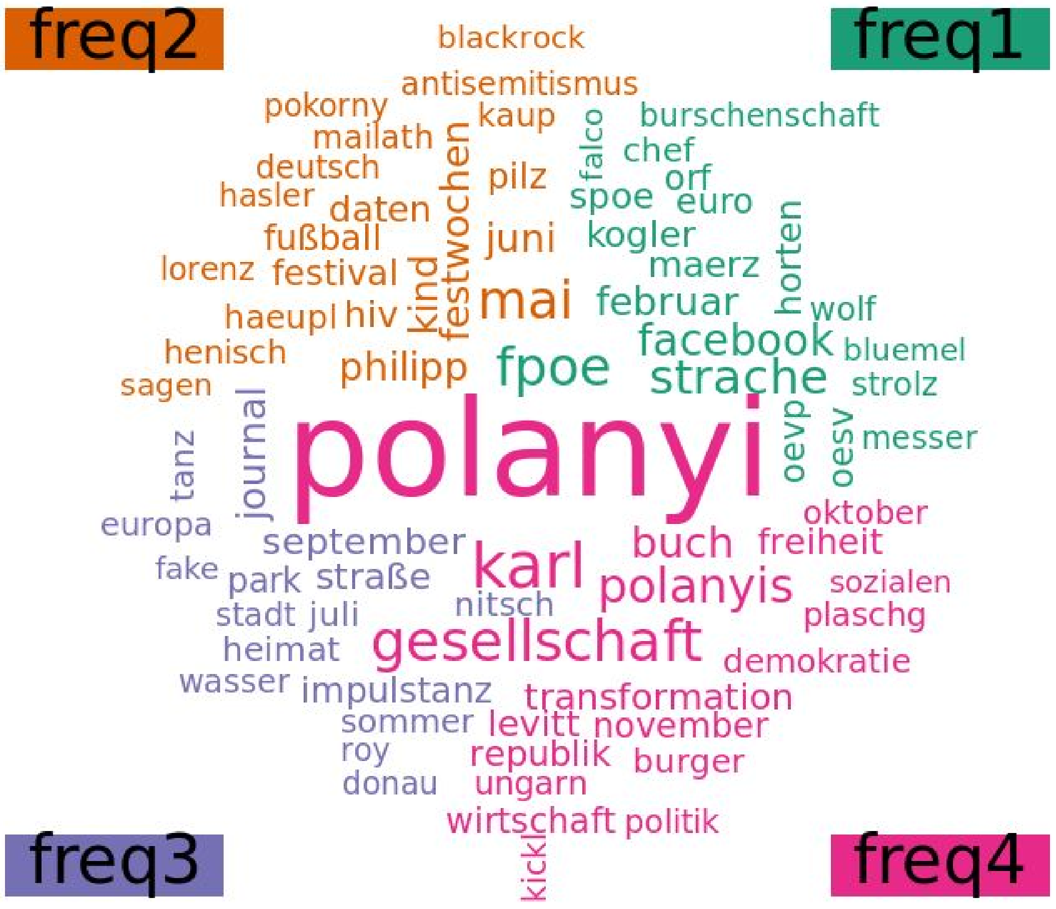


Figure 5.6: Comparison Wordcloud

The name “Karl Polanyi”, who was an Austro-Hungarian economist and sociologist, is the most obvious keyword that can be seen here. This is the result of multiple published issues of *Der Falter* with a focus topic to “Karl Polanyi” in October 2018. As these words are very rare or don’t even exist in all other quarters, they get more highlighted in the comparison wordcloud. The calculation of the size of words can be seen in the methods chapter.
This display of words that appear frequently in one timeframe but not in others can be used to find trends within keywords. The variation of the length of the time periods also allows to distinguish between long and short term trends. Words that do not appear more highlighted when comparing longer timeframes, but do appear on shorter timeframes for

example could be seen as a short term trend. On the other hand a word that appears on longer timeframes but not when comparing subsets of the specific longer timeframe with each other can be seen as a long term trend, as these words are evenly spread throughout the longer timeframe.

Finding recommendations with this method can be done in a very similar way to TF-IDF, TextRank and RAKE, as the wordcloud comparison algorithm computes an importance score, described in Equation 3.4, for every word in the background and uses this score to calculate the size of the word in the wordcloud. When manually calculating these scores for all words, again, a document-term matrix with the scores for every word in every document can be built and via the cosine similarity, articles that are similar to each other can be found.
The full implementation of this, in a non-graphical way for article recommendation, can be found in algorithm 5.4, where all articles published in 2019 are used.

5.5 Stylometry

The implementation of the Stylometry method was done with the help of the *stylo* R-package [ME19]. The authors of this package also wrote a guide for stylometric analysis with this package, which was used as a reference for this work [ERK16].

The first step of preparing the dataset for the stylometry analysis was removing pronouns from the corpus, like “ich”, “mir”, “mein” or “mich”. This was mentioned in the guide of the authors of the R-package to heavily improve the results. Usually they are too strongly correlated with a specific topic or genre, which is an unwanted artefact for a text corpus with various topics and genres [Pen11]. Creating a list of all tokens without the pronouns was done with the help of a German pronoun list provided by the *stylo* R-package itself.

```
txt_tok_list <- list()
doc_ids <- unique(data$doc_id)

for(i in 1:length(doc_ids)){
    txt_tok_list[[doc_ids[i]]] <-
        data[data$doc_id==doc_ids[i],]$lemma %>%
        delete.stop.words(
            stop.words =
                stylo.pronouns(language="German")
            )
    }
```

As it is described in section 3.5, in this method we use word n-grams as well as character n-grams. These features can also be extracted from the corpus with a function of the

Algorithm 5.4: Wordcloud comparison algorithm

```
'%notin%' <- Negate('%in%') #define notin function
#build stopword list
stops <- stopwords("de", source = "stopwords-iso")
data19 <- fread("./falter_data_tok_2019.csv")
data19 <- data19[data19$lemma %notin% stops] #exclude stopwords
data19 <- data19[-grep('^\\d+$', data19$lemma),]
article_data <- fread("./falter_data_full_notext.csv")
article_data <- article_data[article_data$year==2019]
data19 <- merge(data19, article_data[,c("article_id", "date")], by.x=
    "doc_id", by.y="article_id")
data19$month <- substr(data19$date, 6, 7)
#count word per month
counts <- summarise(group_by(data19,month,lemma),count=n())
freqs <- data.frame() #define empty df to be filled with counts
for(m in unique(counts$month)){
    #df is initial -> add word and first count column
    if(dim(freqs)[1] == 0){
        freqs <- counts[counts$month==m,c("lemma","count")]
        colnames(freqs) <- c("lemma",paste("count",m,sep=""))
    #df is not initial anymore -> add new count column
    }else{
        new_dat <- counts[counts$month==m,c("lemma","count")]
        freqs <- merge(freqs,new_dat,by="lemma",all=TRUE)
        colnames(freqs)[ncol(freqs)] <- paste("count",m,sep="")
    }
}
#substitute NAs for non-existing words in months with 0
freqs[is.na(freqs)] <- 0

#calculate score that is used for comparison.cloud
dist <- freqs[,6] - rowMeans(freqs[,-1])
freqs$dist <- dist

data2 <- merge(data19, freqs[c(1,7)], by.x="lemma", by.y="lemma", all
    =FALSE, sort=FALSE)

#calculate cosine similarity
dtm = cast_dtm(data=data2,term=token,document=doc_id,value=dist)
dtm_matrix = as.matrix(dtm)
articleSim = cos_sim(dtm_matrix)
```

stylo package, in this example with the n-gram size of three, and the paramter “features” chosen as “c” meaning characters are used for building the n-grams.

```
feats <- txt.to.features(txt_tok_list,
                         ngram.size = 3,
                         features = "c")
```

These extracted features, the n-grams of characters, now have to be converted to a table with the frequency of every feature for every document. Additionally, only the most frequent 5000 features are used in this example, to reduce the processing time and the needed computing resources.

```
freq_feats <-
    make.frequency.list(txt_tok_list,
                        head=5000)
freqs <-
    make.table.of.frequencies(txt_tok_list,
                              features = freq_feats)
```

Finally, this frequency table can be used for a stylometric analysis of the corpus. Calling the *stylo* function of the same-named R-package does not only compute the stylometric analysis, but can also create various graphical presentations of the result.
In this example, we used the PCR analysis type, which stands for the principal components of all extracted attributes. The parameter “gui” gives the option for a graphical user interface, where further parameters could be set.

```
stylo(frequencies=freqs,
  analysis.type='PCR',
  gui=FALSE)
```

For example, Figure 5.7 shows the reduction of the extracted stylometric features to 2 principal components. These are based on all published articles of the years 2018 and 2019 from the political ressort. Every article is represented by its article id. Stylistic outliers can be found, like the ones on the top right corner, that are further away from the big cluster. These articles are stylistically written differently, for example the article in the top right corner is a satirically written made up interview.

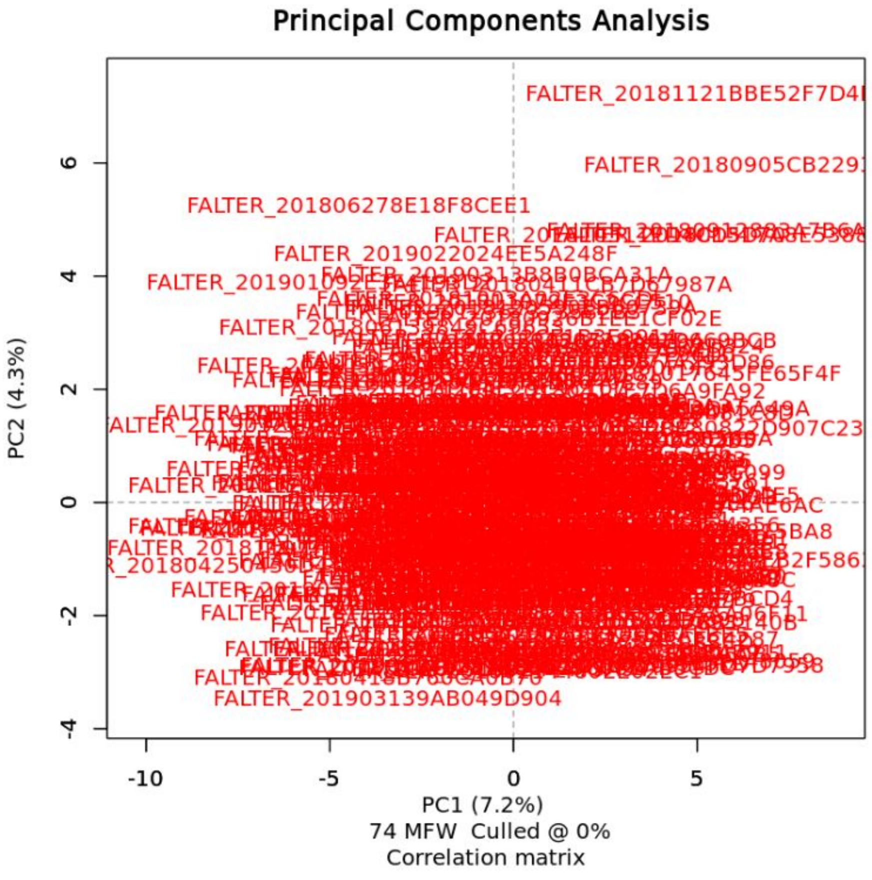


Figure 5.7: Stylometry - Principal Components example

Stylometry also offers a few more possibilities, like the comparison of the articles written by different authors, as it can be seen in Figure 3.4 in the methods chapter, or for cluster analysis as seen in Figure 3.5. But none of the possible options here allow for finding trends within the styles used by the authors.

But what can also be done with Stylometry is calculating a distance between every pair of articles to find possible recommendations. By combining the extracted stylistic attributes into a vector for every document, the distance is calculated via the Manhatten Distance of these vectors [Bur02]. These distances can easily be extracted from the result of the analysis and simply searching for the lowest distances between the articles leads to the most similar ones.

```
stylo_result <- stylo(frequencies=freqs,gui=FALSE)
dist_table <- stylo_result$distance.table
```

The full algorithm for this method can be seen in Algorithm 5.5.

Algorithm 5.5: Stylometry algorithm

```r
data <- fread("./falter_data_tok_2019.csv")
article_data <- fread("./falter_data_full_notext.csv")
data <- merge(data, article_data[,c("article_id", "date")],by.x="doc_id", by.y="article_id", all=FALSE)

txt_tok_list <- list()
doc_ids <- unique(data$doc_id)
for(i in 1:length(doc_ids)){
    txt_tok_list[[doc_ids[i]]] <-
        data[data$doc_id==doc_ids[i],]$lemma %>% delete.stop.words(stop.words=stylo.pronouns(language="German"))
}
#possibly use n-grams as features to be analyzed (3-gram: moped -> mop ope ped)
feats <- txt.to.features(txt_tok_list, ngram.size = 3, features = "c")
#only view most frequent features
freq_feats <- make.frequency.list(txt_tok_list,head=1000) #or use feats instead of txt_tok
#build frequency table of features
freqs <- make.table.of.frequencies(txt_tok_list, features = freq_feats)
#reducing the freqs to only top %age of features
freqs <- perform.culling(freqs, culling.level = 50)

stylo_result <- stylo(frequencies=freqs,gui=FALSE)
dist_table <- stylo_result$distance.table
```

5.6 Dynamic Topic Modelling

Before starting with the Dynamic Topic Modelling itself, a very important preparation step is to find a suitable amount of topics that appear within the corpus. The R-package *stm* not only offers the possibility for the Dynamic Topic Modelling, but also to find a suitable number of topics.
"Suitable" would be a number, where the found topics are not too broad so they don't overlap too much. But at the same time they should not be too specific, to still describe a topic, and not only a few co-occurring words.

First, to later include the dynamic part of this method, the publishing date of the article needs to be added to the dataset from the previously splitted data. But not only the date itself, but rather the timely distance from the first published article in the corpus has to be calculated. In the following example, we used all articles published in the year 2018. Therefore, we used the 1st of January as the starting date and calculated the distance to this date for every article in days.

```
startdate <- as.Date("20180101","%Y%m%d")
data$days <- as.numeric(data$date - startdate)
```

This needs to be done first, as the R-package *stm* does some preparation of the corpus itself and already needs the metadata to all documents added. The preparation steps include things like removing punctuation or removing numbers as well as other things. But the details can be chosen through parameters, and as most of these are already done in our general preprocessing, they are not activated here.

```
meta <- data.frame(data$days)
processed <- textProcessor(data$text,
                           stem=FALSE,
                           removestopwords=FALSE,
                           lowercase=FALSE,
                           language="de",
                           metadata=meta)
out <- prepDocuments(processed$documents,
                     processed$vocab,
                     processed$meta)
```

After these preparation steps, the number of topics that will be chosen has to be found. With the "searchK" function of the *stm* package, the optimal number of topics can be determined by optimizing semantic coherence. We compared the semantic coherence for the number of topics K for 10, 20, 30, 40 and 50. A graphical representation of this comparison can be seen in Figure 5.8, and a more detailed description can be found in section 3.6. Here we want to choose a number of topics K for which the semantic coherence is high while the number of topics is still rather low. In this example the number of topics is set to 40.

```
ntopics <- searchK(out$documents,
                   out$vocab,
                   K = c(10,20,30,40,50))
```

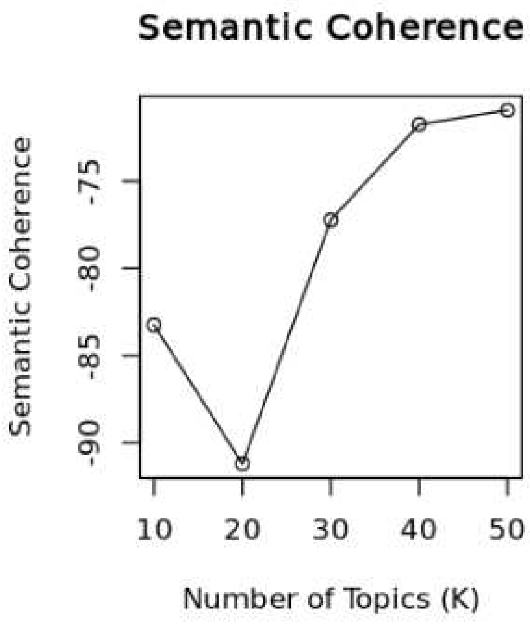


Figure 5.8: Semantic Coherence

With the number of topics defined, we can now do the topic modelling with the help of the *stm* package as well. For this we use the *stm* function, which results in the topic model. The topic model than looks like the example as shown in the methods chapter, Figure 3.7.

```
fit <- stm(out$documents,
        out$vocab,
        K = 40,
        max.em.its = 30,
        data = out$meta,
        init.type = "Spectral")
```

What needs to be done now is the inclusion of the previously added meta-data, so the publishing date of the articles, to find trends within the topics. This can be done with the "estimateEffect" function.

```
prep = estimateEffect(1:40 ~ days, fit, meta = out$meta,
                uncertainty = "Global")
```

Plotting the result now leads to graphical representation of the topics and their trends. For example Figure 5.9 shows the two topics with the steepest decreasing and increasing slopes. One can see for example that the topic “germania liederbuch” was decreasing throughout the year, while “asyl” became more prevalent. But these trends can also be shown in a non-linear way, as shown in Figure 3.9, which can be used to find both short and long term trends within the found topics.

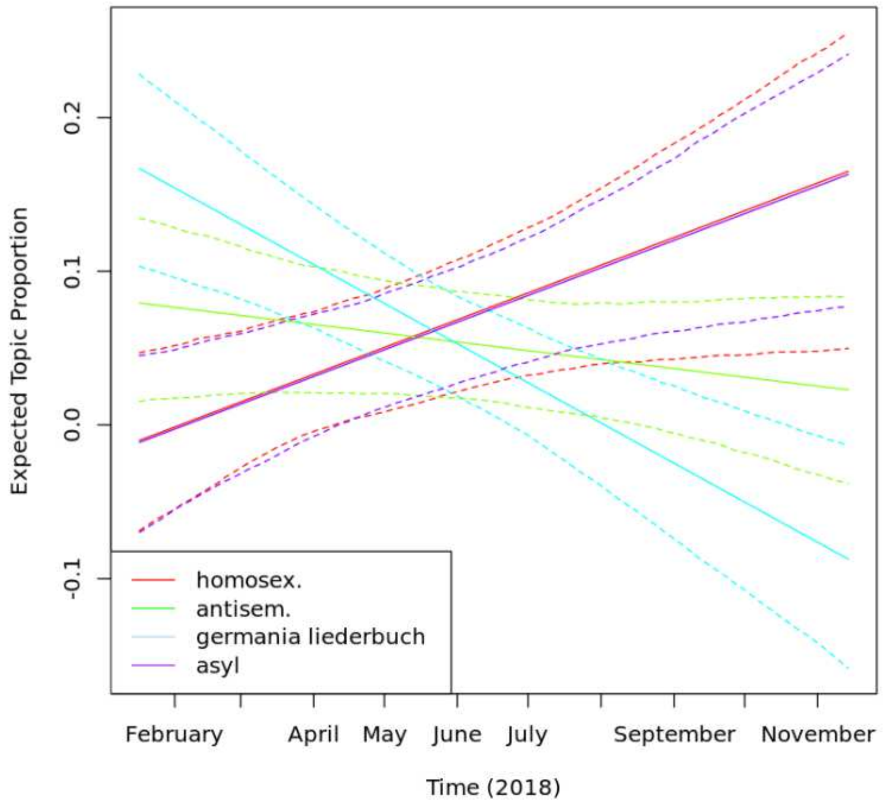


Figure 5.9: Topic trends

After finding both topics and trends within the topics, we want to find a way to utilize this method to find similar articles. For this task we used the functionality of the *stm* package to fit a new document into the already calculated model. By doing so, the already calculated model of topics will not be changed, but for this one article, the correlation to all topics can be extracted. The following code shows an example where one random article from the dataset was chosen and the correlation to each topic was extracted

```
ran_art <- sample(xmldata$doc_id,1)

post_proc <- textProcessor(
    xmldata[xmldata$doc_id==ran_art,]$text_,stem=FALSE,
    removestopwords=FALSE,lowercase=FALSE,language="de",
    metadata=meta[meta$text_==
            xmldata[xmldata$doc_id==ran_art,]$text_,])
post_out <- prepDocuments(post_proc$documents,
            post_proc$vocab,
            post_proc$meta,
            lower.thresh=0)
post <- fitNewDocuments(model=fit,
            documents=post_out$documents,
            newData=post_out$meta,
            origData=out$meta)
```

The variable post$theta now contains a vector with the correlation value of this random article to each topic. We now build a matrix like the document-term matrix for previous methods, but this time not with terms but with topics. Therefore a document topic matrix, with the correlation value of the article to every topic. By doing so, we can again view each document as a vector and calculate the cosine similarity between every pair of articles to find similar ones.

But including articles into an existing model takes a lot of time to compute. Another possibility to create recommendations is to use the topics with the highest correlations to the current article. By using those articles, that have the highest correlation to the same topics, one can build recommendations way faster. The *stm* function has no possibility to extract all correlations between topics and articles besides the tedious way described above. But it offers a function (i.e. "topThoughts") to retrieve the top n articles with the highest correlation to one topic. In the following example, the top 5 articles with the highest correlation to topic number 1 are extracted, which returns the article id of these articles.

```
top_thoughts <- findThoughts(fit, texts=data$text,
                             n=5, topics=1)
```

The implementation of this method, with the help of the R-package *stm* can be seen in multiple algorithms [RST$^+$14]. The full preparation of the data can be seen in Algorithm 5.6. Finding the ideal number of topics as well as plotting the result of the topic model have been shown above.

Algorithm 5.6: DTM preparation algorithm

```
#prepare words to be eliminated
stops <- stopwords("de", source = "stopwords-iso")
# add falter-specific stopwords to the list
stops <- append(stops, c("wien","wiener","oesterreich","falter","the"
    ))
# with already tokenized data
xmldata <- fread("./falter_data_tok_2019.csv")
data <- xmldata
art_data <- fread("./falter_data_full_notext.csv")
art_data <- art_data[art_data$year==2019]
xmldata <- xmldata[xmldata$doc_id %in% art_data$article_id,]
# for every article_id in art_data, collect the tokens into a vector
    and lemmatize_strings() them
xmldata = xmldata %>% group_by(doc_id) %>% summarise(text=paste(token
    , collapse="␣"))
startdate <- as.Date("20190101","%Y%m%d")
xmldata <- merge(xmldata, art_data[,c("article_id","date")], by.x="
    doc_id",by.y="article_id", all = TRUE)
xmldata$date <- paste(substr(xmldata$date,0,4),substr(xmldata$date,6,
    7),substr(xmldata$date,9,10),sep="")
xmldata$date2 <- as.Date(xmldata$date,"%Y%m%d")
xmldata$days <- as.numeric(xmldata$date2 - startdate)
corp <- VCorpus(VectorSource(xmldata$text))
corp <- tm_map(corp, content_transformer(tolower))
corp <- tm_map(corp, removeWords, stops)
corp <- tm_map(corp, removeNumbers)
svec <- convert.tm.to.character(corp)
svec <- lemmatize_strings(svec)
xmldata$text_ = svec
meta <- data.frame(xmldata$days, xmldata$text_)
colnames(meta) <- c("days","text_")
meta <- transform(meta, text_ = as.character(text_))
processed <- textProcessor(xmldata$text_,stem=FALSE,removestopwords=
    FALSE,lowercase=FALSE,language="de",metadata=meta)
out <- prepDocuments(processed$documents, processed$vocab, processed$
    meta)
```

5.7 Word Embeddings

The Word Embeddings method can either be done with an already pre-trained model of word vectors or with an own model, that has to be trained on the given dataset.

So to use both methods, we first have to train our own model. This was done with the help of the R-package *text2vec* [DS18].
The first steps for this are to convert the list of words into a usable format for this package and to define the vocabulary that will be used. Because of our limited computational resources, only words with a minimum of three occurrences in the dataset were used for the training of the model.

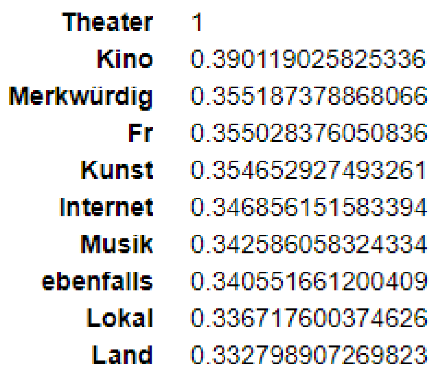
```
it <- itoken(tokens, progressbar = FALSE)
vocab <- create_vocabulary(it)
vocab <- prune_vocabulary(vocab, term_count_min = 3L)
vectorizer <- vocab_vectorizer(vocab)
```

This already concludes the necessary steps to train the model and create the word vectors. Details to this algorithm are given in section 3.7. We chose the skip-grams-window size for the Glove model to be 5, the resulting size of the word vectors to have 150 dimensions and to stop training the model after 30 iterations.

```
tcm <- create_tcm(it, vectorizer, skip_grams_window = 5L)
glove = GloVe$new(word_vectors_size = 150,
                  vocabulary = vocab,
                  x_max = 5)
glove$fit_transform(tcm, n_iter = 30)
word_vectors = glove$components
```

The “word_vectors” variable now contains a list of all words within the defined vocabulary and their representing vectors. For some initial testing, we can now build similarities between all single words. Again, the cosine similarity can be used here. For example we took the word "Theater" from the vocabulary and calculated the 10 most similar words according to this calculated model, which can be seen in Figure 5.10, where of course the word itself has the similarity 1.

```
words <- word_vectors[, "Theater", drop = F]
cos_sim = sim2(x = t(word_vectors),
               y = t(words),
               method = "cosine",
               norm = "l2")
head(sort(cos_sim[,1], decreasing = TRUE), 10)
```



Theater	1
Kino	0.390119025825336
Merkwürdig	0.355187378868066
Fr	0.355028376050836
Kunst	0.354652927493261
Internet	0.346856151583394
Musik	0.342586058324334
ebenfalls	0.340551661200409
Lokal	0.336717600374626
Land	0.332798907269823

Figure 5.10: Similar words - Word Embeddings

The next step is to not only calculate similarities between words, but between articles as well. From this step on, both methods, using the self-trained as well as the pre-trained model, work similarly. The vectors of all words that occur in an article will be combined, via weighing them by their importance within an article with the help of the TF-IDF score and simply adding them up.

So first we have to calculate the TF-IDF score for every word in every article and also the sum of all TF-IDF scores for one article.

```
data2 <- data2[removePunctuation(data2$lemma)!="" &
                    data2$lemma!=" "]
article_words <- count(data2, doc_id, lemma, sort = TRUE)
total_words <- article_words %>%
              group_by(doc_id) %>%
              summarize(total = sum(n))
article_words <- left_join(article_words,
                              total_words, by="doc_id")
article_words <- article_words %>%
              bind_tf_idf(lemma, doc_id, n) %>%
              select(-total) %>%
              arrange(desc(tf_idf))

sums <- aggregate(tf_idf~doc_id, sum, data=article_words)
names(sums) <- c("doc_id","sum_tf_idf")
```

After calculating both the TF-IDF scores and the sum of all TF-IDF scores, we calculate the importance value by dividing the TF-IDF score through the sum of TF-IDF scores for every article.

```
article_words <- merge(article_words, sums, by="doc_id")
```

```
article_words$importance <-
    article_words$tf_idf / article_words$sum_tf_idf
```

And finally we want to create a vector for every article, therefore we multiply each word-vector with its importance value for the article and then sum up all these weighted vectors of one article.

```
word_vectors_t <- word_vectors %>% t() %>%
        data.table(keep.rownames = TRUE)
vector_cols <- paste0("V", seq(1, 150))
scaled_vector_cols <- paste0("X", seq(1, 150))
weighted_vectors <- merge(article_words, word_vectors_t,
                                by.x="lemma", by.y="rn",
                                all.x = TRUE)
weighted_vectors[, scaled_vector_cols] <-
    weighted_vectors[, vector_cols] * weighted_vectors$importance

article_sum_vectors <- weighted_vectors %>%
    group_by(doc_id) %>%
    summarise_at(scaled_vector_cols, sum, na.rm = TRUE) %>%
    data.table
```

With these article vectors we can now build similarities to create recommendations using the cosine similarity between the article vectors. For example calculating the most similar articles for one random article can look like this.

```
ran_art <- sample(meta$article_id,1)
doc <- article_sum_vectors[ran_art,]
cos_sim = sim2(x = data.matrix(article_sum_vectors),
               y = data.matrix(doc),
               method = "cosine", norm = "l2")
cos_sim_df <- as.data.frame(cos_sim)
cos_sim_df <- cos_sim_df[order(-cos_sim_df[ran_art]), , drop = FALSE]
```

The full algorithm, with first calculating the word vectors, which only has to be done for the self-trained model, and then calculating document vectors as well as building similarities between those articles can be seen in Algorithm 5.7 and Algorithm 5.8.

Algorithm 5.7: Word Vector calculation

```
data <- fread("./falter_data_tok_2019.csv")
# collect lemmas of an article back together
datap <- aggregate(lemma ~ doc_id, data=data, paste, collapse="␣")
# tokenize die items
tokens <- space_tokenizer(datap$lemma %>% removePunctuation())

# define relevant vocabulary and vectorize it
it <- itoken(tokens, progressbar = FALSE)
vocab <- create_vocabulary(it)
vocab <- prune_vocabulary(vocab, term_count_min = 3L)
vectorizer <- vocab_vectorizer(vocab)

# train model and create word vectors
tcm <- create_tcm(it, vectorizer, skip_grams_window = 5L)
glove = GloVe$new(word_vectors_size = 150, vocabulary = vocab, x_max
    = 5)
glove$fit_transform(tcm, n_iter = 30)
word_vectors = glove$components

# build similarity between single words
words <- word_vectors[, "Theater", drop = F]
cos_sim = sim2(x = t(word_vectors), y = t(words), method = "cosine",
    norm = "l2")
```

Algorithm 5.8: Document Vector calculation

```
#calculate tf-idf for every word in every article
data2 <- fread("./falter_data_tok_2019.csv")
data2 <- data2[removePunctuation(data2$lemma)!="" & data2$lemma!="␣"]
article_words <- count(data2, doc_id, lemma, sort = TRUE)
total_words <- article_words %>%
                  group_by(doc_id) %>%
                  summarize(total = sum(n))
article_words <- left_join(article_words, total_words,by="doc_id")
article_words <- article_words %>%
                  bind_tf_idf(lemma, doc_id, n) %>%
                  select(-total) %>%
                  arrange(desc(tf_idf))
#calculate the sum of all tf-idf values per article
sums <- aggregate(tf_idf~doc_id, sum, data=article_words)
names(sums) <- c("doc_id","sum_tf_idf")
#calculate the 'importance' (percentage) of a word in the article: tf
   -idf of word / sum of all tf-idfs in article
article_words <- merge(article_words, sums, by="doc_id")
article_words$importance <- article_words$tf_idf / article_words$sum_
   tf_idf
# transform word-vectors and build data-table
word_vectors_t <- word_vectors %>% t() %>% data.table(keep.rownames =
    TRUE)
# word vector columns (word-vector has 150 dimensions)
vector_cols <- paste0("V", seq(1, 150))
# new word vector columns (multiply every word-vector with its
   importance in the article)
scaled_vector_cols <- paste0("X", seq(1, 150))
weighted_vectors <- merge(article_words, word_vectors_t, by.x="lemma"
   , by.y="rn", all.x = TRUE)
weighted_vectors[, scaled_vector_cols] <- weighted_vectors[, vector_
   cols] * weighted_vectors$importance
# grouping and summing scaled vectors to build article-vectors
article_sum_vectors <- weighted_vectors %>%
  group_by(doc_id) %>%
  summarise_at(scaled_vector_cols, sum, na.rm = TRUE) %>%
  data.table
#make doc_id to row_name
article_sum_vectors <- data.frame(article_sum_vectors, row.names=1)
```

CHAPTER 6

Comparison & Evaluation

In this chapter we compare the described and implemented methods. The comparison was based on the research questions, and therefore it focused on four main points:

1. *Finding Keywords/Topics*
   The ability of each method to find underlying keywords or topics within the corpus.
2. *Finding Trends*
   The ability to find trends through the time span of the corpus for keywords or topics.
3. *Processing of a Dynamic Corpus*
   The ability of each method to process a corpus that changes continuously.
4. *Characteristics of Article Recommendations*
   The ability to find similar articles for article recommendation and the attributes of these recommended articles for one specific article.

While the results of the first three points are validated mainly in a qualitative way together in workshops with domain experts from *Der Falter*, point four was evaluated quantitatively.

6.1 Finding Keywords/Topics

The ability of a method to find relevant keywords within the corpus is not only important for the subsequent recommendation of similar articles, but can also yield value for the publishers of the newspaper or authors of articles. They could for example find some keywords or topics, which subjectively spoken, are not used or discussed regularly but suddenly appear more often. Other keywords or topics, while being important for the authors and publishers, might not be used very commonly. For this purpose, the ability of every implemented and analyzed method to not only find and extract keywords, but also to present them in a human readable way was investigated.

Naturally, the mainly keyword based methods are very well suited for this task. This includes TF-IDF Scores, TextRank Algorithm and RAKE Algorithm, which all give the found keywords of the corpus with their importance value as a result. With these, the user afterwards has many possibilities to further process the keywords and analyze them in various ways. Wordcloud Comparison, which gives keywords with importance values as a result as well, not only returns these importance values for further processing. The library also includes a graphical output for a comparison of different sub-corpora, for example for the comparison of different timeframes or different issues of the newspaper. Dynamic Topic Modelling is very well suited for this task as well. Instead of keywords, this method gives topics as a result with their corresponding keywords. So keywords are not weighted for their importance, but are clustered together to topics, which afterwards are weighted for their importance within the corpus. What can be seen as a con for this method is the fact that topics are not automatically labeled by the algorithm itself. With the help of the corresponding keywords, they have to be labeled manually. It is also suggested to analyze the found topics manually through these corresponding keywords to find out if the topic actually represents a fitting cluster of keywords that would be seen as a topic by humans as well.
The methods Stylometry and Word Embeddings on the other hand are not very well suited for the extraction of keywords or topics. Even though they both use keywords in one way or another within the algorithm, both give no way to find important keywords on themselves. For example, Stylometry gives the possibility to find articles that have a strong correlation to some words or wordclusters, but these words or wordclusters are most often not keywords but rather stopwords, that have many occurrences and therefore have influence on the style of the article. Word Embeddings results in a vector for every word of the corpus, which may be useful for example in combination with outlier detection methods to find special words. But doing so is not established or commonly used in the NLP context, therefore it was not included in this work.
An overview of which method is suited for this task can be seen in Table 6.1.

6.2 Finding Trends

Besides finding important keywords and topics within the given corpus of newspaper articles, publishers and authors are heavily interested in occurring trends of keywords and topics. This information can be used by authors and publishers for example to analyze occurring topics and keywords over the years. Or to have more insight in how far any event that has happened, like elections or other events, has been handled by the issues that were published around that timeframe.
But not only for qualitative analysis can trend information be used, it might also be included for news article recommendation. While it comes naturally that more recent articles should be preferred for recommendations, articles that deal with topics or keywords

Finding Keywords or Topics

Well-suited	Unsuited
TF-IDF Scores	Stylometry
TextRank Algorithm	Word Embeddings
RAKE Algorithm	
Wordcloud Comparison	
Dynamic Topic Modelling	

Table 6.1: Method-comparison for finding keywords/topics

that appear to be currently trending might also be preferred as these topics could be more interesting for potential readers. Additionally, trends that have been found in the past could also be used for recommendations, like preferring articles that include topics and keywords which have been trending while the article was published, instead of only looking at articles with keywords and topics that have a high correlation to the article.

For this task, Wordcloud Comparison can be seen as very suitable for more than one reason. First, the algorithm for calculating the importance values of the keywords is based on comparing different sub-corpora of the whole corpus with each other and weighting the keywords based on how important a keyword is in one sub-corpus compared to all the others. When interpreting these sub-corpora as different timeframes, the used algorithm basically compares these with each other and finds keywords that are important in one timeframe relatively seen in comparison to the others. Second, the wordcloud comparison package in R, which was used for this paper, automatically comes with a graphical output for the keywords and their importance values, which can be interpreted very easily. An example for a comparison wordcloud is shown in Figure 3.3, where the years 2015 to 2019 were taken as the timeframes to be compared with each other.
Other than Wordcloud Comparison, the Dynamic Topic Modelling is very well suited for finding trends, which appears obvious, as finding trends within topics was the main motivation for the transition from Topic Modelling to Dynamic Topic Modelling. This method, when using the R-package *stm*, as it was done for this work, comes with a graphical presentation for the trends of the topics, which can easily be interpreted and used by the authors and publishers for analysis purposes. The trends can also be extracted for further processing to include them in recommendations as well.

The keyword based methods, other than Wordcloud Comparison, namely TF-IDF Scores, TextRank Algorithm and RAKE Algorithm, are only mediocre for finding trends. The algorithms themselves do not come with any kind of trend analysis, but as it was already stated in their respective chapters, some possibilities to find trends and combine the resulting keyword importance values with them do exist and can be used. Therefore we classified them in the middle between well-suited and unsuited.

As with finding keywords and topics, the two methods Stylometry and Word Embeddings both are not suited for finding trends within the corpus. Even though for Stylometry a method called *Stylochronometry* exists, which analyses stylistic changes of texts throughout the time, this method is still not suited for finding trends for a newspaper. It is mainly used for comparing texts of only one author and the stylistic changes of this specific author throughout the time. Therefore, it would give results that cannot be interpreted when used for multiple authors, because changes in the style could be an outcome of changing authors and the amount of written articles per author.

An overview of which method is suited for this task can be seen in Table 6.2.

Finding Trends

Well-suited	Partly suited	Unsuited
Wordcloud Comparison	TF-IDF Scores	Stylometry
Dynamic Topic Modelling	TextRank Algorithm	Word Embeddings
	RAKE Algorithm	

Table 6.2: Method-comparison for finding trends

6.3 Processing of a Dynamic Corpus

Another factor that is relevant for a newspaper that continuously publishes new articles is the capability of every method to handle the addition of new articles to the existing corpus. The way, in which every method is able to handle new articles can have influence on different aspects of the analysis of the corpus and the recommendation of articles.

One important issue is the stability of the already existing analysis. Even though new articles get added, the analysis results of the articles from the past should still have relevancy for further analysis. This means that even when the important keywords or topics of the current time period of course will be influenced by new articles, the already calculated values or the results of past analysis runs should not change drastically and completely lose their meaning.
Another issue is the computational effort it takes to rebuild or update the calculated model. When adding new articles only once per month or week, this factor may not be as relevant as the stability, but if articles will be added in a more regular manner, it could become unrealistic to always completely rebuild the model, especially if the computational effort for the model is already very high demanding.

Interestingly, this point is where the two methods, that had their weakness with the previously discussed points, show their strength. Both Stylometry and Dynamic Topic

Modelling are very well suited for processing a regularly changing corpus. The Stylometry library, that was used within this thesis, easily allows for the addition of new articles to the corpus and also to the already calculated model. For the new articles, the stylistic attributes will be computed and based on their results, the articles will be classified and categorized into the existing model. This takes very little computational effort and has no influence on the existing model for the already calculated articles at all.
Dynamic Topic Modelling offers the ability to include new articles into the already calculated topic model as well, which also takes very little computational effort. The new articles will then be fitted into the already calculated topics and their corresponding keywords and the relations of the new article to all existing topics will be built. This process has the advantage that the already calculated topics will be stable and therefore can still be analyzed from the past and their current trends, through the addition of new articles, can always be monitored. But exactly this stability of topics is also the biggest downside of the process, as through doing so, newly emerging topics will never be recognised by the topic model. For the acquisition of new topics, the whole topic model has to be calculated again, through which of course the stability of topics would get lost. So this could be seen as a kind of two-edged sword.
When it comes to Word Embeddings, for the aspect of processing a dynamic corpus we have to differentiate between the pre-trained and the self-trained model. While the self-trained model is not suited for this task, as all word vectors would have to be recalculated, the pre-trained model is very well suited here. As the pre-trained model is calculated on a sufficiently sized set of texts, so that the word vectors can be seen as *final*, the new articles would only have to be translated into the word vectors and afterwards into document vectors, which takes only very little computational effort and would have no influence on the already built relationships of the existing articles.

All other methods do not offer the ability to reasonably add new articles into the corpus without recalculating the whole model. But we still did divide the models into partially suited and not suited, as the RAKE Algorithm, even though it has to recalculate all importance values for every keyword, still is able to do this calculations for the full corpus with the size of the given test dataset (in our case with about 100.000 articles) in an in our opinion acceptable time, which in this case was well under 10 minutes. Every other method, namely TF-IDF Scores, TextRank Algorithm and Wordcloud Comparison takes way more computational effort and therefore was ranked as not suited for this task.

An overview of which method is suited for this task can be seen in Table 6.3.

6.4 Characteristics of Article Recommendations

Finally, we evaluated the ability to recommend other newspaper articles based on one specific given article. The hypothetical situation was given, that a consumer reads a newspaper article on an online platform and based on this currently read article only, without including any potential previously gathered demographical or preference information about the user, other articles should be recommended. As a sophisticated

Processing dynamic corpus

Well-suited	Partly suited	Unsuited
Stylometry	RAKE Algorithm	Word Embeddings self
Dynamic Topic Modelling		TF-IDF Scores
Word Embeddings pre		TextRank Algorithm
		Wordcloud Comparison

Table 6.3: Method-comparison for processing a dynamic corpus

survey is beyond the scope of this thesis, content-based measurements were used here. Measurements, that became more and more popular with evaluating news recommender systems, are serendipity and diversity of recommended articles. Therefore, we put our focus on these measurements.
An overview of measurements is given by Kaminskas et al., where various measurements, including content-based only measurements, are described [KB16]. Following measurements were chosen for the comparison of the recommendations of all methods:

1. *Content-based Diversity*
   This measurement was chosen to measure how similar or diverse the list of recommended articles is.
2. *Content-based Surprise*
   With this measurement in combination with the diversity, the serendipity was tried to be measured as well. If the recommendations are surprising but still relevant, they can be seen as a serendipitous recommendation.

6.4.1 Content-based Diversity

The content-based diversity is defined with the help of using content-labels of articles. The named entities of an article were used as labels for the comparison. The diversity is described as the distance between two articles i and j, where a higher distance means more diversity:

$$dist(i,j) = 1 - \frac{L_i \cap L_j}{L_i \cup L_j}, \tag{6.1}$$

where L_i and L_j are the sets of labels describing the articles i and j, respectively [KB16]. To finally calculate the diversity of all recommendations to one article, the so-called average intra-list diversity was calculated, where the diversity distance is calculated for every possible pair of articles within the list of recommended articles, and the average over all these distances finally is calculated. So the contend-based diversity, as the average intra-list pairwise distance, is defined as:

$$Diversity^{cont}(i) = \frac{\sum_{j \in R} \sum_{k \in R \setminus \{j\}} dist(j,k)}{|R|(|R|-1)}, \tag{6.2}$$

with the definition of the distance *dist* from equation 6.1, where i is the currently read article and R is the set of recommended articles to i [KB16].

Based on this definition, the range for the content-based diversity is 0 to 1, where a lower value describes a very low diversity within the recommended articles and a higher value describes a higher diversity.

6.4.2 Content-based Surprise

The content-based surprise tries to describe how obvious, or rather how not-obvious the recommended articles are in comparison to one specific article. As the content-based diversity, it is defined by Kaminskas et al. [KB16] and is based on the complement of the Jaccard similarity as well. With the same definition of the distance, in our case using named entities as labels, as in Equation 6.1, the content-based surprise of the recommended articles is defined as:

$$obj^{cont}_{surprise}(i) = \min_{j \in R} dist(i,j), \tag{6.3}$$

where i is the currently read article and R are all recommended articles [KB16].

Based on this definition, the range for the content-based surprise is 0 to 1. A higher value means that recommended articles appear as more surprising to the reader, while a lower value means that the recommendations are rather predictable for the reader.

6.4.3 Results

To finally receive results for the above defined measurements, a subset of the full test data set was taken, as the computation over the full test data set is not possible because of computational limitations. The chosen subset contained all articles that were published in 2019, which resulted in 2020 articles. From this subset, a set of 100 articles was randomly chosen, which was stored and used for the evaluation of all methods, so the used articles to create recommendations were always the same. These articles have nearly the same distribution of ressorts as the whole dataset, to utilize stratified random sampling [APSN13]. For every article out of these 100, with every method, the closest 10 articles according to each method, were calculated and the article itself together with these 10 recommendations were evaluated with the measurements defined above.

To give an example, how these recommendations look like, the top two recommendations for a small test sample of two articles for every chosen method were generated and can be seen in Table 6.4.

Base Article	Method	Recomm. Article
Sebastian Kurz ist schuld, sagt es allen!	TF-IDF	Ein Kanzler macht Schluss
		Der Kronzeuge der Kurz-Revolution
	Textrank	Schon wieder neue Pläne für Vice Austria
		Dolchmorde - die Geschichte zweier Städte
	Rake	Kaum Frauen: Weibliche Beute im Meer der Grafiken
		"BÖSE"
	DTM	Der Horror im Haus der Barmherzigkeit
		Stichhaltige Fakten
	Stylometry	"Böse"
		"Gut"
	Word Embeddings	Wir sollten den Druck gemeinsam erhöhen
		Ich brauche in der Politik keine Romanze
	Wordcloud Comparison	Es graust Kurz kurz - es kostet uns lang
		Warum ist jetzt genug?
Kannst du keine Knödel formen, sollst du sie im Ofen schmoren	TF-IDF	Herrscht im Neuen Jahr die Not, gibt's Bohneneintopf mit Brot
		Steht in jeder Küchenbibel: Heiße Suppe aus der Zwiebel
	Textrank	Eintritt frei: Die neunte Nacht der Programmkinos
		Podcast
	Rake	Steht in jeder Küchenbibel: Heiße Suppe aus der Zwiebel
		"BÖSE"
	DTM	Der Horror im Haus der Barmherzigkeit
		Stichhaltige Fakten
	Stylometry	"Böse"
		"Gut"
	Word Embeddings	Vom Glück, Schwein zu haben
		Tu Milch in deine Hendlsoß und mach sie mit Zitrone groß
	Wordcloud Comparison	Dem Masterplan entlang: Rap, Karaoke mit Maschek und Architekturtage
		Elektropop bis zum fidelen Freak-out

Table 6.4: Recommendation Examples

Method	Diversity	Surprise
TF-IDF	0.9665154	0.9108813
TEXTRANK	0.9771817	0.9392942
RAKE	0.9792527	0.9044800
Wordcloud Comparison	0.9862475	0.9587793
Dynamic Topic Modelling	0.9690465	0.9723538
Stylometry	0.9983222	0.9840161
Word Embeddings - selftr.	0.9703393	0.9433950
Word Embeddings - pretr.	0.9694106	0.9320432

Table 6.5: Recommendation Evaluation

The exact results for these measurements can be seen in Table 6.5. A graphical representation of the results can be seen in Figure 6.1

The results of these evaluations do correspond with the expected results in multiple ways. Most importantly, both the diversity and the surprise of the recommendations seem to be lower, the more a method is based directly on the occurring words of the articles themselves. For example the keyword-based methods TF-IDF, TextRank and RAKE, together with word embeddings, appear to have to lowest values for both measurements. Wordcloud Comparison, even though being a keyword-based method as well, has higher values, which can be explained by the fact that the timeframe of the occurring words has a high influence on the method as well, instead of only looking at the words themselves. Stylometry has the highest value for both measurements, which fits the expectations as well, as this method, in comparison to the other methods, focuses more on stopwords like articles, while other methods mostly ignore these words. Because the recommendations are built up on the style of the article, the diversity and surprise, which both are more based on the content of the articles, not the style, are higher than for other methods.
The fact that the pretrained word embeddings model has a lower diversity and lower surprise than the selftrained model was expected as well, because the pretrained model was expected to have more fitting word vectors and therefore create closer relations between the documents. What comes as a surprise is how little the difference in the results is, even though the pretrained model used millions of wikipedia articles, while the selftrained model only used about 2000 articles.
Interestingly, Dynamic Topic Modelling is the only method, where the content-based surprise exceeds the content-based diversity. This could only mean that the diversity between the recommended articles is rather low, while the diversity between the recommended article and the currently read article seems to be rather high. Then again, this can be explained by the chosen way of finding recommendations through using the highest correlating topic of the article and then using multiple highest correlating articles to this topic.

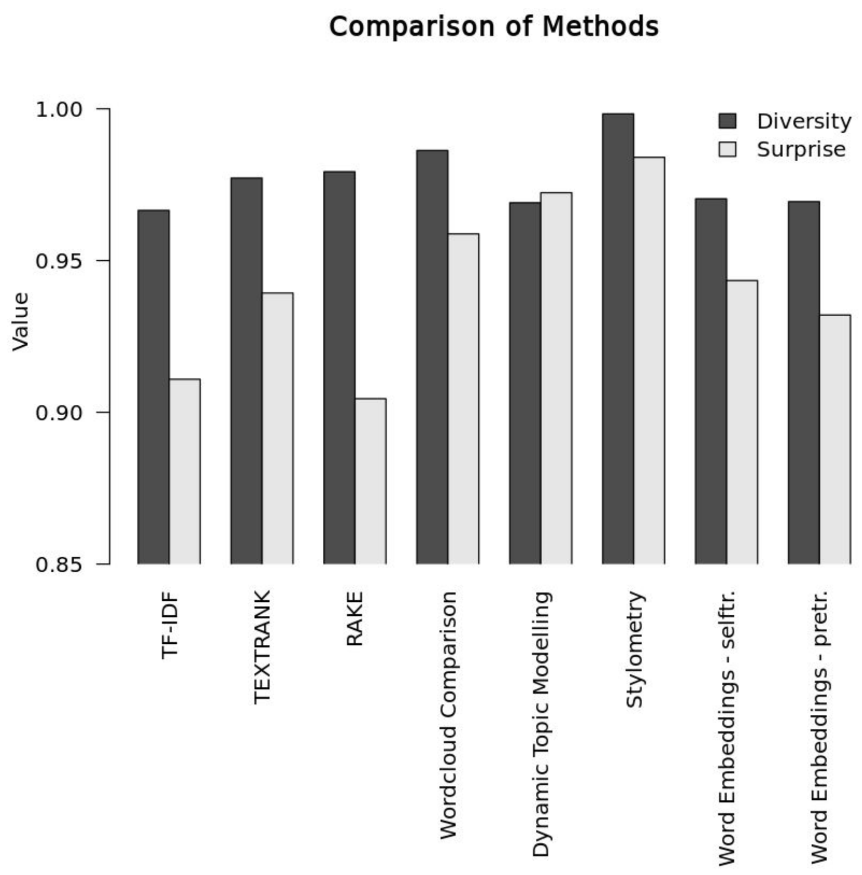


Figure 6.1: Comparison Results

CHAPTER 7

Conclusion

With this thesis we give new insights into the strengths and weaknesses of content-based methods with attention to a dynamic corpus, as well as an evaluation of content-based recommendations other than accuracy.
To further conclude this work, a summary of the answers to the research questions, a quick overview of found strengths and weaknesses of methods and an outlook for possible future work is given in this chapter.

7.1 Research Questions

In the first chapter of this thesis, research questions were stated. The following chapter summarizes the findings of this work and how the research questions are answered by those.

7.1.1 What methods, concepts or algorithms represent suitable mechanisms for the continuous analysis and subsequent classification and clustering of news articles based on general topics or specific keywords?

This question was partly answered by the literature research and partly by the implementation and evaluation of the methods.
The literature research concluded in the methods that are analyzed in this work, throughout the analysis of related literature, only those methods were finally chosen for further analysis, that were already successfully used by others for similar tasks. Therefore all analyzed methods are suitable for the analysis of news articles. Classification and clustering were analyzed through finding similar articles, or articles that are "close to each other".

As described in the comparison & evaluation chapter, all keyword based methods, which includes TF-IDF, TextRank, RAKE, but also Wordcloud Comparison, can be used to find important keywords. Dynamic Topic Modelling does not focus on single keywords, or groups of words, but rather finds important topics within the corpus.
Stylometry and Word Embeddings on the other hand are not very suitable for finding keywords or topics. While both have possibilities to find outstanding words, they can not be used to find specific keywords or topics that are important throughout a corpus of news articles. But they still can be used for classification and clustering of articles. Especially Stylometry, which has a very different approach than all other methods, by using mostly those words that are ignored by the other methods, is very suiatable for clustering and classification, the R-package that implements this methods even comes with an out of the box function for both these tasks.

But this question also asks for the continuous analysis, meaning the possibility to include further articles into an already calculated model.
For this task, the keyword based models are all not optimal, they all would have to recalculate the whole model after adding a single article, otherwise the calculated values would not fit the corpus anymore. Dynamic Topic Modelling is a double edged sword when it comes to this. While it is possible to easily include new articles into an existing model, by doing this for a longer period of time new topics that might have emerged will never be found, but new articles will only be fitted into the existing topics. With Word Embeddings it is important to differentiate between the self- and pre-trained model. The self-trained model would have to be completely recalculated with every new article, but the pre-trained model does not need any recalculation and together with Stylometry, which also can flawlessly include new articles, appear to be the best choices for this.

7.1.2 Which methods allow to recognize short or long term changes of e.g. key word clusters or topic correlations and which are not really suited to analyze changes at all?

Both methods, that are best suited for the continuous analysis of a corpus, i.e. Stylometry and Word Embeddings, appear to be the worst choices when it comes to finding trends. While both methods have variations that might be able to do this, as mentioned in the excluded methods chapter, their base methods, which are analyzed in this thesis, are unsuited for this task.
The keyword based methods, TF-IDF, TextRank and RAKE, all are moderately suitable for finding trends. As it has been shown throughout this thesis, they can be used to find trends within the keywords, as well as distinguishing between short- and long-term trends. But these options have to be fully self implemented and are not established at all.
Both Wordcloud Comparison and Dynamic Topic Modelling are very suited for finding trends, as both methods are basically created for finding trends, and both can easily be used to also differentiate between short- and long-term trends. With the only difference that Wordcloud Comparison analyses keywords while Dynamic Topic Modelling analyses topics.

7.1.3 What methods build close relations between news articles yielding results in a more focused way and what methods lead to a broader spectrum of relations generating serendipitous connections?

To answer this question, the content-based diversity and the content-based surprise were used as characteristics of article recommendations. The final result of a comparison done with the same subset of articles can be seen in Figure 6.1.
Based on this analysis, it can be said that the keyword based methods result in the least surprise of recommendations, but when it comes to the diversity, they are at a similar level as Wordcloud Comparison, Dynamic Topic Modelling and Word Embeddings. Stylometry results in both the highest diversity as well as the highest surprise of recommendations. Interestingly, Dynamic Topic Modelling resulted in second place when it comes to surprise, only beaten by Stylometry, but on the other hand it comes in last place regarding the diversity of a set of recommendations.

7.2 Strengths and Weaknesses of All Methods

To conclude this comparison, it can be said that every method has its strengths and weaknesses. No method could be found that exceeds all other methods in all aspects that we considered.
A conclusion of strengths and weaknesses of all methods is given in the Tables 7.1 to 7.7.

TF-IDF Scores

Pros	Cons
Finding keywords	Prefers rare words if they appear in only one document
Widely established for finding keywords	Inclusion of new articles not possible
Rather fast	Finding trends possible but not easy
Possible variations (named entities only, average values, etc.)	Little diversity and surprise in recommendations
Very easy to implement	
Fitting, close recommendations	
Possibility to find trends	

Table 7.1: Pros and cons of TF-IDF Scores

The TF-IDF scores method, as a keyword based method, obviously has its strength in finding keywords. It is a very well established method, that is also easy to implement. It is also versatile when it comes to further processing the resulting scores, for example for finding trends within the scores. The resulting recommendations of articles based on the TF-IDF scores can be seen as rather close.

TextRank Algorithm

Pros	Cons
Finding keywords	Parameter optimization needed to fit expectations
Finding keyword groups	Inclusion of new articles not possible
Established algorithm to find relations/-connections	Finding trends possible but not easy
Easy to use frameworks exist	Little diversity and surprise in recommendations
Possibility to find trends	

Table 7.2: Pros and cons of TextRank Algorithm

The TextRank algorithm can not only be used to find keywords, but also to find keyphrases or important groups of words, therefore finding keywords/groups is one of its main strengths. Easy to use frameworks exist that can be used to calculate the scores. It gives the possibility to find trends, but not out of the box, so while it is possible, it has to be done manually. Recommendations based on the TextRank algorithm can be seen as rather close, with little surprise in the recommendations. The parameters for this method, like types of words to include and exclude as well as the length of the wordgroups have to be optimized to meet the expectations.

RAKE Algorithm

Pros	Cons
Finding keywords	Parameter optimization needed to fit expectations
Finding keyword groups	Inclusion of new articles not possible
Established algorithm to find relations/-connections	Finding trends possible but not easy
Easy to use frameworks exist	Little diversity and surprise in recommendations
Possibility to find trends	
Very fast	

Table 7.3: Pros and cons of RAKE Algorithm

The major strength of RAKE, which can be used to find keywords and phrases, is its computational speed. It is very fast, even for big datasets, and yields in comparable results regarding the quality of the extracted keywords in comparison to TF-IDF and TextRank. Recommendations can also be seen as rather close, same as for the other keyword based methods. Again, frameworks that can easily be used exist, so no implementation of the algorithm is needed. The possibility to find trends with the method exists, but is not established and has to be done manually. Same as with TextRank, the parameters for this method have to be optimized to meet the expectations.

Wordcloud Comparison

Pros	Cons
Finding keywords	Inclusion of new articles not possible
Visual representation	Rather slight connections between articles
Intuitively understandable through representation	Little parameters to optimize results
Easy to find temporally trends	Quality of found keywords lower than other keyword-based methods
Easy to use frameworks exist	Keywords focus more on temporal difference in occurrences than keyword importance
Fast	

Table 7.4: Pros and cons of Wordcloud Comparison

Wordcloud Comparison is very well suited for finding keywords, it also comes with a graphical representation of both the found keywords as well as a comparison of keywords for different subsets of the data, which are easily understandable and already widely used. Finding trends can be done via the Wordcloud Comparison method, for which frameworks exist. It is also a very fast method, even with big datasets. The resulting recommendations based on this method are also rather close, with little surprise in the recommended articles. A downside of the found keywords is that they don't focus on the importance of the corpus itself, but rather on the differences between the subsets that are compared, but to overcome this issue, simple wordclouds, without the comparison algorithm, could be used.

Dynamic Topic Modelling

Pros	Cons
Finding topics and corresp. keywords	Computational intensive / slow
Finding correlations between documents and topics	Complex to use, even though frameworks exist
Based on established topic modelling algorithm	Tweaking of parameters takes effort and know-how
Frameworks to use exist	Number of topics is fixed
Inclusion of new articles	Recalculation of topics computationally very expensive
Visualization of temporal trends	

Table 7.5: Pros and cons of Dynamic Topic Modelling

The Dynamic Topic Modelling method has its major strength in not finding only keywords, but whole topics within the corpus. Because of existing frameworks, it is also easy to use, with little self implementation needed. The framework also comes with the possibility to build correlations between topics and articles, as well as with a way to find trends within the topics and a graphical representation of these trends.
On the other hand, this method is rather slow and needs a lot of computational resources. It is also not very easy to find the ideal number of topics within a corpus, and adding new articles to a calculated model can only be done by fitting it into the already found topics, with no regard to possibly new emerging topics.

Stylometry

Pros	Cons
Comparing styles of authors	Finding topics or keywords not possible
High diversity and surprise in recomm.	Computationally extensive to initially calculate model
Correlating most used words to authors	Recommendations might appear as random
Inclusion of new articles easily possible	Not an established method for text recommendation
Different approach than other methods	
Easy to use frameworks exist	
Visual representations	

Table 7.6: Pros and cons of Stylometry

While Stylometry can not be used to find keywords, topics or trends, it still can be used to compare articles, or subsets of articles, with each other. Its strength comes from its different approach for finding similar articles, not based on important keywords, but based on the style of the author. Therefore, the recommendations based on this method are rather diverse and surprising compared to the other methods. Easy to use frameworks do exist, that also come with a graphical representation of the results.
The high diversity and surprise of the recommendations can also be seen as a downside, because they might appear as random to the reader. Stylometry also is not an established method for text recommendation.

Word Embeddings

Pros	Cons
Using self-trained or pre-trained model possible for different results	Computationally very expensive
Close connections between recommended articles	Training of own model takes very long and vast amounts of memory
Very different approach compared to other methods	Results not humanely interpret able - vectors seem arbitrary
Very easy to include new articles with pre-trained model	No parameters for optimizing results
Possibility to experiment with results of self-trained model	Frameworks exist, but still need know-how to use
Current state of the art for text analyses	

Table 7.7: Pros and cons of Word Embeddings

For the Word Embeddings, both a self- and a pre-trained model can be used, which both have different strengths and weaknesses. While the self-trained model might result in more diverse and surprising results of recommendations, the pre-trained model can be used continuosly with very little computational effort to include new articles. It is also seen as one of the current state of the art methods for text analyses with various new developments, which might allow for finding trends in the near future as well.
One big downside is the very high computational demand of the method. Calculating the word-vectors for the self-trained model takes quite some time and big amounts of memory, but also calculating recommendations with he pre-trained model uses high amounts of memory, when using high-dimensional word vectors.

7.3 Practical Implications

Based on our findings, we recommend newspapers like *Der Falter*, that are planning to implement a recommender system, to keep the following in mind:
For finding trends, both Wordcloud Comparison and Dynamic Topic Modelling can be used to find trends. They both enable the user to include the trends into recommendations as well as visualizing them in an intuitive way. But these methods both come with the drawback that they are practically unable to include new articles into the model for a longer period of time.
To overcome this, both of these methods would have to be not used alone for finding trends and building recommendations, but to combine them with other methods that are able to include new articles. For example using Word Embeddings for building recommendations and combining those with the trends from the Dynamic Topic Model. This would lead to a more stable model with lasting recommendations that is also able to find trends and include those into the recommendations.

7.4 Future work

The main focus of this thesis was to find and compare content-based methods for recommendation, keyword extraction, and trend detection in the domain of online news. It was concluded that all methods have its strengths and weaknesses in different aspects. Future work mainly concerns the deeper analysis of the tested methods with more diverse datasets, the inclusion of methods which were not included into this analysis due to various reasons like no existing implementation guides or frameworks, the combination of different methods to maybe compensate their downsides, or the implementation of a full recommender system with the inclusion of one or more methods described in this thesis.

- Deeper analysis with more diverse dataset
  The results from this work are all based on one test dataset, which is written in German, or sometimes even with bits of Viennese dialect. Using different datasets, for example in other languages, or with a different type of text might yield different results.
- Inclusion of other methods
  Some methods that might have been promising for this thesis have been excluded for different reasons like no existing frameworks during the selection process of the methods. This might already have changed since then.
  New methods for text analysis and news recommendation might emerge throughout the next years, which prompts further analysis of these new methods and comparing them against older and already established methods.
- Combination of different methods
  One way to continue this work would be the further combination of different methods for trying to combine the strengths of methods. On a very small scale this

was already done in this thesis, for example when combining the Word Embeddings with TF-IDF scores to create meaningful document vectors out of the word vectors. But the possibilities of combining these methods goes way further and might result in better results than each of the analyzed methods in this thesis have achieved. Especially the combination of methods to improve the compared abilities within this thesis, like combining Stylometry for diverse and serendipitous recommendations with Dynamic Topic Modelling for finding trends while still allow for an easy continuous analysis of the corpus, for example might be interesting to further look into.

- Implementation of a recommender system
  Another obvious next step would be using the methods analyzed in this thesis, maybe with combining multiple methods, and implementing a news recommender system. This would also suggest further analysis with real people to examine and verify the the recommendations.

List of Figures

List of Tables

List of Algorithms

Bibliography

[APSN13] Anita S Acharya, Anupam Prakash, Pikee Saxena, and Aruna Nigam. Sampling: Why and how of it. *Indian Journal of Medical Specialties*, 4(2):330–333, 2013.

[AS08] Ana Isabel Rojão Lourenço Azevedo and Manuel Filipe Santos. Kdd, semma and crisp-dm: a parallel overview. *IADS-DM*, 2008.

[AWC$^+$07] Shlomo Argamon, Casey Whitelaw, Paul Chase, Sobhan Raj Hota, Navendu Garg, and Shlomo Levitan. Stylistic text classification using functional lexical features. *Journal of the American Society for Information Science and Technology*, 58(6):802–822, 2007.

[BGJM17] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[BL06] David M Blei and John D Lafferty. Dynamic topic models. In *Proceedings of the 23rd international conference on Machine learning*, pages 113–120, 2006.

[BNJ03] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[Bur02] John Burrows. 'delta': a measure of stylistic difference and a guide to likely authorship. *Literary and linguistic computing*, 17(3):267–287, 2002.

[CT20] Tomas Kalibera CRAN Team, Duncan Temple Lang. *XML: Tools for Parsing and Generating XML Within R and S-Plus*, 2020. R package version 3.99.

[Dia20] Gene Diaz. Stopwords iso. `https://github.com/stopwords-iso`, 2020.

[DRB19] Adji B Dieng, Francisco JR Ruiz, and David M Blei. The dynamic embedded topic model. *arXiv preprint arXiv:1907.05545*, 2019.

[DS18] Qing Wang Dmitriy Selivanov. *text2vec: Modern Text Mining Framework for R*, 2018. R package version 0.5.1.

[ea19] Tyler Rinker et al. *lexicon: Lexicons for Text Analysis*, 2019. R package version 1.2.1.

[Ede13] Maciej Eder. Mind your corpus: systematic errors in authorship attribution. *Literary and Linguistic Computing*, 28(4):603–614, 2013.

[ER11] Maciej Eder and Jan Rybicki. Stylometry with r. In *DH*, pages 308–310, 2011.

[ERK16] M Eder, J Rybicki, and M Kestemont. Stylometry with r: a package for computational text analysis. r journal 8 (1): 107–21, 2016.

[fas] Wiki word vectors. `https://fasttext.cc/docs/en/pretrained-vectors.html`. Accessed: 2019-10-29.

[Fel18] Ian Fellows. *wordcloud*, 2018. R package version 2.6.

[FPSS96] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, 1996.

[GDBJ10] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 257–260, 2010.

[GDQ20] Emil Hvitfeldt Os Keyes Kanishka Misra Tim Mastny Jeff Erickson David Robinson Julia Silge Gabriela De Queiroz, Colin Fay. *tidytext: Text Mining using 'dplyr', 'ggplot2', and Other Tidy Tools*, 2020. R package version 0.2.5.

[GWFC11] Anatole Gershman, Travis Wolfe, Eugene Fink, and Jaime G Carbonell. News personalization using support vector machines. 2011.

[HBB$^+$11] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2011.

[HK07] Martin J Halvey and Mark T Keane. An assessment of tag presentation techniques. In *Proceedings of the 16th international conference on World Wide Web*, pages 1313–1314. ACM, 2007.

[Hol98] David I Holmes. The evolution of stylometry in humanities scholarship. *Literary and linguistic computing*, 13(3):111–117, 1998.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*, page 134. Springer Science & Business Media, 2009.

[Hua08] Anna Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, volume 4, pages 9–56, 2008.

[KB10] Michal Kompan and Mária Bieliková. Content-based news recommendation. In *International conference on electronic commerce and web technologies*, pages 61–72. Springer, 2010.

[KB16] Marius Kaminskas and Derek Bridge. Diversity, serendipity, novelty, and coverage: a survey and empirical analysis of beyond-accuracy objectives in recommender systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 7(1):1–42, 2016.

[KB19] Akitaka Matsuo Kenneth Benoit. *spacyr: Wrapper to the 'spaCy' 'NLP' Library*, 2019. R package version 1.2.

[KBB$^+$09] Barbara Kitchenham, O Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, 51(1):7–15, 2009.

[KFD12] Evan Kirshenbaum, George Forman, and Michael Dugan. A live comparison of methods for personalized article recommendation at forbes. com. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 51–66. Springer, 2012.

[KJJ18] Mozhgan Karimi, Dietmar Jannach, and Michael Jugovac. News recommender systems–survey and roads ahead. *Information Processing & Management*, 54(6):1203–1227, 2018.

[KV15] Carmen Klaussner and Carl Vogel. Stylochronometry: Timeline prediction in stylometric analysis. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 91–106. Springer, 2015.

[LK08] Sungjick Lee and Han-joon Kim. Automatic keyword extraction from news articles using tf-idf model. *Networked Computing and Advanced Information Management*, 2, 2008.

[LS18] Hao Li and Yuan Sun. English education text recommendation technology based on word embedding. In *2018 International Conference on Big Data and Artificial Intelligence (BDAI)*, pages 82–86. IEEE, 2018.

[ME19] Mike Kestemont Steffen Pielstroem Maciej Eder, Jan Rybicki. *stylo: Stylometric Multivariate Analyses*, 2019. R package version 0.7.1.

[MK18] Michael Jugovac Mozhgan Karimi, Dietmar Jannach. News recommender systems - survey and roads ahead. *Information Processing Management*, 54:pp. 1203–1227, 2018.

[ML14] David Mimno and Moontae Lee. Low-dimensional embeddings for interpretable anchor-based topic inference. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1319–1328, 2014.

[MSC$^+$13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[MT04] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 404–411, 2004.

[MWHL17] Hualong Ma, Xiande Wang, Jianfeng Hou, and Yunjun Lu. Course recommendation based on semantic similarity analysis. In *2017 3rd IEEE International Conference on Control Science and Systems Engineering (ICCSSE)*, pages 638–641. IEEE, 2017.

[NSF$^+$17] Tempestt Neal, Kalaivani Sundararajan, Aneez Fatima, Yiming Yan, Yingfei Xiang, and Damon Woodard. Surveying stylometry techniques and applications. *ACM Computing Surveys (CSUR)*, 50(6):1–36, 2017.

[PBM13] Joshua T Pyle, Nicholas W Barrett, and Christopher Markley. Textual document analysis using word cloud comparison, March 19 2013. US Patent 8,402,030.

[Pen11] James W Pennebaker. The secret life of pronouns. *New Scientist*, 211(2828):42–45, 2011.

[R$^+$03] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. Piscataway, NJ, 2003.

[RECC10] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. Automatic keyword extraction from individual documents. *Text mining: applications and theory*, 1:1–20, 2010.

[RST$^+$14] Margaret E Roberts, Brandon M Stewart, Dustin Tingley, et al. stm: R package for structural topic models. *Journal of Statistical Software*, 10(2):1–40, 2014.

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[SK17] Jieun Son and Seoung Bum Kim. Content-based filtering for recommendation systems using multiattribute networks. *Expert Systems with Applications*, 89:404–412, 2017.

[SNW19] Mete Sertkan, Julia Neidhardt, and Hannes Werthner. Documents, topics, and authors: Text mining of online news. In *2019 IEEE 21st Conference on Business Informatics (CBI)*, volume 1, pages 405–413. IEEE, 2019.

[TGB15] Poonam B Thorat, RM Goudar, and Sunita Barve. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110(4):31–36, 2015.

[Tra07] Josef Trappel. The austrian media landscape. *European media governance. National and regional dimensions*, pages 63–72, 2007.

[UDPU08] JW Uys, ND Du Preez, and EW Uys. Leveraging unstructured information using topic modelling. In *PICMET'08-2008 Portland International Conference on Management of Engineering & Technology*, pages 955–961. IEEE, 2008.

[wep] Corola-based word-embeddings. `https://www1.ids-mannheim.de/fileadmin/kl/CoRoLa_based_Word_Embeddings.pdf`. Accessed: 2020-01-30.

[WH00] Rüdiger Wirth and Jochen Hipp. 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*, pages 29–39. Springer-Verlag London, UK, 2000.

[Wij19a] Jan Wijffels. *textrank: Summarize Text by Ranking Sentences and Finding Keywords*, 2019. R package version 0.3.

[Wij19b] Jan Wijffels. *udpipe: Tokenization, Parts of Speech Tagging, Lemmatization and Dependency Parsing with the 'UDPipe' 'NLP' Toolkit*, 2019. R package version 0.8.3.