Automated Identification of Paid Articles from Online News Platforms

Theresa Obermayr

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2019

Declaration

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

Hagenberg, June 24, 2019

Theresa Obermayr

Contents

Abstract

Paid news articles are listed within the daily news feed and have to be tagged as sponsored due to legal reasons, but there is no unified labelling. The goal of the thesis is to find suitable automated approaches to distinguish paid and non paid articles. It should be shown if such a distinguishment is feasible and if so different algorithms will be applied to the dataset and compared to each other to find out which one performs best, while keeping over- and underfitting in mind. Further it is investigated if the performance of the machine learning models can be increased by including the sentiment of the articles into the classification. All approaches handled in this thesis are implemented in the project. The approaches can be evaluated in the thesis based on the evaluation results of the project. Also the challenges and problems that occurred from data collection to implementation are tackled in this thesis.

Kurzfassung

Bezahlte Nachrichtenartikel werden im täglichen Newsfeed aufgelistet und müssen aus rechtlichen Gründen als gesponsert gekennzeichnet werden, es gibt jedoch keine einheitliche Kennzeichnung. Ziel der Arbeit ist es, geeignete automatisierte Ansätze zur Unterscheidung von bezahlten und nicht bezahlten Artikeln zu finden. Es soll gezeigt werden, ob eine solche Unterscheidung möglich. Wenn die Unterscheidung möglich ist werden unterschiedliche Algorithmen auf den Datensatz angewendet und deren Genauigkeit miteinander verglichen um herauszufinden, welche davon am Besten funktionieren. Dabei wird auf die Über- und Unterpassung der Daten Acht gegeben. Weiterhin wird untersucht, ob die Leistung der automatisierten Lernmodelle durch die Einbeziehung der Stimmung der Artikel in die Klassifizierung gesteigert werden kann. Alle in dieser Arbeit behandelten Ansätze sind im Projekt umgesetzt. Die Ansätze können in der Arbeit auf Basis der Bewertungsergebnisse des Projekts evaluiert werden. Auch die Herausforderungen und Probleme, die von der Datenerhebung bis zur Implementierung aufgetreten sind, werden in dieser Arbeit behandelt.

Chapter 1

Introduction

Automated text classification is no new research area but its history dates back to the beginning of the 1960s. At this time text classification was not fully automated but dependent on human defined heuristic methods and a set of rules defined by experts. This approach is highly inefficient as new rules always have to be introduced by humans to adapt to new data. Also the detection of complex patterns and relationships requires a lot of time for humans. Nowadays, the massive increase of textual data available online, such as online documents, news articles or social media posts renewed and intensified the interest in the research areas of automated text classification and data mining. The focus nowadays is on fully automated classification and clustering of the data provided as the amount of data often is too big to be able to manually analyse all of it [26].

When classifying text it can be distinguished between two main categories: binary and multiclass classification. The goal of the thesis project is to distinguish paid from non paid news articles, which is a binary classification problem. Therefore, this thesis focuses on binary text classification.

1.1 Problem Definition

Paid news are articles sponsored by a person or company to promote themselves, their company or products. Those articles are published on a news platform and listed within the daily news feed. News platforms must tag articles as sponsored due to legal reasons. Announcements, recommendations and other contributions and reports, for whose publication in periodical media a fee was paid, must be marked as *Anzeige* (advertisement), *entgeltliche Einschaltung* (paid insertion) or *Werbung* (promotion) as instructed in § 26 media law in [20]. Since there are three terms accepted to mark an article as sponsored there is no unified labelling. Also the placement of the identifying keyword differs along platforms. While some platforms provide this information instead of the author tag, others display it instead of the category or as an image banner with the text "Anzeige" at the end of the article.

1.2 Research Question

To find out if paid news articles can be automatically distinguished from non paid news articles, multiple algorithms are implemented, optimized, evaluated and compared to each other. Based on the evaluation and comparison the following research question will be answered: To which extent can paid articles be automatically identified from online news platforms?

1.3 Solution Approach

The identification of paid news articles is approached with text classification, which is a field in machine learning. The approach to identify paid news articles is similar to the approaches of fake news and spam detection. These approaches deal with binary classification problems and can therefore be partially applied to the identification of sponsored news articles. Since there are numerous solving approaches it is necessary to detect which one fits best on this particular use case. To tackle this problem the following approaches will be used: Support Vector Machine, Logistic Regression and Sentiment Analysis.

As data is one of the most important factors to get good results, a lot of high quality data is needed. Therefore the first big step of the project is to gather enough data, clean the data and finally label this data in paid and non paid articles.

As a result it should be shown if paid news articles can be distinguished from non paid articles using machine learning. To achieve this the selected algorithms will be evaluated and compared to each other. For the evaluation the accuracy, precision, recall and f1-score as well as the confusion matrix will be calculated. The goal is to find the best suitable method to section articles in two categories: paid and non paid.

1.4 Thesis Structure

The thesis is structured into 5 main parts. The first chapter covers the technical foundation and provides basic knowledge about the technologies used in the thesis project. In the second chapter, State of the Art, similar solution approaches are analyzed, summarized and their relevance for this thesis is shown. The methodology the implementation is based on is handled in the third chapter, followed by the implementation itself. Chapter 6 contains the evaluation and comparison of all algorithms used, as well as a summary and conclusion of the results. The last part of the thesis tackles conflicts that occured and possibilities for further research.

Chapter 2

Technical Foundation

2.1 Crawling and Scraping

Scraping is the automated process of downloading, parsing and extracting data from web pages. This automation accelerates the process of gathering data from online platforms a lot, as no human has to manually access the pages and copy the desired information anymore. The outsourcing of this task to a computer program is much faster and correcter. If not only one but multiple pages should be scraped, *crawler* come into play as they crawl across one to many pages. The distinction between crawling and scraping is very vague and therefore these two terms are often used interchangeably [3].

2.2 Machine Learning

Machine learning can broadly be divided into two main strategies: supervised and unsupervised learning. In supervised learning labeled training data is provided to the system to learn from it. For example labeled news articles with the labels paid true or false are fed to the system. From this labeled data the system deciphers features of each label which are then taken to classify new articles as paid true or false. In unsupervised learning the training data provided to the system is not labeled. The system deciphers features and groups the data based on similarities. The articles for example are then not group as paid true or false but as groups of similar texts. There are two main approaches how the system can predict something: Regression and Classification. Regression forecasts continuous variables, such as the price of a house or the temperature for a specific time. Classification is used for data that has a few distinct outcomes, such as paid or non paid / sunny, foggy or rainy. A typical algorithm for a regression (continuous variable) is the Linear Regression and for classification (discrete variable) the Logistic regression [1, S. 1–2].

Figure 2.1 shows a hierarchical selection of the machine learning approach based on data, domain and division. When looking at the hierarchy from top to bottom, the first decision is whether to choose supervised or unsupervised learning. If the data is not labeled clustering, which is a methodology of unsupervised learning, is chosen. If the data is labeled a supervised approach is chosen. To decide which model fits best, the domain of the data is inspected. If the data domain can not or should not be divided, a

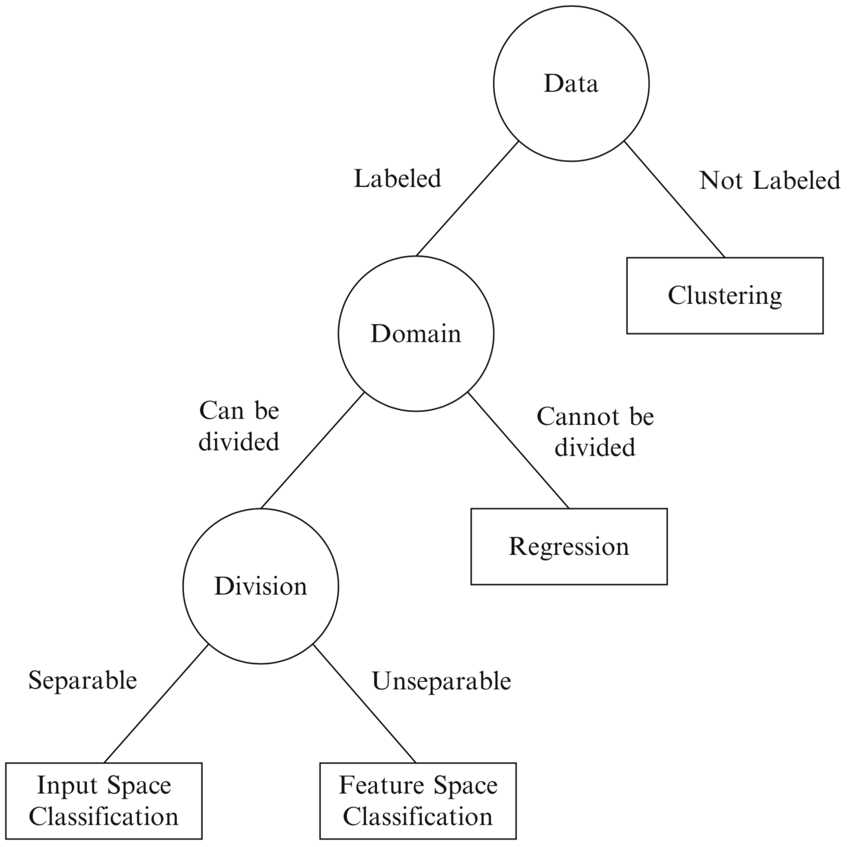

Figure 2.1: Hierarchical structure of distinction between regression, classification and clustering based on data-domain-division [12].

regression model is chosen. With a regression model continuous variables are forecasted, such as the temperature or the price of something. However, if the data domain can be divided (for example into a few distinct outcomes such as paid or non paid), the next level of the hierarchy, which deals with the the ease of the domain division, will be included as well. If the data points of the classes are separable, the original data domain can be divided and the classification can be applied (input space classification). If it is not possible to separate the data points of the classes, the original data domain has to be transformed into a feature space before the classification can be applied (feature space classification) [12].

2.3 Text Classification

Text classification, also known as document classification, is the process of assigning text documents one or more classes out of a set of predefined classes. This process can be automated using supervised or unsupervised machine learning techniques. Text documents can range from a phrase, a sentences to even one to many paragraphs of text. Unsupervised text classification does not require prelabeled data. With this approach the aim is to find patterns and cluster the data accordion to those patterns. For supervised learning on the other hand prelabeled training data is needed, from which the features are then derived. So even for automated supervised text classification, at first some data needs to be labeled manually. There are several types of text classification that differ in in how many classes the data should be divided into and how many classes are to be predicted for a data sample. The three types are: binary classification, multi-class classification and multi-label classification. Binary classification predicts one of two classes for a data sample. So the outcome is either the one or the other class. A use case for such a classification is for example spam filtering. Either the email is spar or not, there are no other options. Multi-class classification, also known as multinomial classification, predicts one class out of a set of more than two classes for one data sample. An example for this is the classification of fruits. The fruit can either be an apple, a banana or a strawberry but not more than one at the same time. Multi-label classification is used if for one data sample more than one class can be predicted. This option comes into play for example when predicting the category of a news article. One article can belong to the category politics and at the same time also to the category education [9].

2.4 Logistic Regression

For the master thesis project the classes paid and non paid will be predicted, which are discrete values. As already mentioned above, a typical algorithm for predicting discrete variables is the logistic regression. The function used for the calculation of a logistic regression is the sigmoid function. The feature of such a function is that the output varies from 0 to 1 as it plateaus after a certain threshold. Any predictions above 1 are capped to 1 and any prediction below 0 are floored at 0. Therefore, as an output the probability associated with an event is retrieved. Figure 2.2 shows such a sigmoid curve.

2.5 Support Vector Machine

The support vector machine can be used for classification as well as for regression. Regarding the amount of data which is necessary to achieve good results, the Support Vector Machine is a great choice for small datasets. A Support Vector Machine is used to find the optimal decision boundary to separate two classes. To build the optimal hyperplane (decision boundary), support vectors are needed. The support vectors are the points of each class that are closest to the hyperplane. To find the optimal hyperplane the margin, which is the space between the support vectors, needs to be maximized.

Figure 2.3 shows that there are multiple correct hyperplanes possible to separate the data. The blue hyperplane H1 as well as the red hyperplane H2 separate the data linearly. However these two solutions differ a lot concerning the margin of each hyperplane. H2 represents the optimized solution with the maximum margin whereas H1 is a random solution that also seperates the dataset correctly. In the Figure it can be seen that H1 is much closer to the data points of each class, therefore the margin associated with this hyperplane is small. To find the optimal position of the hyperplane the support vectors

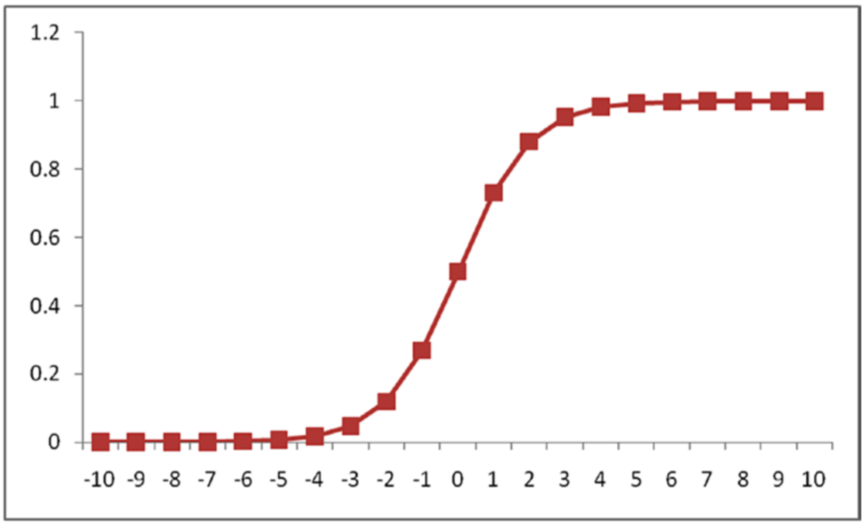

Figure 2.2: A sigmoid curve [1, S. 52].

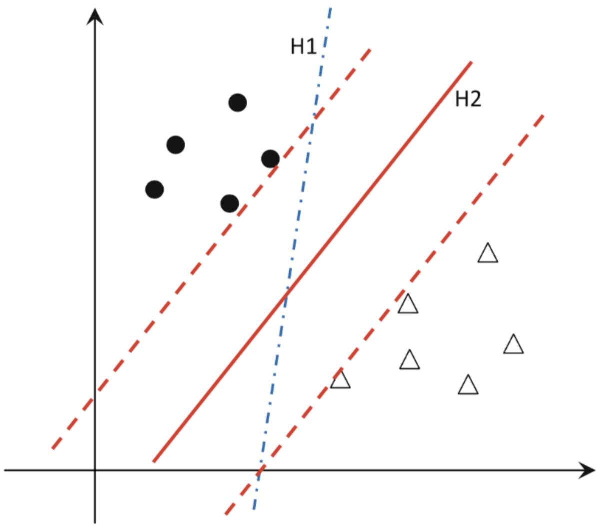

Figure 2.3: Example of maximum margin separator [16].

come into play. The position and alignment of H2 is defined by the position of the two red dashed lines, which are aligned along the support vectors [16].

When the data is not linearly separable in its input space, the input space can be mapped to a feature space using a kernel function so that the data becomes linearly separable in the feature space (as shown in Figure 2.4).

If there is noise in the data, the Support Vector Machine can be subject to overfitting. To avoid overfitting some training errors have to be tolerated in order to reach a satisfying balance between a maximum margin and a minimum error rate to sustain a

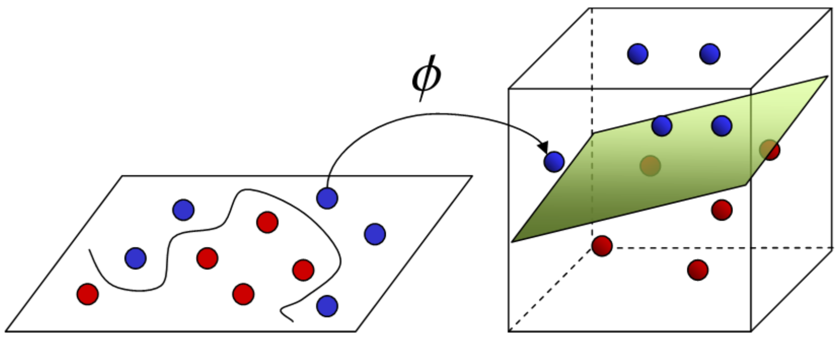

Figure 2.4: Input space (left) to feature space (right) conversion [33].

proper generalization power of the algorithm. This trade-off can be controlled via the soft margin constant C [4].

2.6 Neural Network

A Neural Network is a supervised machine learning algorithm and consists of multiple neurons that are interconnected to one or more other neurons. This interconnections form the network [17]. Neural networks are well suited for complex shapes within the data that can not be calculated with linear or logistic functions. The more complex the problem is the more complex the function needs to be. With the complexity of the function also the accuracy of the Neural Network increases [1, S. 135].

Figure 2.5 shows an abstract overview of the structure of a Neural Network. The left side represents the input layer, which is composed of the independent variables. These inputs are taken to predict the dependent variables. This prediction is modeled as the output layer on the right side. The number of nodes of the output layer is defined by the number of classes available in dependent variable (distinct values of the dependent variable). In a regression problem for example, there will be only one node in the output layer. Between the input and the output layers, there can be multiple hidden layers. These layers transform the input variables into a higher order function. These hidden layers are essential for identifying complex relations and patterns [1, S. 136].

2.7 Hyperparameter Tuning

Hyperparameter tuning is the process of finding the optimal combination of hyperparameters for an algorithm to increase its performance. Hyperparameters are parameters, which are defined before the construction of the model. Such parameters are for example the soft margin constant C for the Support Vector Machine or the class weight for the Support Vector Machine/Logistic Regression. There are two main approaches to optimize the hyperparameters for an algorihm: Grid Search and Random Search. For the grid search a set of hyperparameters and their possible values is defined. Then all possible combinations out of these parameter values are search through exhaustively.

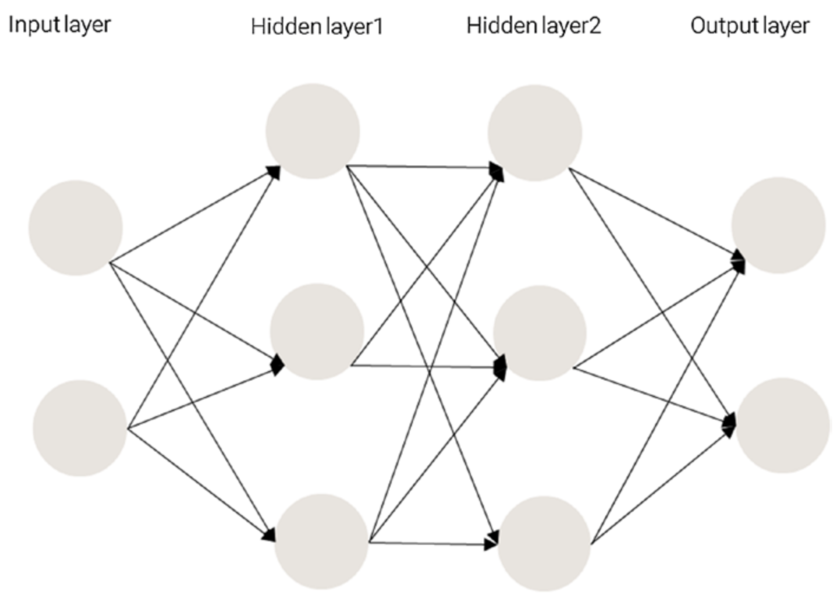

Figure 2.5: Neural network structure [1, S. 136].

The advantage of this method is that it guarantees that the best parameter combination will be found. The drawback is that this exhaustive search can be computationally expensive and time consuming. The Random Search scores with lower processing time as the hyperparameters are searched randomly. However this comes with the drawback that it is not guaranteed that the optimal parameter combination will be found [18].

2.8 Feature Extraction

Since machine learning models cannot understand texts directly but only numeric representations, numeric features have to be extracted from the input data. There are multiple approaches of how to extract features from textual data such as the TF-IDF model, Bag of Words or the one hot encoding. One problem with the Bag of Words model is that the feature vectors are based on absolute term frequencies. As a consequence words that occur less often can easily be overshadowed by words that occur in nearly all documents. The problem is that words that occur in nearly all documents are not significant for classifying the texts into different categories. In fact, words that occur less often are way more informative and relevant to distinguish classes. The TF-IDF (term frequency-inverse document frequency) strategy solves this problem, as words that occur often weigh less that words that occur less frequently [9].

For a Neural Network, features are extracted from the texts using one hot encoding. Given a "numeric representation of any categorical feature with m labels, the one hot encoding scheme, encodes or transforms the feature into m binary features, which can only contain a value of 1 or 0. Each observation in the categorical feature is thus converted into a vector of size m with only one of the values as 1 (indicating it as active)."

As a numeric representation of the categorical feature is needed, all text labels have to be converted to numeric representations beforehand [10].

2.9 Sentiment Analysis

Sentiment analysis is the process of extracting emotions from texts to conclude their polarity. One big challenge when analysing the sentiment is the complexity of the language. Words can have different meanings depending on the context they are used in. Their meaning is dependent on the knowledge domain, the area of expertise and if they are combined with a negation. The automation of sentiment analysis requires complex and well-defined boundaries to bring clarity to that ambiguity. Furthermore, the sentiment analysis is also dependent on the domain and media the texts come from. Articles from a news platform for example require a different analysis that texts from social networks such as Facebook or twitter. Articles from news platforms follow grammatical rules and usually contain no misspellings. Social media posts on the other hand do not follow grammatical rules, sentences are often incomplete, misspellings are occasionally contained in the posts and emoticons are included to show emotions. The results from a sentiment analysis can be either the polarity or the range of polarity. The polarity means that the result is either positive, neutral or negative. An example for the range of polarity are star ratings or rankings on a scale from 1 to 10 [15].

2.10 Technologies Used

The project is implemented in the programming language Python. The libraries used are scikit-learn, NLTK and Keras.

Scikit-learn: Scikit-learn is a free machine learning library for Python which offers various tools for among others data mining and data analysis. It is open source, commercially usable and build on NumPy, SciPy, and matplotlib. The main applications of scikit-learn are [7]:

- Classification: the identification the class a data point belongs to.
- Regression: the prediction of a continuous output variable for a data point.
- Clustering: the grouping of similar data points.
- Dimensionality reduction: the reduction of considered random variables.
- Model selection: the comparison and validation of models and hyperparameters.
- Preprocessing: the process of data preparation and feature extraction.

NLTK: NLTK, which stands for Natural Language Toolkit, is a package for building text analysis programs in python. It consists of the most common algorithms for natural language processing such as tokenization, stemming, sentiment analysis and many more. NLTK is well documented and provides a hands-on guide in which the underlying principles behind the supported language processing tasks are explained [23; 28].

Keras: Keras is a Neural Networkk API written in Python. It is a high-level API and therefore easy to use. With keras easy and fast prototyping is possible because it is modular, user friendly and easy to extend. Regarding the modularity the aim of keras is to encapsulate all functionalities in standalone modules which can later be combined when creating new models [19].

Chapter 3

State of the Art

This chapter analyses relevant state of the art approaches of automated text classification. Several papers are taken into consideration that tackle similar tasks to the one's in the master thesis project, such as binary text classification, the imbalance of data or finding the optimal model for a certain task. The chapter is divided into two main parts: the sequential analysis of existing papers in Section 3.1 and a summary and the relevance of the papers for the master thesis project in Section 3.2.

3.1 Analysis

In this section multiple papers are analysed in terms of their classification approaches. The analysis focuses on the complete pipeline developed from data acquisition to evaluation, which includes among others data preparation and text preprocessing, classification algorithms, handling of imbalanced data as well as the system architecture.

3.1.1 Automatic Text Classification of Crawled News

The research in [14] intends to find the appropriate algorithm to automatically classify a news article in Indonesian language. The classification process from textual input to categorical output is shown in Figure 3.1 and consists of five main steps:

- Gather the textual input via web crawling and scraping,
- perform multiple text preprocessing steps,
- select relevant features from the preprocessed textual data,
- perform the classification based on the extracted features
- and finally receive the predicted category as output.

Since the classification process of the master thesis project also ranges from data acquisition via crawling to the prediction of a class for a certain article, the classification process introduced in Figure 3.1 is a good starting point to develop the classification process for the project. Similar to the approach of this paper, the data for the master thesis project is also gathered using web crawler. Regarding text preprocessing it is also experimented with several approaches such as lemmatization or the removal of stop words. One big difference is that the paper tackles multinominal classification whereas the task of this thesis project is binary classifiaction.

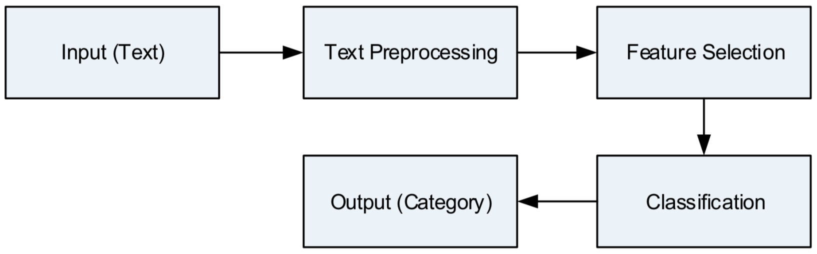

Figure 3.1: Classification process from input data to category output [14].

3.1.2 Comparison of multiple Machine Learning Algorithms for the Classification of Imbalanced Data

The approach stated in [8] can help to solve the task of the master thesis project as both problems are similar in terms of imbalanced data. Additionally, both approaches deal with binary classification of news articles. For the classification of the news articles in [8] multiple machine learning algorithms are considered to build binary classifiers. Based on these models an evaluation was performed to identify the best model.

Besides the algorithms it is also covered which data is available when scraping as well as which problems can occur regarding the data. Attention is drawn to the fact that the desired data might not be available for every article as the provided data can sometimes be very limited. The title as well as the body text of an article is no problem but the author of the article is often not provided or simply replaced with an agency. It is also noted that images must be ignored as the features for the models are based only on textual data. These remarks have to be kept in mind for the master thesis project as the data is also gathered using web crawling and scraping.

To find out which method works best for feature selection multiple variations were experimented with. Models were trained without any feature selection, meaning with the plain text, as well as with multiple feature selection variations in which it was experimented with the removal of stop words and stemming among others. Based on the analysis of the results it is claimed in [8] that stemming can have either a negative, a positive or no impact on the effectiveness of the model. From this they result that the effect of stemming cannot be generalized but is highly dependent on the input data. All in all, this concludes that feature selection is always different depending upon the algorithm and data used. It shows that there is not one best solution on how to prepare the data for an algorithm. It always depends on the algorithm and the data provided and has to be worked out individually through experimenting. The goal of [8] was to compare multiple algorithms and find out which one fits their data best. For the comparison two main measurement categories were defined: Effectiveness and Efficiency.

Four effectiveness measures have been selected which depend on the confusion matrix output, which are: True Positive, False Positive, True Negative, and False Negative. The following four effectiveness measures have been chosen in [8]:

- precision,
- recall,

Table 3.1: Effectiveness of F-Measure [8].

Combination	*SVM*	*C4.5*	*Naive-Bayesian*
Plain	**98.95**	96.91	97.96
Stop Word Removal	**100**	96.91	98.97
+ Stemming	98.95	98.95	**98.97**
+ Chi-Square FS	**100**	98.95	98.97

- accuracy
- and f-measure (micro-averaging).

The above stated effectiveness measures will also be used in the master thesis project to compare which variation of an algorithm performs best. In addition to effectiveness, the following efficiency measures have been chosen in [8]:

- The calculation time needed for model creation,
- the size of the model,
- the number of features for a model and
- further algorithm specifics such as the number of support vectors for Support Vector Machine models.

Concerning the efficiency measures only the first two items may become relevant when evaluating the thesis project.

Table 3.1 shows the results from [8] for the f-measure. From these results it can be seen that in three of four cases the Support Vector Machine (SVM) performed the best. However, with performances ranging from 96.91% to 100% it can be concluded that all three models result in a satisfying performance. Even though the Support Vector Machine outperformed the other models, it is mentioned in [8] that one big downside of the Support Vector Machine is a very high calculation time. Concerning the train and test split they settled for a ratio of 70/30 as this ratio showed the most promising results after several experiments.

When preparing the dataset of the master thesis project similar problems to the ones mentioned in [8] might occur with feature extraction because of incomplete data, such as missing authors. Similar to the data and the classification method of this paper, the master thesis project also deals with imbalanced data and binary classification. In the paper there is an equal focus on effectiveness and efficiency when comparing the results, whereas in the master thesis project the main focus is on effectiveness. The effectiveness measures for the master thesis project are similar to the measures from [8]. Efficiency measures will only be taken into consideration if the effectiveness of the models does not differ a lot.

3.1.3 Automated Text Classification using a Cost-Sensitive Support Vector Machine

When dealing with imbalanced data the aim of [6] is to increase the performance for the majority class while at the same time keeping the error rate of the minority class low. It is stated that the Support Vector Machine is a popular choice for such a use case as risk minimization is possible with this model. Another point that makes the Support Vector Machine a suitable approach is the fact that this model is able to generalize well on limited data, in contrary to models such as Neural Networks that require a lot more data. Further, the Support Vector Machine is well suited for imbalanced datasets, as the penalties associated with classes can be adapted. With adjusting the penalties it can be defined which classification errors are worse and therefore should be penalized higher [6].

A new Support Vector Machine, called Biased Support Vector Machine (BSVM) is introduced in [6]. This BSVM is based on the metrics recall and specificity. It is pointed out that a cost-sensitive Support Vector Machine usually provides better results in terms of recall than a standard Support Vector Machine. Nevertheless, it is criticised that even the cost-sensitive Support Vector Machine does not provide a way to regulate the level of recall. This is why the BSVM is introduced, which is said to be able to regulate the level of recall. According to [6] the main difference to the Support Vector Machine is that the BSVM always focuses on the priority class, regardless of which class the majority class is. The performances of the Support Vector Machine and the BSVM are evaluated and compared to each other based on an imbalanced dataset. For the evaluation it was decided to not use the accuracy, as this metric calculates the total correct classified instances, regardless of if the instances are correctly classified as negative or positive. This means that every correct classification weights the same. If one class, either positive or negative, is of greater importance the accuracy is not the right metric to choose as this metric can not favor one specific class. Taking this in consideration it was decided in [6] to use the precision and the f-value as metrics. The precision was chosen because the cost of False Positives should be high and the f-value was chosen because it keeps a balance between precision and recall (high cost associated with False Negatives). In the experiments of [6] a lower bound for the recall measure was defined. Since this bound was applied to the training data, the recall of the test data was not always greater or equal than this lower bound. However, it is claimed that BSVM produces good classification results in terms of recall as the recall of the test set is always close to the previously defined lower bound.

Since a Suppport Vector Machine is used in the master thesis project, one of the most important insights from [6] is that it is stated that a cost-sensitive Suppport Vector Machine is better in terms of recall than a regular Suppport Vector Machine. Although it is claimed that a BSVM delivers much better results, a cost-sensitive Suppport Vector Machine will be used in the master thesis project this algorithm is proven to be a promising method.

3.1.4 The Impact of Text Preprocessing on Text Classification

The importance of text preprocessing for text classification is shown in [13]. To highlight the importance, the impact of preprocessing on text classification is extensively examined. In this research several aspects such as text domain and language, classification accuracy and dimension reduction are considered. All possible combinations of the most common used preprocessing tasks are evaluated and compared to each other on two different domains (e-mail and news) and in two different languages (Turkish and English). After analyzing benchmark datasets it is believed that certain combinations of preprocessing tasks, depending on the domain and language of the texts, could significantly improve the accuracy of an algorithm [13].

The preprocessing tasks experimented with in [13] are: tokenization, stop-word removal, lowercase conversion and stemming. The extensive experiments were carried out in multiple languages, with different feature selection approaches and with multiple datasets namely binary and multiclass as well as balanced and imbalanced datasets. Since the text language of the master thesis project is neither English nor Turkish but German, the results of both languages are taken into consideration. The texts of the master thesis project are news articles, therefore the main focus is on the results of the news domain of [13]. The news datasets, both from the paper and the master thesis project, are imbalanced. Further, the classification algorithm used in the research of the paper as well as the master thesis project is, among others, the Support Vector Machine.

For the experiments in [13] four preprocessing tasks were selected. Since all possible combinations of the four selected preprocessing tasks are tested this leads to 16 different combinations in total. The possible values tested for each preprocessing task are:

- Tokenization is either alphanumeric or alphabetic,
- stop-word removal is either applied or not,
- lowercase conversion is either applied or not
- and stemming is either applied or not,

The results of these experiments show the importance of alphanumeric tokenization, as in feature set of the English news 10% of the selected terms contained numeric characters. Further investigations showed that especially business related news contained a huge amount of numeric terms, such as '1st" and "2nd". If alphabetic tokenization would have been applied to this dataset those terms would have been converted to "st" and "nd" and therefore their significant information would have been lost. The results also highlight that stop words removal is much dependent on the language, since it had a positive impact on English news but not on Turkish news. With stemming it is the opposite case: on Turkish news it has a positive impact and on English news not. According to the results, lowercase conversion should always be applied, regardless of domain and language because it reduces the feature size.

As a conclusion of these extensive experiments it is stated in [13] that certain optimal preprocessing combinations, which are always dependent on the data used, may significantly improve the performance of an algorithm. However, if the preprocessing combination is chosen badly this can also effect the algorithm the other way around and decrease its performance. Consequential it is stated that there is no best parameter combination that improves every model independent of data and language. From these results it can be deduced that for every new text classification task the different preprocessing tasks should be experimented with. Many preprocessing tasks influence the performance of the model differently depending on domain and language, which concludes that there is no best overall solution. However, it is concluded that preprocessing can improve the classification results notable. Consequently, text preprocessing will be an important step when preparing the data for the classification in the master thesis project.

3.1.5 Research if Sentiment Analysis can help to Identify Spam Messages

The research of [5] tries to find out if sentiment analysis can help to identify spam messages. The focus of this research is to improve spam detection with a conjunction of sentiment analysis and other text mining and natural language processing techniques. In order to be able to work with the sentiment, the sentiment of each message of a dataset first needs to be calculated and added as a new attribute. Then the classification results with and without the calculated sentiment attributes are compared. As a metric for the comparison the accuracy is used.

In their research the first step was to find out which classifiers with which hyperparameters perform best when identifying spam messages. As they only considered the Bayesian classifier, they experimented with multiple variations of it to find its optimal configuration to fit the problem. Before applying the classifiers of course multiple text preprocessing steps were performed on the dataset. The second step was to calculate the sentiment score of each email and add this score as a new attribute to the dataset. To determine the sentiment they considered two approaches: develop an own sentiment classifier and use an existing classifier. Their own sentiment classifier is based on SentiWordNet[1] with which the average sentiment polarity of each email was calculated. As an existing classifier the TextBlob[2] classifier was used which provides a simple API for the desired NLP tasks. To identify the optimal threshold for the sentiment analysis, a previously labelled dataset was taken and relabelled using the two sentiment classifiers mentioned above. Finally, the best performing classifiers identified in step one were applied to the dataset, which was extended with the sentiment features, and the results of the classifiers with and without the calculated sentiment score were compared to each other.

The main insight from [5] is the fact that spam messages are more likely to result in a positive sentiment than non spam messages. Based on this result it is claimed that the polarity of an email helps to improve the performance of the classifier in most cases. In the master thesis project it is examined if the polarity of texts helps to identify paid news articles, as it is believed that not only spam messages but also paid news articles have the tendency to result in a positive sentiment.

3.1.6 Comparison of Supervised Machine Learning Algorithms in terms of Accuracy when Classifying News Articles

Three supervised machine learning algorithms are used and to classify Nepali news articles and compared to each other in terms of accuracy in [11]. The features were extracted from the textual data using the TF-IDF approach. Then the same features were used in input data for the Naive Bayes, the Neural Network as well as for the Support Vector Machine. Finally the results from all three models are compared to each other. The news articles were collected from multiple news platforms with crawling. All in all, they collected news articles of 20 different topics such as business, health or sport. There are some minor imbalances in the news topic distribution but all in all there is no majority class.

[1] http://ontotext.fbk.eu/sentiwn.html
[2] https://textblob.readthedocs.io/en/dev/

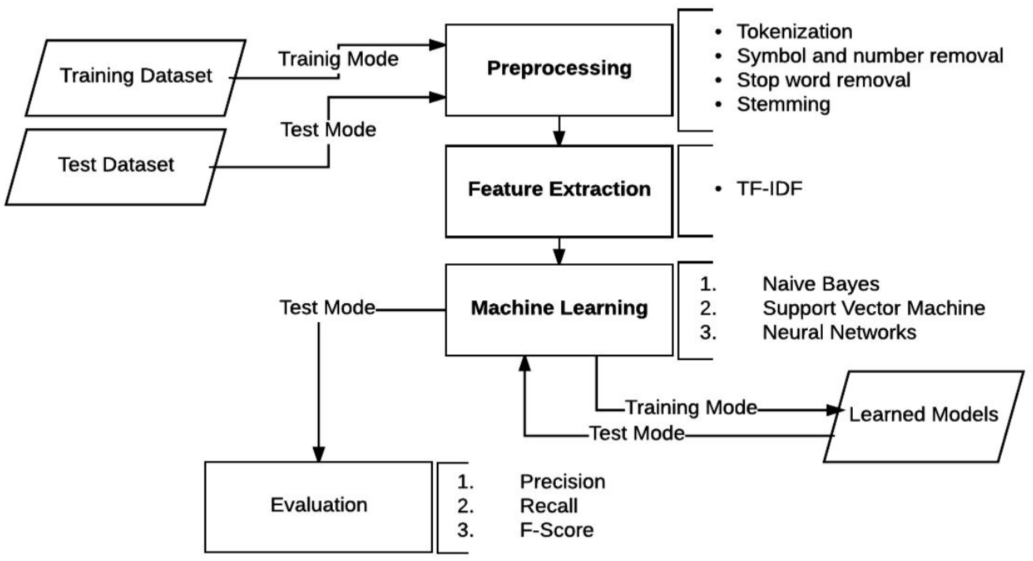

Figure 3.2: News classification system pipeline [11].

Figure 3.2 shows the whole text classification process introduced in [11]. This process consists of the following steps:

- The preprocessing, which again consists of four steps namely tokenization, special symbol and number removal, stop word removal and stemming.
- The feature extraction which is done using the TFIDF vectorizer.
- The machine learning itself, which is performed using either the Naive Bayes, the Support Vector Machine or a Neural Network. The hyperparameters were analyzed and optimized for each model to reach the best possible classification results.
- The evaluation of the three models using the metrics accuracy, precision, recall and f-score.

Table 3.2 shows the mean performance of each algorithm. In each case, the mean value results from the five different runs of an algorithm. As listed in Table 3.2, the Support Vector Machine, both with RBF and linear kernel, outperforms the other algorithms. Nevertheless, after the Support Vector Machine the Multilayer Perceptron (MLP) Neural Network follows with slightly less percentages of precision and recall. Only the Naive Bayes did not perform that well. It is claimed in [11] that taking everything in consideration the Support Vector Machine outperformed the other algorithms.

The data gathering approaches from [11] and the master thesis project are the same, both gather data from multiple news platforms using web crawler. Also the classification process from text gatehering, text preprocessing to vectorization and evaluation is similar. In the master thesis proejct it is also experimented with stop words removal and stemming among others. For the evaluation the metrics recall and f-score are relevant as well. The optimization of the hyperparameters for the algorithms is performed using the grid search for the master thesis project as well as in [11]. The Support Vector Machine

Table 3.2: Results of the four models used [11].

Measures	*Naive Bayes*	*SVM (Linear)*	*SVM (RBF)*	*MLP*
Accuracy	68.31	74.62	**74.65**	72.99
Precision	69.2	75.2	**75.4**	73
Recall	68.2	74.6	**74.6**	73
F Score	67.2	74	**74.4**	72.2

and the Neural Networks are experimented with in the master thesis project as they show promising results in Table 3.2.

3.2 Summary and Findings

A good overview of a text classification workflow in introduced in [14] and is shown in Figure 3.1. A detailed illustration of the pipeline of the whole text classification process introduced in [11] is shown in Figure 3.2. These two workflow approaches are good starting points to set up the classification workflow for the master thesis project.

The data for the master thesis project was gathered using web crawler, similar as presented in [14]. The meta-data provided by web pages is limited. This problem came up when cleaning the gathered data. Paper [8] also stated the same problem regarding data limitations. They highlight that for example the author, if provided, has no fixed location. Regarding the class distribution, the dataset of the master thesis project is imbalanced and the classification type is binary. [8] and [6] deal with binary classification of imbalanced data. It is shown in in [6] that a cost-sensitive Support Vector Machine performs better on imbalanced data than a standard Support Vector Machine (see Section 3.1.3). The impact of text preprocessing on text classification is shown in [13] in which several preprocessing tasks namely tokenization, stop-word removal, lowercase conversion and stemming in a binary and multi class setting are approached (see Section 3.1.4). The results presented in [13] highlight that there is no overall best preprocessing setting. In fact, the best preprocessing setting for a certain task is always domain and language specific.

One part of the master thesis project is to examine if a sentiment analysis helps to improve the identification of paid news articles. In [5] a similar study is presented, in which the aim to find out if the polarity of an email can increase the accuracy of an algorithm when identifying spam emails (see Section 3.1.5). Looking at the results from [11], in which different machine learning algorithms are applied to classify news articles, Support Vector Machines and Neural Networks show the most promising results (see Section 3.1.6).

Regarding the evaluation of the algorithms, paper [8] gives detailed insight in their algorithm performance measurement. The algorithms are evaluated using two different categories namely effectiveness and efficiency (see Section 3.1.2). A detailed explanation of the metric chosen for the evaluation is also presented in [6] (see Section 3.1.3).

Chapter 4

Methodology

The implementation of the project is divided into five main steps, which are shown in Figure 4.1. Data gathering includes all steps that lead to the final database, which are mainly crawling, scraping and data cleaning (see Section 4.1). The next step, data preprocessing, deals with text preprocessing methods such as tokenization, stop words removal, stemming and lemmatization (see Section 4.2). Feature extraction (see Section 4.3) is done using the tfidf (term frequency inverse document frequency) approach, which means that words that occur often in the texts weight less than words that occur rarely. Another feature, which is extracted and tested if it improves the performance of a classifier, is the sentiment of the texts. To get the sentiment score, a sentiment analysis is performed for every text and the result is stored in an additional database column. The fourth step is the classification itself (see Section 4.4). To classify the texts three different approaches are implemented and compared to each other. The approaches used are: Support Vector Machine, Logistic Regression and Neural Networks. The algorithms themselves are optimized by hyperparameter tuning. To find the optimal parameter combinations a grid search is performed. The final step is the evaluation and comparison of all optimized algorithms. For this the k-fold cross validation is used (see Section 4.5).

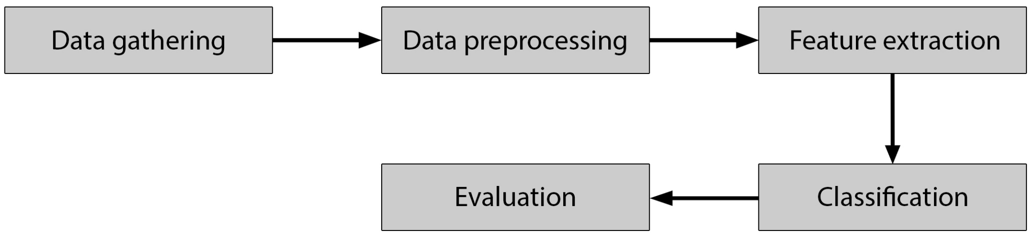

Figure 4.1: Process of the project implementation.

4.1 Data Gathering

The data was collected from multiple news platforms using crawling and scraping approaches (see Section 4.1.1). Afterwards the data was cleaned, meaning the removal of irrelevant articles, the extraction of authors and many more (see Section 4.1.2).

4.1.1 Crawling and Scraping

As a first step, multiple news platforms were crawled to gather the urls of news articles. Since every news platform has their own HTML structure and their own way how to provide news articles, a seperate crawler has to be implemented for every platform. Some news platforms are easy to crawl as they provide an archive. The crawler for the platforms without an archive are more complex and need more initial effort. Some articles are identified by an id (for example krone.at) which makes the articles easy to crawl as the url can to be called with iterating over a defined set of numbers as the id. Also a google search crawler is implemented for every selected news platform. The google search crawler gather all articles for multiple predefined search terms in the tabs “all” and “News”.

After the urls are collected, every url is called and the content of the news article is scraped. A separate scraper has to be implemented for every selected platform, as the HTML structure is different for every platform. The crawler and scraper are implemented in Python with BeautifulSoup and Selenium. Selenium is needed because some platforms load their archive/data asynchronously and therefore parts of the content are missing on the initial page load. For those cases a Selenium scraper will be implemented, which awaits the content load with a timeout.

BeautifulSoup Scraper: With the BeautifulSoup library[1] one element as well as multiple elements which match a given selector can be selected. One element can be selected using the `find()` method, which returns a single BeautifulSoup element. The method `find_all()` returns a list of elements that can be looped through afterwards. To define which elements to select, a HTML tag is passed to the find-function. To refine the search a css class or an id can optionally be passed to the find-function as well. Then the text of the selected elements can be parsed with the `get_text()` function. This text data can be stored for example in a database after parsing. To find out which HTML elements with which classes/ids need to be parsed, the web page is first manually inspected using the google chrome dev tools. Before the desired elements can be selected with BeautifulSoup, the HTML of the page needs to be downloaded for example with the python request library[2].

Selenium Scraper: Selenium[3] is a web driver that enables python to control the browser via OS-level interactions. With Selenium a web page can be requested using the `browser.get()` function. If the page needs time to load, Selenium can be commanded to wait using the `time.sleep()` function. In this project, after the page is fully loaded it

[1]https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[2]https://2.python-requests.org/en/master/
[3]https://www.seleniumhq.org/

Table 4.1: Initial draft of the news article database.

Name	*Type*
id	INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE
url	TEXT UNIQUE
author	TEXT
category	TEXT
title	TEXT
body	TEXT
date	TEXT
sponsored	TEXT
credits	TEXT

is handed over from Selenium to BeautifulSoup and the data is extracted as explained above.

The data is stored in a SQLite database. Table 4.1 shows the initial draft of the database. It was the goal to gather the following information for every news article: url, author, category, title, body, date, credits and if this article is sponsored or not. One important detail is that the url has be be unique in order to be able to perform multiple crawling iterations without storing any article more than once.

4.1.2 Data Cleaning

Before the actual data cleaning and conversion to same formats can be done, irrelevant articles have to be removed from the database. Irrelevant articles are: image galleries (because only textual content can be processed), polls (because most of them are already expired), epapers, pdfs and weather reports. More irrelevant article categories can occur during the actual implementation. After the irrelevant articles are removed from the database, a separate cleaning script is implemented for every news platform, same as for crawling and scraping, because there are big differences in the textual content and the format of for example the date. General cleaning steps that are the same for every platform are:

- Every date string is converted into the datetime format to have a uniform representation of the date for later visualisations.
- If the author is missing, the body text is checked for some patterns to extract an author if possible (for example string in brackets at the end of the text).
- Excessive whitespaces are removed from all texts.
- Every new line is replaced with one white space.
- Newsletter sign up texts are removed if contained in the body text.

Since the goal is to distinguish paid from non paid articles, all gathered articles

Table 4.2: Stem and Lemma of inflected words.

Word	*Stem*	*Lemma*
study	study	study
studying	study	study
studies	studi	study

have to be labelled as sponsored true or false. To achieve this, multiple sponsored news articles of every news platform were manually inspected to retrieve the corresponding identifiers for sponsored articles.

4.2 Data Preprocessing

When preparing the data for the classifiers, the following preprocessing options will be experimented with:

- stemming,
- lemmatization,
- stop word removal,
- the maximum document frequency (`max_df`) of a word to be included as a feature
- and minimum document frequency (`min_df`) of a word to be included as a feature.

Stemming and lemmatization are text normalization techniques that both convert inflected words to their root form. The difference is that with stemming the root form does not need to be a valid word whereas with lemmatization the root form always is a valid word. Stemming reduces the words to their root form, also called Stem, by removing the suffixes or prefixes. This can often lead to word stems that are no grammatically valid words. Lemmatization converts inflected words properly which ensures that the root form, also called Lemma, is a grammatically valid word. This is achieved using lexical knowledge to get correct root form [24]. Table 4.2 shows the difference between stemming and lemmatization. Lemmatization produces the same Lemma for all three words, whereas stemming produces two different Stems.

Stop words are words that occur is nearly all documents and do not help to separate texts into classes. As those words are not significant they can be removed from all texts. This can be done by using default lists such as the list “english” provided by sklearn or by passing a custom list of stop words. Another possibility to regulate stop words is through the parameter `max_df`.

The parameters `max_df` and `min_df` define thresholds for the maximum and the minimum document frequencies of words. `max_df` defines the maximum document frequency a word is allowed to have to be included when building the vocabulary. All terms that have a document frequency higher than `max_df` will be ignored. `min_df` works the other way around. It defines the minimum document frequency a word must have to be included when building the vocabulary. All terms that have a document frequency lower than `min_df` will be ignored.

4.3 Feature Extraction

The data is vectorized using the `TF-IDF` approach. The `TfidfVectorizer` from sklearn converts the data into a matrix of `TF-IDF` features. This vectorizer accepts the preprocessing options mentioned above as parameters. The stemming and the lemmatization can be applied by passing them as the TF-IDF's tokenizer. The other preprocessing options can be passed using the vectorizers same-named parameters. Before building the vocabulary all words will be converted to lowercase, as suggested in [13]. For this no parameter needs to be set as it is the `TfidfVectorizers` default. The term `TF-IDF` is composed of the `TF` (Term Frequency) and the `IDF` (Inverse Document Frequency), which causes rare words to have higher weights than words that occur more often.

As an additional feature the sentiment for every news article will be calculated. Then it will be evaluated for every algorithm used if the sentiment scores improve its performance. To calculate the sentiment the Google Cloud Natural Language API will be used[4]. This API returns a sentiment score as well as a magnitude. To interpret those values thresholds for the score as well as for the score-magnitude combination need to be defined.

4.4 Classification

To find the algorithm which fits best when identifying paid news articles, multiple algorithms will be taken into consideration and experimented with. The approaches used in this project are: Support Vector Machine, Logistic Regression and Neural Networks. The optimal hyperparameters for the Support Vector Machine as well as for the Logistic Regression will be identified using the `grid search`[5]. With a grid search the optimal values for the parameters for a certain model can be determined. It is hard and time consuming to manually find the best hyperparameter combinations. Grid search automates this process and performs a cross validation to get the best performing hyperparameter combination. To use a grid search, the following parameters have to be provided:

- `estimator:` the model which is used,
- `param_grid:` a list of parameters to be optimized; for every parameter an array of values has to be provided.

It is also possible to change the scoring function of the cross validation. As a scoring function precision, recall, f1-score and others can be chosen (see model documentation [6]). The scoring functions used in the master thesis project are recall and f1-score.

Hyperparameters to optimize for the Support Vector Machine: The following hyperparameters were considered for the optimization of the Support Vector Machine:

- `kernel`,
- `C`,
- and `class_weight`.

[4]https://cloud.google.com/natural-language/
[5]https://scikit-learn.org/stable/modules/grid_search.html
[6]https://scikit-learn.org/stable/modules/model_evaluation.html

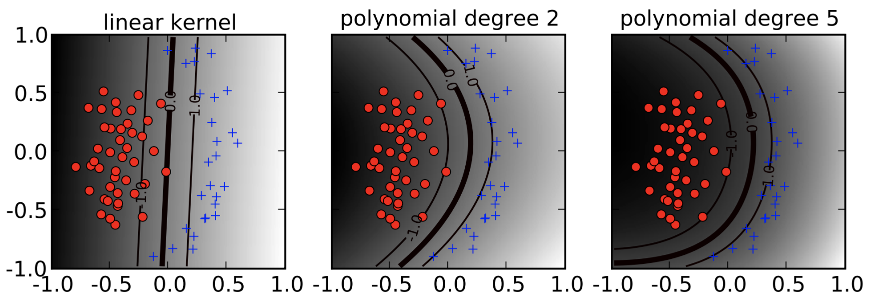

Figure 4.2: Support Vector Machine hyperparameter kernel [2].

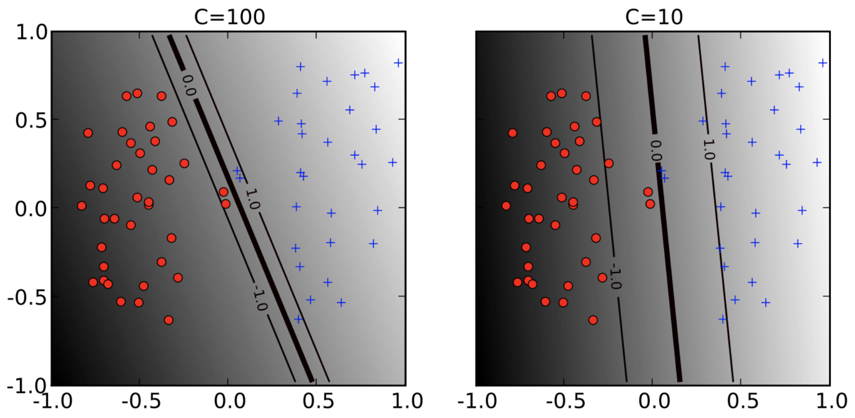

Figure 4.3: Support Vector Machine hyperparameter C (soft margin constant) [2].

The first hyperparameter is the `kernel`, which defines the flexibility of the classifier. The higher the degree of the polynomial, the more flexible the decision boundary gets [2]. Figure 4.2 shows how the `kernel` parameter affects the hyperplane. The linear kernel leads to a straight line whereas the polynomial kernel leads to a curve with a bend depending on the degree .

The second hyperparameter is `C`, also called soft-margin constant. The larger the value of C is, the larger is the penalty that is assigned to errors/margin errors [2]. Figure 4.3 shows how the parameter `C` affects the hyperplane of the Support Vector Machine. The left illustration shows a Support Vector Machine with a high value for `C`. This leads to a hyperplane which is close to several data points and has a narrow margin. For the right illustration a low value is chosen for `C`, which leads to a hyperplane with a much larger margin. Not only the margin but also the orientation of the hyperplane is affected by the value of `C`.

The third hyperparameter is the `class_weight`, which is an optional parameter for the Support Vector Machine. If the `class_weight` is not provided every class has the same weight. If the `class_weight` is set to be `balanced` then the weights for the classes are automatically adjusted inversely proportional to class frequencies, leading minor classes to have higher weights.

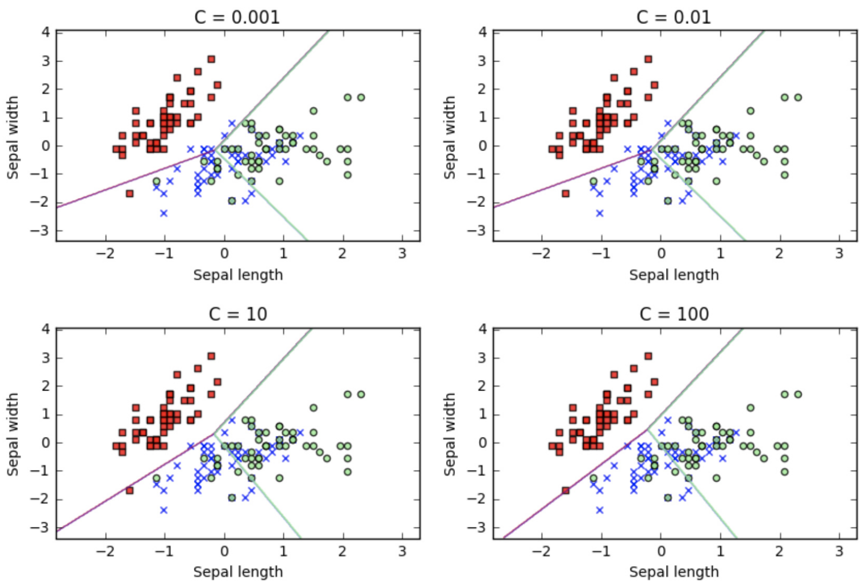

Figure 4.4: Decision regions of Logistic Regression depending on hyperparameter C [22].

Hyperparameters to optimize for the Logistic Regression: The following hyperparameters were considered for the optimization of the Logistic Regression:

- `solver`,
- `C`,
- and `class_weight`.

The first hyperparameter named `solver` is the algorithm used in the optimization problem. The values considered for this parameter are: `liblinear` (a good choice for small datasets, limited to one-versus-rest schemes) and `lbfgs` (recommended for use for small data-sets, used by default because its robustness) [29]. The hyperparameter `C` is the regularization parameter ($C = \frac{1}{\lambda}$). The effect of λ is described in [22] as follows:

> Lambda (λ) controls the trade-off between allowing the model to increase it's complexity as much as it wants with trying to keep it simple. For example, if λ is very low or 0, the model will have enough power to increase it's complexity (overfit) by assigning big values to the weights for each parameter. If, in the other hand, we increase the value of λ, the model will tend to underfit, as the model will become too simple.

The parameter C works the other way around. If a small value is chosen for C the regularization strength is increased. As a consequence the model is kept simple and therefore is prone to underfit the data. If on the other hand a big value is chosen for C the regularization strength is low. As a consequence the model can get very complex and therefore is prone to overfit the data. Figure 4.4 shows how the decision regions change depending on the value chosen for C [22].

The hyperparameter `class_weight` defines the weight associated with each class. If the `class_weight` is set to be `balanced` the weight for each class is automatically adjusted inversely proportional to the class frequencies. If not set every class has the same weight.

Parameters to optimize for the Neural Network: For the optimization of the Neural Network it will be experimented with the following parameters:

- `vocabulary_size`,
- `input_length`,
- `epochs`,
- and `batch_size`.

The first parameter, `vocabulary_size`, is no direct parameter of the algorithm but used in the feature extraction step. In contrary to the `TfidfVectorizer` (used for the Support Vector Machine and the Logistic Regression) the vocabulary size needs to be defined for the one hot encoding. Therefore the parameter `vocabulary_size` is used to define the maximum vocabulary size, meaning the amount of features extracted. Multiple values will be experimented with to see how the vocabulary size affects the performance of the model.

As the `input_length` has to be passed to the model, every one hot encoded article has to be brought to the same length. To achieve this, a concrete `input_length` is defined and the all articles are cut to this length. This is done using the `pad_sequences` function, which either cuts the array at the end if it is bigger than the defined length or fills up the array with zeros at the end until the defined length is reached.

One Epoch is when the entire dataset is passed forward and backward through the Neural Network once. If the number of `epochs` it set to just one epoch the model will underfit. Multiple epochs need to be performed until the Neural Network reaches its optimum. However, if the number of epochs is too high, to model will be overfitted [31]. It will be experimented with the number of epochs to whether over nor underfit the model.

The `batch_size` is the total number of training examples present in a single batch. It is used to not have to pass the entire dataset into the Neural Network at once but in several batches with a defined `batch_size` [31]. If not specified, the batch size will have a default value of 32 [25].

4.5 Evaluation

An extensive evaluation is performed to achieve three main goals:

- Evaluate the performance of each model to optimize its performance by the tuning of hyperparameters. The goal is to tune the hyerparameters in such a way that the model best fits the given data without over- or underfitting the model.
- Compare the optimized models of all algorithm used to each other to find the best performing classifier. The goal is to rank the Support Vector Machine, the Logistic Regression and the Neural Network in terms of model performance.
- Compare the results of the optimized models with and without including the results of the sentiment analysis. The aim is to see if the sentiment feature improves the performance of the model.

Table 4.3: Confusion Matrix.

		Predicted Label	
		Positive	*Negative*
Actual Label	*Positive*	True Positive	False Negative
	Negative	False Positive	True Negative

To achieve goal one, the optimization of one model, the algorithms themselves will be evaluated and optimized using the `recall`, `f1 score` and the `confusion matrix`.

The *precision* (see Formula 4.1) indicates how many of all predicted positives are actual positives. This metric is a good choice when the cost associated with False Positives should be high. An example for high costs associated with False Positives is the email spam detection. Emails that are no spam messages (actual negative) should in no case be classified as spam (False Positive). The *precision* is calculated as follows:

$$precision = \frac{TruePositive}{TruePositive + FalsePositive}. \tag{4.1}$$

The *recall* score shows how many positives out of all actual positives were classified correctly. This metric is a good choice when the cost associated with False Negatives should be high. Regarding the master thesis project, the aim is identify the paid articles. Therefore, if a paid article (Actual Positive) is predicted as non paid (Predicted Negative), the cost associated with such errors should be high. The formula for the *recall*, which calculates how many of all actual positives are classified as positive, is

$$recall = \frac{TruePositive}{TruePositive + FalseNegative}. \tag{4.2}$$

If the aim is to keep a balance between *precision* and *recall*, the $f1score$ (see Formula 4.3) is a good choice, because it takes the *precision* as well as the *recall* into account. In [32] it is stated that the $f1score$ is the metric to choose for an uneven class distribution. Since in the master thesis project there is an uneven class distribution the main focus will be, alongside with the recall, on the f1-score while evaluating and optimizing the algorithms. The $f1score$, in which the *precision* and the *recall* are integrated, is calculated as follows:

$$f1score = 2 \times \frac{Precision \times Recall}{Precision + Recall}. \tag{4.3}$$

Also the confusion matrix will be analyzed to see how many paid articles are not identified as paid ones (see Table 4.3). Those articles are represented as False Negatives.

The final evaluation and comparison of the Logistic Regression and the Support Vector Machine, which approaches goal two and three, is done with the k-fold cross validation[7]. The metrics used are `recall` and `f1-score` and the amount of folds is ten.

[7]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

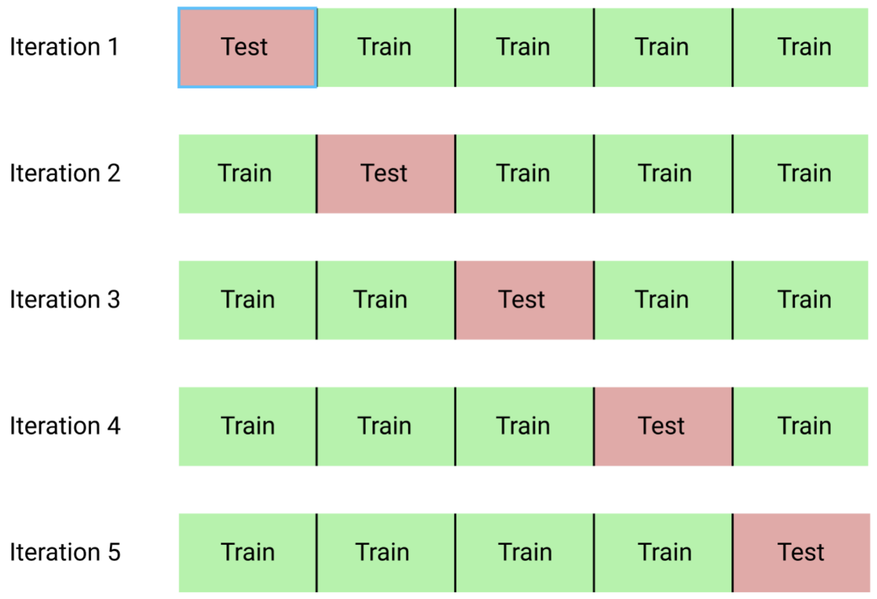

Figure 4.5: 5-fold cross validation [30].

The metric used for the Neural Network is the `accuracy` and the amount of folds is also ten. When performing a cross validation the initial dataset is partitioned into a number of subsets. Each iteration one set is held out and the model is trained on the remaining sets. Then the model is tested on the set which was held out. The k in `k-fold cross validation` represents the number of partitions the dataset will be split into. When a specific value for k is chosen, it can be used instead of k. Meaning if $k = 10$ it becomes the 10-fold cross-validation. Figure 4.5 shows the iterations of a 5-fold cross validation. In [30] it is illustrated which partitions are used for training and which partition is used for testing for every iteration.

The process of the 5-fold cross validation is describe in [30] as follows:

- take one group as a holdout set for testing,
- fit a model on the remaining groups,
- evaluate the model on the holdout set,
- keep the evaluation score and discard the model,
- calculate the overall model performance by taking the evaluation scores from all iterations into consideration.

After the evaluation of the optimized models with the 10-fold cross-validation the last step is to evaluate if the sentiment feature improves the performance of the model. To achieve this the Logistic Regression is trained with and without the sentiment feature and the results from these two models are compared to each other.

Chapter 5

Implementation

5.1 Data Gathering

Figure 5.1 shows the news distribution of the final database per year. The most articles are from 2018, as the iterations of the crawling process were maily executed in 2018. From the past years fewer articles were available as many news platforms either provided no archive or in some cases even deleted articles after some time. During the scraping process it became clear that articles are deleted from time to time as the longer the time gap was between url crawling and content scraping the more articles were not available anymore at the previously crawled url. The moment this issue was detected the crawler and scraper were always executed at the same day from this time on. From a few platforms news even before 2014 were collected. However it was decided to cap all articles before 2014 since sufficient articles were only available for the minority of the platforms.

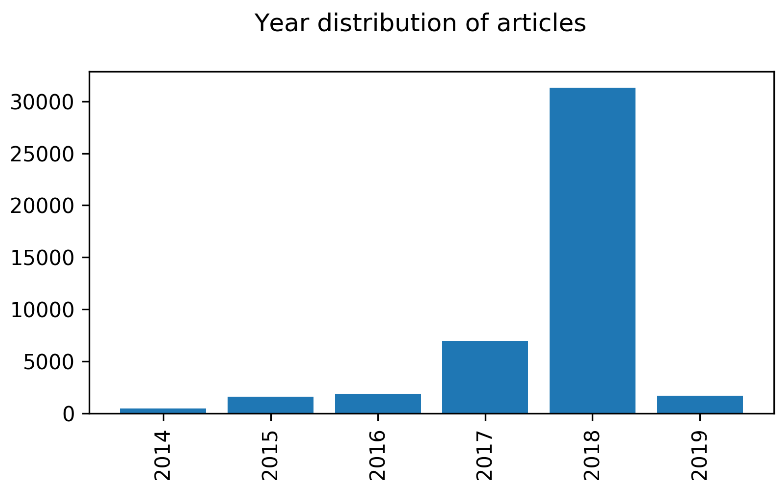

Figure 5.1: Distribution of the articles of the final database per year.

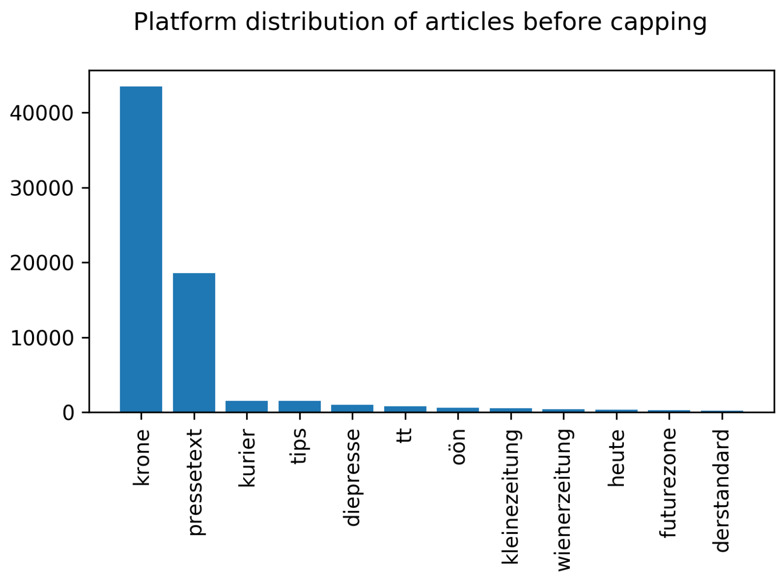

Figure 5.2: Platform distribution of articles before capping krone.

5.1.1 Crawling and Scraping

The news articles were collected from 12 different news platforms. Their initial platform distribution is shown in Figure 5.2. At first the majority of the articles were from "krone". In fact, nearly half of all articles were from this single source. To prevent that the classifiers are biased because of this huge imbalance, the "krone" articles were cut to the same amount of the "pressetext" articles. The new platform distribution after the capping is shown in Figure 5.3. There is still some imbalance because these two sources make up more than half of all articles. Nevertheless, the imbalance of the news platform distribution was decreased a lot by cutting the "krone" articles. The two majority platforms can not be capped any further as this would result in having too few data to work with. In order to really solve this imbalance the crawling and the scraping processes would need to be executed over a much longer period to gather enough data from every platform.

For every platform mentioned in Figure 5.3 two crawler were implemented: one initial crawler and a second one which only collected sponsored articles. The initial crawler gathered as much urls as possible from the platform itself, the archive of the platform (if provided) and via the google search. These crawlers were executed multiple times over months to always include the newly published articles. As after those crawling iterations the imbalance of paid and non paid articles was too high (only about 10% paid articles) a second crawler was implemented for every platform, which only gathers paid articles and ignores non paid articles to compensate this imbalance. Finally a distribution of 29% paid and 71% non paid articles was reached, as shown in Figure 5.4. The classifiers were trained on the highly imbalanced (10% paid) and again on the fewer imbalanced (29% paid) data. The results showed that the distribution of 29% paid and 71% non paid is sufficient to lead to good results. Of course the imbalance of the data still needs

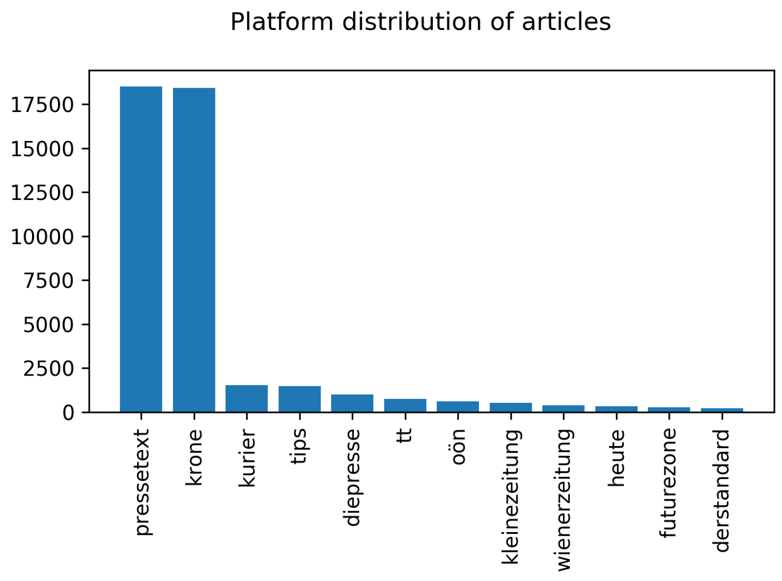

Figure 5.3: Platform distribution of the articles of the final database.

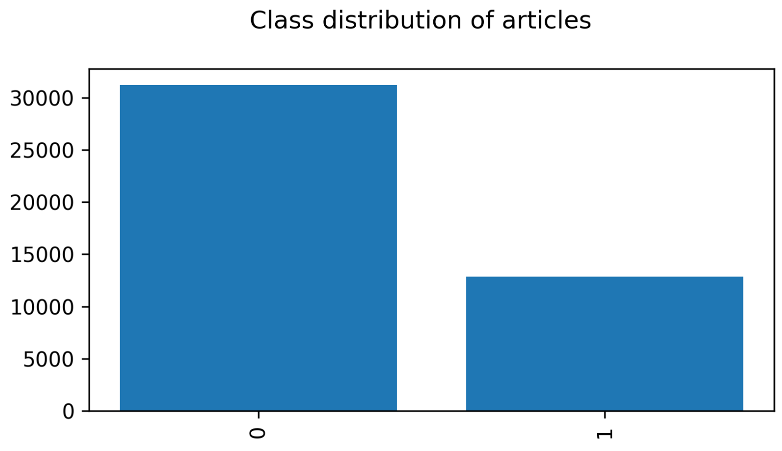

Figure 5.4: Class distribution of articles.

to be considered when implementing the classifiers.

After every crawling iteration, the content of the newly gathered urls was scraped. To achieve this a separate content scraper was implemented for every platform. One big challenge with the scraper was that some platforms had multiple different HTML layouts for their articles, depending on the article category. This led to many special cases which needed to be integrated iteratively into the corresponding platform scraper.

Table 4.1 shows the database which was created before the scraping. It shows the attributes that wanted to be collected about an article. The following attributes are available for every article:

- id,

- url,
- title,
- body
- and if the article is sponsored.

Also the date is available for nearly all articles. Only 61 out of 44058 articles do not provide the date.

The remaining attributes are not available for all articles and therefore not included when classifying the articles. The author and the credits (which agencies contributed to the articles; from where the content is taken) were extracted from the body text for some articles. However, this information could not be extracted for many articles and therefore is not relevant for the classification. The category was an interesting attribute as it could show tendencies which categories are sponsored more often. This attribute was dropped too because of lack of data.

During the crawling and scraping process several problems occurred. The first problem is that after multiple hours of scraping one scraper was blocked because of too many requests. The same problem occurred with the google search selenium scraper. After too many executions the google “i am not a robot” captcha was triggered on every new execution try. Another challenge, which was already mentioned above, was that if the time gap between crawling and scraping was too big, some articles had already been deleted and therefore the content could not be scraped anymore. Additionally, for some urls it was not possible to scrape any content because only textual data can be processed in this specific task and some pages contained mainly image galleries, videos, polls, epapers or similar non textual content.

5.1.2 Data Cleaning

Before the actual cleaning steps are performed, the database needed to be cleaned up, meaning that articles that are irrelevant for classifying the news articles had to be removed. Irrelevant articles that were contained in the database are:

- Test articles,
- image galleries and slideshows: for example baby image galleries (because only text content is relevant);
- video content: video livestream, viral videos (because only text content is relevant);
- result tables: sport results, election outcome percent table;
- articles in which the body text is missing (for example https://www.krone.at/1607987);
- e-papers (for example http://epaper.heute.at/#/documents/171014_HEU);
- weather reports, horoscopes, quizzes, live ticker and many more.

After the content of the articles was scraped and the irrelevant articles were removed, all the gathered information needed to be cleaned. The conversion of the data to a common format and the extraction of data for additional database columns was achieved using RegEx[1]. To get the datetime for every article, the date and time information often needed to be extracted from strings which held not only the date and time information but in some cases filler words, such as “last updated at”, or even the author of the

[1] https://www.regular-expressions.info/

article. Therefore a separate RegEx to extract the datetime was implemented for every platform.

Since some platforms use shortcuts for their authors, one attempt to extract the author was to create a list of author shortcuts and check if some of the shortcuts were contained at the end of a body text. An example list is: `['apa', 'dpa', 'tmn', 'jba', 'spe', 'sar', 'sch', 'rer', 'KOB', 'KOP', 'mei', 'red', 'JSt', 'ufi', 'afp', 'mja']`. However, the information could not be extracted for a sufficient amount of aritcles to be relevant for the classification.

The process of labelling the articles as paid or non paid was also different for every platform. Some platforms use one single identifications string (for example “Bezahlter Inhalt” for kleinezeitung) while others use multiple identifications strings (for example “Promotion”, “Advertorial” or “Bezahlte Anzeige” for krone). One platform even used an image banner to mark the article as sponsored. In this case it was checked if the class associated with the promotion image banner was contained in the HTML markup of the news articles from this platform.

After all semantic cleaning steps were finished, all excess white spaces were removed. First all new lines were converted into a single white space. Then the excess white spaces were removed so that just one white space separates the words. In the end, the beginning and the trailing white spaces were removed using the `txt.lstrip()` and the `txt.rstrip()` functions.

5.2 Data Preprocessing

The parameters considered for the preprocessing are `stemming`, `lemmatization`, `max_df`, `min_df` and stop word removal, as described in Section 4.2.

To test the impact of stop words removal a list of German stop words was taken from PyPI[2]. The overall score is exactly the same with and without the removal of those stop words. The only thing that changed with the stop words removal is that numbers in the confusion matrix shifted. The number of False Negatives decreased by 6 but simultaneously the number of False Positives increased by 6. Looking at these results it was decided to not include stop words removal in the final preprocessing steps.

As with the parameter `max_df` an upper limit for the frequency of terms can be defined, it was decided to use this parameter to regulate more frequent words, such as stop words, instead of using a predefined list of stop words. `Max_df` is more flexible because all words that have a document frequency higher than its value will be ignored, therefore document specific stop words can easily be removed without building a list of custom stop words.

Also the parameter `min_df` is believed to be important, as words that occur too infrequently are not significant enough when classifying the news articles.

The Tables 5.1 and 5.2 show that the impact of `min_df` is much bigger than the impact of `max_df` when classifying this imbalanced dataset. The bigger the value of `min_df` is, the worse the performance of the classifier gets. This means that if `min_df` is too big, a lot of significant words will be ignored. Therefore `min_df` is set to 4 for the final algorithm, which means that words that occur in less than 4 documents will be

[2]https://pypi.org/project/stop-words/

Table 5.1: Impact of max_df tested on the Support Vector Machine.

max_df	*Recall*	*F1-score*
1.0	0.97	0.97
0.9, 0.8, 0.7, 0.6, 0.5	0.96	0.97

Table 5.2: Impact of min_df tested on the Support Vector Machine.

min_df	*Recall*	*F1-score*
1, 4, 5, 6	0.97	0.97
2, 3	0.96	0.97
0.01	0.96	0.95
0.05	0.94	0.89
0.1	0.92	0.84
0.2	0.86	0.74
0.3	0.84	0.72

ignored. For the parameter `max_df` the value 0.7 is chosen, as the values from 0.5 to 0.9 performed equally well and 0.7 is the middle of those parameters. With a value of 0.7 for the `max_df`, words that occur in more than 70% of the documents will be ignored.

Stemming and Lemmatization both did not improve the results a lot. The improvement of the overall score was under 1% for the WordNetLemmatizer[3]. The result with the SnowballStemmer[4] was even slightly worse (0.002% worse than without a tokenizer). Therefore it was decided to use neither stemming nor lemmatization.

5.3 Feature Extraction

To extract features from the news articles two main approaches needed to be implemented. The first approach is the `TFIDF`, which is used for the Support Vector Machine and the Logistic Regression. The second approach is the `one hot encoding`, which is used to prepare the data for the Neural Network. Additionally to these two feature extraction approaches, the sentiment score of each article was calculated and added as a feature. To store the feature the final database was extended by an additional column called *sentiment*. The algorithms were trained with and without this additional feature to test if it can increase the performance of the model.

TFIDF: The body texts of all news articles are converted to a matrix of TF-IDF features using the `TfidfVectorizer`[5]. The `TfidfVectorizer` computes the word counts,

[3]https://www.nltk.org/_modules/nltk/stem/wordnet.html
[4]https://www.nltk.org/_modules/nltk/stem/snowball.html
[5]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Program 5.1: One hot encoding of train and test set for the Neural Network.

```
1 vocab_size = 30000
2 X_train = [one_hot(d, vocab_size,filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~',lower=
                                              True, split=' ') for d in X_train]
3 X_test = [one_hot(d, vocab_size,filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~',lower=True
                                              , split=' ') for d in X_test]
```

idf (inverse document frequencies) and tfidf (text frequency inverse document frequency) values all at once [21]. The preprocessing parameters explained above are passed to the `TfidfVectorizer` as follows: `TfidfVectorizer(max_df=0.7, min_df=4)`. The dateset is split into train and test set with a test set size of 25%. Therefore the training set contains 33043 articles and the test set 11015 articles. The training set has a shape of `(33043, 87690)` after applying the `TfidfVectorizer`. The first number of the shape (33043) represents the amount of articles `TfidfVectorizer` the second number (87690) represents the amount of features. In this case the vocabulary of the training dataset consists of 87690 items (features). Each item of this vocabulary has its own idf (inverse document frequency) score. The first 15 entries of the vocabulary created by the `TfidfVectorizer` are as follows: `{’pcc’: 56117, ’se’: 65643, ’hat’: 34706, ’zum’: 85928, ’oktober’: 54656, ’2018’: 604, ’zwei’: 86477, ’neue’: 53142, ’anleihen’:
5325, ’einer’: 20392, ’verzinsung’: 79653, ’00’: 0, ’laufzeit’: 45504, ’fünfeinhalb’: 28667, ’jahre’: 39548}`. The corresponding first 15 items of the idf array are: `[4.49384799, 2.82186557, 8.22754146, 9.79615737, 9.61383582, 9.00770001, 8.84064593, 9.00770001, 9.20837071, 9.79615737, 8.84064593, 9.61383582, 9.20837071, 9.61383582, 9.20837071]`. The values in the idf array belong to the words in the dictionary linked by the index. The third item of the vocabulary (“hat”) for example has an idf score of 8.22754145533445.

One Hot Encoding: Before the actual one_hot encoding, the two classes (’0’: non paid, ’1’: paid) had to be converted to binary labels instead of strings and the vocabulary size had to be defined. Multiple experiments show that the vocabulary size has a big influence on how well the model performs. These experiments are shown in Table 5.11. Program 5.1 shows the one_hot encoding of the train and test set. When encoding the texts, multiple characters, such as punctuation and special characters (see parameter “filters” in line 2 and 3), are filtered from the texts since they are not relevant to the task. As already discussed in Section 4.3 all words should be converted to lowercase, which is handled through the parameter `lower=True`.

As the length of the input data needs to be passed to the model, it has to be ensured that the data always has the same length. So the encoded `X_train` and the `X_test` from Program 5.1 had to be converted to a predefined length using the `pad_sequences`[6]. For illustration purposes a sequence length of 100 was chosen. Figure 5.5 shows how the feature array of one article looks like after `pad_sequences` was applied to it. As one can see, the feature array was filled up with zeros at the end of the array until the defined

[6]https://keras.io/preprocessing/sequence/

```
[ 5822 10417 13330 10346  4947  2595 14955 11658  4191  1015 18181 16136
   107 19098  3131  2595   322  2369 19964 17124  3794 12034 11824   124
 15758 17467  9996 12034 16389 12898  9661  8135 11941  9050  1534 14626
 14192 19964  3976 18095  5936 16457 19585 11941  2020 17961 18095 18181
 15318  7556 16240  3924  7123 11824 13066 13534  5480 10081     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0]
```

Figure 5.5: One hot encoded data after `pad_sequences`

Program 5.2: Sentiment calculation of one text.

```
1 def calculateSentiment(body):
2     document = types.Document(
3         content=body,
4         type=enums.Document.Type.PLAIN_TEXT)
5     sentiment = client.analyze_sentiment(document=document).document_sentiment
6     print('Sentiment: score {}, magnitude {}'.format(sentiment.score, sentiment.
                                        magnitude))
```

length of 100 was reached.

Sentiment Analysis: The Sentiment Analysis was implemented using the Google Cloud Natural Language API. One sentiment calculation is shown in Program 5.2. To analyze a text the text itself has to be passed via the `content` parameter (line three). Additionally the type of the content has to be defined. This is done via the `type` parameter (line four). In this case the type is `PLAIN_TEXT`, as only textual data from the news articles will be passed to the API. The Google API is called with the parameters described (line five) and returns the sentiment score (`sentiment.score`) as well as the magnitude of the sentiment score (`sentiment.magnitude`).

Based on the score and the magnitude it is differentiated if the sentiment of an article is either positive, neutral or negative. To be able to map the results to one of these three options, thresholds for the sentiment score in combination with the magnitude had to be defined. The magnitude can have values from 0.0 to infinity. Every sentiment, regardless of whether it is positive or negative, adds up to the magnitude score. A low magnitude indicates that there are few expression of mood in the text and therefore results in a neutral sentiment. The thresholds for positive, neutral and negative sentiment outcomes are as follows:
Positive Sentiment:

- A sentiment score greater than 0.5 always results in a positive sentiment.
- If the sentiment score is between 0.1 and 0.5 not only the sentiment score but also the sentiment magnitude is considered. A sentiment magnitude greater that 0.15 leads to a positive sentiment whereas a sentiment magnitude less than or equal to 0.15 leads to a neutral sentiment.

Neutral Sentiment:

- A sentiment score between 0.1 and 0.5 in combination with a sentiment magnitude less than or equal to 0.15 results in a neutral sentiment.
- A sentiment score between -0.1 and 0.1 always results in a neutral sentiment.
- A sentiment score between -0.5 and -0.1 in combination with a sentiment magnitude less than or equal to 0.15 results in a neutral sentiment.

Negative Sentiment:

- A sentiment score less than or equal to -0.5 always results in a negative sentiment.
- If the sentiment score is between -0.5 and -0.1 not only the sentiment score but also the sentiment magnitude is considered. A sentiment magnitude greater that 0.15 leads to a negative sentiment whereas a sentiment magnitude less than or equal to 0.15 leads to a neutral sentiment.

For illustration purposes and to first test the sentiment thresholds three executions of the code from Program 5.2 were performed with different textual input. The results are analyzed in terms of sentiment score and sentiment magnitude based on the thresholds. For the first execution the following positive sentence was passed to the API: “Das Wetter ist schön heute! Ich freue mich schon auf das Treffen mit meinen lieben Freunden”. With a sentiment score of 0.8 and a sentiment magnitude of 1.7 this sentence clearly resulted in a positive sentiment according to the thresholds. As a second example the neutral sentence “Das ist ein Beispiel für eine Gefühlsanalyse mit der Google NLP API” was taken. As expected the sentence resulted in a neutral sentiment with a sentiment score of -0.1 and a sentiment magnitude of 0.1. For completeness, as a third example the following negative sentence has also been passed to the API: “Ich mag diese Buch überhaupt nicht! Es is langweilig und ich werde ich sicher nicht weiterempfehlen”. This sentence resulted in a clearly negative sentiment with a sentiment score of -0.9 and a sentiment magnitude of 1.8.

Figure 5.6 shows the sentiment distributions of all paid articles. 63.99% of these articles resulted in a positive sentiment and only 9.98% resulted in a negative sentiment. This shows that the majority of the paid articles indeed results in a positive sentiment and confirms the assumption that sponsored articles tend to be positive. However, the significance is with under 70% not as high as assumed, since a quarter of all paid articles resulted in a neutral sentiment.

Figure 5.7 shows the sentiment distribution of all non paid articles. On the contrary to the distribution of the paid articles, the non paid articles have a slight tendency to result in a negative sentiment. However, the distribution of positive (26.78%), neutral (30.69%) and negative (42.53%) articles is way more balanced for the non paid articles, as all of the three classes are under 50%. Figure 5.6 and Figure 5.7 show that the paid and the non paid articles have the opposite tendency of sentiment, as the paid articles have the tendency to be positive and the non paid articles have the tendency to be negative.

5.4 Classification

Since the aim is to optimize not the majority but the minority class, custom scoring functions needed to be defined for the grid search. These scoring functions are defined as shown in Program 5.3. Line two defines the `recall` scoring function, line three the

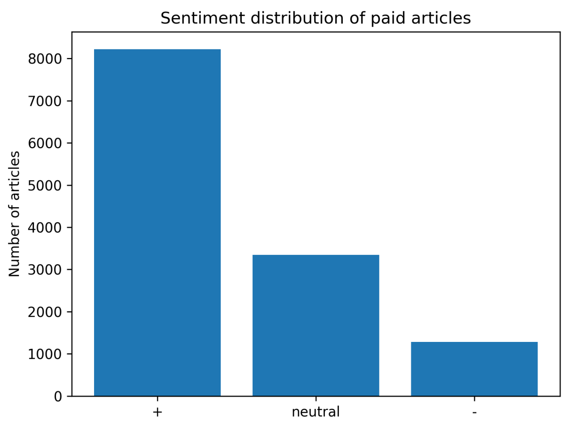

Figure 5.6: Positive, neutral and negative sentiment distribution of the paid articles.

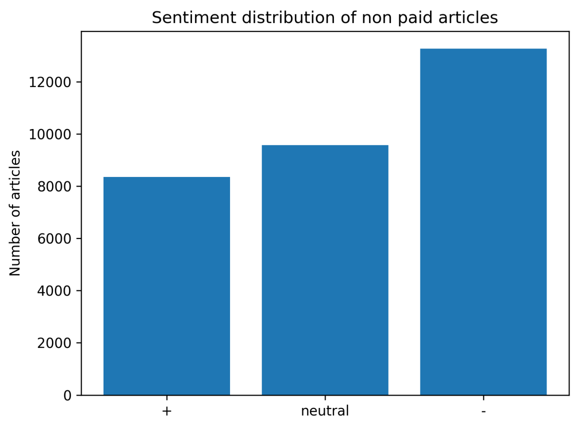

Figure 5.7: Positive, neutral and negative sentiment distribution of the non paid articles.

Program 5.3: Scoring functions for the grid search.

```
from sklearn.metrics import make_scorer, recall_score, f1_score
recallscore = make_scorer(recall_score, pos_label='1', average='binary')
f1score = make_scorer(f1_score, pos_label='1', average='binary')
```

`f1-score` scoring function. The parameter `pos_label='1'` tells the scorer to optimize for the label `1`, which represents the paid articles. The parameter `average='binary'` causes that only results for the class specified by `pos_label` will be reported. These scoring functions will be applied to the grid search of the Support Vector Machine and the Logistic Regression.

Program 5.4: Grid search for the Support Vector Machine.

```
 1 tuned_parameters = [{'kernel': ['rbf'], 'gamma': ['auto', 1e-3, 1e-4], 'C': [1, 10,
                                           100], 'class_weight': ['balanced', None]
                                           },
 2                     {'kernel': ['linear'], 'C': [1, 10, 100], 'class_weight': ['
                                           balanced', None]}]
 3 scores = [recallscore, f1score]
 4
 5 for score in scores:
 6     vect = TfidfVectorizer(max_df=0.7, min_df=4)
 7     clf = GridSearchCV(SVC(), tuned_parameters, cv=5, scoring=score)
 8     pipe_lr = Pipeline([('vect',vect),  ('clf',clf)])
 9     pipe_lr.fit(X_train, y_train)
10     print("Best parameters set found on development set:" + clf.best_params_)
```

Table 5.3: Support Vector Machine: best performing models for recallscore.

Recall	*Variance*	*Parameters*
0.961	(+/-0.010)	{'C': 1, 'class_weight': 'balanced', 'kernel': 'linear'}
0.957	(+/-0.010)	{'C': 10, 'class_weight': None, 'kernel': 'linear'}
0.957	(+/-0.011)	{'C': 10, 'class_weight': 'balanced', 'kernel': 'linear'}
0.956	(+/-0.012)	{'C': 100, 'class_weight': 'balanced', 'kernel': 'linear'}
0.956	(+/-0.012)	{'C': 100, 'class_weight': None, 'kernel': 'linear'}

5.4.1 Support Vector Machine

The grid search performed to find the optimal hyperparameter combination for the Support Vector Machine is shown in Program 5.4. A shortened version of the code is shown to highlight the most important points. Line one and two each define a list of hyperparameters with an array of suggested values. For each list a model for every possible combination of the parameters will be fitted. All results from both lists will be compared to each other to find the best performing combination (line seven of the Listing). Line three defines which scoring functions to use. The grid search will be performed for each scoring function (loop in line five). After the execution two best performing parameter combinations will be outputted, one for every scoring function. Line ten prints out those best performing combinations. The original, not shortened, implementation also prints out a list of all fitted models with their parameter combination and the according score of the model. The Tables 5.3 and 5.4 show the best performing models.

The five best performing Support Vector Machine models and their score are shown in Table 5.3 for the `recallscore` and in Table 5.4 for the `f1score`. From these tables it can be seen that the best performing parameter combinations are the same for both scoring functions. Therefore the parameter combination chosen for the final Suuport Vector Machine is `{'C': 1, 'class_weight': 'balanced', 'kernel': 'linear'}`.

Table 5.4: Support Vector Machine: best performing models for f1score.

F1 score	*Variance*	*Parameters*
0.970	(+/-0.004)	{'C': 1, 'class_weight': 'balanced', 'kernel': 'linear'}
0.970	(+/-0.005)	{'C': 10, 'class_weight': None, 'kernel': 'linear'}
0.969	(+/-0.005)	{'C': 10, 'class_weight': 'balanced', 'kernel': 'linear'}
0.969	(+/-0.005)	{'C': 100, 'class_weight': 'balanced', 'kernel': 'linear'}
0.969	(+/-0.005)	{'C': 100, 'class_weight': None, 'kernel': 'linear'}

Program 5.5: Final algorithm of the Support Vector Machine.

```
1 vect = TfidfVectorizer(max_df=0.7, min_df=4)
2 clf = svm.SVC(C=1, class_weight='balanced', kernel='linear')
3 pipe_lr = Pipeline(('vect',vect), ('clf',clf))
4 y_pred = pipe_lr.fit(x_train, y_train).predict(x_test)
```

After analyzing the results from the grid search as stated above, the final algorithm of the Support Vector Machine was implemented as shown in Program 5.5. Line one defines the `TfidfVectorizer` as described in Section 5.3. Line two defines the Support Vector Machine model with the optimized hyperparameters. These two steps are chained in line three using a pipeline[7]. The model is then fitted using this pipeline, the training data and the training labels in line four (`fit()`). The prediction is also performed in the same line (`predict()`) using the test data.

The overall score of the model defined in Program 5.5 is 98% and the runtime was 32 minutes. Table 5.5 shows the full classification report of the model. This report shows the precision, recall and f1-score for the class 0 as well as the class 1. From this report it can be seen that the model performs slightly better for the classification of the non paid articles. This is due to the imbalance of the dataset, as way more non paid articles were provided to the model to learn and generalize from. However, with a precision, recall and f1-score of 97% the model performed well for the classification of paid articles too. The confusion matrix is shown in Table 5.6. It shows that 84 articles are False Negatives, which means that they are predicted to be non paid but actually are paid articles. It also shows that 96 articles are False Positives, which means that they are classified as paid articles although they are actually non paid articles. All in all, there is a misclassification of 1.6%.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

Table 5.5: Classification report of the final Support Vector Machine.

Class	*Precision*	*Recall*	*F1-score*	*Support*
0	0.99	0.99	0.99	7779
1	0.97	0.97	0.97	3236
micro avg	0.98	0.98	0.98	11015
macro avg	0.98	0.98	0.98	11015
weighted avg	0.98	0.98	0.98	11015

Table 5.6: Confusion Matrix of the final Support Vector Machine.

		Predicted Label	
		Positive	*Negative*
Actual Label	*Positive*	7695	84
	Negative	96	3140

Program 5.6: Grid search for the Logistic Regression.

```
 1 tuned_parameters = [{'solver': ['liblinear'], 'penalty': ['l1', 'l2'], 'C': [1, 10,
                                            100], 'class_weight': ['balanced', None]
                                            },
 2                     {'solver': ['lbfgs'], 'penalty': ['l2'], 'max_iter': [100, 1000,
                                             4000], 'C': [1, 10, 100], 'class_weight
                                            ': ['balanced', None]}]
 3 scores = [recallscore, f1score]
 4
 5 for score in scores:
 6     vect = TfidfVectorizer(max_df=0.7, min_df=4)
 7     clf = GridSearchCV(LogisticRegression(), tuned_parameters, cv=5, scoring=score)
 8     pipe_lr = Pipeline([('vect',vect), ('clf',clf)])
 9     pipe_lr.fit(X_train, y_train)
10     print("Best parameters set found on development set:" + clf.best_params_)
```

5.4.2 Logistic Regression

The grid search performed to find the optimal hyperparameter combination for the Logistic Regression is similar to the one for the Support Vector Machine stated above. The program works in the same way as explained for Program 5.4. The two main differences are the hyperparameters and the algorithm used. The parameter combinations to experiment with are defined in line one and two. In line seven instead of the Support Vector Machine the Logistic Regression is defined as the algorithm to use. The Tables 5.7 and 5.8 show the best performing models.

The five best performing Logistic Regression models and their score are shown in Table 5.7 for the `recallscore` and in Table 5.8 for the `f1score`. Other than for the SVM,

Table 5.7: Logistic Regression: best performing models for recallscore.

Recall	*Variance*	*Parameters*
0.964	(+/-0.008)	{'C': 10, 'class_weight': 'balanced', 'penalty': 'l2', 'solver': 'liblinear'}
0.964	(+/-0.008)	{'C': 10, 'class_weight': 'balanced', 'max_iter': 100, 'penalty': 'l2'}
0.964	(+/-0.008)	{'C': 10, 'class_weight': 'balanced', 'max_iter': 1000, 'penalty': 'l2', 'solver': 'lbfgs'}
0.964	(+/-0.008)	{'C': 10, 'class_weight': 'balanced', 'max_iter': 4000, 'penalty': 'l2', 'solver': 'lbfgs'}
0.962	(+/-0.006)	{'C': 100, 'class_weight': 'balanced', 'max_iter': 100, 'penalty': 'l2', 'solver': 'lbfgs'}

Table 5.8: Logistic Regression: best performing models for f1score.

F1 score	*Variance*	*Parameters*
0.970	(+/-0.003)	{'C': 100, 'class_weight': 'balanced', 'penalty': 'l2', 'solver': 'liblinear'}
0.969	(+/-0.003)	{'C': 10, 'class_weight': 'balanced', 'penalty': 'l2', 'solver': 'liblinear'}
0.969	(+/-0.003)	{'C': 10, 'class_weight': 'balanced', 'max_iter': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.969	(+/-0.003)	{'C': 10, 'class_weight': 'balanced', 'max_iter': 1000, 'penalty': 'l2', 'solver': 'lbfgs'}
0.969	(+/-0.v)	{'C': 10, 'class_weight': 'balanced', 'max_iter': 4000, 'penalty': 'l2', 'solver': 'lbfgs'}

the Logistic Regression has different best performing parameter combinations for the recall and the f1-score. The best performing parameter combination for the recall is `{'C': 10, 'class_weight': 'balanced', 'penalty': 'l2', 'solver': 'liblinear'}` as shown in line one of Table 5.7 whereas the best performing parameter combination for the f1-score is `{'C': 100, 'class_weight': 'balanced', 'penalty': 'l2', 'solver': 'liblinear'}` as shown in line one of Table 5.8. The only difference of the two best performing models is the parameter C. The value of C is 10 for the `recallscore` and 100 for the `f1score`. For the final algorithm the best performing parameter combination from the `recallscore` is taken to keep a balance between a too simple and too complex model to neither underfit nor overfit the data.

Program 5.7 shows the final implementation of the Logistic Regression. Same as for

Program 5.7: Final algorithm of the Logistic Regression.

```
vect = TfidfVectorizer(max_df=0.7, min_df=4)
clf = LogisticRegression(C=10, class_weight='balanced', penalty='l2', solver='
                                           liblinear')
pipe_lr = Pipeline([('vect',vect), ('clf',clf)])
y_pred = pipe_lr.fit(x_train, y_train).predict(x_test)
```

Table 5.9: Classification report of the final Logistic Regression.

Class	*Precision*	*Recall*	*F1-score*	*Support*
0	0.99	0.99	0.99	7779
1	0.97	0.97	0.97	3236
micro avg	0.98	0.98	0.98	11015
macro avg	0.98	0.98	0.98	11015
weighted avg	0.98	0.98	0.98	11015

the Support Vector Machine, the `TfidfVectorizer` is defined as described in Section 5.3. Line two defines the Logistic Regression model with the parameter combination chosen above. In line three a pipeline is used again to chain the vectorizer and the model from line one and two. In line four the model is fitted using this pipeline, the training data and the training labels (`fit()`). The prediction is also performed in the same line (`predict()`) using the test data.

The overall score of the model defined in Program 5.7 is 98% and the runtime was 28 seconds, which is 70 times faster than the runtime of the Support Vector Machine from Program 5.5. Table 5.9 shows the full classification report, which shows the precision, recall and f1-score for the class 0 as well as the class 1. It can be seen that the model performs slightly better for the classification of the non paid articles as well. Same as stated above for the Support Vector Machine, this is due to the imbalance of the dataset, as way more non paid articles were provided to the model to learn and generalize from. However, with a precision, recall and f1-score of 97% the final model of the Logistic Regression performed well for the classification of paid articles too. The confusion matrix from Table 5.10 shows that 87 articles are False Negatives, which means that they are predicted to be non paid but actually are paid articles. It also shows that 104 articles are False Positives, which means that they are classified as paid articles although they are actually non paid articles. All in all, there is a misclassification of 1.7%.

5.4.3 Neural Network

In order to be able to use the same evaluation metrics as for the Support Vector Machine and the Logistic Regression, the `recall` and the `f1-score` metrics needed to be implemented for the Neural Network. The implementation can be seen in Appendix B.1. Before passing the data to the Neural Network the features needed to be extracted using

Table 5.10: Confusion Matrix of the final Logistic Regression.

		Predicted Label	
		Positive	*Negative*
Actual Label	*Positive*	7692	87
	Negative	104	3132

Table 5.11: Impact of the vocabulary size on the performance of the Neural Network.

vocabulary size	*time*	*recall*	*f1*	*loss*
1000	6min 6sec	0.989	0.905	0.299
10000	28min 23sec	0.940	0.949	0.148
50000	2h 10min 2s	0.966	0.961	0.123
80000	3h 36min 37sec	0.967	0.961	0.114

the one hot encoding as described in Section 5.3 paragraph “one hot encoding”. The optimization of the model was done by experimenting with the data as well as with the model itself. Concerning the data it was experimented with the size of the vocabulary as well as the input length of the model.

Table 5.11 shows the impact of the vocabulary size on the performance of the Neural Network. The number of epochs was set to 20 for these experiments to keep the calculation time low. The input length was set to the same value as the vocabulary size. The vocabulary sizes of 50000 and 80000 performed best, only with small differences in recall and loss. The vocabulary size of 50000 performs a bit worse but scores with considerably shorter calculation time. Since the f1 score is the same for both and the recall and loss does not differ a lot, the value 50000 is chosen as the vocabulary size for the final Neural Network.

Table 5.12 shows the impact of the input length on the performance of the Neural Network. For the experiments the number of epochs was set to 20 and a vocabulary size of 80000 was chosen. The figure shows that with an increase of the input length the computation time increases rapidly. The recall and the f1-score also increase but much slower. The input length of 80000, which is the same value as the vocabulary size, delivers the best results. Therefore the input size will be set to the same value as the vocabulary size.

Table 5.13 shows how the parameter `batch_size` affects the performance of the model. For these experiments the number of epochs was set to 20 to keep the calculation time low. The results from the table show that the larger the batch size gets, the worse the model performs. A batch size of 33043, which is equal to the number of training samples available, performs the worst. Looking at these results the default value of 32 will be used for the `batch_size` as this value led to the best result.

Table 5.14 shows how the Neural Network improved over the epochs. It can be seen that after epoch 3 the model did not improve anymore. The highest validation recall

Table 5.12: Impact of the input length on the performance of the Neural Network.

input length	*time*	*recall*	*f1*	*loss*
500	5min 2sec	0.939	0.950	0.115
1000	5min 56sec	0.938	0.952	0.122
10000	29min 22sec	0.934	0.954	0.132
50000	2h 21min 2sec	0.948	0.963	0.113
80000	3h 36min 50sec	0.951	0.962	0.121

Table 5.13: Impact of the batch size on the performance of the Neural Network.

batch size	*time*	*recall*	*f1*	*loss*
32	4min 41s	0.959	0.963	0.100
256	3min 44s	0.948	0.960	0.073
512	3min 36s	0.943	0.957	0.068
1536	3min 34s	0.942	0.956	0.071
3304	2min 35s	0.816	0.886	0.145
6608	2min 36s	0.761	0.838	0.284
16521	2min 36s	0.391	0.5189	0.485
33043	2min 41s	0.196	0.297	0.526

Table 5.14: Improvement of the Neural Network over epochs.

Epoch	*loss*	*recall*	*f1-score*
1	0.0775	0.9555	0.9553
2	0.0579	0.9616	0.9647
3	0.0571	0.9642	0.9646
4	0.0620	0.9544	0.9619
5	0.0627	0.9581	0.9630
6	0.0636	0.9610	0.9644

was reached in epoch 3 (0.9642), the highest validation f1-score in epoch 2 (0.9647). The lowest validation loss was reached in epoch 3 (0.0571). Therefore it can be concluded that 3 epochs are sufficient to get good results and the value 3 is chosen for the number of epochs of the final algorithm.

Program 5.8 shows the final and optimized implementation of the Neural Network. The metrics used to evaluate the model are `fmeasure_pred`, which is the implementation of the f1-score, and `recall_pred`, which is the implementation of the recall score (see

Program 5.8: Final algorithm of the Neural Network.

```
model = Sequential()
model.add(Embedding(50000, 8, input_length=50000))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid')
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[fmeasure_pred,
                                                    recall_pred])
model.fit(X_train, y_train, epochs=3, validation_data=(X_test, y_test))
```

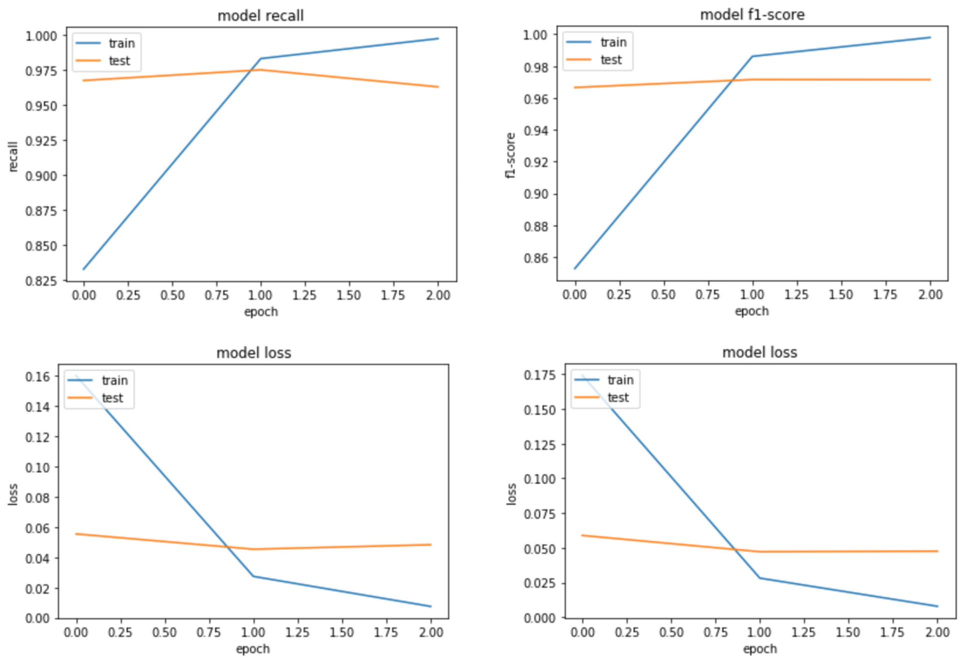

Figure 5.8: Results of the final Neural Network of its epochs

Appendix B.1). The vocabulary size as well as the input length is set to 50000 and the model is training over three epochs. The plotted results over the epochs are shown in Figure 5.8.

Figure 5.8 shows the performance of the final model over its three epochs. The blue lines represent the results for the training set and the orange lines represent the results for the test set. The upper left image shows the recall score and the lower left image the loss of the model for the recall. The right side shows the same information as the left side, but for the f1-score. The recall and the f1-score as well as their losses are almost the same for the training set. However one can see that for the test set the recall score is slightly decreasing after epoch two whereas the f1 score is not.

The accuracy of of the model defined in Program 5.8 is 98% and the runtime was 21 minutes and 30 seconds. Table 5.15 shows the detailed classification report of the model after all three epochs. This report shows the precision, recall and f1-score for the class 0

Table 5.15: Classification report of the final Neural Network.

Class	*Precision*	*Recall*	*F1-score*	*Support*
0	0.98	0.99	0.99	6207
1	0.98	0.96	0.97	2605
micro avg	0.98	0.98	0.98	8812
macro avg	0.98	0.98	0.98	8812
weighted avg	0.98	0.98	0.98	8812

Table 5.16: Confusion Matrix of the final Neural Network.

		Predicted Label	
		Positive	*Negative*
Actual Label	*Positive*	6161	46
	Negative	108	2497

as well as the class 1. The precision is exactly the same for the classes 0 and 1. However, the recall as well as the f1 score are with a score of 99% both slightly better for class 0. This is due to the imbalance of the dataset as stated above for the Support Vector Machine and the Logistic Regression. Nevertheless, the performance of the model for class 1 is with a recall of 96% and a f1 score of 97% still pretty satisfying. The confusion matrix in Table 5.16 shows that 46 articles are False Negatives, which means that they are predicted to be non paid but actually are paid articles. It also shows that 108 articles are False Positives, which means that they are classified as paid articles although they are actually non paid articles.

Chapter 6

Evaluation

The evaluation is split into four main parts. The first part, Section 6.1, tackles the 10-fold cross validation of the Support Vector Machine, the Logistic Regression and the Neural Network. In the section the evaluation results of all models are compared to each other to see if which one performs the best. Section 6.3 evaluates and analyses the impact of the sentiment scores on the performance of the model. In this section it is shown if the sentiment scores can help improving the models. Section 6.4 recaps and analyses the main findings from the previous sections. Furthermore, in this section the models are evaluated not only regarding their recall and f1-score, but also in terms of their run time performance. This section also answers the research question stated in Section 1.2. The last part, Section 6.5, draws a conclusion of the evaluation results.

6.1 10-Fold Cross Validation of the Support Vector Machine and the Logistic Regression

In this section, a 10-fold cross validation is performed for the Support Vector Machine and the Logistic Regression. As scoring methods the scoring functions `recallscore` and `f1score` from Program 5.3 are used again. The models are evaluated once for each metric and the results are compared to each other independently for each metric.

Program 6.1 shows the implementation of the 10-fold cross validation. First the Support Vector Machine and the Logistic Regression classifiers and their hyperparameters are defined in line two and three. For this the final and optimized models from Section 5.4.1 and 5.4.2 are taken. The data is vectorized using the the optimized TfidfVectorizer (line one) as described in Section 5.2. Line five and six define the pipelines for vectorization and classification, one for each classification algorithm. Then an array of models to evaluate is created, to be able to loop over the models and store the results in a corresponding array for later visualisations. In line 15 the scoring function is defined among others. In this example the `recallscore` is used. This code is executed a second time using the `f1score` instead. The 10-fold cross validation itself is performed according to the code from line 13 to 16. The results, stored in the `results` array, are then used to create a boxplot, which is shown in Figure 6.2 for the `recallscore` and in Figure 6.1 for the `f1score`.

Figure 6.1 shows the result of the 10-fold cross validation of the Support Vector

Program 6.1: Implementation of 10-fold cross validation.

```python
vect = TfidfVectorizer(max_df=0.7, min_df=4)
clf_logreg = LogisticRegression(C=100, class_weight='balanced', penalty='l2', solver='liblinear')
clf_svm = svm.SVC(C=1, class_weight='balanced', kernel='linear')

pipe_logreg = Pipeline([('vect',vect), ('clf',clf_logreg)])
pipe_svm = Pipeline([('vect',vect), ('clf',clf_svm)])

models = []
models.append(('LR', pipe_logreg))
models.append(('SVM', pipe_svm))

results = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=42, shuffle=True)
    cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=recallscore)
    results.append(cv_results)
```

```
LR: 0.973321 (0.002506)
SVM: 0.973083 (0.002375)
```

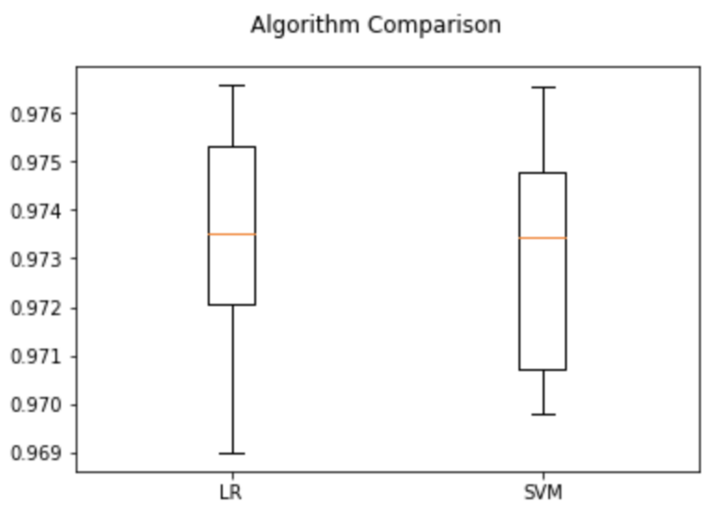

Figure 6.1: 10-fold cross validation of Support Vector Machine and Logistic Regression with f1 scoring.

Machine (SVM) and the Logistic Regression (LR) using the `f1score`. The results for the f1-score are almost the same for both models, with a difference of only 0.001692%. Also the standard deviation of both models is almost the same (0.00126 difference).

Figure 6.2 shows the result of the 10-fold cross validation of the Support Vector Machine (SVM) and the Logistic Regression (LR) using the `recallscore`. Same as for

```
LR: 0.972541 (0.005527)
SVM: 0.973930 (0.005568)
```

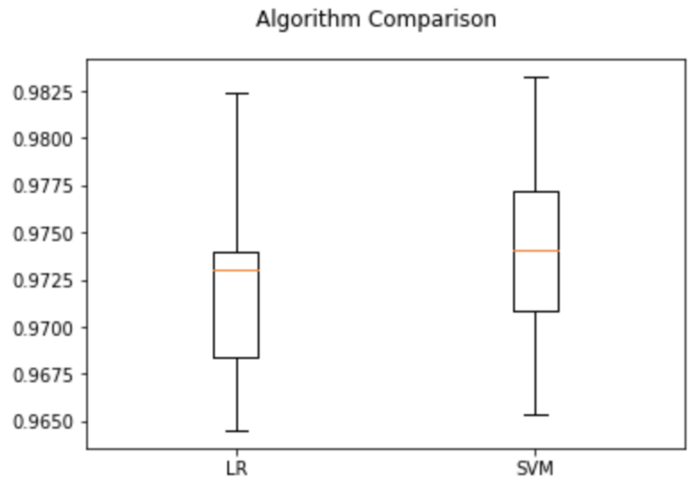

Figure 6.2: 10-fold cross validation of Support Vector Machine and Logistic Regression with recall scoring.

the f1-score, the results for the recall for both models do not differ a lot. The recall for the Support Vector Machine is 0.9697 and for the Logistic Regression 0.9691 (only 0.000617 difference). Also the standard deviation of both models is almost the same (0.001454 difference).

6.2 10-Fold Cross Validation of the Neural Network

Since the default metric defined for the Neural Network is the accuracy and during the experiments of applying the custom scoring functions the the 10-fold cross validation of the Neural Network a lot of problems occurred, it was decided to keep to accuracy metric for the evaluation. The accuracy is simply the number of total correct predicted instances divided by the total number of instances. Program 6.2 shows the implementation of the 10-fold cross validation of the Neural Network. Line one to seven define and return the compiled model. In line nine the keras model is wrapped using a Keras Classifier[1] wrapper so that it can be used by scikit-learn. This wrapped network is then taken in line 12 to perform the cross validation.

Figure 6.3 shows the result of the 10-fold cross validation from Program 6.2. From these results it can be seen that the Neural Network performed approximately equally well than the Support Vector Machine and the Logistic Regression as the score (mean accuracy) of both models is 0.98.

[1] https://keras.io/scikit-learn-api/

Program 6.2: 10-fold cross validation of Neural Network.

```python
def neuralNetwork():
    model = Sequential()
    model.add(Embedding(vocab_size, 8, input_length=max_length))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
    return model

neural_network = KerasClassifier(build_fn=neuralNetwork, epochs=3, verbose=2)
results = []

scores = cross_val_score(neural_network, X_train, y_train, cv=10)
results.append(scores)
```

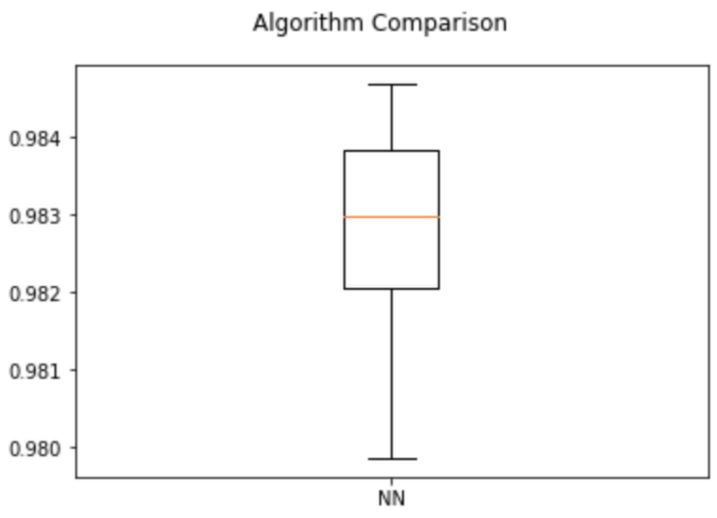

Figure 6.3: 10-fold cross validation of Neural Network with accuracy score.

6.3 Impact of Sentiment Scores on Model Performance

The assumption that most paid articles result in a positive sentiment turned out to be true with a percentage of 63.99% positive paid articles (see Figure 5.6). However with just slightly over half of the paid articles resulting in a positive sentiment this feature looks not as promising as expected for the identification of paid articles. Nevertheless, experiments with the feature of the body texts and the sentiment scores were performed as follows.

As the Logistic Regression scores with the lowest computation time, the experiments with the sentiment addition were carried out with the optimized model of the Logistic Regression. Table 6.1 shows the precision, recall and f1-score for the classes “0” (not sponsored) and “1” (sponsored) for the model with and without included sentiment feature. From the results it can be seen that there is not much difference in the performance

Table 6.1: Performance of Logistic Regression with and without sentiment analysis.

sentiment	*class*	*precision*	*recall*	*f1-score*
not included	0	0.99	0.99	0.99
included	0	0.99	0.98	0.98
not included	1	0.97	0.97	0.97
included	1	0.95	0.97	0.96

Table 6.2: Confusion matrix of Logistic Regression with and without sentiment analysis.

		Without Sentiment		*With Sentiment*	
		Predicted Label		*Predicted Label*	
		Positive	*Negative*	*Positive*	*Negative*
Actual Label	*Positive*	7673	106	7618	161
	Negative	103	3133	97	3139

neither for class “0” nor for class “1”.

In addition to the scoring metrics also the confusion matrix was calculated for the model with and without included sentiment scores. Table 6.2 shows the comparison of both confusion matrices. Table 6.1 showed that there is not much difference in the performance. Table 6.2 on the other hand gives more insights on how the results changed. With the inclusion of the sentiment scores it can be seen that the false positive rate decreased but at the same time the false negative rate increased. This shows that the misclassified items just shifted but were not minimized.

6.4 Results

As stated above, the performance of the Support Vector Machine, the Logistic Regression as well as the Neural Network does not differ a lot. So it can be concluded that this performance is the upper limit for the performance that can be reached with the data provided.

The sentiment result turned out to be not significant enough to increase the performance of the model. However, in case of doubt the sentiment result could be inspected for marginal articles in future research. One thing that is certain is that if an article results in a negative sentiment it is not likely to be sponsored as the rate of negative paid articles is under 10% (see Figure 5.6). However the positive sentiment rate of the sponsored articles is not high enough to contribute enough to be a relevant feature and was therefore decided to not be included.

Since all three algorithms performed approximately equally well, they were also analyzed in terms of computation time, which is shown in Table 6.3. This table shows that the Logistic Regression was by far the fastest computed model with around 27 seconds, followed by the Neural Network (NN) with around 22 minutes and the Support

Table 6.3: Computation time of final models.

	SVM	*Logistic Regression*	*NN*
Computation time	32min 59sec	26.9 sec	21min 47sec

Vector Machine (SVM) with around 33 minutes. Therefore, in terms of computation time, the Logistic Regression outperformed the other algorithms by far.

To answer the research question it can be said that it is definitely possible to distinguish between paid and non paid news articles. The question *to which extent can paid articles be automatically identified from online news platforms* can be answered by taking a closer look at the recall score for class “1” of the optimized models:

- With the Support Vector Machine, it is possible to automatically classify 97% of the paid articles correctly as paid ones from online news platforms.
- With the Logistic Regression, it is possible to automatically classify 97% of the paid articles correctly as paid ones from online news platforms.
- With the Neural Network, it is possible to automatically classify 96% of the paid articles correctly as paid ones from online news platforms.

6.5 Conclusion

Neural networks are extremely flexible and can identify patterns and significant features from any data structure. However, this is just possible if a sufficient amount of training data is provided. Small and medium sized dataset usually contains too little training example for Neural Networks to generalize well and achieve good results. Moreover, the amount of parameters available to modify is very high for a Neural Network. Therefore, lots of experiments have to be performed to find the optimal parameter combination for the optimal results.

The amount of free parameters in a Neural Network can easily get very high. Due to this the model is endangered of overfitting if not enough training points are provided to prevent it. Additionally it will be very time consuming to train datasets that are large enough to avoid overfitting [27].

Since with linear models, such as the Support Vector Machine and the Logsitic Regression, fewer hyperparameters can be adjusted as for a Neural Network, the grid search for those models is less time consuming. Therefore, good results can be achieved faster and easier. With the Support Vector Machine even a global optimum can be guaranteed. To avoid overfitting and underfitting certain regularisation parameters can be experimented with. The amount of data needed for such linear models is smaller than for the Neural Network. Therefore, the Support Vector Machine and the Logistic Regression are good choices for small to medium sized datasets.

All in all, the Support Vector Machine, the Logistic Regression and the Neural Network performed approximately equally well. The improvements of the Neural Network were stopped as soon as reasonable results were achieved since the aim was to include a simple Neural Network to see if such a model can compete with linear models, such as the Support Vector Machine and the Logistic Regression, when classifying news articles. From the results it can be concluded that with an increase of labelled news data available, a Neural Network can get very relevant when choosing the optimal model.

Chapter 7

Conflicts and Further Research

This chapter highlights the main issues in limitations that occurred during the implementation of the project in Section 7.1, as well as the possibilities for further research and improvement in Section 7.2.

7.1 Issues and Limitations

As discussed in Chapter 5, after crawling urls for new articles the content of those articles needes to be scraped timely. Otherwise it could happen that some articles are not available anymore as they were deleted shortly afterwards.

Another big challenge is the distribution of paid and non paid articles. Within the daily news feed way more not sponsored than sponsored articles get published. Therefore, if all news articles are taken the ratio between paid and non paid articles would be too imbalanced to achieve good classification results. To counteract this so that the dataset gets more balanced, in another crawling and scraping iteration only paid articles were stored and the non paid articles were discarded. This was repeated until a reasonable distribution was reached. Also not all platforms provide an archive. Therefore, news from previous years can't be taken into consideration since those articles are just available for a few but not the majority of all platforms.

7.2 Possibilities for Further Research

Since the performance of the algorithms was limited to the quality of the data, for further research it is recommended to continue gathering data and solve the imbalance of the platform distribution the articles come from.

The amount of data available is sufficient for the Support Vector Machine and the Logistic Regression, but to further improve the results of the Neural Network it is crucial to collect way more data. Then the experiments and algorithm comparisons should be repeated to see if the Neural Network can outperform the Support Vector Machine and the Logistic Regression.

Appendix A

CD-ROM/DVD Contents

Format: CD-ROM, Single Layer

A.1 PDF

Path: /

Obermayr_Theresa_2019.pdf	Master Thesis

A.2 Source Code

Path: /project

project.zip	Source Code of Algorithm Implementation and Evaluation

A.3 Online-Literature

Path: /online-sources

literature	Online sources referenced in the thesis

Appendix B

Additional Source Code

Program B.1: Implementation of metrics precision, recall and f1-score for NN

```python
from keras import backend as K

def precision(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def recall(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def fbeta_score(y_true, y_pred, beta=1):
    # If there are no true positives, fix the F score at 0 like sklearn.
    if K.sum(K.round(K.clip(y_true, 0, 1))) == 0:
        return 0
    p = precision(y_true, y_pred)
    r = recall(y_true, y_pred)
    bb = beta ** 2
    fbeta_score = (1 + bb) * (p * r) / (bb * p + r + K.epsilon())
    return fbeta_score

def f1_score(y_true, y_pred):
    return fbeta_score(y_true, y_pred, beta=1)
```

References

Literature

[1] V. Kishore Ayyadevara. *Pro Machine Learning Algorithms. A Hands-On Approach to Implementing Algorithms in Python and R.* Berkeley, CA: Apress, 2018 (cit. on pp. 3, 6–8).

[2] Asa Ben-Hur and Jason Weston. “A User’s Guide to Support Vector Machines”. *Methods in Molecular Biology* 609 (Jan. 2010), pp. 223–239 (cit. on p. 24).

[3] Seppe vanden Broucke and Bart Baesens. *Practical Web Scraping for Data Science. Best Practices and Examples with Python.* Berkeley, CA: Apress, 2018, pp. 155–172 (cit. on p. 3).

[4] Nello Cristianini and Elisa Ricci. “Support Vector Machines”. In: *Encyclopedia of Algorithms.* Ed. by Ming-Yang Kao. Boston, MA: Springer, 2008, pp. 928–932 (cit. on p. 7).

[5] Enaitz Ezpeleta, Urko Zurutuza, and José María Gómez Hidalgo. “Does Sentiment Analysis Help in Bayesian Spam Filtering?” In: *Hybrid Artificial Intelligent Systems: 11th International Conference.* Vol. 9648. Apr. 2016, pp. 79–90 (cit. on pp. 16, 18).

[6] Luis Gonzalez-Abril et al. “Handling binary classification problems with a priority class by using Support Vector Machines”. *Applied Soft Computing* 61 (2017), pp. 661–669 (cit. on pp. 13, 14, 18).

[7] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 9).

[8] Tarik S. Zakzouk and Hassan I. Mathkour. “Comparing text classifiers for sports news”. *Procedia Technology* 1 (2012), pp. 474–480 (cit. on pp. 12, 13, 18).

[9] Dipanjan Sarkar. *Text Analytics with Python. A Practitioner’s Guide to Natural Language Processing.* Berkeley, CA: Apress, 2019, pp. 275–342 (cit. on pp. 5, 8).

[10] Dipanjan Sarkar, Raghav Bali, and Tushar Sharma. *Practical Machine Learning with Python. A Problem-Solver’s Guide to Building Real-World Intelligent Systems.* Berkeley, CA: Apress, 2018, pp. 203–205 (cit. on p. 9).

[11] Tej Bahadur Shahi and Pant Ashok Kumar. “Nepali news classification using Naïve Bayes, Support Vector Machines and Neural Networks”. In: *2018 International Conference on Communication Information and Computing Technology (ICCICT).* Feb. 2018, pp. 1–5 (cit. on pp. 16–18).

[12] Shan Suthaharan. *Machine Learning Models and Algorithms for Big Data Classification. Thinking with Examples for Effective Learning.* Springer, Oct. 2015, pp. 123–128 (cit. on p. 4).

[13] Alper Kürşat Uysal and Serkan Gunal. “The impact of preprocessing on text classification”. *Information Processing Management* 50.1 (2014), pp. 104–112 (cit. on pp. 14, 15, 18, 23).

[14] Rini Wongso et al. “News Article Text Classification in Indonesian Language”. *Procedia Computer Science* 116 (2017), pp. 137–143 (cit. on pp. 11, 12, 18).

[15] Peter Zadrozny and Raghu Kodali. *Big Data Analytics Using Splunk.* Berkeley, CA: Apress, 2013, pp. 255–257 (cit. on p. 9).

[16] Xinhua Zhang. “Support Vector Machines”. In: *Encyclopedia of Machine Learning and Data Mining.* Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer, 2017, pp. 1214–1220 (cit. on p. 6).

Online sources

[17] Arthur A. *First neural network for beginners explained (with code).* 2019. URL: https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf (visited on 06/23/2019) (cit. on p. 7).

[18] Tara Boyle. *Hyperparameter Tuning.* 2019. URL: https://towardsdatascience.com/hyperparameter-tuning-c5619e7e6624 (visited on 05/23/2019) (cit. on p. 8).

[19] François Chollet et al. *Keras: The Python Deep Learning library.* 2015. URL: https://keras.io/ (visited on 05/29/2019) (cit. on p. 10).

[20] Bundesministerium für Digitalisierung und Wirtschaftsstandort. *Kennzeichnung entgeltlicher Veröffentlichungen gemäß § 26 Mediengesetz.* Jan. 2018. URL: https://www.usp.gv.at/Portal.Node/usp/public/content/brancheninformationen/information_und_kommunikation/kennzeichnung/66074.html (visited on 03/18/2019) (cit. on p. 1).

[21] Kavita Ganesan. *Tfidftransformer Tfidfvectorizer Usage Examples and Differences.* 2019. URL: http://kavita-ganesan.com/how-to-use-tfidftransformer-tfidfvectorizer-and-whats-the-difference/#.XMa695MzaL4 (visited on 04/29/2019) (cit. on p. 35).

[22] Jose Parreño Garcia. *Tuning parameters for logistic regression.* 2016. URL: https://www.kaggle.com/joparga3/2-tuning-parameters-for-logistic-regression?scriptVersionId=534495 (visited on 06/14/2019) (cit. on p. 25).

[23] Techopedia Inc. *Natural Language Toolkit (NLTK).* 2019. URL: https://www.techopedia.com/definition/30343/natural-language-toolkit-nltk (visited on 05/29/2019) (cit. on p. 9).

[24] Hafsa Jabeen. *Stemming and Lemmatization in Python.* 2018. URL: https://www.datacamp.com/community/tutorials/stemming-lemmatization-python (visited on 04/07/2019) (cit. on p. 22).

[25] Keras.io. *The Sequential model API.* 2019. URL: https://keras.io/models/sequential/ (visited on 05/05/2019) (cit. on p. 26).

[26] Bernd Klein. *Text Categorization and Classification.* 2018. URL: https://www.python-course.eu/text_classification_introduction.php (visited on 03/18/2019) (cit. on p. 1).

[27] Dmitry Laptev. *Neural networks vs support vector machines: are the second definitely superior?* 2019. URL: https://stats.stackexchange.com/questions/30042/neural-networks-vs-support-vector-machines-are-the-second-definitely-superior (visited on 05/29/2019) (cit. on p. 53).

[28] Avinash Navlani. *Text Analytics for Beginners using NLTK.* 2018. URL: https://www.datacamp.com/community/tutorials/text-analytics-beginners-nltk (visited on 05/29/2019) (cit. on p. 9).

[29] scikit-learn. *Generalized Linear Models.* 2018. URL: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (visited on 04/06/2019) (cit. on p. 25).

[30] Raheel Shaikh. *Cross Validation Explained: Evaluating estimator performance.* 2018. URL: https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85 (visited on 04/04/2019) (cit. on p. 28).

[31] Sagar Sharma. *Epoch vs Batch Size vs Iterations.* 2017. URL: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9 (visited on 05/05/2019) (cit. on p. 26).

[32] Koo Ping Shung. *Accuracy, Precision, Recall or F1?* 2018. URL: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9 (visited on 04/01/2019) (cit. on p. 27).

[33] Drew Wilimitis. *The Kernel Trick in Support Vector Classification.* 2018. URL: https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f (visited on 05/19/2019) (cit. on p. 7).