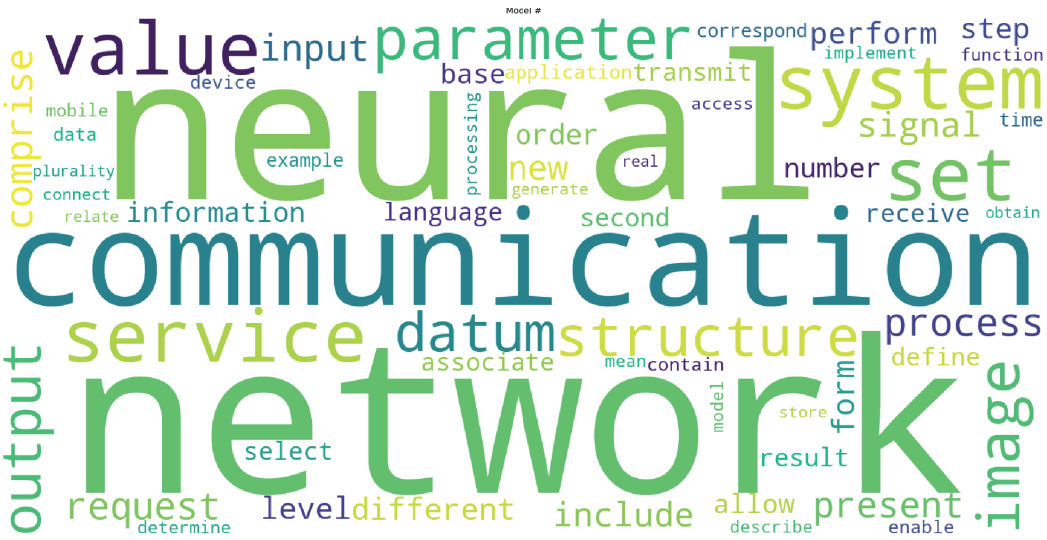
René-Marcel Kruse/Benjamin Säfken/
Alexander Silbersdorff/Christoph Weisser (Eds.)

Learning Deep Textwork

Perspectives on Natural Language Processing and Artificial Intelligence



Universitätsdrucke Göttingen

René-Marcel Kruse /Benjamin Säfken/
Alexander Silbersdorff/Christoph Weisser (Eds.)

Learning Deep Textwork

erschienen in der Reihe der Universitätsdrucke
im Universitätsverlag Göttingen 2021

René-Marcel Kruse
Benjamin Säfken
Alexander Silbersdorff
Christoph Weisser (Eds.)

Learning Deep Textwork

Perspectives on Natural Language Processing and Artificial Intelligence

Universitätsverlag Göttingen
2021

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.dnb.de> abrufbar.

Adresse der Herausgeber
Georg-August-Universität Göttingen
Lehrstühle für Statistik und Ökonometrie
Humboldtallee 3
D-37073 Göttingen
https://www.uni-goettingen.de/de/411195.html
E-Mail: rene-marcel.kruse@uni-goettingen.de
E-Mail: asilbersdorff@uni-goettingen.de
E-Mail: benjamin.saefken@uni-goettingen.de
E-Mail: c.weisser@stud.uni-goettingen.de

Satz und Layout: Christoph Weisser

https://univerlag.uni-goettingen.de
ISBN: 978-3-86395-501-4
doi:10.17875/gup2021-1608

Learning Deep Textwork - Perspectives on Natural Language Processing and Artificial Intelligence

A. Silbersdorff[1,2], **B. Säfken**[1,2], **R. Kruse**[1], **and C. Weisser**[1,2]

[1]Georg-August-Universität Göttingen, Germany
[2]Campus-Institut Data Science (CIDAS), Germany

Preamble

The rise of artificial intelligence has come to the forefront of both academic and public discussion in recent years and accordingly the topic has gained considerable interest among students. Many of the recent advances in and the growing use of natural language processing are built around the applications of deep learning algorithms. In the winter semester 2020/21, we thus decided to offer a new seminar on deep learning algorithms for text processing to students of the masters programme in applied statistics at the University of Göttingen. Following the Humboldtian model of higher education, we aimed to allow students to conduct their own research into the basic ideas, mechanics and practical applications of deep learning algorithms and natural language processing and thereby learn about the issue more deeply than by conventional lecture-based teaching. Their findings were presented in the seminar and subsequently portrayed in article-styled seminar papers. The results of this seminar, both in terms of the advancements made by many students in their understanding and the quality of many of the submitted seminar papers, were strikingly positive and deserving of publication in our eyes. Thus we decided to give the students the chance to publish their work in this edited volume. The best seminar papers were thus selected for publication and the selected students went through a full review process conducted by two researchers active in the fields of deep learning and natural language processing. Upon successfully addressing the issues raised by the reviews, the articles were included in this volume. The contributions are structured as follows: The first paper by Bennet Sohns provides a sentiment analysis contrasting the performance of different neural networks. Using a dataset of manually labelled text massages from a Telegram chat group, they find that n-gram vectorization shows the best results while classical preprocessing like aggregating emojis and lemmatization was implemented.

The second paper by Carsten Dolle and Kolja Holzapfel trains classifiers on highly imbalanced data to detect fraudulent job posting. An encoder based BERT-variant and densely connected deep neural network are implemented and various over- and undersampling techniques are applied. Moreover, model ensembling is used to increase detection rates.
The third paper by Johannes Brachen and Alisa Rothe evaluates common preprocessing techniques for a Twitter sentiment analysis with a Long Short Term Memory (LSTM) model. They find minor benefits for lowercasing, normalizing repeated letters and mentions and automatic spelling correction, but a clear decrease in performance for stop words removal.
The fourth paper by Kim Sarah Meier and Henrike Meyer is concerned with the impact of different levels of preprocessing on the predictive performance of deep learning models. The study uses a Twitter dataset with pre-labelled tweets with either positive or negative sentiment. A binary classification with a LSTM model is used.
The fifth paper by Malte Kramer and Claire Reiß compares Convolutional Neural Networks and LSTM models with Support Vector Machine, Multinomial Naive Bayes and Random Forest Classifiers for a sentiment analysis on tweets about COVID-19. They find that the LSTM clearly outperforms Convolutional Neural Networks and the standard machine learning classifiers.
The sixth paper by Katharina Linke and Jasmin Wiebke Schilling is concerned with the sentiment classification of Twitter data on the US Election 2020 with deep learning models. They implement various approaches such as a sentiment analysis with the TextBlob library or a clustering based methods.
The seventh paper by Joanna Simm and Sophie Potts provides a deep learning analysis on German news articles contrasting the performance of different classification algorithms. Using an imbalanced dataset of news articles it finds that Complement Naive Bayes yields superior results to any of the implemented Convolutional Neural Networks.
The eighth and last paper by Markus Heidenreich and Nils Muttray uses social network data from Twitter to analyze the use of Twitter by ISIS followers. They point out that the effective identification of terrorist content on Twitter is a research application with high practical relevance. Their suggested solutions is based on Recurrent- as well as Convolutional Neural Networks and automatically detects tweets with terrorist content on Twitter.
From these contributions of the selected students participating in the seminar, we hope on the one hand that the reader gains insights into the topics addressed in the contributions. On the other hand, these contributions hopefully portray the scope and depth of the understanding developed by students when left to explore deep learning algorithms in a deep manner following the Humboldtian model of higher education.
We thank the Campus-Institut Data Science (CIDAS) for funding this project.

Contents

Malte Kramer, Claire Reiß

Coronavirus tweets NLP

Katharina Linke, Jasmin Wiebke Schilling

Labelling and predicting sentiments in Twitter Data on the US Election 2020

Sentiment Analysis of Text Messages

Bennet Sohns

1 Introduction

Sentiment Analysis has become a highly used tool for different commodities. It is used by a wide range of companies for marketing purposes and even by investors at the stock market for the outperformance of the market (Feldman 2013). The sentiment analysis can be either based on a two options categorism, i.e. positive and negative, or consist of a n-point scale, e.g. very positive, positive, neutral, negative, very negative (Prabowo & Thelwall 2009). Sentiment analysis is a method, which aims to extract opinions from a large amount of data. The first successful experiments were made in the 2000s. In the following time large public available data and new models let the topic of sentiment analysis grow (Mozetič et al. 2016). Moreover, the increasing popularity of social media has created many data and cases for sentiment analysis. While Twitter data is used often for different task, i.e. Hate Speech Detection (Badjatiya et al. 2017), this paper however uses text messages from a telegram group. To analyse the sentiment three different neural networks will be used with the given data. In this paper a simple Artificial Neural Network, a Convolutional Neural Network and a Recurrent Neural Network will be used. Moreover, various kinds of preprocessing tools will be applied.

2 Methods

2.1 Data

The Data consists of less than 6000 samples. The data has been retrieved from a telegram chat with the python package telethon and transformed to a csv-file. The API offers many different information, but for this analysis only the message itself will be used. The chat speech is German. The new dataframe has three columns, the chat message and the two sentiment ratings. The first varies from zero to four, where zero is a negative statement, one a slightly negative, three a neutral and 3 and 4 are slightly positive and positive sentiment. The latter varies from zero to two, where zero indicates a negative sentiment, one a neutral sentiment and two a positive sentiment. The two different options for the classes will be compared later on. The labels are added manually. The intuition for two different labels is, that even for a

human it is challenging, rather a sentiment is very good or just slightly good. The difference in the performance will be examined later. The User ID is removed and to maintain anonymity, the names of the chat participants in each of the messages is pseudonymized. A rather equal distribution among the three classes is given, where bad has 1978 samples, neutral has 1824 and good has 1772 samples.

Therefore the data set in the three class option can be considered balanced, which is important for the interpretation of the results later on. The following messages show an example per class starting from bad to neutral and the last is a good sentiment: "Deswegen echt nicht so lustig", "Wann reset?", "Ja aber witzig". To increase the data consistency, each text message is preprocessed.

2.2 Preprocessing

The used emojis are grouped, rather implying a positive sentiment or a negative one. These emojis are translated in "lachen" for positive meaning and "man" for a negative meaning. The chat Messenger Telegram offers the implementation of a bot. A bot can be added to a chat and his actions can be triggered with certain commands, usually starting in Telegram with "/" as indicator of a bot command. The chat therefore has many bot commands that have been posted by users to get information from the bot, followed by a bot message, which delivers the requested information. Both types of messages have been removed for the given data set, since these have no value for the sentiment analysis. Moreover, the bot triggers on different content e.g. pictures and displays in a message the votes of each user. The vote is related to the content, which another user has posted into the group. Therefore, this bot message is seen as a sentiment statement itself. For simplicity the different votes are aggregated into one vote by calculating the mean of the given votes from the number of users. After the preprocessing of the strings lemmatization. Lemmatization is a technique to derive the given word into its base, a lemma. It is considered to be a core task in text analysis. However, the common natural language processing package NLTK does not provide a lemmatization for the German language (Lezius et al. 1998). Therefore another package is needed. The HanTa package does offer a German lemmatizer. The model will be computed with and without it to evaluate its influence on the performance.

As the data consists of strings, these have to be changed to vectors, to be readable for the neural network.

Two common model options exist, n-gram and sequence type. The first one splits the text message either in a couple of words, a single word or a single character. The latter one is using the whole text message i.e. the sample as a sequence.

Google developers suggest to chose the depending model by a simple calculation. If the sample size divided by the average words per sample is smaller than 1500, one should chose the n-gram model. Since the sample size of the retrieved data is close to 6000 and the average words per sample are 4, the outcome of the equation is around 1500. This means, that both options are a possible choice. Sequential models are known to perform better with larger sample sizes. Both potential models will be tested, to score the best results for the given data set (Developers 2021).

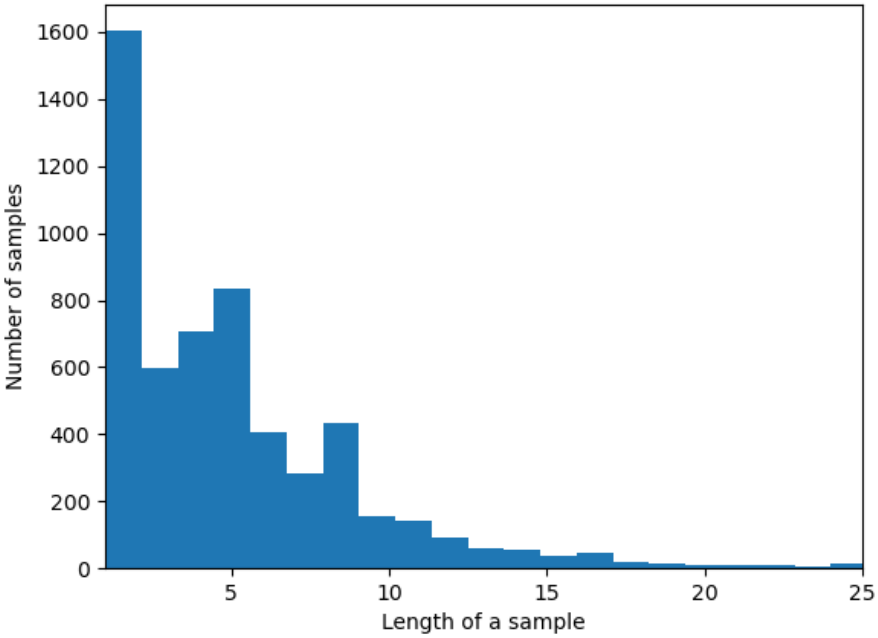


Figure 1: Sample length distribution

2.3 Vectorization

For the n-gram model a vectorizer will be used. There are different options for the vectorization, e.g. Bag of Words approach, HashVectorizer, CountVectorizer (sklearn) or the TFidfVectorizer. The HashVectorizer does not have the ability to do a inverse transformation, this might be usefull to see, which words actually have an important effect on the validation. Therefore the HashVectorizer will not be used. The other two options will be compared in a GridSearchCV from SKLearn. The TFidf stands for term-frequency and idf for inverse document-frequency. This specific vectorizer is useful to actually keep the focus on words that are used less often and not on words which are appearing on a high frequency in the corpus, e.g. “das”, “ein”, “ist” in German. The TFidfVectorizer is equal to the CountVecotrizer followed by the TfidfTransformer. The first one is used to convert a collection of string documents to a matrix of token counts. The output is a scipy.sparse.csr.matrix. The GridSearchCV suggest to chose the CountVectorizer without the TFidfTransformer. The intuition behind it is, that chat messages are rather short and dense text messages, which usually only contain the most important words in a comprehensive form. Fill words

might not be used very often. Therefore the use of the TfidfTransformer can remove certain key words, which occur quite often, e.g. "lachen" or "nicht" (Figure 2). Moreover the GridSearch suggests a N-Gram form of (1,2).

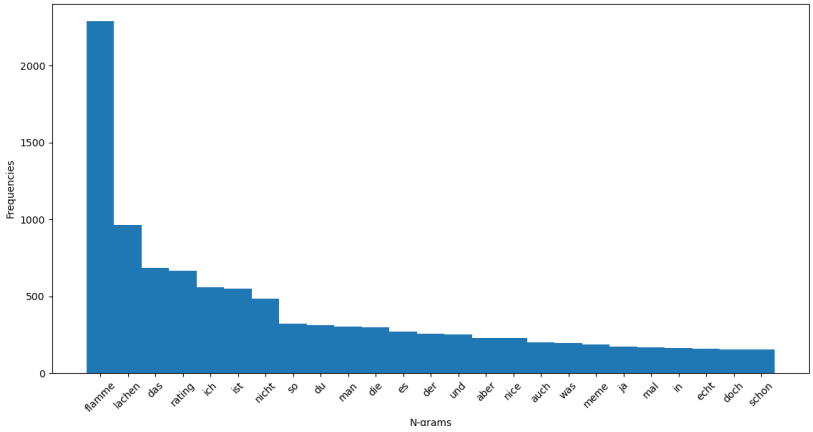


Figure 2: Frequency distribution of n-grams

2.4 Artificial Neural Networks

This sentiment analysis will focus on the n-point scale. Therefore it has three classes:

$$C = c_1, c_2, c_3 \tag{1}$$

One of the models that will be used in this paper is a Multi-Layer Perception (MLP) model. It is part of the Artificial Neural Networks (ANNs), which are inspired by the human brain. They are supervised and generate a nonlinear function model, which make an input data based prediction of output data possible. The MLP model can be expressed as (Taud & Mas 2018):

$$y = f(z) \text{ and } z = \sum_{i=0}^{n} w_i \cdot x_i. \tag{2}$$

The MLP is a layered feed-forward network and uses a learning algorithm back-propagation. Feed-forward means, that information is lead directly through the input layer to the output layer through the hidden layers (Bishop et al. 1995). Moreover, back-propagation is a mechanism, where the correction of each weight starts at the output layer and goes backwards through the layers, while correcting them through propagating (Du & Swamy 2014). The used model has a dense layer with the activation function ReLu each followed by a dropout layer. The last dense layer has only 3 units, for the three given outputs and in comparison to the other dense layers it has the

activation function softmax. This activation function will create a vector with the length being equal to the number of units of the output layer. For the other model with 5 classes, the last layer has consequently 5 units.

2.5 Convolutional Neural Network

Another promising neural network architecture is a convolutional neural network. Usually these kind of models are very popular for image classification with 2D Convolutional layers. Since the data set has strings the intuition is to use a 1D Convolutional layer, where the filters work similar to n-grams. While a traditional neural network usually does a multiplication of Matrices of parameters to describe the relation between input, i.e. every input relates to every output, convolutional neural networks have a different approach. They have a sparse interaction, where the kernel is smaller than the input. For example an image has millions of pixel, but to detect important features within the image not every single pixel as to be analyzed. This does reduce the amount of parameters, which have to be stored. Therefore the memory needed can be reduced as well and the amount of operations for creating the output can be lowered. It can be summarized, that the convolutional approach is more statistical efficient and needs less memory. The layer setup of a convolutional neural network usually starts with a convolutional layer, this is followed by a non linear activation function such as ReLu. After this step a pooling functions follows. A popular pooling function is the max function. This function chooses the maximum output within a rectangular (Goodfellow et al. 2016). The max function is used in this paper as well, as it leads to a faster convergence (Scherer et al. 2010). Generally speaking the purpose of the pooling layer itself is to reduce the size of the given feature maps, and thereby reduce the complexity of the model (O'Shea & Nash 2015).

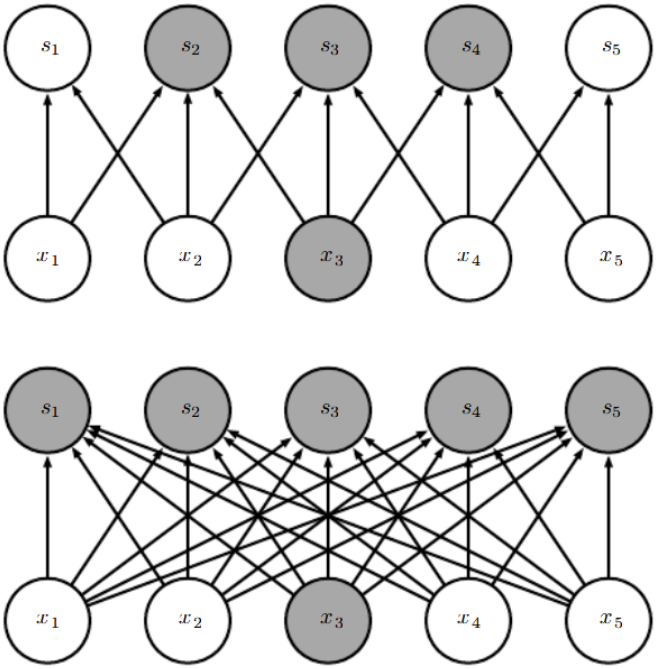


Figure 3: Sparse connectivity view from below (O'Shea & Nash 2015)

An important part is parameter sharing. As seen in the figure, in the top part the kernel has a width of five, but only three of the outputs are effected by x. In comparison in the bottom example, all outputs are influenced by x. This demonstrates parameter sharing very well, where a parameter is used not only for one function per model but for multiple functions (O'Shea & Nash 2015).

2.6 Recurrent Neural Network

The third model which is used in this paper is a recurrent neural network. This kind of model is often used for sequence of data, such as time series, e.g. stock prices. The intuition to use a recurrent neural network for a sentiment analysis of chat messages is, that a many-to-one case usually works well with RNNs/LSTMs. As shown in the analysis of the data set, the sentences have a small amount of words and chat messages are often very simple texts. The idea is, that repetitive sentences might work well with the memory of an RNN. Figure 4 shows, how a RNN cell looks. In comparison to an artificial neural network is has a loop. The way how it is constructed allows the network to actually keep information within the RNN and use it later on. In the right part of the Figure the network is displayed unrolled. Each cell indicates the state at some point in time t (Olah 2015).

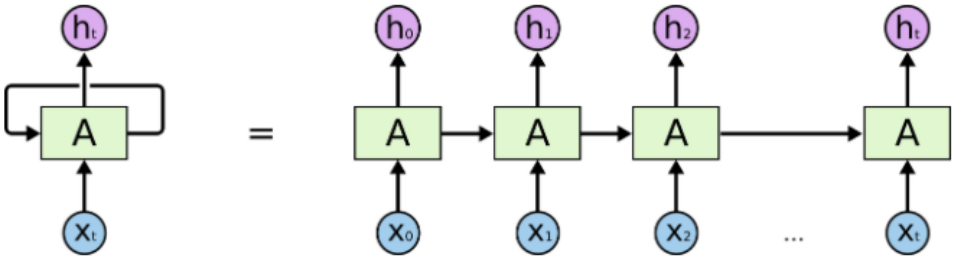


Figure 4: An unrolled recurrent neural network (Olah 2015)

Rather than using a standard RNN, the network will be extended with the long-short term memory. Which is a popular addition to avoid the problem of the vanishing gradient. This term describes a well known problem, where the error term vanishes over time through the back-propagation. This leads to the problem, that RNNs can´t learn long term dependencies very well. In comparison LSTM networks perform well even for time lags of 1000 (Hochreiter 1998). The construction of the LSTM layer is displayed in Figure 5. This kind of layer has the same chain like build up as a simple RNN had.

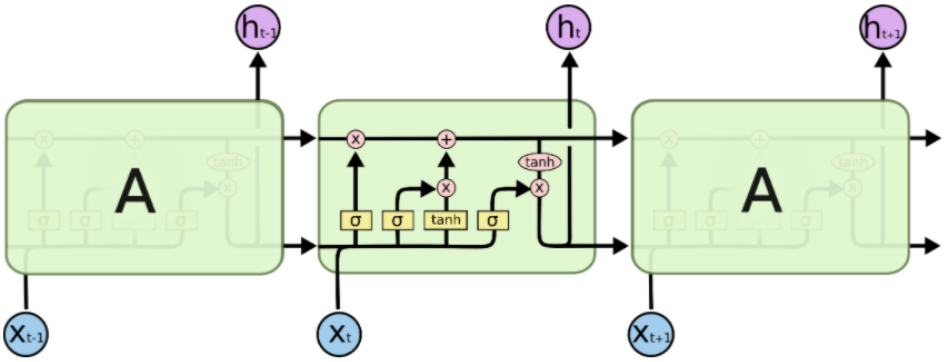


Figure 5: Long Short Term Memory (Olah 2015)

Compared to the RNN the structure is more complex. It has 4 so called “gates”, which regulate the actual memory of the LSTM cell. The top horizontal line is the cell state and it can be changed by the cell itself depending on the other inputs, i.e. information can be added or removed. The first vertical line is called the “forget gate layer”. It has a sigmoid function and can be therefore either be 0 or 1. The next vertical line is called the “input layer”, a new vector with information is calculated by a tanh function and the flow is regulated with another sigmoid function. And the last layer sets the actual output of the cell. The setup of the LSTM helps to diminish the problem of the vanishing gradient and might therefore be a better option for the given task and data set. Both presented kinds will be used to find the optional model for the sentiment analysis (Olah 2015).

2.7 Hyperparameter

For the optimizer adam is used and the loss function sparse categorical cross-entropy is applied. Adam is one of the most used optimizers. It is very simple to handle and does not need a lot of specification in terms of parameters which have to be set. Moreover the tuning process of adam is relatively simple. Another advantage is, that it is very efficient The pseudocode for the adam optimizer is displayed below (Kingma & Ba 2014):

Algorithm 1 Adam: Adaptive moment estimation. Default parameters are $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$

Require: α : Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: ϵ : Small integer to avoid division by zero
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0 : Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
  $v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$(Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 6: Pseudocode Adam (Kingma & Ba 2014)

Three different types of neural networks have been presented and used for the sentiment analysis. For each model the hyper-parameters were optimized to get the best results for the given data. Exemplary for all networks the optimization of an artificial neural network will be shown. The ANN has many hyper-parameters to optimize. For the given data set a gridsearch is used to find the optimal amount of layers and units for the model. For one, two or three layers and 20, 40, 60, 80, 100 and 120 units the performance of the ANN is examined.

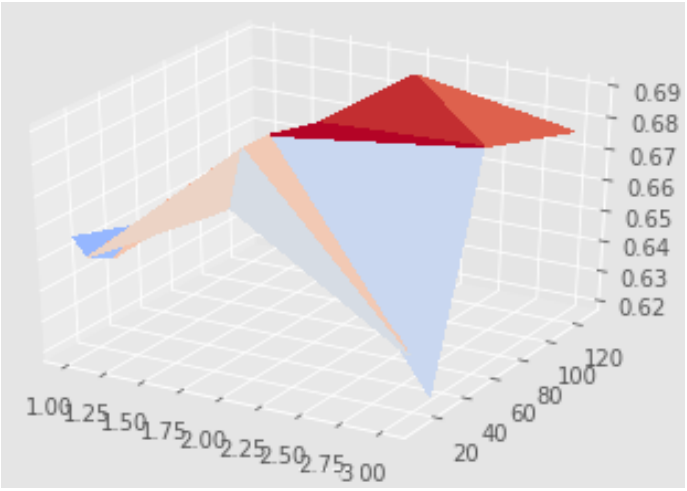


Figure 7: Gridsearch for Model

2.8 Overfitting

To avoid overfitting regularization is necessary to optimize the results of the network. Overfitting is a term in statistics which describes the specification of a model which has to many parameters. Speaking about the ANN, it means that the model learns the patterns in the given data set very well. So well, that the model sticks to close to the particular data set. Therefore it has problems with unknown data, because it takes random noise in the given training data as a concepts, which leads to a negative performance in other unknown data. Speaking generally for machine learning, overfitting is also called overtraining, because a model is trained rather than fit. Usually a model with overfitting has a higher variance due to to many parameters. On the other hand a model can also underfit, this is the case if the model has a simple build up, which overgeneralizes the given data. Therefore the relations in the input data are only vaguely learned by the model. If a model does underfit it has a lower variance in comparison to a model which overfits, but it suffers from a relatively large bias and has for this reason a large prediction error (Raschka & Mirjalili 2017).

Early Stopping is the easiest way to prevent the model from overfitting. This method uses the duration of the model training as a hyperparameter to optimize the outcome. The duration is measured in epochs, whereas one epoch is the period, in which the whole data set is seen once by the model. If the minimization of the loss is the goal, the callback can be used for this kind of signal catching. After each epoch the early stopping function implemented with a callback can check if the loss has had a significant change in the correct direction. A parameter “patience” can be used to accept a certain amount of epochs, in which the loss can be unchanged but the callback does not stop the training process of the given artificial neural network (Prechelt 1998).

A method to prevent overfitting is dropout. Dropout is a technique to randomly drop out units and the connections of the units. This will stop units from co-adapting to much. The figure below shows this process in detail. While in the left part of the figure no dropout is applied and therefore all units are connected with each other, this is not the case in the right part of the figure. In b) dropout is applied. The units which are marked with the cross are removed and not longer counted or connected to the other units, which remain in the neural network.

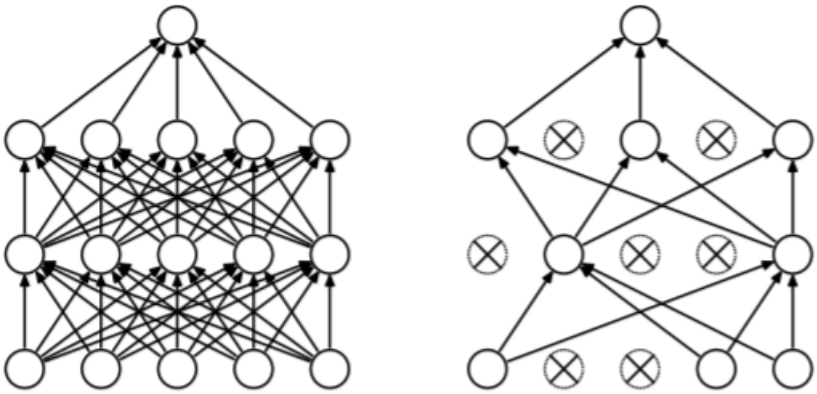


Figure 8: With and without Dropout (Srivastava et al. 2014)
aufgenommen

In the model which is used in this paper dropout layer will be applied. This kind of regularization does address the weights of the neural network (Srivastava et al. 2014).

3 Results

Different models with different hyperparameters and preprocessing have been trained with the given data set.

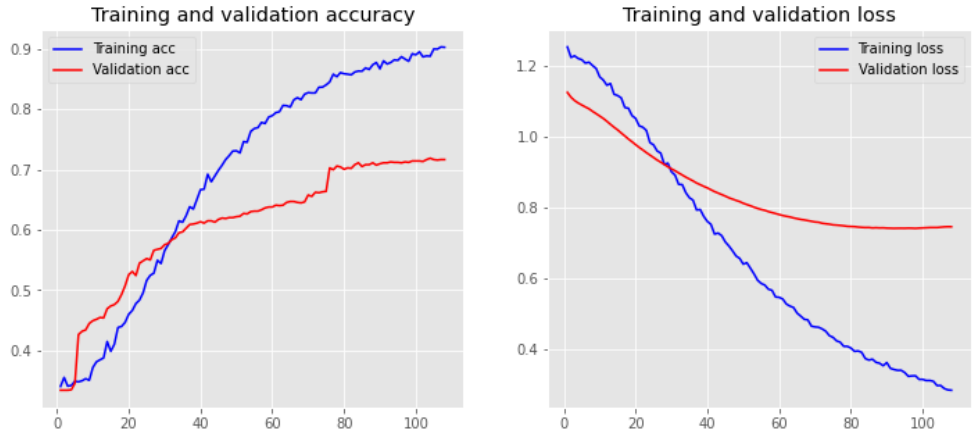


Figure 9: MLP with 3 classes

The best model has been the n-gram MLP with 2 layers and 64 units with the CountVectorizer and without the Hanta Lemmatization. The model reached more than 0.7 accuracy, as shown in the graph. Moreover, the loss accuracy goes down to 0.8. Since the model is trained with a balanced data set, the accuracy is an appropriate tool to measure the performance of the artificial neural network. The model has a callback function applied. This will stop the training process if the validation loss does not decrease in 10 consecutively periods. The batchsize is set to 150 as this showed the best results. The epochs are set to 300. But due to the implemented callbacks this amount of epochs is not needed at all. The MLP with five classes does not perform a lot worse than the MLP with 3. The other models did not perform as well. The table displays the complete results.

Table 1: Comparison of the models

Model	Validation Accuracy
MLP with 3 classes	0.72
CNN with 3 classes	0.70
RNN with 3 classes	0.67
MLP with 5 classes	0.62

Potential problems of the model with the data set, are displayed within a confusion matrix. For each class the matrix shows the predicted outcome and the correct class. A value on the diagonal of the matrix, implies a correct prediction of the model. If

the value is not on the diagonal, the model did do a false prediction. This kind of matrix can expose the weaknesses of a model, i.e. if the model does a specific miss classification over and over again.

Figure 10: Confusion matrix for the ANN with 3 classes

As the confusion matrix in Figure 10 shows, the model has a high rate of correct classified samples. Also there is no clear evidence for an ongoing misclassification in a certain class.

To get an intuition of the abilities of the created artificial neural network, the following table will show a small sample of correct and misclassified examples. In the first column are selected examples displayed and in the second column is the prediction of the presented ANN with the highest accuracy. Table 2 displays the likelihood of the bad, neutral and good sentiment respectively for the given example.

Table 2: Correct and Misclassified Examples

Example	Prediction
Die Beweise sprechen gegen dich	[0.980, 0.017, 0.002]
Ich kringel mich vor lachen	[0.009, 0.154, 0.835]
Starker Auftakt heute von User2	[0.160, 0.695, 0.143]
Lass deinen Frust echt mal woanders aus	[0.503 , 0.226, 0.270]

As shown in the table the first two examples are clearly classified correctly with 0.98 probability for a bad sentiment in the first case and a 0.835 probability in the second example. Especially the latter sentence has a clear indication with the word “lachen”. This word was already named as one of the highest mentioned words in the data set. For the third example the result is not correct. While the sentence implies a positive sentiment, the artificial neural network does give the highest value in the vector to the neutral sentiment. However, the sentiment is not as clear for this sentence compared two the first two examples. In the last example the sentences is correctly classified, but has only a 0.503 value in the correct sentiment class given by the model. This is

not as clear as it was for the first two examples. The sentence itself can be considered semantically more complex. The model seems to be relatively good at classifying simple sentences and struggling with increasing complexity of the semantics of the sample. This might be due to the lack of a large sample size for the given task.

4 Conclusion

In this paper various kinds of neural networks were applied to the data set. The data set was downloaded with the python package Telethon from a Telegram chat group. Moreover, different preprocessing methods were used to score the best possible result for the sentiment analysis of chat messages. The data consistency was improved by aggregating emojis and removing content without any value for the process and with lemmatization the whole data set was edited. The artificial neural network with the n-gram vectorization showed the best results. Surprisingly the lemmatization toolkit specially developed for the German language did not improve the results, however the dropout layer at 0.5 reduced overfitting. The MLP with five classes does not perform a lot worse than the MLP with three classes.

With only about 6000 samples the data set is relatively small compared to usual sentiment tasks with for example tweets with sample sizes above 1 million. With an increasing sample size the convolutional neural network might have been the best choice, since this does perform better with larger data sets. Due to the fact, that the data itself has been labeled by hand for this paper the data is limited. Keeping this fact in mind the results are quite encouraging.

5 Appendix

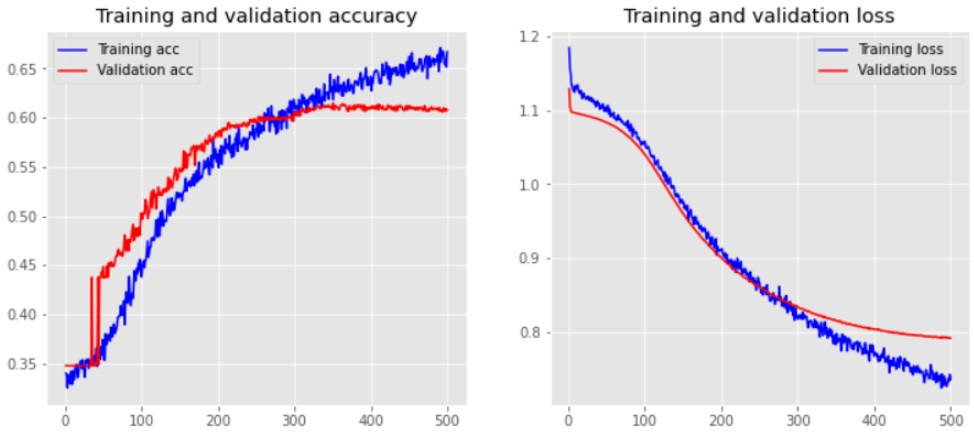


Figure 11: MLP Model with 5 classes

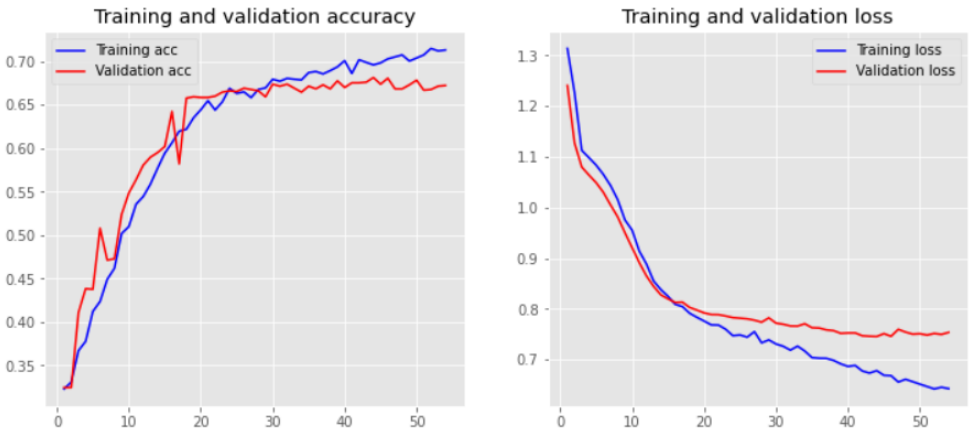


Figure 12: GRU with 3 classes

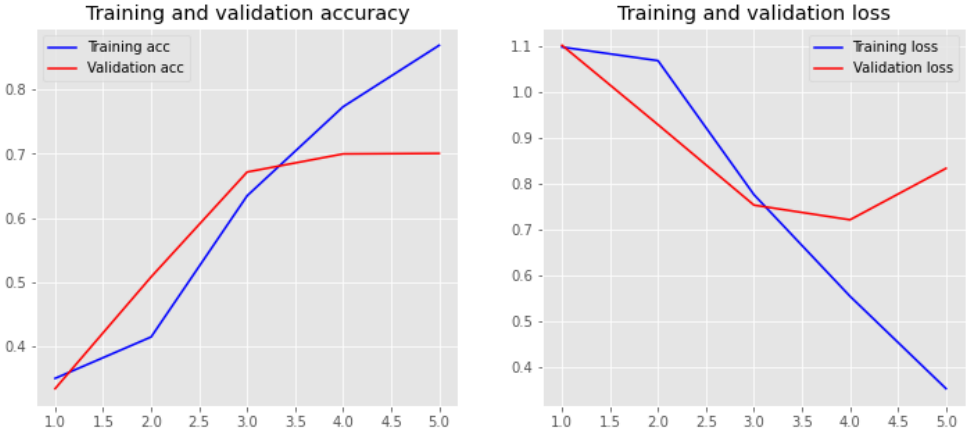


Figure 13: CNN with 3 classes

References

Badjatiya, P., Gupta, S., Gupta, M., & Varma, V. 2017, in Proceedings of the 26th international conference on World Wide Web companion, 759–760
Bishop, C. M. et al. 1995, Neural networks for pattern recognition (Oxford university press)
Developers, G. 2021, Text classification
Du, K.-L. & Swamy, M. 2014, in Neural Networks and Statistical Learning (Springer), 15–65
Feldman, R. 2013, Communications of the ACM, 56, 82
Goodfellow, I., Bengio, Y., & Courville, A. 2016, Deep Learning (MIT Press), `http://www.deeplearningbook.org`
Hochreiter, S. 1998, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6, 107
Kingma, D. P. & Ba, J. 2014, arXiv preprint arXiv:1412.6980
Lezius, W., Rapp, R., & Wettler, M. 1998, arXiv preprint cs/9809050
Mozetič, I., Grčar, M., & Smailović, J. 2016, PloS one, 11, e0155036
Olah, C. 2015, Understanding LSTM Networks
O'Shea, K. & Nash, R. 2015, arXiv preprint arXiv:1511.08458
Prabowo, R. & Thelwall, M. 2009, Journal of Informetrics, 3, 143
Prechelt, L. 1998, in Neural Networks: Tricks of the trade (Springer), 55–69
Raschka, S. & Mirjalili, V. 2017, Scikit-Learn, and TensorFlow. Second edition ed
Scherer, D., Müller, A., & Behnke, S. 2010, in International conference on artificial neural networks, Springer, 92–101
Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. 2014, The journal of machine learning research, 15, 1929
Taud, H. & Mas, J. 2018, in Geomatic Approaches for Modeling Land Change Scenarios (Springer), 451–455

Dealing with Imbalanced Data for Fraudulent Job Postings - An Application of BERT and Sampling Techniques

Carsten Dolle, Kolja J. Holzapfel

1 Introduction

Imbalanced data sets are quite common in the real world. We see imbalances occur e.g. in medical data (Davagdorj et al. 2020), in text data from Twitter or various other social media platforms (Paul & Saha 2020; Casula & Tonelli 2020), or, like in our case, in classification tasks dealing with fraud.

In recent years, great advances have been made in machine learning using text data to generate predictions. One of those was the advent of the Transformer architecture as a Natural Language Processing (NLP) deep learning model (Vaswani et al. 2017) and the subsequent arrival of pre-trained transformer-based models like BERT, short for Bidirectional Encoder Representations from Transformer (Devlin et al. 2018), which led to new state-of-the-art results. At the same time however, deep learning methods in combination with imbalanced data are according to Johnson & Khoshgoftaar (2019) understudied. With our paper we want to focus on exactly this gap by combining deep learning with in our case sampling methods.

To accomplish that task we use a dataset that we obtained from Kaggle (Bansal 2020). It contains 17880 obsevations of job posting offers of which 866 observations are marked as fraudulent. One observation consists of 16 variables measured and the variable of interest which is a dummy indicating whether a variable is fraudulent or not. The bulk of information are given in text fields, with the exception of dummies for the possibility to telecommute, whether the ad contains a company logo and whether the ad contains screening questions. Furthermore, data on employment type and required education per ad are coded as factor variables, while data on the salary range are given as an integer range. All other variables contain text information. A tabular overview of the different variables can be found in the appendix.

As can be seen in Figure 2, our data set is highly imbalanced with only around 5 percent of the job postings being fraudulent ones. Also, our set consists of a mix of different variable types. Additionally, we found that not every text variable contains actual text but is often left blank. Therefore we decided to double use these

information and extract on the one hand the text and on the other hand convert these variables to another dichtome variable with the information 'left blank or not'. We ended up conducting this procedure with all text variables and discarded only the variable `job_id` from the beginning, since it is just a running number and should not contain any useful information.

Figure 1: The left wordcloud describes the text found in non-fraudulent ads, the right plot shows the text of fraudulent ones.

We conducted some preliminary data exploration and focused on differences between class 0 non-fraudulent and 1 fraudulent in both, factor structures and text structures. Figure 3 shows two exemplary analysis plots for factors. Here we calculated the percentage of observations in either class, that, had a company description in their job posting. We can see, that the percentage differs greatly between class 0 and 1, letting us believe it is worthwhile to use this information. However, some variables barely differ across classes like the the inclusion of a benefits description in the ad. Therefore we conducted a more thorough feature selection via the use of a Pearson's X^2-test. We also surveyed text information with the help of a wordcloud which can be seen in Figure 1 and found some differences like e. g. the more prominent use of 'project' in fraudulent data and the use of 'bachelor degree' in non fraudulent data versus 'high school' for fraudulent data. Both wordclouds have been taken from the Kaggle Notebook of Mranal Jadhav[1]. Because of these variances in the data we conclude, that a two pronged approach via a deep text classification model combined with an easier model dealing with categorical data could yield good predictions.
From this preliminary analysis alone two problems become apparent. We are dealing with a highly imbalanced data set and have a mixture of different variable types at hand. The variable mixture directly influenced our model choice. For the text part of our paper we chose a BERT-model which is a popular choice and sets benchmark values not only in text classification (Minaee et al. 2020). For the comparatively small amount of values for our categorical data, we chose a simple Feed Forward Network (FFN) with three layers. To tackle the prediction problem of different variable types, we chose a two-pronged approach with a simple implementation of model ensembling via gridsearch as described by Chollet (2018).
To improve the prediction of imbalanced data, many routes are possible. Johnson and Khoshgoftaar mention three categories: Preprocessing methods, cost sensitive learning methods and algorithm-centered approaches (Johnson & Khoshgoftaar 2019).

[1] https://www.kaggle.com/mranaljadhav/real-fake-job-post

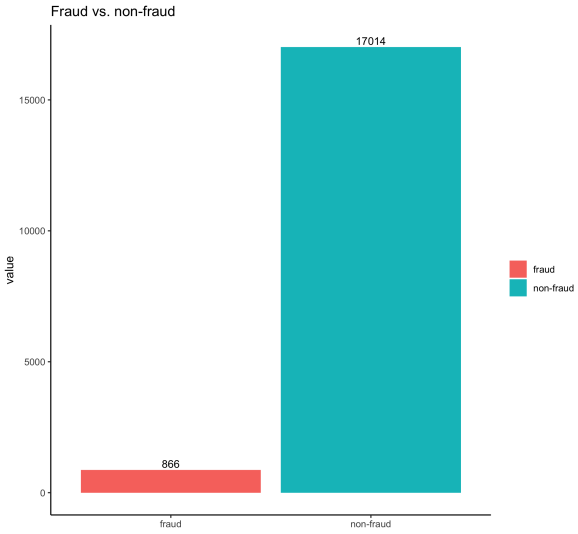


Figure 2: Imbalance of the data set visualized

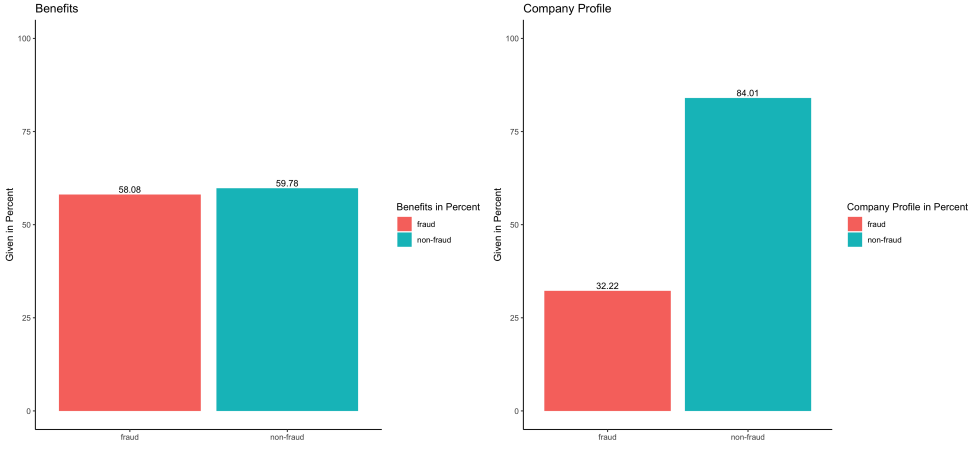


Figure 3: Left plot distribution of the benefits variable, right plot distribution of having a company profile or not

We chose a preprocessing approach, namely sampling methods like over- and undersampling, because it does not require us to change the BERT model, keeping the scope of the paper clear and narrow and also making it comparable to other BERT-based approaches.

All in all, we have identified two problems we want to tackle in this paper. First, we have a highly imbalanced data set and want to evaluate the influence of preprocessing methods on both a feed forward network for categorical variables and in particular on a BERT instance. Second, we want to obtain solid prediction values from our data set and use model ensembling to account for different data structures contained in the set.

The rest of the paper is structured as follows: In section 2 we first give an overview about the methods and the subsequent implementation. Therefore, we clarify which sampling techniques we used and introduce then the Transformer architecture and BERT. Following that, we give an overview about the categorical model we used. Afterwards we describe the preprocessing we have done for every model. At the end of the section we introduce and describe the implementation of our model ensembling approach. Section 3 deals with the results obtained. In section 4 we close the paper with a short discussion and an outlook about further research.

2 Methods

In this section we first give an overview of different sampling techniques we implemented to tackle imbalanced data. In the next subsections we then subsequently introduce our chosen models and follow up with a description of our model implementations.

Our model architecture is due to the data structure based on two approaches. We first used a BERT instance for the text part of our data and used a feed forward neural network for the categorical part. Since BERT is heavily based on the Transformer architecture we provide a short introduction here and complete the part with a look into BERT-specific theory and data processing. Following that we give a description of the feed forward network and describe the preprocessing done for the second model part.

The last section of this chapter is comprised of an introduction to model ensembling and our subsequent implementation approach.

2.1 Dealing with Unbalanced Data – Over and undersampling

The occurrence of fraudulent examples in our dataset is highly skewed: There are many more legitimate than fraudulent samples. This can lead to overfitting of the trained neural network with regard to the majority class. The weights of the neurons are adjusted to the training dataset during each epoch and through the imbalance the weights get more adjusted to the majority class. This effect is pronounced since the majority class outweighs the minority by the factor 12. Oversampling tries to mitigate this problem by introducing new examples to the dataset. In the easiest form of oversampling, random oversampling balances the dataset by randomly copying observations of the minority class to balance the dataset. This in turn brings in a

new chance to overfit to the data because the same examples are used to adjust the weights of the neural network multiple times. To counteract this problem synthetic example generation was proposed and employed by Chawla et al. with the method of 'Synthetic Minority oversampling Technique', in short SMOTE (Chawla et al. 2002). This technique creates new synthetic datapoints by selecting a random sample, from which the k-nearest neighbors, with the default setting of 5-nearest neighbors, are drawn. The distance in the feature-space between the sample and the nearest selected neighbor out of the five is calculated and multiplied with a random number between 0 and 1. The result is then added along the corresponding axis of the sample to create the values of the new synthetic sample. Then this process is repeated with the next-nearest neighbor until the desired ration of synthetic samples have been created. This process tends to create 'larger and less specific decision regions' (Chawla et al. 2002), which in turn should lead to a better generalization.

Several researchers have proposed changes to the Synthetic Minority oversampling Technique which change the distribution of synthetic samples. Some of these techniques are aggregated and provided in the `imbalanced-learn`-package by Lemaître, Nogueira and Aridas (Lemaître et al. 2017). Those techniques can be differentiated by the distribution of newly synthesized samples as well in the way these samples get created: The 'Borderline-SMOTE' variant, looks trough all samples of the minority class and sorts those in three categories: If the k-nearest neighbors are from the majority class then this samples are categorized as noise, if the k-nearest neighbors are in most cases from the minority class they are classified as 'safe', if the k-nearest neighbors are in most cases from other classes then thess samples are labeled as 'DANGER'. The samples of the last category are then used to generate the synthetic samples (Han et al. 2005). This should improve the detection of fringe cases, which are potentially hard to classify. SVM-SMOTE also generates synthetic samples in borderline regions the improve the performance, regarding the classification of fringe cases. In this method, samples close to the border between the minority and majority classes are considered, but in this method the synthetic samples are extrapolated, if the k-nearest neighbors are from the minority class, or interpolated, if the k-nearest neighbors are from the majority class (Nguyen et al. 2011). Another variation is K-means SMOTE, which works in three steps: clustering, filtering and oversampling. In the clustering step the input space is clustered in k groups using k-means clustering, which is a method for identifying naturally occurring groups in data. In the filtering step the clusters with more minority samples than majority samples are identified, in which the samples are used to generate synthetic samples in the oversampling step. Thus cleaning the decision boundaries between the clusters (Last et al. 2017), which should also improve the decision clarity for fringe cases. Oversampling should remove the bias towards the majority class of the dataset, while simultaneously increasing the size of the dataset. This increase in datapoints also increases the amount of training steps needed.

Undersampling removes samples from the majority class to obtain a balanced dataset and thus reduces the size of the dataset. A basic way to balance the dataset is through random undersampling. Samples from the majority class are randomly discarded until a specific ratio of all classes is reached. Usually the aim is to reach an equilibrium between all classes. Random undersampling bears the danger of losing potentially useful information, especially if borderline-samples are randomly

removed. `Imbalanced-learn` offers some undersampling approaches that each employ a different strategy to minimize the potential loss of information through removing random samples (Lemaître et al. 2017). Analogous to the techniques employed in oversampling the methods aim to remove specific cases of the dataset, but in contrast to previous methods they don't target borderline samples. Instead they reduce the denseness of clusters or other redundant datapoints, noise or misleading samples, under the presumption that training on those datapoints would net no performance gain and could even hamper the accuracy. The 'Neighborhood Cleaning Rule' (NCR) proposed by Jorma Laurikkala, is designed to remove noisy data with the 'Wilson's edited nearest neughbor rule' (ENN) (Wilson & Martinez 2000) while also removing datapoints which are misclassified by the three nearest neighbors, if those come from a class which accounts for more than 50 percent of the dataset (Laurikkala 2001). This method balances the dataset only to an amount, predetermined by the properties of the dataset. All in all, undersampling techniques reduce the amount of samples in the dataset und thus decrease training time. The loss of information in the majority class, could potentially lead to a strong loss in detection accuracy.

The benefits, namely clearer decision boundaries and shorter computing time, of over - and undersampling can be combined and some flaws, e.g. longer computing time or loss of information, can be overcome by combining the two approaches. In these combined techniques samples from the majority class are removed with undersampling techniques and synthetic samples are produced until the desired ratio between the majority and minority classes is reached. The SMOTEENN variant offered by `imbalanced-learning` (Lemaître et al. 2017), proposed by Batista et al. (2004), first oversamples the dataset with the SMOTE technique to obtain a balanced dataset. Afterwards ENN is used to remove datapoints from all classes, thus removing samples that are misclassified by its 3-nearest neighbors.

Over- and undersampling techniques were designed with datapoints in mind, that can be modified along the feature-space. This is without preparation not possible for textual data-entries. Embedding methods can recode textual data into numeric values, that with the right preparation, allows for modification along the axes of the feature-space. `GloVE` is an embedding of textual data designed with this goal in mind (Pennington et al. 2014). Davis Liang showed that the Embeddings produced by the encoder architecture of BERT-models, also seem to show similar properties. However, the BERT-Embedding rely to a larger degree on contextual information (Liang 2019). Thus, a modification of BERT-Embeddings along the vector space should be done with precaution.

2.2 Transformer architecture

The big difference between the Transformer architecture introduced by Vaswani et al. (2017) and more traditional approaches of networks with memory is the eschewing of sequentiality and the sole reliance on so-called self-attention for memorizing and learning data dependencies. This is advantageous for two reasons. First, long-range dependencies are hard to model via sequential approaches. Say, we have for example a sequence of words and try to model dependencies between them. The length of computational paths between any combination of inputs and outputs are critical for

this ability. Transformers tend to have shorter paths and are therefore well suited to learn dependencies. Second, the Transformer architecture allows for a more efficient learning phase due to an enhanced ability to parallelize processes as well as needing in general less computations.

In general, the Transformer architecture is an encoder/decoder structure. Because BERT uses only the encoder part, we will focus on that during the section. An encoder stack here consists of multiple Transformer blocks, in the BERT case 12 or 24, in Vaswani et al. (2017) six blocks are used. One block consists of two sublayers of which one is a self-attention block followed by a simple fully connected feed-forward network.

The key here is a multi-headed attention mechanism in the self-attention block. Vaswani et al. use a 'scaled dot product attention' (Vaswani et al. 2017). Figure 4 shows a stylized diagram of how this mechanism works. Via fitting matrices, three linear projections for every single input vector are created. In the paper they are called Query, Key and Value. They have a smaller dimension $d_k < d_{model}$ than the input vector with d_{model}. This becomes important when multi-headed attention is considered. For each query vector the mechanism calculates the dot product with every key vector. The resulting scores are in the next step multiplied by a scaling factor $\frac{1}{\sqrt{d_k}}$. Afterwards a softmax function is applied that presses the scaled scores as weights in regions between 0 and 1. These weights are an expression of the dependence between inputs. The scaling factor is supposed to counteract large scores for single values, which would lead to partially very low softmax scores. This procedure results in a vector with values between 0 and 1 that sums up to exactly 1. This weighting vector is multiplied by a matrix containing the original value vectors corresponding to the keys to form one weighted average representation of the input.

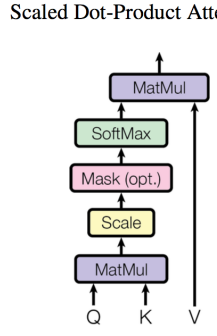


Figure 4: Scaled dot product attention

To simultaneously execute the attention mechanism, query, key and value vectors are bundled up in the respective matrices Q, K and V. Outputs in matrix form are then give by

$$ATTENTION(Q, K, V) = softmax\left(\frac{Q^T K}{d_k}\right) V$$

The above mentioned multi-headed attention is simply a certain number h of attention heads computed in parallel and independently. Therefore, the output consists of h d_k-dimensional value vectors. Since the next transformer block expects just one single vector of size d_k, it needs to be concatenated and linearly projected to fit the next block again. Through downsizing the input vectors to d_k instead of using the full model dimension, it is possible to keep the computational cost on par with a single attention head of size d_{model}. Vaswani et al. (2017) use here an input dimension of 512 and with $h = 8$ follows $d_k = d_{model}/h = 64$. The advantage of using multiple heads consists in the ability of parallel and independent attention mechanisms to learn different dependencies of the sequences at the same time.

2.3 BERT

Devlin et al. (2018) introduced BERT (Bidirectional Encoder Representations from Transformers) as a pre-trained stack of Transformer encoder blocks. The basic architecture is similar to the above mentioned encoder block used by Vaswani et al. (2017) with only minor adjustments for the numbers of blocks L, the number of attention heads A and the input dimensions H. In the paper, two different models are proposed, which are called $\mathbf{BERT_{BASE}}$ and $\mathbf{BERT_{LARGE}}$. They also differ in parameter size due to different values for blocks (L), input dimensions (H) and attention heads (A) with $\mathbf{BERT_{BASE}}$ set to $L = 12, A = 12, H = 768$ giving an overall parameter size of 110M. This choice was motivated by comparison purposes with respect to the OpenAI GPT framework by Radford et al. (2018). $\mathbf{BERT_{LARGE}}$ is set to $L = 24, A = 16, H = 1024$, yielding a parameter size of 340M. Throughout the paper we are going to use $\mathbf{BERT_{BASE}}$ to keep the computational time feasible.

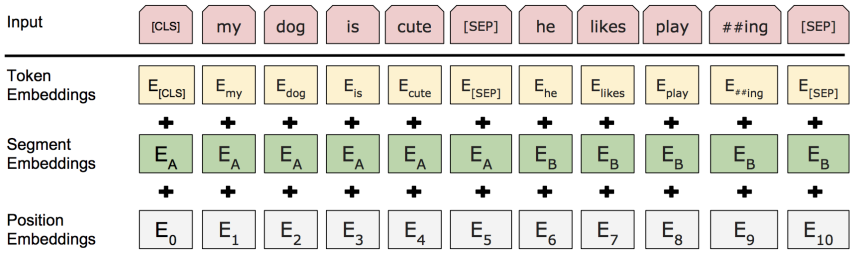


Figure 5: Schematic overview of BERT input

In general BERT takes as input tokens in form of an H-dimensional vector. A schematic overview can be seen in Figure 5. Tokens are according to the graphic the sum of three subtokens that describe one word in a sentence (Token Embedding) as well as the position of the word (Position Embedding) and if more than one input sentence is provided (this is important e. g. in question answering tasks but not important for this paper) to which sentence the word belongs to (Sequence Embedding). If only one sentence is provided, the last token is just the same for all inputs. To create the word embeddings, WordPiece is used. The special tokens `CLS` and `SEP` do not describe words but are used for classification tasks as an output token and the token `SEP` describes the point in the input sequence where one sentece ends and a second one starts. This again is not important for our classification task at hand since only one sentence is provided.

BERT is not the first transformer based NLP model with e. g. the OpenAI GPT framework being released several months prior (Radford et al. 2018). The novelty in BERT lies in its pre-training regimen. Unlike OpenAI, Devlin et al. use a technique called Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), that allows the model in training to train dependencies between words not just in a unidirectional way but bidirectionally. This is according to the paper intuitively

more powerful than just training from left-to-right or right-to-left. Training sets were the BookCorpus by Zhu et al. (2015) and the English Wikipedia with an overall size of 3,300M words.

The Masked LM part of pre-training works by simply masking 15% of tokens of the input sequence which are subsequently predicted by the algorithm. The final output token of the masked word is then fed into a softmax function over the WordPiece vocabulary. Because the masked token can be in the middle of the sequence with the algorithm having simultaneously access to all other words in the sequence, it is possible to form bidirectional dependencies.

NSP works by feeding two sentences into BERT. The second sentence is then to 50% a sentence that directly follows the first one and to 50% a randomly picked sentence. Via the `CLS` classification token BERT is supposed to predict, whether the sentences are immediately connected to understand relationships between sentences. This is important for e.g. Question Answering tasks.

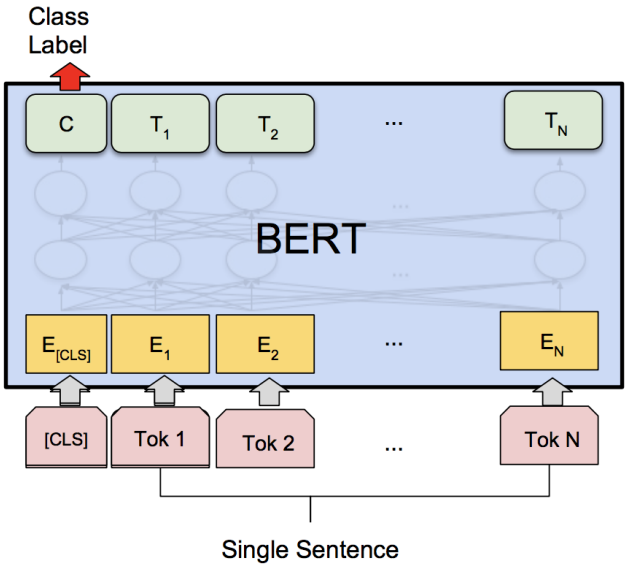


Figure 6: BERT classification process

Fine tuning for single sentence classification tasks is pictured in Figure 6. Inputs are one `CLS`-token and n word tokens belonging to a single sentence of the training data. The output of interest for our task at hand is the transformed classification token `C`, which is subsequently fed into a simple output layer, as suggested Devlin et al. (2018). Afterwards, the loss is calculated with an Adam loss function and weights are updated. The authors also suggest hyperparameter values for fine tuning for the learning rate, the number of epochs and batch size. They are given by a Batch size of 16 or 32, a number of epochs of two to four and a lerning rate of 2e-5, 3e-5, 4e-5 or 5e-5.

2.3.1 Implementation BERT

Proceeding from our research question, in terms of BERT we only modify the data preprocessing by using under- and oversampling. BERT itself is thus always the exact same model implementation and will differ in the end only through different chunks of data for finetuning. Throughout the rest of this section, we will give an overview about the preprocessing we have done and hyperparameter values for BERT.

For the implementation of the fine tuning and prediction process we first had to discriminate between categorical and textual information. Therefore we dropped for BERT all categorical variables and kept only the text of the titles, department descriptions, the company profiles, the job description, the requirement fields, the benefits of the job post and industry and function descriptions. However, we were faced with the problem, that we could not feed all text data into BERT because it has a maximum sequence length of 512 words. Additionally, we decided to cap the maximum input at a length of 200 to keep computation times reasonably low. This left us with the problem, that for example a very exhaustive description of the company in a job ad left us with no space to include other probably also important text fields. On top of that, not every field was used by different ads and therefore we often had inputs consisting of completely different fields. To make the observations more comparable, we subsequently wrote a function that extracted a certain share, loosely based on the average text length of each variable, to capture all fields available. The selected text has text artifacts like line breaks or tabs removed, is set in lower case only and stripped of orthographic signs.

We use cross validation by splitting our data into three chunks, test, training and validation data. We do this in a two stage process, to ensure we test, train and validate all models on the exact same data. By that, we want ensure comparability across models. First, we split our dataset into test and train data and in the second stage we split the train data again to obtain a validation set. This ensures, that we do not end up sampling with different and ultimately different sizes of training data sets.

Table 1: BERT Model - Model overview

Layer (type)	Output shape	No. of Param	Connected to
word ids (Input)	None, 200	0	-
input mask (Input)	None, 200	0	-
segment ids (Input)	None, 200	0	-
BERT layer (BERT)	None, 768	109,482,241	input layers
slicing (-)	None, 768	0	BERT layer
sigmoid (dense)	None, 1	769	slicing
Total parameter	109,483,010		
Trainable parameter	109,483,009		
Non-trainable parameter	1		

Up until now the preprocessing has been identical for all BERTs we implemented. We fork the processing procedure into the undersampling method to obtain UnderBERT, the oversampling method to obtain OverBERT and in direct input into a BERT instance to obtain our baseline model OnlyBERT. It is to note that the pre-trained model as shown in table 2 is the same for all three preprocessing approaches.

For the sampling methods we used functions from the library `imbalanced-learn` (Lemaître et al. 2017). We sampled training and validation data separately. For UnderBERT and OverBERT we wanted to test how the models react if different sample ratios are used for preprocessing. We achieved that by using a loop function to fine tune the respective model with certain sample ratios k times and obtain predictions that we averaged to account for the randomness during sampling. In the case of oversampling, we started with a sample ratio of 1, with the sample ratio being

$$SR = \frac{n_{min}}{n_{maj}}$$

with n_{min} as the (sampled and unsampled) number of observations of the minority class and n_{maj} as the number of observations in the majority class. We then subsequently lowered the ratio in stepsizes of 0.2 and by that reduced the number of oversampled observations of the minority class. In the case of undersampling we used a similar method to account for the random discarding of observations by again fine tuning and obtaining predictions for k models. We started with a ratio of 1 and subsequently lowered it by a stepsize of 0.2 again. The difference to oversampling lies in the sampled quantity, for oversampling we deleted less observations of the majority class to obtain lower ratios. After prediction we simply averaged the resulting confusion matrices.

For hyperparameterization we chose for every model a batch size of 24 (32 worked slightly better, but ran into memory issues) with a learning rate of $2e-5$ and 3 epochs. These values are well in the range of the suggestions given by Devlin (Devlin et al. 2018). Our decision on the learning rate parameter was based on Casula & Tonelli (2020) and Paul & Saha (2020), who chose $2e-5$ as well for text classification tasks. Both papers chose larger numbers of epochs, five and ten respectively, but our model especially with sampling methods showed signs of overfitting when using more than three epochs.

2.4 Categorical Deep Neural Network

The BERT-model we employed only considered the textual entries of the dataset. We created and employed a densely connected deep-learning model to capture information contained in the categorical part of our dataset. This model also allowed us to compare the performance of the transformer-based models with a more basic version of deep-learning. It also allowed us to test various other over- and undersampling techniques because the non-textual datapoints can be transformed over the feature-space.

2.4.1 Feature Selection and Implementation of CAT

To determine which variables had influence on our dependent variable we employed a Pearson's chi-squared-test. The p-values are listed in detail in the appendix. For this we transformed textual and categorical variables into binary, stating only if any

Table 2: Categorical model CAT - Model overview

Layer (type)	Activator-Function	Output shape	No. of Param
x-cat-train (Input Layer)	-	None, 41	0
layer 1 (dense)	relu	None, 42	1764
layer 2 (dense)	relu	None, 42	1806
output (dense)	sigmoid	None, 1	43
Total parameter	3,613		
Trainable parameter	3,613		
Non-trainable parameter	0		

input was given or not. Only the variables of the job location and benefits offered by the employer showed a non-significant influence (p-value >0.1), and thus got discarded. We followed a liberal approach, including more datapoints to increase the information-pool the model can learn from. The variable of job requirements got included because the variance in words used was large and thus including this variable should yield an information gain. The variables with $n > 2$ categories which were the type of employment and the required education and experience for a job got recoded by one-hot encoding them into n distinct dichotomous variables. The dichotomous variables were used as found in the original dataset. The variables for salary range, as well as industry and departmetn of the job offer were recoded to 1, for having an entry and 0 if no entry was present. The textual variables containing information about the company profile, the job description and job requirements were used to feature engineer new variables. The amount of words were counted and the wordcount were normalized to create a new set of variables. For the wordcount the textual data got cleansed of unwanted characters, signs, numbers and stop words.

For hyperparameter optimization we used the 'Talos'-library provided and developed by Mikko Kotila (2019). This library allows the user to run experiments with a range of hyperparameters, defined by the user, that test some set amount of possible combinations to reduce the training and testing phase. It also provides tools, e.g. tables and plots, to evaluate the experiment-runs. Through iterations of experiments the user tries to narrow down on the best performing hyperparameter settings and combinations. To arrive at our final hyperparameter constellation we conducted ten experiments, with in total over 1000 test-runs. The categorical model CAT is a densely-connected deep neural network containing two dense layers with 42 neurons, with a relu-activation function, each. For optimizing the loss-function we employed Adam, designed by Kingma & Ba (2017) with a learning rate of 0.05, which is 5 times the standard setting. The training is conducted in 120 epochs with a batch-size of 16. The output-layer is a one-neuron densely connected layer with a sigmoid activation function. The CAT-model is rather small (3,613 parameter) and thus has a relatively low computational cost in training time and prediction calculation. The oversampling techniques we employed with this model where: Random oversampling, SVMSMOTE, Borderline-SMOTE. Undersampling was conducted using the random undersampling

and the NCR techniques. A combined approach was taken with SMOTEENN. To account for random numerical variation in the sampling techniques and training we conducted 10-runs with each technique to arrive at the measures as stated in the chapter 3 Results.

2.5 Model Ensembling

Model ensembling itself is useful when models capture different aspects of the data. Our approach consists of a BERT instance that works with the text portion of our data and a classical neural network that uses categorical data as input. We therefore think, the main assumption of model ensembling, that the two models describe the data in different ways, is fulfilled and to tackle our prediction problem with all information we have access to, we decided to implement an instance of model ensembling. If ensembling works, then the prediction strength and weaknesses of the ensembled approaches should balance out leading to better predictions overall.

In essence, ensembling as proposed in Chollet (2018) makes use of a weighted average between two or more models. The weights are learned or found by a gridsearch or a simple maximisation algorithm, trained on the validation data set. We opted to implement a simple gridsearch algorithm for two models. Given a certain stepsize this algorithm tests every possible combination of weights and checks whether the prediction on the validation data is better than previous ones. If yes, these weights are kept until either a better one is found or the search has ended.

2.6 Implementation Model Ensembling

For the model ensembling we chose to ensemble all three BERT models with two categorical models in a pairwise fashion leaving us with a total of six ensembled models. We view pairwise ensembling as a reasonable choice here, since ensembling should combine models that capture different aspects of data. Enhancing the amount of models with another Cat or another BERT should therefore not add too much information. We chose as categorical models the RandomUnderCAT and one that utilised the SVMSMOTE-technique. This choice was grounded in the performance of the models. The RandomUnderCAT showed the best detection rate of positive observation while the SVMSMOTE-CAT had the best F1-score of all categorical models.

We split the data in the exact same way as before to have access to the same validation data that were used in the training process as well. Then we loaded the pre-trained and fine tuned model instances and obtained predictions from each model of the validation data. We then constructed a grid of a given stepsize s in $[0, 1]$ and calculated prediction vectors for every possible combination of weights such that

$$PV_{ensemble} = w_i * PV_1 + (1 - w_i) * PV_2$$

with $PV_{ensemble}$ being the prediction vector of the ensembled model and PV_1, PV_2 the prediction vectors of the respective base models. After trying rougher stepsizes of $S = 0.1$ and $S = 0.05$ and noticing, that often no meaningful results emerged, we chose a finer stepsize of $s = 0.005$ leading to 200 individual points. However, this process

can of course be made more accurate. Afterwards we constructed confusion matrices based on the ensembled prediction vectors for every grid point. The grid points with the lowest amount of misclassified observations, expressed by the sum of false positive and false negative classifications, were then identified and the corresponding weights taken as result. One problem we have to mention here, is that with this simple approach only the first best weight value is taken. If successive grid points show the same amount of misclassification, the weight value saved does not change.

3 Results

In this section we first discuss the results obtained by our BERT-based models and then switch to prediction results obtained by CAT-based models. To round things up, we show results from our model ensembling approach. For our results we used the number of misclassifications and mostly the macro-averaged F1 score as success measurements. This score is an average taken over the F1 score for every class considered with the F1 score being the harmonic mean of recall and precision.

$$F1_{macro} = \frac{1}{N} \sum_{i=0}^{N} F1_i$$

This metric is well suited for imbalanced data, since it gives equal weight to both classes.

3.1 CAT Results

In line with previous research, cf. section 2.1, all sampling-techniques increased the detection of samples from the minority class, see Tables 3 and 4. The cost of the sampling techniques, the increase of misclassified observations of the majority class that the techniques cause with respect to the model without sampling, is larger for undersampling techniques, see Table 4 and 5. Oversampling improves the detection of samples from the minority class by a large margin, but it in general came with a large increase in computational resources needed for the model training. The amount of training steps conducted increased by the factor 1.4 for random oversampling with the BERT-model and 1.9, with oversampling techniques on CAT. The oversampling techniques could increase the true positive detection, while having less majority class misclassifications compared to undersampling, while the undersampling approaches show a greater increase in falsely negative classified samples. The macro F1 increases in all oversampling approaches of the CAT model. This is not the case for the undersampling methods, since we also see here sometimes drastic increases in misclassified majority class observations.

The overall highest macro F1 score of all CAT models is reached by the SVMSMOTE-CAT. The best rate of true positive detection is reached by the RandomUnder-CAT, albeit to the cost of very large misclassification of majority observations. Thus these two models got selected to be used in model ensemble.

The combinatory approach only yielded mediocre results; the cost is lower than in the random undersampling techniques but the minority class detection is only slightly better than the rates achieved by the oversampling approaches. The NCR technique showed the least deviation from the baseline model, it could improve the detection rate slightly, while only increasing the cost marginally, yielding no great departure from the baseline CAT model.

Table 3: Results - Categorical model - Oversampling techniques

Sampling Strategy	Without Sampling	Random Over	SMOTE	SVM SMOTE	Borderline SMOTE
True Negative	3363	3127.3	3129.1	3197.5	3167.8
False Negative	32	267.7	265.9	197.5	227.2
False Positive	139	44.9	47.8	50.6	54.9
True Positive	42	136.1	133.2	130.4	126.1
macro F1	0.6523	0.7089	0.7057	0.7376	0.7147
Weighted F1	0.9425	0.9278	0.9273	0.9399	0.9328

Table 4: Results - Categorical model - Undersampling and combination techniques

Sampling Strategy	Without Sampling	Random Under	NCR	SMOTEENN
True Negative	3363	2603.5	3319	3091.3
False Negative	32	791.5	76	303.7
False Positive	139	22.8	127	41.1
True Positive	42	158.2	54	139.9
macro F1	0.6523	0.5723	0.6588	0.6976
Weighted F1	0.9425	0.8352	0.9388	0.9219

3.2 BERT Results

We estimated in total three different BERT-based models. Table 5 shows our results in form of classifications and macro as well as weighted F1-scores. We also estimated the over- and undersampled models for several times, accounting for the randomness of the sampling techniques. The values given in Table 5 are the averaged values. In total, we used five oversampled models, this low amount is due to high computational cost, and ten undersampling models. The baseline BERT-model was also estimated five times on the same data.

Results can be found in Table 5. The baseline model OnlyBERT showed the lowest misclassifications of positive observations. It also had the highest F1-score of all BERT models and thus provides a strong baseline. OverBERT showed on average slightly

Table 5: Results - BERT model

Sampling Strategy	Without Sampling	Random Over	Random Under
True Negative	3386.4	3364.8	3031.2
False Negative	8.6	30.2	363.8
False Positive	30.4	25.0	11.4
True Positive	150.6	156.0	169.6
macro F1	0.9398	0.9208	0.7083
Weighted F1	0.9888	0.9847	0.9181

lower misclassifications of minority observations, but the amount of misclassified majority observations was higher. This is also reflected in an on average slightly lower macro-F1 score. The undersampling models provide an interesting example. The F1-scores are considerably lower than the scores of both previous models, but this stems only from a large amount of majority class misclassifications. In contrast to that, the amount of wrongly predicted minority observations is the lowest of all BERT-models.

Another result we obtained is the predictive behaviour of the models when changing the sample ratios. For the undersampling approach, we estimated five UnderBERTs for each ratio from 0.2 to 1, with later averaging of the resulting confusion matrices for every ratio step. The undersampling approach was estimated with ten OverBERT models per ratio step of which the confusion matrices were averaged as well. The results can be found in table 6 and 7.

Table 6: BERT Undersampling - Comparison of Ratios

Ratio	1.0	0.8	0.6	0.4	0.2
True Negative	3031.2	3134.6	3198.6	3247.3	3285.7
False Negative	363.8	260.4	194.4	147.7	109.3
False Positive	11.4	14	16.4	19.5	17.5
True Positive	169.6	167	164.6	161.5	163.5
macro F1	0.7083	0.7535	0.7889	0.8169	0.8508
Weighted F1	0.9181	0.9374	0.9499	0.9589	0.9679

Here we can see, that the predictive behaviour changed only slightly for the oversampling models when we varied the ratio. The amount of majority class misclassifications decreases slightly while the amount of minority class misclassifications appears to be very similar. We however do get a more pronounced picture when looking at the undersampling models. When increasing the ratio we obtain an increasing number of wrongly classified majority observations and a decreasing number of wrongly classified minority observations.

Table 7: BERT Oversampling - Comparison of Ratios

Ratio	1.0	0.8	0.6	0.4	0.2
True Negative	3364.8	3370	3368.2	3370	3374.4
False Negative	30.2	25	26.8	25	20.6
False Positive	25	27.4	23.4	24.2	26.8
True Positive	156	153.6	157.6	156.8	154.2
macro F1	0.9208	0.9233	0.9276	0.9286	0.9299
Weighted F1	0.9847	0.9853	0.9860	0.9863	0.9866

3.3 Model Ensembling Results

Results for our model ensembling approach can be found in table 8. We ensembled all three BERT-models with two different CAT-models namely an SVMSMOTE approach as well as a random undersampling approach. We chose the cat models due to their solo performance as discussed before. The table gives again an overview of misclassified and correctly classified observations as well as the corresponding F1-scores. Additionally we reported the weights found in the gridsearch we conducted. The weights w_{BERT} are always the weights assigned to the BERT-model with the CAT-model receiving a weight of $1 - w_{BERT}$.

Table 8: Results - Model Ensembly

Employed Models	Employed	Sampling	Technique			
BERT	-	-	Over	Over	Under	Under
CAT	Under	SVMSm.	Under	SVMSm.	Under	SVMSm.
w_{BERT}	0.55	0.51	0.725	0.53	0.615	0.51
True Negative	3383	3385	3394	3383	3186	3310
False Negative	12	10	4	1	209	85
False Positive	29	40	31	31	16	39
True Positive	152	141	150	150	165	142
macro F1	0.9376	0.9210	0.9452	0.9495	0.7802	0.8388
Weighted F1	0.9883	0.9854	0.9898	0.9907	0.9471	0.9672

OnlyBERT ensembled with categorical models did not show any improvements. OverBERT combined with either of the categorical models however, showed considerable reductions in misclassifications of majority observations. Those were also the lowest values we obtained for every model we considered. Subsequently, the macro-F1 score of the UnderBERT combined with SVMSmote-CAT with 0.9495 is the highest we could obtain. A similar picture can be seen for the UnderBERT. In both ensembled models the combination was able to considerably lower the amount of misclassifications of majority observations in comparison to the standalone UnderBERT. This is interesting, since especially the categorical model using undersampling showed the

worst ability to detect and classify majority observations correctly. Another result worth mentioning is, that the weights were in all cases giving BERT-models a higher impact.

4 Conclusion

In this paper we used different sampling methods to evaluate the influence of some preprocessing methods on deep learning approaches and to generate good predictions from an imbalanced dataset. We also used model ensembling with two quite different model specifications to account for our special data structure of mixed textual and categorical data. Our main findings were that for our BERT model sampling methods worked only mediocre. The oversampling procedure did in our view not show any considerable advantages over our baseline OnlyBERT without any preprocessing, especially considering the higher computational cost. Even the different ratios we tried did not change the general model prediction tendencies. The undersampling procedure showed good results, albeit to the cost of greatly increased false negative rates. This is also a result we could find in another study by Weissenbacher et al. (2021), who also found an increased recall for pure undersampling methods. A very good impression was left by our baseline model OnlyBERT, which reached an averaged macro F1 score of 0.9398.

For our CAT-models we can state, that sampling methods in general lead to better minority predictions with the cost of sometimes drastically enlarged misclassifications of majority observations. Finally, we managed to make good predictions and created via model ensembling the only models that were able to beat our baseline OnlyBERT. Therefore, the combination of two deep learning models looking at different aspects of a dataset via model ensembling proved quite successful.

A big drawback of our findings is the missing ability to generalize. We only resampled the data from one single train-test split and did not use more sophisticated methods like k-fold cross validation. Also, we only used an averaged value as evidence and therefore we can not make any valid statements about the variability of our model predictions. Finally, we only used 5 to ten repetitions due to computational limits of Google Colab, our computation platform of choice, further hindering generalisation. Therefore, further research in the field might be conducted via systematic testing of different models using cross validation and significance tests like the Friedman test (Demšar 2006) or cross validated t-tests (Dietterich 1998).

Furthermore, since simple random over- and undersampling did not show unconstrained positive effects, it might be worthwhile to look into different resampling methods. One for us particularly interesting heuristic method consists of using e.g. the `GloVe` word embedding to translate minority text samples and use the results as new samples.

5 Appendix

Table 9: List of independent variables

Variable	Text / Cat
job_id	-
title	Text
location	Text
department	Text
salary_range	Cat
department	text
company_profile	Text
description	Text
requirements	Text
benefits	Text
telecommuting	Cat
has_company_logo	Cat
has_questions	Cat
employment_type	Cat
required_experience	Cat
required_education	Cat
industry	Text
function	Text

Table 10: p values obtained by feature selection for categorical or categorized variables

Variable	Text / Cat
location	0.937
salary_range	0.000
department	0.098
company_profile	0.000
requirements	0.375
benefits	0.505
telecommuting	0.000
has_company_logo	0.000
has_questions	0.000
employment_type	0.005
required_experience	0.000
required_education	0.006
industry	0.125
function	0.288

References

Bansal, S. 2020, [Real or Fake] Fake JobPosting Prediction, https://www.kaggle.com/mranaljadhav/real-fake-job-post, accessed November 15 2020

Batista, G., Prati, R., & Monard, M.-C. 2004, SIGKDD Explorations, 6, 20

Casula, C. & Tonelli, S. 2020, in CEUR Workshop Proceedings, Vol. 2769, Proceedings of the Seventh Italian Conference on Computational Linguistics, CLiC-it 2020, Bologna, Italy, March 1-3, 2021, ed. J. Monti, F. Dell'Orletta, & F. Tamburini (CEUR-WS.org)

Chawla, N., Bowyer, K., Hall, L., & Kegelmeyer, W. 2002, J. Artif. Intell. Res. (JAIR), 16, 321

Chollet, F. 2018, Deep learning with Python (Shelter Island, New York: Manning Publications Co), oCLC: ocn982650571

Davagdorj, K., Lee, J. S., Pham, V. H., Ryu, K. H., et al. 2020, Applied Sciences, 10, 3307

Demšar, J. 2006, The Journal of Machine Learning Research, 7, 1

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. 2018, arXiv preprint arXiv:1810.04805

Dietterich, T. G. 1998, Neural computation, 10, 1895

Han, H., Wang, W.-Y., & Mao, B.-H. 2005, in International conference on intelligent computing, Springer, 878–887

Johnson, J. M. & Khoshgoftaar, T. M. 2019, Journal of Big Data, 6, 1

Kingma, D. P. & Ba, J. 2017, Adam: A Method for Stochastic Optimization

Kotila, M. 2019, Autonomio Talos, http://github.com/autonomio/talos

Last, F., Douzas, G., & Bação, F. 2017, CoRR, abs/1711.00837

Laurikkala, J. 2001, Proc 8th Conf AI Med Eur Artif Intell Med, 63

Lemaître, G., Nogueira, F., & Aridas, C. K. 2017, Journal of Machine Learning Research, 18, 1

Liang, D. 2019, Are BERT Features InterBERTible?, https://medium.com/thelocalminima/are-bert-features-interbertible-250a91eb9dc, accessed February 20 2020

Minaee, S., Kalchbrenner, N., Cambria, E., et al. 2020, arXiv preprint arXiv:2004.03705

Nguyen, H. M., Cooper, E. W., & Kamei, K. 2011, International Journal of Knowledge Engineering and Soft Data Paradigms, 3, 4

Paul, S. & Saha, S. 2020, Multimedia Systems, 1

Pennington, J., Socher, R., & Manning, C. D. 2014, in Empirical Methods in Natural Language Processing (EMNLP), 1532–1543

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. 2018, Improving language understanding by generative pre-training

Vaswani, A., Shazeer, N., Parmar, N., et al. 2017, CoRR, abs/1706.03762

Weissenbacher, D., Rawal, S., Magge, A., & Gonzalez-Hernandez, G. 2021, medRxiv

Wilson, D. & Martinez, T. 2000, Machine Learning, 38, 257

Zhu, Y., Kiros, R., Zemel, R., et al. 2015, in Proceedings of the IEEE international conference on computer vision, 19–27

Stop Removing Stop Words: An Evaluation of Preprocessing Techniques for Twitter Sentiment Analysis with a Deep Learning Approach

Johannes Brachem, Alisa Rothe

1 Introduction

In Twitter Sentiment Analysis (TSA), as in many other machine learning tasks, deep learning techniques using multi-layered neural network architectures have seen increased use in recent years. One of their main advantages, compared to classic machine learning approaches like Support Vector Machines or Maximum Entropy, is their ability to automatically learn to extract relevant features from complex input data like human language Bengio et al. (2013); Tang et al. (2015). This greatly reduces the amount of ingenious data preprocessing required by humans.

Still, it is not entirely obvious, just how far this independence from preprocessing goes. The advent of sophisticated Recurrent Neural Networks (RNN) removes the need for techniques like grouping words into n-grams to capture multi-word patterns. But what about more basic data cleaning operations? A clear consensus on the matter does not appear to exist. While some authors do not report any preprocessing in their deep learning models for TSA Minaee et al. (2019); Huang et al. (2016); Severyn & Moschitti (2015); Chen & Wang (2018), others apply similar techniques as seen in classic machine learning approaches, like lowercasing, stemming or the removal of stopwords Sohrabi & Hemmatian (2019); Ramadhani & Goo (2017); Wazery et al. (2018); Rehman et al. (2019). The usefulness of such techniques has been evaluated in a number of different ways for classic machine learning, but not for deep learning models.

Here, we present an empirical evaluation of the efficacy of the most popular preprocessing techniques in TSA with a Long Short Term Memory (LSTM) model architecture. The rest of this report is structured as follows: In Section 2, we cover the purpose of preprocessing, relevant literature, and briefly explain word embeddings and the main operating principles of LSTM models. In Section 3, we present details concerning the investigated preprocessing techniques, our experiment procedure, and

the structure of our deep learning model. In Section 4, we present the results. In Section 5, we summarise our work and discuss limitations and implications. A link to the code and data used for this report is available in Appendix 6.

2 Theoretical Foundations

2.1 Preprocessing

In 2016, Giachanou & Crestani published a comprehensive review of Twitter Sentiment Analysis methods, including a description of the most important challenges that researchers face in TSA. Among others, the list includes :

- *Incorrect English* – Tweets contain lots of big and small spelling errors and non-standard expressions.
- *Data Sparsity* – In Twitter data, there are often many terms that appear only a few times in hundreds of thousands of tweets, which makes it hard to identify the meaning of these terms.
- *Negation* – Negating words can completely turn around the meaning of a sentence.
- *Stop Words* – Natural language contains lots of words with negligible discriminatory value, leading to an unfavorable signal to noise ratio in the data.

The preprocessing techniques investigated here all constitute attempts to meet these challenges. For example, automatic spelling correction aims at reducing the noise caused by incorrect English, while lowercasing, stemming, and lemmatization reduce data sparsity. But the intended benefit from preprocessing often comes at a price. Some techniques, like lowercasing, remove information that might be meaningful – people often write words in all uppercase to add emphasis. Others, like lemmatization or automatic spelling correction, are imperfect and can remove noise from the data only at the cost of introducing different noise. We elaborate on the pros and cons of the individual techniques in more detail in Section 3.3.

In the machine learning community, some studies have been conducted on the effect of preprocessing on classification accuracy in TSA for non-deep learning approaches. The most comprehensive of these was done by Angiani et al. (2016), using a Naïve Bayes Multinomial (NBM) model. They grouped a number of data cleaning operations together to form a single basic preprocessing step. This basic preprocessing did indeed improve classification accuracy compared to no preprocessing. Additionally, the authors reported that performance could be further improved by individually adding stemming, stop word removal, emoticon regularization or negation regularization to the set of basic techniques, while automatic spelling correction impaired performance. Saif et al. (2014) focused specifically on the removal of stop words, investigating a number of approaches on different datasets, classifed using Maximum Entropy and Naïve Bayes. They found that stop word removal using pre-compiled lists, though very popular, hurts performance. Instead, they recommend to simply remove singleton words. Other authors have taken less detailed approaches, mostly comparing

performance of a number of combined preprocessing techniques with unprocessed data. For example, in the evaluation of Support Vector Machines, Naïve Bayes, and Maximum Entropy Haddi et al. (2013) and Alam & Yao (2019) found that generally, preprocessing increased classification accuracy, although to differing degrees depending on the specific model.

Since the challenges identified above are present for deep learning models just as they are for other machine learning approaches, it appears reasonable to be curious about the degree to which basic preprocessing might affect classification accuracy in deep learning scenarios, possibly boosting their capability to extract meaningful features. For our experiment, we chose a model architecture with a self-trained word embedding layer followed by an LSTM (Long Short Term Memory) layer. LSTMs are one of the most successful and popular deep learning approaches for natural language processing in general, as well as for sentiment analysis in particular (see, for example Karpathy (2015); Tang et al. (2015)), which makes them a good pick for our experiment.

2.2 Word Embeddings

Most deep learning models rely on word embeddings to map a vocabulary onto a dense vector space. The distinct pieces of information in a text are points in a high-dimensional space; each entry in the vocabulary adds another dimension. The value of word embeddings lies in representing ("embedding") these points in a space of much lower dimension. In this space, similarity and nuanced differences between words can be represented by proximity or distance on multiple dimensions.

In practice, the dimensionality can be freely chosen by the researcher, with common values lying between 50 and 300 dimensions.

2.3 LSTM Layer

A key challenge for any machine learning model that is directed at understanding natural language is the evaluation of time-dependent relationships between input values: The ending of a sentence might alter the meaning of previous words, for example. To capture these relationships, Recurrent Neural Networks (RNN) have been developed. The cells of these networks process each token of an input vector (which might represent a single tweet) in sequence, so that the output of previous steps can be used as additional input in later steps. There is also another problem, known as the problem of vanishing gradient. Owing to the technical characteristics of the backpropagation algorithm, the gradient used to update the weights of the network can easily grow too small too quickly, impairing the ability of the network to understand long-term relationships. At this point, Long Short Term Memory (LSTM, Hochreiter & Schmidhuber (1997)) models take the stage, since they provide one of the most successful and most widely used solutions to this issue.

Each LSTM cell step produces an output vector h_t and contains a cell state vector c_t, also known as carry state, where t indicates the current step. When an input vector x_t is processed by an LSTM cell, the information passes through three so-called gates, which are a series of data transformations using trainable weights. Figure 1 provides a

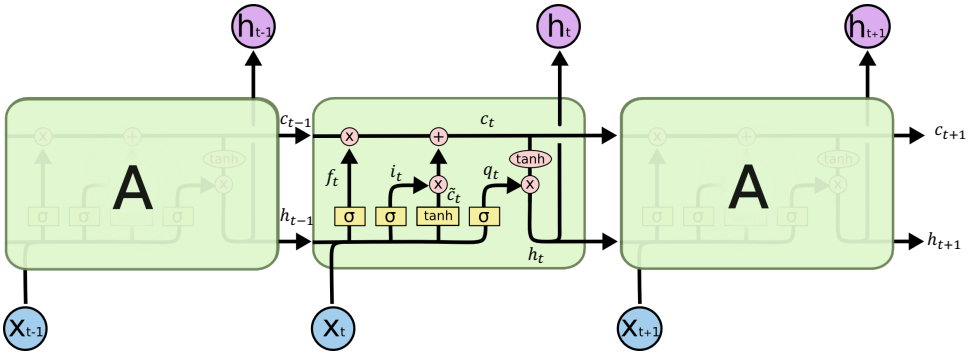


Figure 1: Schematic depiction of an unrolled LSTM unit. The diagram is reproduced from Olah (2015), with added annotations from ourselves.

visualization to accompany the following explanations. The first gate is often referred to as the *forget gate*, where a sigmoid unit is used to obtain values between 0 and 1 based on x_t and the output h_{t-1} of the previous step.

$$f_t = \sigma\left(U_f x_t + W_f h_{t-1} + b_f\right),$$

where σ symbolizes the sigmoid function. These values are multiplied element-wise, indicated by the symbol $\odot$, with the previous cell state c_{t-1}, causing the cell state to forget or retain information in Equation 5. The second gate updates the cell state with information from the current input vector, which is why it is called the *input gate*. It can be viewed as consisting of two parts: A sigmoid unit controls, which values in the cell state will be updated (3), and a tanh unit produces the proposed update values (4). In combination, the forget and input gate update the cell state to c_t (5).

$$i_t = \sigma\left(U_i x_t + W_i h_{t-1} + b_i\right) \tag{3}$$
$$\tilde{c}_t = tanh\left(U_k x_t + W_k h_{t-1} + b_k\right) \tag{4}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t. \tag{5}$$

Finally, the third gate produces the output h_t, which is why it is appropriately called the *output gate*. It is constructed similar to the input gate: A sigmoid unit produces a filter q_t (6), which controls the extent to which values from the tanh-activated cell state will be returned (7)

$$q_t = \sigma\left(U_q x_t + W_q h_{t-1} + b_q\right) \tag{6}$$
$$h_t = q_t \odot tanh(c_t). \tag{7}$$

In the equations above, U_* and W_* refer to the different weight matrices that are trained in an LSTM model and b_* refers to different equation-specific bias terms, which are trained alongside the weights. During backpropagation, the cell state c_*

functions as a pathway that allows the gradient to flow without danger of vanishing overly quickly. The LSTM presentation here is based on Goodfellow et al. (2016) and Olah (2015).

3 Methods

In this section, we describe our testing data, the investigated preprocessing techniques, the experiment procedure, and the architecture of our LSTM model.

3.1 Data

We used the Sentiment140 dataset, which was collected by Go et al. (2009). The authors queried the Twitter API for tweets that contained positive or negative, but not both emoticons like ':)' and ':(' . These emoticons serve as noisy labels, indicating a positive or negative sentiment. To prevent overfitting, they were removed from the data. Further, neutral tweets, as well as repeated tweets and retweets are not included. Altogether, the dataset encompasses 1.6 million tweets from about 660,000 different authors. Half of those tweets are labeled as positive, the other half as negative. Since the data was collected in 2009, the maximum length of a tweet was 140 characters, which implies that rather short sentences were being used – the majority of tweets consists of less than 20 words. Nonetheless, the data contains around 800,000 unique words, but only a small fraction of these, i.e. about 84,000 were used more than five times.

3.2 Data cleaning

There are some data cleaning operations that we apply uniformly in all conditions. First, before any other preprocessing step, we unescape HTML expressions, for example the sequence `&quot;` is turned into double quotes (`"`). After all other preprocessing is done, we remove linebreaks and tabs, as well as the following symbols: `!"#$%&()*+,-./:;<=>?@[\]^_‘{|}~`. Because of the overwhelming number of lone tokens that appear only once in all 1.6 million tweets, we retain only the 10,000 most frequent tokens, based on the recommendation to remove singleton words Saif et al. (2014)[1].

3.3 Preprocessing techniques

To be able to better handle many different techniques, we grouped them into basic and advanced techniques and applied them in two different stages. The grouping is based on a subjective rating of the complexity of the task. An overview of all used techniques is given in Table 1.

Table 1: Overview of pre-processing techniques with examples. Note that we filter out punctuation and symbols like # in all conditions.

Technique	Example (processed version in *italics*)
Stage 1	
Replace mentions	Hello @daniel, check out https://wikipedia.com #wiki *Hello username, check out https://wikipedia.com wiki*
Replace URLs	Hello @daniel, check out https://wikipedia.com #wiki *Hello @daniel, check out url wiki*
Remove Hashtags	Hello @daniel, check out https://wikipedia.com #wiki *Hello @daniel, check out https://wikipedia.com*
Mark Hashtags	Hello @daniel, check out https://wikipedia.com #wiki *Hello @daniel, check out https://wikipedia.com HTwiki*
Normalize Repetitions	I looooove this day *I loove this day*
Lowercasing	I LOVE this day *i love this day*
Normalize Negations	I don't like this day *I not like this day*
Stage 2	
Correct spelling	I liek this day *I like this day*
Remove stop words (a)	I don't like this day *I like day*
Remove stop words (b)	I don't like this day *I not like day*
Stemming	Nice days are good, but nice weeks are better *Nice day are good, but nice week are better*
Lemmatization	Nice days are good, but nice weeks are better *Nice day be good, but nice week be well*

3.3.1 Basic preprocessing techniques

Normalize URLs and mentions.

URLs and @-mentions are used very frequently on Twitter, and we do not expect them to generally aid in detecting the sentiment of a tweet. Still, there might be valuable information in the mere fact that another user was mentioned or that a URL was posted, which is why frequently, mentions and urls are replaced by uniform strings, e.g. *username* and *url*. In our case, most of the original URLs and mentions are filtered out through our word limit, since each specific individual mention or URL

[1] Pre-testing revealed no difference in performance between training on 10,000 and 20,000 tokens.

is quite rare on its own – so this technique is a way of retaining some information that is lost completely otherwise. We test the replacement of URLs and mentions independently from another.

Normalize hashtags.

On Twitter, as on many social media platforms, users can mark their tweets with so-called hashtags, which are keywords preceded by `#`. They are used in various ways, although the most common one is to connect a tweet to a specific topic. Users often concatenate several words into a single hashtag and integrate hashtags seamlessly into their tweets. We investigate three options to deal with hashtags: (1) Remove the `#` symbol and retain the keyword text. This is actually the Stage 1 baseline, since we filter out the `#` automatically in any case. (2) Remove the hashtag completely, including the keyword. (3) Mark the hashtag by replacing `#` with the string `HT`, for example `#hashtag` is turned into `HThashtag`.

Normalize repeated letters.

It is not uncommon to find words with repeated letters in the informal speech used on Twitter. These repetitions might sometimes be spelling errors, but often they are a way of adding emphasis. The underlying problem is, that the number of added letters varies a lot, making the pattern so noisy that our models might have a hard time to extract any meaning from it. So, in order to facilitate the detection of emphasis through letter repetition, we normalize the number of repeated letters to two.

Lowercasing.

The casing of words does carry meaning. For example, the tweet "I've had enough cookies"might be interpreted differently from the tweet "I've had ENOUGH cookies." But for machine learning models, there is a trade-off here, because even two words that differ only in the casing of a single letter are treated as separate tokens from the start, if casing is preserved - which means that the association between them has to be learned in training. It is not obvious, that this added complexity is a price worth paying for the information that can be retrieved from the case of words. For this reason, one of the most widely used preprocessing steps is to transform all words to lowercase, reducing data sparsity.

Normalize negations.

Natural language is littered with different versions of negation - *can't*, *won't*, *hasn't*, etc. There is a lot of variation. One can argue that this variation does not add much information and that it is not indeed necessary for a language model to learn each form of negation. So, the different forms of negation might be replaced with a uniform "not".

3.3.2 Advanced techniques

Automatic spelling correction.

Tweets contain a lot of spelling errors that introduce noise into the data and make it harder for our network to discern the meaning of certain tokens or sentences. So it seems like a sensible idea to consider some form of automatic spelling correction in order to reduce noise. But the noise reduction achieved through automatic spelling correction comes at a price, because corrections can be erroneous. When working with Twitter data, this added noise through spelling correction might be especially pronounced, since people use a lot of internet slang on Twitter. This slang is especially prone to erroneous correction, for example the common acronym "lol", which stands for "laughing out loud", is wrongly corrected to "Lola" by the correction algorithm used here. The question then is, whether the removed noise or the introduced noise carries heavier weight.

We employ the Python library *PyEnchant* Kelly (2020) to generate suggestions of corrected versions for each word. If the change from the raw version to the best suggestion requires changing less than three characters, we use the suggestion. Otherwise, we retain the raw version of the word. This approach was based on a recipe for automated spelling correction provided by Perkins (2014).

Stop word removal.

Some of the most common words in the English language are so-called *stop words*, which means that they are viewed as carrying only negligible amounts of information. This observation quite naturally leads to the idea to simply remove stop words from the data in order to reduce data sparsity. Still, these words do have some function. Often, they give an orientation as to where a sentence is directed.

We use a pre-compiled list of English stopwords provided by the *Natural Language Processing Toolkit* (NLTK; Bird et al. (2009)) Python package for filtering. Because the list of stop words includes many negating terms, we include two conditions of stop word removal. In the naive condition (a), we simply remove stop words as-is, i.e. all stop words including negations are removed. In the negation-preserving condition (b), we first normalize negations by replacing them uniformly with the string "not" and strike "not" from the list of stop words before removing the latter from the data. That way, in (b) negations are preserved and can be detected by our model. Based on the work of Saif et al. (2014) in the context of Naïve Bayes and Maximum Entropy classifiers, we suspect stop word removal to be more likely to harm performance, than to improve it. Still, it is a popular technique and is therefore included in the investigation.

Stemming.

Stemming is the process of removing the endings from words in order to reduce them to their stems. For example, the word "singing" is reduced to "sing". Stemming is supposed to remove noisy complexity from the input data: Instead of learning associations for all possible forms a word, they are learned only for the stem. The

purpose of stemming is not to produce the correct base form or lemma of a word, only to map different forms of a word to the same form. We utilize Porters Porter (1980) stemming algorithm, implemented in the NLTK library. While stemming can reduce data sparsity, it can harm precision. For example, the PorterStemmer reduces all of the following words to "`oper`": `operate, operating, operates, operation, operative, operatives, operational` Manning et al. (2008).

Lemmatization.

While stemming is rather blunt in the sense that it simply removes the ending of words when the algorithm deems it appropriate, lemmatization is a related take on the same task with a more fundamental approach: In lemmatization, words are reduced to their lemma, its most basic form. For lemmatization to work properly, the algorithm needs to know the context in which a word is used. For example, the word "meeting" might be an inflected form of "(to) meet", or a noun describing a gathering. In the first case, the lemma is "(to) meet", while in the second case the lemma is "meeting". For this reason, we need to identify the role of a word in a sentence before we can lemmatize, a process that is known as *part-of-speech-tagging*.

We use the NLTK library for lemmatization, more specifically the default pretrained function for part-of-speech-tagging and the *WordNetLemmatizer* for looking up and inserting the lemmas. Applied to the stemming example from above, lemmatization produces a more nuanced result, returning: `operate, operate, operate, operation, operative, operative, operational`. Still, as Manning et al. (2008) notes, lemmatization is imperfect. For example, it might change the expression "operating system" to "operate system", losing the distinct meaning of the original term. Another possible source of error lies in imperfect part-of-speech recognition, which might be especially difficult with the informal nature of tweets.

3.4 Experiment procedure

Since we examine many different preprocessing techniques, it is not feasible to investigate all possible combinations of the techniques – the number of possible conditions is too great for the scope of this project. For this reason, we take a two-stage approach that is detailed below.

Stage 1.

In Stage 1, we apply each basic preprocessing technique to the cleaned data, creating separate datasets for all techniques. We feed the processed data into our model and compare the performance to the baseline model, which operates on the cleaned data.

Stage 2.

In Stage 2, we start by applying all preprocessing techniques that lead to improved performance in Stage 1, thereby creating a new baseline dataset with basic preprocessing. We then apply each of the advanced preprocessing techniques to this dataset and feed the processed data into our models. We evaluate performance with respect

to both the Stage 2 baseline. Regardless of the efficacy of normalizing negations in Stage 1, we include a condition with normalized negations in conjunction with stop word removal, as explained above.

Sample size.

Some preprocessing techniques that are intended to reduce data sparsity might be much more helpful if the amount of available data is limited. For example, the nuances of meaning contained in the casing of words can only be discovered, if there are enough observations with varying casing present in the data. If that is not the case, casing might effectively be nothing more than noise. In order to make a potential pattern like this discoverable in our experiment, we fit our model to three versions of each dataset, i.e. the full dataset, a subset of 100,000 observations, and a subset of 10,000 observations. We use a single random draw of row numbers to determine which observations to include in the subsamples, which means that all subsamples of one size consist of the same observations being selected from the full data. This ensures comparable results. In the following, we will refer to the different samples as the full sample, the 100k sample, and the 10k sample.

3.5 Model architecture

Structure.

A schematic overview of the model architecture is shown in Figure 2. First, the texts are tokenized and turned into sequences of integers. These sequences serve as input for the first layer of model, the embedding layer. We train our own embedding layer with 150 dimensions to have the embeddings tuned specifically to the preprocessed data and the task at hand. The embedding layer is followed by a single LSTM layer with 16 units. This structure performed well during pretests. We use glorot uniform initialization for the kernel weight matrices U_* Glorot & Bengio (2010) and orthogonal initialization for the recurrent kernel weight matrices W_*. The bias terms b_* are initialized with zeroes, except for the forget gate bias b_f, which is initialized to 1, following Jozefowicz et al. (2015). In training, we use a dropout of 25% after the LSTM layer, but no recurrent dropout within it[2]. Finally, a fully connected layer with sigmoid activation aggregates the information to a single value ranging from 0 to 1. In our case, this value can be interpreted as the estimated probability of positive sentiment.

Training and testing.

Data are randomly split into training and testing data, including 85% and 15% of observations respectively. We train our model using *binary-crossentropy* loss and hold out 15% of training observations for validation in each epoch. We use the RMSprop optimizer with a learning rate of 0.001. We chose RMSprop, because it is

[2] Trials with 50% dropout and 50% recurrent dropout indicated no major differences in results.

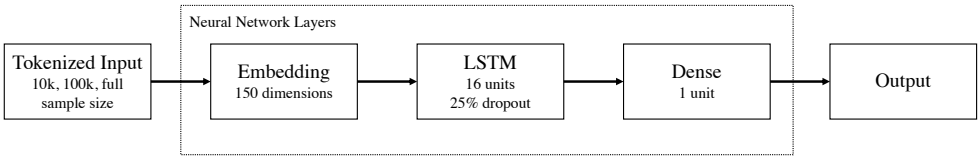


Figure 2: Schematic overview of our neural network architecture. Dropout is applied only in training.

recommended by Chollet (2018) and trial runs with the Adam optimizer indicated no major differences in results. Data is passed into the models in mini-batches of 32 observations. The models are allowed to train for a maximum number of 20 epochs, but will stop training if the loss does not decrease for five epochs in a row. In case of such early stopping, the version of a model with lowest loss will be restored and used for evaluation on the testing data. We report the performance on testing data. Model fit is done using Keras Chollet & Others (2015) as an interface for TensorFlow in Python Abadi et al. (2016).

Performance metrics.

When evaluating the performance of our models, we take both the classification accuracy and the loss into account. The classification accuracy is the share of total observations that were correctly classified. This is an easy-to-understand metric and can be appropriately used in our case, because we have an equal number of positive and negative observations. The loss is binary crossentropy, which measures the distance between the predicted probabilities and the actual classes. The loss offers a little more nuance. Take, for example, two tweets labelled as positive. If a model outputs a probability of 0.15 for the first, and 0.49 for the second, both will simply count as wrongly classified in the accuracy metric. The loss, on the other hand, will indicate that the first prediction was worse than the second one. So, because we care about both the easily interpretable classification accuracy and the more nuanced loss, we take both metrics into account. For the decision about which preprocessing techniques from Stage 1 to use in Stage 2, we favor nuance and thus rely on the loss.

4 Results

In the following, we describe the experiment results individually for each stage of the experiment.

4.1 Stage 1

The results are displayed in Table 2. With increasing sample size, model performance increased and variation between models fitted on differently pre-processed data decreased.

Table 2: Accuracy and loss for Stage 1 and Stage 2 models. The column headings *10k*, *100k*, and *full* indicate the sample size in the respective model. Preprocessing techniques in italics were used in the baseline for Stage 2. Rows are sorted in ascending order based on loss in models with full samples. In stop word removal, (a) refers to the naive condition and (b) to the negation-preserving condition.

		Accuracy (%)			Loss		
	Technique	10k	100k	full	10k	100k	full
Stage 1							
b, c	*Lowercasing*	74.9	79.1	81.7	0.526	0.445	0.406
b, c	*Normalize Mentions*	74.5	79.0	81.3	0.528	0.450	0.413
b, c	*Normalize Repetitions*	74.6	79.6	81.1	0.538	0.443	0.415
	Baseline 1	76.0	78.6	81.2	0.499	0.454	0.416
b	Normalize Negations	74.8	79.2	81.2	0.540	0.451	0.416
	Remove Hashtags	74.0	78.3	81.2	0.518	0.454	0.416
b	Mark Hashtags	72.1	78.8	81.2	0.562	0.451	0.417
	Normalize URLs	73.2	78.2	81.1	0.542	0.458	0.417
Stage 2							
b, c	Spelling Correction	70.9	80.2	82.1	0.546	0.431	0.399
	Baseline 2	72.8	79.5	81.8	0.546	0.441	0.403
a, b	Stemming	73.0	79.8	81.9	0.556	0.436	0.404
a	Lemmatization	75.4	79.1	81.7	0.512	0.449	0.406
a	Remove Stop Words (b)	75.1	77.3	79.5	0.512	0.473	0.440
	Remove Stop Words (a)	71.9	71.4	75.4	0.549	0.556	0.500

[a] Lower loss than baseline model in run with 10,000 samples
[b] Lower loss than baseline model in run with 100,000 samples
[c] Lower loss than baseline model in run with full sample

When the full sample was included, lowercasing the data, replacing mentions and normalizing repeated characters lead to reductions in loss compared to the baseline. The biggest reduction was achieved by lowercasing: Loss was reduced by 0.01, i.e. from 0.416 to 0.406. The reduction in loss was also reflected in an increase in accuracy by 0.5 percentage points, from 81.2% to 81.7%. For all other techniques, the differences to the baseline data were very small: The second best model loss was only 0.003 below baseline, and the worst loss was only 0.001 above baseline. Such small differences might even be attributable to minimal variations in the optimization process.

In the 100k subsample, the performance-increasing preprocessing techniques included all three of the above, with the addition of normalized negations and marked hashtags. The distinct performance benefit of lowercasing observed in the full sample was not present in the same way in this subsample. Instead, the biggest reduction in loss was achieved by normalizing repetitions.

In the smallest subsample, including only 10,000 tweets, not a single preprocessing technique lead to increased performance. In fact, the baseline was ahead of the next best technique, lowercasing, by 1.1 percentage points in accuracy.
We might draw some additional insight from a look at the fitting histories, which are displayed in Figure 3. First, as can be expected, we see a lot of overfitting in both experiment conditions with reduced sample size and virtually none in the full sample condition. Second, the variation in performance between preprocessing techniques shrinks with growing sample size to such a small amount, that the models are practically indistinguishable in the full sample condition on the given scale.

4.2 Stage 2

In Stage 2, we can identify a more pronounced hierarchy of preprocessing techniques from looking at the results in Table 2. The removal of stop words was clearly harmful to the performance of our models in the full sample and 100k subsample. The difference in loss between naive stop word removal and the Stage 2 baseline amounted to 0.097, which is reflected in a difference of a full 6.4 percentage points lower accuracy for naive stop word removal, both in the full sample condition. Even when negations were preserved, stop word removal still fared notably worse than the baseline, with a difference of 0.037 in loss that was reflected in a 2.3 percentage points lower accuracy of negation-preserving stop word removal compared to the Stage 2 baseline.

For the remaining three techniques, stemming, lemmatization, and automatic spelling correction, the differences in performance compared to the Stage 2 baseline in both the full sample and 100k observations conditions were small. In the full sample, only automatic spelling correction led to a reduction in loss with regard to the baseline, with the loss difference of 0.004 amounting to a 0.3 percentage points increase in accuracy.

In the smallest subsample of 10k observations, neither naive nor negation-preserving stop word removal were clearly distinguishable from the remaining techniques or the Stage 2 baseline in terms of loss or accuracy. In fact, negation-preserving stop word removal even led to a lower loss and higher accuracy than the baseline, the differences being 0.034 in loss and 2.3 percentage points in accuracy. This made negation-preserving stop word removal the second best preprocessing technique in the 10k condition, beaten only by lemmatization. Note however, that no Stage 2 model lead to better performance than the Stage 1 baseline in the 10k subsample.

A look at the fitting history in Figure 3 confirms our observations. Stop word removal, especially when negations are not preserved, clearly harmed performance in both bigger-sample conditions. The naive stop word removal condition was the only one that did not trigger an early stop during the 20 training epochs. Stemming, lemmatization, and spelling correction were practically indistinguishable from the baseline, especially in the full sample and 100k conditions.

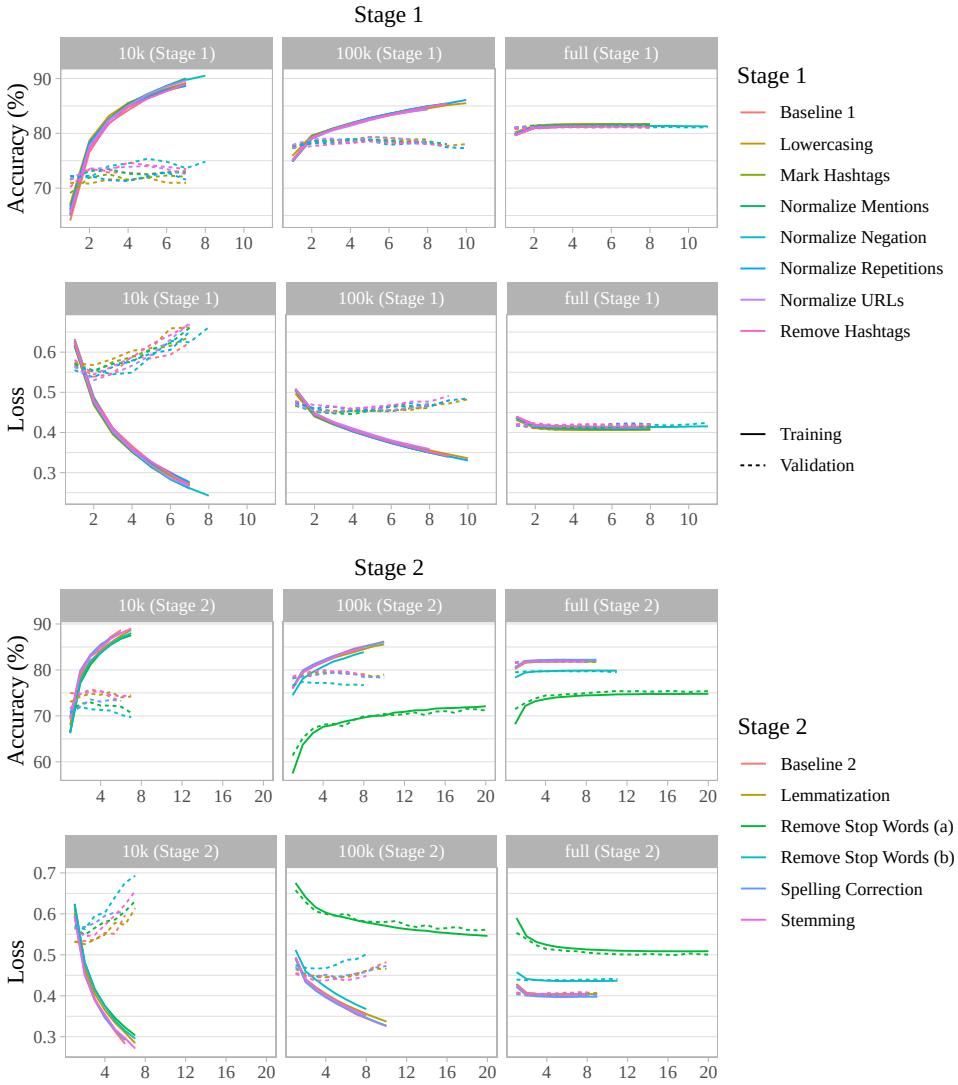


Figure 3: Fitting history for Stage 1 and Stage 2 models. The columns show the data for the three sample sizes, 10k, 100k, and full sample. In stop word removal, (a) refers to the naive condition and (b) to the negation-preserving condition.

5 Discussion

Our goal was to investigate the effectiveness of different preprocessing techniques for a deep learning approach to Twitter sentiment analysis. In a two stage plan, we compared the loss and accuracy achieved by a neural network with self-trained word embeddings and a single LSTM layer on different sized data sets. In the first stage, we evaluated lowercasing, normalization of negations and repeated letters, replacement of URLs and mentions, and the removing or marking of hashtags. The second stage included all techniques that led to an improvement in Stage 1, supplemented by more advanced preprocessing techniques such as spelling corrections, stop word removal, lemmatization and stemming.

Classification accuracy of models fit to preprocessed data either remained on a similar level as the baseline, increased slightly or even diminished. The removal of the stop words seemed to have a particularly negative impact, especially the removal of negations. The latter caused a drop of 6.4 percentage points in classification accuracy compared to baseline in the full sample condition. On the other hand, small benefits can be drawn from lowercasing, mention and repetition normalization, and automated spelling correction. The combination of these four techniques improved classification accuracy by 0.9 percentage points in the full sample condition.

Notably, sample size made a difference: In the 100k subsample, the group of beneficial preprocessing techniques contained the ones mentioned above with the addition of negation normalization, hashtag marking, and stemming. In the 10k subsample, not a single preprocessing technique lead to a better classification accuracy than the Stage 1 baseline. The largest influence of sample size was independet of preprocessing techniques: More data, as would be expected, lead to better performance. A lack of sample size was not compensated by preprocessing.

In conclusion, our results indicate first and foremost that stop word removal significantly hurts the performance of deep learning models for Twitter sentiment analysis. Lowercasing, the normalization of mentions and repeated letters, and automatic spelling correction in combination lead to a small improvement in accuracy, while other techniques showed neither notable improvement nor notable harm.

6 Experiment Code and Data

All code and data used for this study are available from `https://owncloud.gwdg.de/index.php/s/PPyAGV6AJoYp2K0` (password: `dl2021`). The files are available in a compressed archive. The file size is quite large, because we created many different datasets and included all of them for reproducibility. Please refer to the file `README.md` for detailed guidance on how to navigate the files.

References

Abadi, M., Barham, P., Chen, J., et al. 2016, in Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (Savannah, GA: USENIX), 265–283

Alam, S. & Yao, N. 2019, Computational and Mathematical Organization Theory, 25, 319

Angiani, G., Ferrari, L., Fontanini, T., et al. 2016, in Proceedings of the 2nd International Workshop on Knowledge Discovery on the Web, Cagliari, Italy

Bengio, Y., Courville, A., & Vincent, P. 2013, IEEE Transactions on Pattern Analysis and Machine Intelligence, 35, 1798

Bird, S., Klein, E., & Loper, E. 2009, Natural Language Processing with Python (Sebastopol, CA: O'Reilly)

Chen, N. & Wang, P. 2018, in Proceedings of the 5th International Conference on Cloud Computing and Intelligence Systems (CCIS) (Nanjing, China: IEEE Press), 684—-687

Chollet, F. 2018, Deep learning with Python (Shelter Island, New York: Manning Publications Co), oCLC: ocn982650571

Chollet, F. & Others. 2015, Keras (Version 2.4.3) [Computer software], `https://keras.io`

Giachanou, A. & Crestani, F. 2016, ACM Computing Surveys, 49

Glorot, X. & Bengio, Y. 2010, in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), Vol. 9 (Sardinia, Italy: Journal of Machine Learning Research), 249–256

Go, A., Bhayani, R., & Huang, L. 2009

Goodfellow, I., Bengio, Y., & Courville, A. 2016, Deep Learning (MIT Press), 785

Haddi, E., Liu, X., & Shi, Y. 2013, Procedia Computer Science, 17, 26

Hochreiter, S. & Schmidhuber, J. 1997, Neural Computation, 9, 1735

Huang, M., Cao, Y., & Dong, C. 2016

Jozefowicz, R., Zaremba, W., & Sutskever, I. 2015, in Proceedings of the 32nd International Conference on Machine Learning (Lille, France: Journal of Machine Learning Research)

Karpathy, A. 2015, The unreasonable effectiveness of recurrent neural networks [Blog post]

Kelly, R. 2020, PyEnchant (Version 3.2.0) [Computer software], `https://pyenchant.github.io/pyenchant/`

Manning, C. D., Raghavan, P., & Schütze, H. 2008, Introduction to information retrieval (Cambridge, United Kingdom: Cambridge University Press)

Minaee, S., Azimi, E., & Abdolrashidi, A. A. 2019

Olah, C. 2015, Understanding LSTM networks [Blog post]

Perkins, J. 2014, Python 3 text processing with NLTK 3 cookbook, 2nd edn. (Birmingham, United Kingdom: Packt Publishing)

Porter, M. F. 1980, Program: electronic library and information systems, 14, 130

Ramadhani, A. M. & Goo, H. S. 2017, in Proceedings of the 7th International Annual Engineering Seminar (Yogyakarta, Indonesia: IEEE)

Rehman, A. U., Malik, A. K., Raza, B., & Ali, W. 2019, Multimedia Tools and Applications, 78, 26597

Saif, H., Fernandez, M., He, Y., & Alani, H. 2014, in Proceedings of the 9th International Conference on Language Resources and Evaluation, (European Language Resources Association), 810–817

Severyn, A. & Moschitti, A. 2015, in Proceedings of the 38th International Conference on Research and Development in Information Retrieval (Santiago, Chile: Association for Computing Machinery), 959–962

Sohrabi, M. K. & Hemmatian, F. 2019, Multimedia Tools and Applications, 78, 24863

Tang, D., Qin, B., & Liu, T. 2015, WIREs Data Mining and Knowledge Discovery, 5, 292

Wazery, Y. M., Mohammed, H. S., & Houssein, E. H. 2018, in Proceedings of the 14th International Computer Engineering Conference (Cairo, Egypt: IEEE)

Preprocessing in Sentiment Analysis with Twitter Data

Kim Sarah Meier, Henrike Meyer

1 Introduction

Natural Language Processing (NLP) is defined as a research area that studies how natural language texts can be understood and manipulated by computers. Sentiment Analysis (SA) as a part of NLP is the task of finding a judgement, feeling or opinion in language, text, tweets or data based on document, sentence or aspect level. Predicting the polarity of a given sentence is one of the basic tasks in sentiment analysis. The goal is to find out whether a sequence of words is meant to express a positive or negative opinion, which is a binary classification task. Due to the increasing use of microblogging websites such as Twitter or Facebook, the amount of data that can be analyzed through SA is growing rapidly. Thus, the potential of financial benefit gains accelerating attention in companies. SA offers to monitor opinions in real time and use the knowledge for their own benefit. (Feldman 2013; Go et al. 2009; Effrosynidis et al. 2017; Agarwal et al. 2011; Bengio et al. 2015; Kharde & Sonawane 2016)

In the literature, deep neural nets - especially using long-short term memory (Lstm) layers - are frequently used for sentiment analysis. For example, Wazery et al. (2018) found that a network with three Lstm layers outperformed Naive Bayes, Decision Tree, K- Nearest Neighbor and Support Vector Machine classifiers on three different datasets. Chandra & Jana (2020) also showed that a deep neural network model with Lstms obtained better results than many machine learning approaches such as (Multiple) Naive Bayes Classifier, Bernoulli Classifier, Logistic regression and more.

An aspect that is less paid attention to is the question whether and how much preprocessing of the input data affects the performance of Lstm classifiers. In this paper, we will take a closer look at different preprocessing techniques and their success in improvement of an optimized model's classification performance.

The remainder of this paper is organized as follows: In Section 2, a descriptive analysis of the dataset and its characteristics is conducted. The different preprocessing techniques, network architecture as well as training and validation process are presented in Section 3. Afterwards, the results are shown and discussed in Section 4 and in Section 5 the present work is summarized.

2 Twitter Data

The dataset sentiment140 [1] consists of nearly 1,600,000 tweets extracted from the Twitter API by Go et al. (2009). Every two minutes in the period of April 6th 2009 to June 25th 2009 a query for ":)" and ":(" was executed which each retrieved 100 tweets. Therefore, the tweets do not have a specific topic in common. Originally, the dataset consists of 6 columns: "target" indicates the polarity of the tweet. In "ids", "date", "flag" and "user", descriptive attributes of each tweet can be found. These will not be taken into account because they do not serve our purpose. In "text", the original text of the tweet is saved. Emoticons are already completely stripped from the data. In the data retrieval, tweets with the emoticons ":)", ":-)", ":)", ":D", and "=)" were assigned a positive label ("target" = 4). A negative sentiment ("target" = 0) was allocated to tweets containing ":(", ":-(" and ": (". After retrieval, the text of the tweets was already processed in the following way: All emoticons listed above were stripped off. Tweets that comprised both positive and negative emoticons were removed from the dataset. If a tweet contained a retweet, it was also removed. This was done in order to not put enhanced importance to the retweeted tweet, which may happen because the text of the retweeted tweet is copied into the new tweet. Additionally, tweets with ":P" were removed. The objective of this was to counteract Twitter's API that returns tweets with ":P" as a result of a query of ":(". The authors argue that tweets with ":P" do not usually imply a negative sentiment. Lastly, duplicates of tweets were removed for the same reason as retweets. After the preprocessing, 800,000 tweets of each sentiment were taken over into the sentiment140 dataset available on kaggle (Go et al. 2009).

When taking a closer look at the data we discovered that not all duplicates were discarded. Additionally, other problematic tweets which needed handling in the preprocessing stood out. These irregularities are further discussed in Section 3. The direct mapping of emoticons to the target sentiment of the tweet results in noisy labels, which will be considered in Section 4.

Before data preprocessing, the dataset included tweets written by 659,775 different users. In 2009, Twitter restricted the maximum number of characters per tweet to 140. The tweets have an average word count of 13 with a minimum of one word in a tweet and a maximum number of 64 words. The distribution of word counts per tweet can be seen in Figure 1. Only those word counts that occurred in the dataset are displayed. In Figure 2, the most common words after the advanced preprocessing are shown, which means that so-called stopwords were already removed (see Section 3.1 for details). Mostly positive words can be found in the positive labelled tweets, e.g. "good" and "love". Unexpectedly the word "not" is the most used word in positive and negative labelled tweets. One might not directly associate it with positive sentiment but negations are commonly used in both type of tweets for different reasons. The placeholder "laughing" used in the advanced preprocessing for replacing laughter is the second most frequently used word in positive labelled tweets. Non-surprisingly, "bad_word" – a placeholder for all swear words – is frequently used in negative labelled tweets.

[1] `https://www.kaggle.com/kazanova/sentiment140`

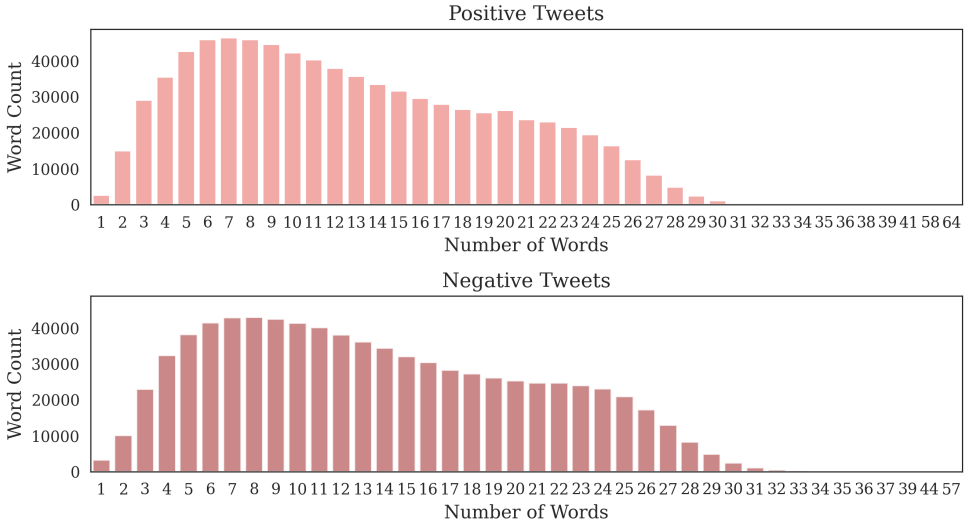


Figure 1: Number of words in tweets separated by sentiment

In the course of the preprocessing, the number of tweets was reduced for different reasons which are further discussed in the Section 3.1. An overview of the numbers can be found in Appendix 6.1. Even after reducing the number of tweets the dataset is sufficiently large and also balanced for further evaluation.

3 Methods

In order to train a good binary classifier, three different preprocessings were executed, each with more effort. These three types of preprocessing were compared based on the performance of one model with similar neural network architecture.

3.1 Preprocessing

There are a lot of acronyms, spelling mistakes and emoticons in the data. Also, mentions (@user) and hashtags (#topic) are included. It has been shown that pre-processing a dataset extensively affects a model's performance noticeably. Subsequently, the choice of different sentiment analysis models can have a rather small effect on the performance (Goularas & Kamis 2019; Angiani et al. 2016; Effrosynidis et al. 2017). For the present analysis, so called basic, advanced and stemming preprocessing techniques were conducted.

Basic Preprocessing

The basic preprocessing includes the necessary neutralization of the text to make information extraction for the neural network possible. The following steps were conducted:

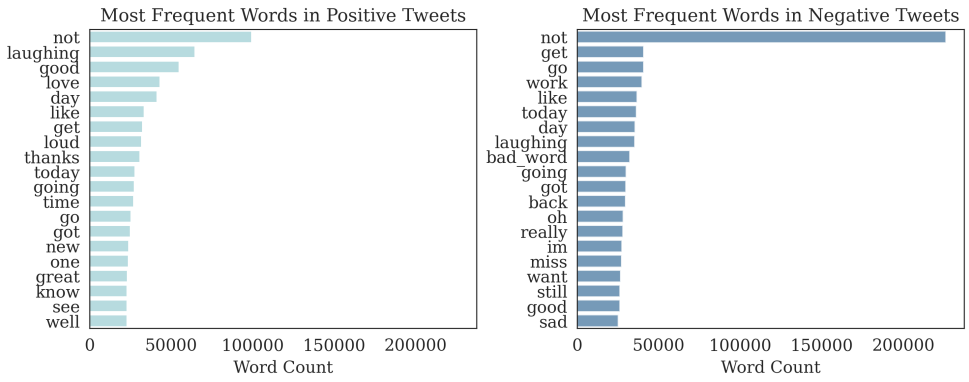


Figure 2: Most common words in tweets separated by sentiment

- Removal of duplicates: Although the dataset was supposed to be free from duplicates, we still found more than 18,000 duplicates and removed them from the dataset. Otherwise, unwanted emphasis could lay on duplicate tweets or a duplicate could unwillingly be part of the training as well as the testing data and therefore, the classifier would be trained on something it is tested with afterwards.
- Removal of redundant spaces: Additional whitespace is removed for further processing of the tweet.
- Deleting URLs, mentions (@username) and all punctuation.
- Removal of Stopwords: These were downloaded from the *nltk* package[2] by Bird et al. (2009). This includes words like "not", "doesn't" and "couldn't", which could be problematic with regard to the negation that is removed from the tweet.
- Lowercase: All tweets were converted to lowercase.

There was no further selection in terms of acronyms or handling of negation. The original dataset includes 18,545 acronyms and 248,247 negations which point to the necessity of further data processing. These and other challenges are addressed more extensively in the advanced preprocessing.

Advanced Preprocessing

Here, the following steps were executed in addition to the basic preprocessing:

- Hashtags: The sign # of hashtags were removed, but the word behind it was kept. We did not see significant differences between keeping and deleting hashtag words and therefore decided to keep them. This is beneficial in case the hashtag was used within a sentence and contributes to the meaning of the sentence (e.g. in "Our #car did not #start today").

[2] `https://github.com/nltk/nltk`

- Laughter: All possible compositions of "haha", e.g. "hahaahahah", "hhhaahahahh", "aahahha" were replaced by the word 'laughing' in order to enable the algorithm to identify them as the same token.
- Repeated characters: Each time the same character would be repeated at least 3 times in a row, the sequences was replaced by three repetitions of that character. This also has the aim to facilitate identifying e.g. "eeeeeewwww" and "eeeeeewww" as the same token.
- Acronyms: Commonly known acronyms were replaced with their meaning. Thus, "ASAP" was replaced by "as soon as possible".
- Negations: All negations were replaced with "not". This allows the algorithm to identify negations in a more reliable and consistent way.
- Insults: Insults and swearwords were substituted by "bad_word" with the same aim of identifying similar associations.
- With regard to the removal of stopwords please note the following: This was conducted after replacing acronyms in order to delete stopwords also from sequences that took place instead of their acronyms. In the above example, "ASAP" would eventually become "soon possible", since "as" is a stopword. As already mentioned above, "not" is in the list of stopwords in the *nltk* package. Since we wanted to keep it due to the power it has to change a sentence's meaning, "not" was removed from the list of stopwords and therefore remained in the tweets.
- Language and other symbols: Although the tweets were supposed to be only in English, there were many tweets with cryptic characters which were presumably written in another language. Additionally to those tweets, we decided to remove the words "amp", "quot" and "lt", since they are remnants of former signs (e.g. "<").

Lists of the acronyms, swearwords and negations can be found under `https://github.com/hengoe/SA_cleaners_new/blob/master/clean_words.py`. The advanced preprocessing was mainly based on Effrosynidis et al. (2017) and Angiani et al. (2016).

Preprocessing with Stemming

Based on the advanced preprocessing, an additional step was taken here to prepare the available data with the aim to enable better performance of the classifier.

- Stemming: This is a technique of reducing each word to its stem in order to further normalize the data. The suffix of the word is stripped to reduce it to the basic meaning of the word. E.g. "connecting" and "connected" are both reduced to "connect". Therefore, similar words can be interpreted in the same way and similar information can be retrieved by the classifier.

The algorithm was introduced by Porter (1980). It is a difficult task and results may vary from real words as it can lead to inappropriate stems. But overall, stemming is useful to improve interpretation of similar words. In this analysis the Snowball Stemmer from the *nltk* python tool kit by Bird et al. (2009) is used with the language

setting English. Additionally a word vector representation is trained on the stemmed tweets as the word stems might not occur in a pretrained vocabulary. This vector representation is then used as the weights of an embedding layer in the model.

Two exemplary tweets of the dataset can be found in Table 1 and 2 with the respective output after performing each preprocessing. The first tweet in Table

Table 1: Example Tweet 1

Original Tweet	@mialuna1 LMAOOOOOO !! omg hahaha!!! yeah yeah yeah! I DIDN'T GET A MESSAGE !!
Basic	lmaoooooo omg hahaha yeah yeah yeah get message
Advanced	lmaooo oh god laughing yeah yeah yeah not get message
Stemmed	lmaooo oh god laugh yeah yeah yeah not get messag

2 showcases the deletion of mentions, stopwords and punctuation, the lower case setting for basic preprocessing and additionally the transformation of acronyms for the advanced preprocessing. Here, the different handling of "not" can also be seen. The stemming shows the reduction to "messag" which isn't ideal. The second reference

Table 2: Example Tweet 2

Original Tweet	Studying for my biology final is killing me sleep isn't really an option when this thing is on monday afternoon. Fuck #finals!
Basic	studying biology final killing sleep really option thing monday afternoon fuck finals
Advanced	studying biology final killing sleep not really option thing monday afternoon bad_word finals
Stemmed	studi biolog final kill sleep not realli option thing monday afternoon bad_word final

tweet in Table 2 indicates the keeping of hashtag words but deletion of the hashtag sign. The advanced preprocessing substitutes insults with "bad_word". Furthermore, the stemming technique can be observed where words are no longer necessarily part of the dictionary (e.g. "studi").

After these differently extensive preprocessings, some tweets were reduced to an empty string. When taking a closer look at the respective tweets it became clear that these were mainly tweets were the user only mentioned another user. Apparently, these tweets were often used when users wanted to make their friends aware of something. Since there is no logical sentiment in those empty tweets, they were removed from

the dataset. The amounts of remaining tweets for each preprocessing separated for positively and negatively labelled tweets compared to the original data can be found in Table 4.

After the preprocessing and prior to running the classifier, different techniques were applied to the data in order to make it processable for the network. Thus, the text needs to be converted into numeric tensors, which is called vectorizing. Text-vectorization is a process of applying tokenization and afterwards allocating numeric vectors to the generated tokens. In the present work, the vectorizing was applied on word level. We decided to use word embeddings, where dense, low-dimensional vectors are learned from data. As already mentioned above, we decided to both use pretrained word embeddings as well as training embeddings ourselves (Chollet 2018). After converting the sequences of words into sequences of indexes representing unique words with the aid of keras' tokenizer, we applied padding. The sequences of different length, meaning different number of words in the tweets, were padded to the maximum tweet length seen in training data. This means, that the sequences were filled up with zeros until they all had the same length. These numeric tensors are the tweet representation that was fed into the neural networks.

3.2 Network Architecture

Using Lstm layers for sentiment analysis is a widely used approach (Gulli & Pal 2017; Chandra & Jana 2020; Jalaj 2017). Lstm cells are a special form of recurrent neural networks which exploit their input's sequential nature. In these types of input, such as time series or text, an element's occurrence in the sequence depends on other elements that came up prior to it. (Gulli & Pal 2017)

The neural network used in the analysis of the described dataset consists of four different layer types. The model was implemented using the keras framework by Chollet & Others (2015).

Embedding layer

This first layer of the network can be described as a dictionary that allocates the unique indices for words to dense vectors representing their meaning. Similar words or rather their indices are assigned similar dense vectors. These dense vectors can either be learned from scratch or pre-trained word embeddings can be used.

Using trained embeddings is advantageous in speeding up the training process of the model. This is due to setting the starting weights of the embedding layer in accordance with the pre-trained vectors. The relationship between words therefore is already established and the pre-trained weights can be used from the beginning of training. (Chollet 2018) In our advanced preprocessing model pre-trained word vectors from the Global Vector for Word Representation (GloVe) by Pennington et al. (2014) were incorporated. These word vectors are particularly trained for Twitter data. The used word vectors are of dimension 100 and have been trained on a dataset of 2 billion tweets resulting in an available vocabulary of 1.2 million words.

For comparison of performance custom word vectors for the vocabulary of the dataset were calculated in the stemming model with *Word2Vec* using the implementation by Řehůřek & Sojka (2010). Moreover, this was necessary because stemming did not allow the use of pre-trained word vectors since many words were not to be found in the dictionary anymore (e.g. "messag"). The word vectors were calculated using the continuous skip-gram model, an unsupervised learning technique to classify words based on other words in that sentence (Mikolov et al. 2013). The window of words used for calculation was set to five. The obtained word vectors have dimension 100. All words in the dataset are used for building the vocabulary. The size may vary between trainings as it depends on the used training data.

Long short-term memory network

The Lstm unit is built to handle sequential data with long term dependencies and was first introduced by Hochreiter & Schmidhuber (1997). Feedback connections between its own layers are used to keep information received previously. Each Lstm cell consists of an input, output and a forget gate which regularize what information is important to save for further analysis. For each part of a sequence an Lstm cell is used to evaluate and hand over the information from that word of the tweet. The advantage of Lstm units is their safekeeping of word dependencies. Furthermore, the use of multiple Lstm layers improves the performance of the system. (Hochreiter & Schmidhuber 1997; Goularas & Kamis 2019)

Dropout layer

To avoid co-adaption of neurons in the Lstm layer and therefore the extraction of similar (hidden) features in the data the dropout layer is used. A fraction of the layer's neurons are randomly chosen and set to zero. The remaining neurons are upscaled to remain the same sum value for the output. This enables the neural network to work effectively even in settings in which certain information is not available. Therefore, it is an effective opportunity to avoid overfitting of the model (Buduma & Locascio 2017).

Sigmoid output layer

For the transformation at the end of the network a fully connected dense layer with a sigmoid activation function is used to receive a label for the tweet belonging to the respective class (0: negative, 1: positive). The sigmoid function

$$f(z) = \frac{1}{1 + e^{-z}}$$

maps values on a range from zero to one. The resulting value represents the probability of the tweet belonging to class 1 (Buduma & Locascio 2017). Here, this refers to a positive sentiment of the tweet.

The network architecture that was applied in the present work is visualized in Figure 3. The input layer and an embedding layer are followed by two sets of Lstm- and dropout layers. After that, a fully connected dense layer returns one value between zero and one, which can be interpreted as the probability of a positive sentiment.

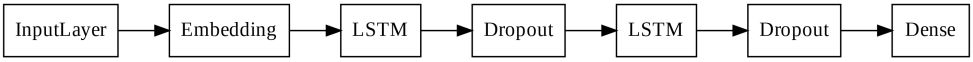


Figure 3: Network Architecture

The updating of the weights of the neural network in the course of the model training is also called loss minimization. For this optimization task a loss function and an appropriate optimizer have to be chosen. Optimization of the model is performed with keras' implemented Adam optimizer which is a stochastic gradient descent based optimizer. Adam was introduced by Kingma & Ba (2014) as a computationally efficient, little memory using optimizer which is invariant to diagonal rescaling of gradients. Additionally, it tackles problems with noise and sparse gradients and also large data or parameter problem solving tasks can be resolved. Therefore, Adam is a popular optimizer for a wide range of problems (Gulli & Pal 2017).

The loss function used for this two-class classification task is binary cross-entropy

$$BCE\ (p,t) = -t\log(p) - (1-t)\log(1-p)$$

with t is the target and p as the resulting probability calculated by the classifier (Gulli & Pal 2017). This function is used as it measures the difference between the predicted probability distribution and the ground-truth distribution (Chollet 2018, Chapter 3). The pre-implemented keras binary crossentropy function was used for the process of optimization to update the weights during model training to optimize the sentiment prediction.

3.3 Training and Validation Process

For training and testing purposes, the dataset was split up into 10% testing and 90% training and validation data. The training and validation sets were used to train the model, while the test set was used for the final evaluation in order to avoid overfitting on the validation set as mentioned in Chollet (2018, Chapter 4.2). The different models' performance on the testing data is presented in Section 4.

In order to measure the goodness of a model, accuracy is determined as the most important measure of the correctness of a classifier and as well as precision, sensitivity and F1 score are assessed as seen e.g. in Kharde & Sonawane (2016); Angiani et al. (2016). They are defined as follows

$$\begin{aligned} Accuracy &= \frac{TP+TN}{TP+TN+FP+FN}, \\ Precision &= \frac{TP}{TP+FP}, \\ Sensitivity &= \frac{TP}{TP+FN}, \\ F1\ score &= 2 * \frac{Precision * Sensitivity}{Precision + Sensivity}, \end{aligned}$$

with TP as the number of true positives, TN as true negatives, FP as false positives and FN as false negatives. The purpose of this project task is a binary classification of positive or negative sentiment for each tweet. For this reason accuracy seems to be

the most probable metric as it accounts for the correctly predicted labels out of the total amount of predicted tweets. It does not capture unequal misclassification costs of the different labels, which is not needed in the present work anyways. Precision and sensitivity denote the relevance of selected items for multilabel or ranked classification and therefore are less important for this project task. The F1-score is a summarizing metric of model performance. Since we decided for the accuracy to be the measure to pay attention to primarily, the binary accuracy was tracked throughout the training and validation process. Additionally, the value of the loss function as well as the amounts of correctly and incorrectly as positive and negative classified tweets were monitored over the epochs of training. From these, a confusion matrix was built to evaluate the classification (Gulli & Pal 2017).

Cross validation was used to evaluate the stable performance of the trained models. In the course of a training process, the 90% training data from above was randomly split into three parts in order to perform a 3-fold cross validation. After each epoch, the model's performance on the training as well as the evaluation data were assessed. Model hyperparameter choices were examined based on averaged performance of the model of each fold.

In the first fitted models, the training accuracy increased in a much steeper way than the validation accuracy and the curves intersected after few epochs of training for each fold of the cross validation. A model that is performing better on the training than on validation data is assumed to be overfitting. Therefore, different regularization techniques were applied.

Firstly, we observed a very steep learning curve when we fitted deep models without regularization. Thus, we introduced dropout layers after each Lstm layer and thereby flattened the curve without decreasing the performance.

In addition to that, the learning rate was reduced when the training accuracy did not improve anymore. For this, keras' ReduceLROnPlateau was used with a factor of 0.1 and a minimum learning rate of 0.01. This means, that the new learning rate will be calculated by multiplication of the old learning rate with 0.1, but will not sink below 0.01 in total.

Since we still noticed overfitting of the models within the first few epochs of training, we decided to implement a L1 regularizer. Thereby, a cost is added to the loss function, which is proportional to the absolute value of the weight coefficients (Chollet 2018). Consequently, the curve of training accuracy flattened recognizably and drew closer to the validation accuracy more slowly without intersecting. But the model did not accomplish accuracies as high as the model without L1 regularization were able to. Also with a grid search over different parameters, no significant improvement could be achieved. Furthermore, the training epochs needed for acceptable accuracies increased from approximately 5 to 7 to much over 30, which slowed down the training process significantly. Thus, we decided to remove the L1 regularization from our model.

In order to stop the training process in the moment overfitting occurs, an early stopping mechanism was introduced in the model. Since we did not observe typical characteristics of overfitting such as a halted improvement of the validation loss close to the moment when the training accuracy outperforms the validation accuracy (see Figure 4), keras' pre-implemented EarlyStopping did not serve our purpose. Instead,

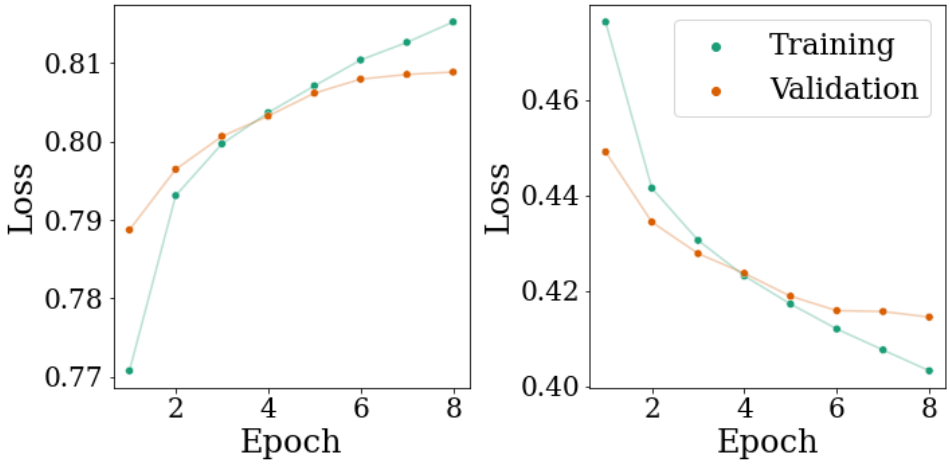


Figure 4: Custom Early Stopper. Training without Early Stopping

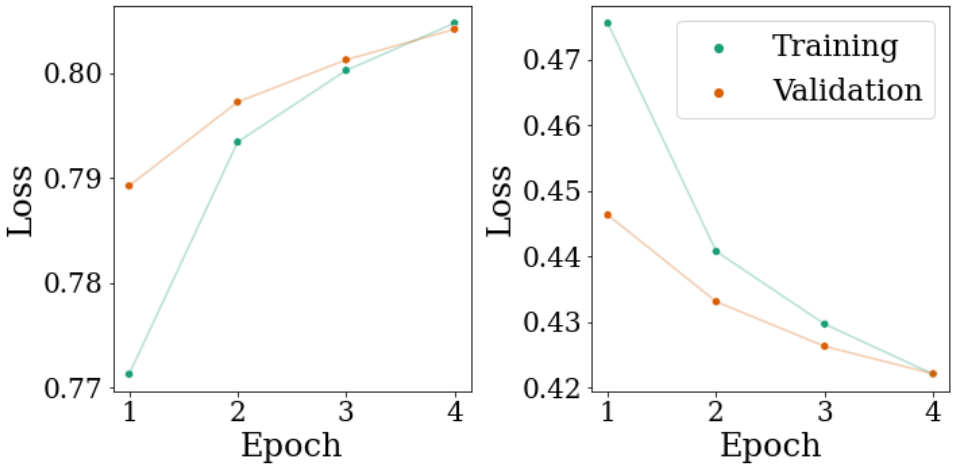


Figure 5: Custom Early Stopper. Training with Early Stopping

we introduced a custom early stopper, that ends the training as soon as the training accuracy exceeds the validation accuracy. The model is saved after this epoch. The effect of the implemented custom early stopper is displayed exemplary in Figure 5. Here, the early stopper terminates training after the fourth epoch as the model performs better on the training set than on the validation set. The model is saved after the fourth epoch for further evaluation.

The training length is determined by two implementation choices. Within each cross validation fold, the model was trained up to 15 epochs, whereas the amount of epochs represents how many times the model is exposed to the training set. With each iteration through the set, the weights are adjusted by the optimizer so that the crossentropy loss function is minimized. As already mentioned above, an early stopper was implemented and therefore most of the models that we considered were trained for approximately 5 to 8 epochs only. Therefore, the training length may differ between models.

4 Results

As already mentioned above, we tried different preprocessing techniques. In this section, we present the performance of the different models on the testing data of 160,000 tweets and point out differences to the baseline model.

Per trial and error, multiple combinations of Embedding, Lstm and Dropout Layers were tried. For the final network architecture as seen in Figure 3 the best results were obtained. Thus, we decided to optimize the selection of hyperparameters for this model and different preprocessing techniques respectively.

To achieve the best performance a random search was conducted for which the results were inconclusive. Therefore, a gridsearch for special parameter combination was performed which led to better results. In Appendix 6.2, the different grid searches conducted for the model are displayed. This was done with the aid of the *sklearn* package by Pedregosa et al. (2011). For each preprocessing technique the performance for the same hyperparameter combinations was evaluated. All parameter combinations were examined with a 3 -fold cross-validation and in the Tables 5, 6 and 7, the mean values of the folds are displayed. All in all, we saw very stable results in between the different folds, which we interpreted as a sign that the chosen model is valid. It is clear to see that within one preprocessing technique, the results did not vary much between different parameter combinations. The best model was chosen based on the best mean accuracy in relation to the complexity of the model. The training progress for the best models can be seen in Appendix 6.3.

For each preprocessing the best model was chosen and then tested on the testing set. In Table 3, the performance of these best models is presented. In particular, the models' accuracy, precision, sensitivity and F1-score can be seen. For each preprocessing one model from training during cross validation was chosen based on balanced correct classification and accuracy. Additionally, a baseline model for performance comparison was trained and tested in the same manner (see below for specifications of the baseline model). A steady improvement in performance can be observed from the baseline model to basic, advanced and stemming preprocessing.

Table 3: Performance comparison

	Baseline	Basic	Advanced	Stemming
Accuracy	0.5714	0.7903	0.8002	0.8022
Precision	0.5594	0.7890	0.7967	0.8129
Sensitivity	0.6673	0.7917	0.8054	0.7844
F1-score	0.6086	0.7903	0.8010	0.7984

Baseline Model

As the baseline model, we decided for a model based on the basic preprocessing. The network consists of an embedding layer with random initial weights, a simple recurrent neural network layer with 32 units and a dropout layer with a dropout rate of 0.2, followed by a dense layer with a sigmoid activation function. To avoid overfitting, we additionally implemented an early stopper as for the other models. Although we included helpful features for a text classification task, the baseline model's performance on the testing data was rather poor. The simple design of the baseline model does not seem to fit the data appropriately. In Table 3 and in the confusion matrix (see Appendix 6.4 Figure 13) it is clear to see that the simple structure of the baseline model is not sufficient. Therefore the model structure described before in Section 3.2 is used for the training of models for the preprocessing.

Basic Preprocessing

In Table 5, the results of a grid search for the basic model's parameter finetuning can be seen. Because the results did not vary much between the different parameter combinations, we decided for model Basic 1. It stands out with a high validation accuracy in combination with the smaller Lstm size. Generally, when an easy and a more complicated model perform similarly, it is recommended to decide for the easier one (Chollet 2018; Gulli & Pal 2017).

In the chosen model, there is a clear improvement with regard to the baseline model in all metrics. Therefore, the results justify the chosen model structure. Improvement was achieved by setting pre-trained initial weights for the embedding layer, using Lstms as a special form of recurrent neural networks in combination with dropout layers and stacking two sets of the latter after one another.

In the confusion matrix (see Figure 6), it is clear to see that the portion of false-positives and false-negatives is balanced. Also, the relation of true-positives and true-negatives seems very even. To summarize, the basic classifier is able to predict the true sentiment for positive and negative tweets nearly equally accurately.

In general, setting the initial weights of the embedding matrix with the aid of the pretrained GloVe-embedding as well as introducing two Lstm layers resulted in a significant improvement of performance. With the aid of various regularization techniques as described in Section 3.3, training the model to a high validation accuracy without overfitting was achieved.

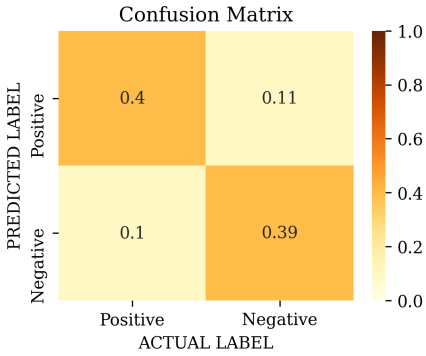


Figure 6: Confusion Matrix for the best Model using Basic Preprocessing

Advanced Preprocessing

The results of the parameter tuning grid search for the advanced preprocessing can be seen in Table 6. As for the basic preprocessing, the parameter finetuning does not affect the validation accuracy significantly. We decided for model Advanced 12, where the accuracy of predictions for the testing data reached 0.8002. This is an improvement to the basic preprocessing model of around 0.01. For the other metrics, we also see clear improvements with regard to the baseline as well as to the basic preprocessing model.

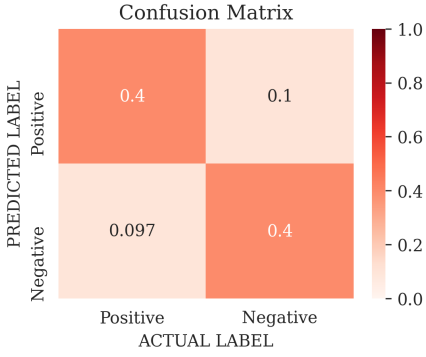


Figure 7: Confusion Matrix for the best Model using Advanced Preprocessing

Similar to the basic model the confusion matrix for the advanced preprocessing is well-balanced and can be seen in Figure 7. In comparison with the basic confusion matrix the percentage of misclassified tweets is slightly less. In summary, the trained advanced classifier can evaluate positive and negative sentiment equally accurately.

Although the effect was rather low, we could show that preprocessing input data better can result in an improvement of performance. In the course of the advanced preprocessing, approximately 18,000 acronyms were replaced by their meaning and

about 248,000 negations were turned into “not”. We suspect that with more application possibilities, the effect of the advanced preprocessing might have been even more noticeable in the performance evaluation.

Preprocessing with Stemming

For the model that was based on preprocessing with stemming, the grid search results can be found in Table 7. The Stemming 8 model performed best with an average training accuracy of 0.8067 and validation accuracy of 0.8054. Here, an improvement can be seen with regard to the advanced model in accuracy and precision, but a decrease in sensitivity and F1-score. The model does not seem to be able to detect positive sentiment as well as the advanced model but still performs better than the basic and baseline models.

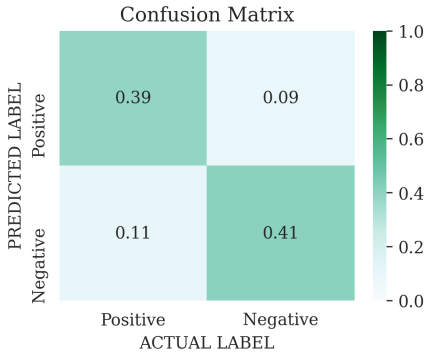


Figure 8: Confusion Matrix for the best Model using Preprocessing with Stemming

In the model’s confusion matrix, no significant differences can be seen in comparison with the basic and advanced model. The stemming classifier seems to be slightly more successful in predicting negative labels than the advanced classifier but is less successful in determining positive sentiment. With more extensive preprocessing the model was able to perform better regarding the chosen metric accuracy, meaning the right predictions of the sentiment of tweets.

In Figure 9, one can see the distribution of probabilities the different models predicted for the positive sentiment. For all three models, the plots look very symmetrical. As it already seemed in the confusion matrices, it does not look like one sentiment is predicted better or worse than the other.

Fortunately, the upper graphs for correctly classified tweets show a U-shape that indicates that the classifier works and for many tweets, is pretty certain about the tweet’s sentiment. For the basic preprocessing, the U shape is flatter than for the advanced and stemmed models, which means that the latter two assigned more extreme values to the tweet. The basic model labelled tweets more moderately which leads to the conclusion, that although the difference in metrics of basic, advanced and stemmed models were not remarkable, there is a clear improvement in predictions for the more complex preprocessing techniques.

In the bottom graphs, it is visible that a flat but undeniable peak of misclassified tweets had a probability around 0.5, which shows that the classifier was not very certain about the decision. But the tails of the lower graphs are not as thin as it would be desirable. A possible explanation for this could be the noisy labels of the data.

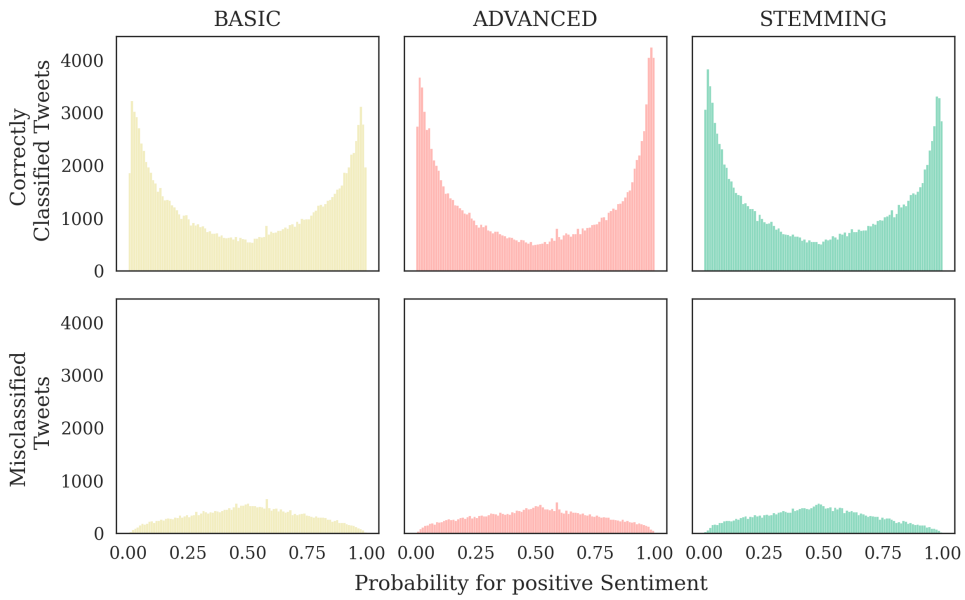


Figure 9: Classification of Test Data

When evaluating the results of all models it should be taken into account that the original labelling of the data was conducted with regard to the emoticons used in the tweets. This resulted in noisy labels, because this approach ignores the possibility of irony. For example, a tweet "My key just broke in the lock :) :) :)" would be classified as positive although the emoticons are clearly meant ironically and the intended sentiment of this tweet would most certainly be negative. In addition, the present binary classification does not take into account a neutral sentiment. It is a rather limited way to classify tweets as not every message of a tweet necessarily can be sorted into one of those two classes. Also it does not represent real-world applications, as those tweets should not be ignored (Go et al. 2009).

For the basic and advanced preprocessing model, a small irregularity can be seen in the upper graphs: With regard to the bars around, there is a slightly higher bar at approximately 0.59. When taking a closer look, we figured that this is due to a specific characteristic of tweets in the testing data. When a tweet exclusively or mostly contains words that the tokenizer does not know, the prediction of the model seems to be the same. This would for example be the case if a tweet "@otherUser, ewwww you about to??" would, with the removal of stopwords, punctuation and the mention, be reduced to 'ewww'. The tokenizer does not know the word and

assigns an out-of-vocabulary token. All tweets consisting of exclusively one word that was not learned by the model will after tokenization consist of exactly the same out-of-vocabulary token. Therefore, it is not surprising that the same sentiment will be assigned to these tweets.

The irregular pattern cannot be observed for the stemmed classifier. This is a result of the trained global word vectors based on the training data for each model. As described before, the continuous skip-gram algorithm was used. It conducts word vectors based on the words surrounding the current word. Therefore the trained model is able to predict the sentiment for the words in the vocabulary more efficiently and unknown tokens are better handled by the classifier.

For a baseline model, the original work related to the dataset of Go et al. (2009) could also be considered. The authors applied three machine learning algorithms (Naive Bayes, Maximum Entropy, and SVM) with the feature extractors unigrams, bigrams, unigrams and bigrams, and unigrams with part of speech tags. The authors reported accuracies between 78.8 and 82.7. But they generated the data their classifiers were tested on differently than the training data. Additionally, the sentiments for their testing data were manually assigned instead of the automatic labelling on the basis of emoticons that was found in the training data. This prevents a meaningful comparison with our results. Therefore, further comparison and discussion of their results is omitted.

5 Conclusion

For the sentiment140 dataset three different types of preprocessing were conducted and evaluated with appropriate validation and evaluation techniques. A grid search for optimal hyperparameter tuning was performed. Each preprocessing's best model was chosen and evaluated based on suitable metrics. The resulting classifier for different types of extensive preprocessing show an increase in accuracy. Therefore it can be concluded that more effort in preprocessing leads to better modelling results. In this dataset, evaluation based on the target labels needs to be handled with care as the noisy labels might distort the correct classification results. Lastly, the limitations of this binary classification need to be taken into account, which do not account for a neutral level of sentiment.

In conclusion, the type of extensive preprocessing should be chosen depending on the area of application. A thorough preprocessing can improve the deep learning model's performance remarkably. For faster training it is more suitable to use pre-trained word vector models for the embedding layer instead of training them on the data as the results do not seem to be significantly better.

6 Appendix

6.1 Number of Tweets

This table shows a more detailed segmentation of tweets after further analysis of the dataset. Non-Unique tweets refers to the found duplicates in the dataset. After performing preprocessing some tweets came out empty, if e.g. the deletion of mentions was done to a tweet which only contained one. The available tweets were used for training.

Table 4: Number of tweets

Type of Tweet	Label	Original	Basic	Advanced	Stemmed
Non-Unique	Positive:	8,719	8,719	8,719	8,719
	Negative:	9,815	9,815	9,815	9,815
Empty	Positive:	0	3,372	3,374	3,374
	Negative:	0	3,198	2,900	2,900
Available	Positive:	791,281	787,909	787,907	787,907
	Negative:	790,184	786,986	787,284	787,284
Total		1,599,999	1,599,999	1,599,999	1,599,999

6.2 Grid Search

The results of the hyperparameter grid search are represented in the following tables with regard to each preprocessing technique.

6.3 Model Training

Each figure in this section gives an overview about the training performance of the respective preprocessing. The implemented early stopper was a precaution for overfitting and stopped training as soon as the training set performed better than the validation set.

6.4 Baseline Model Results

The confusion matrix of the baseline model shows evident problems in the classification. Particularly, positive labelled tweets are classified more inadequately than negative ones.

Table 5: Basic Preprocessing

Preprocessing	Batch Size	Dropout Rate	LSTM Size	Loss	Accuracy	Val. Loss	Val. Accuracy
BASIC 1	256	0.3	64	0.44	0.7937	0.44	0.7930
BASIC 2	256	0.3	128	0.44	0.7938	0.44	0.7930
BASIC 3	256	0.5	64	0.44	0.7924	0.44	0.7917
BASIC 4	256	0.5	128	0.44	0.7927	0.44	0.7919
BASIC 5	512	0.3	64	0.44	0.7919	0.44	0.7913
BASIC 6	512	0.3	128	0.44	0.7942	0.44	0.7928
BASIC 7	512	0.5	64	0.44	0.7933	0.44	0.7915
BASIC 8	512	0.5	128	0.44	0.7931	0.44	0.7926
BASIC 9	1024	0.3	64	0.45	0.7907	0.44	0.7903
BASIC 10	1024	0.3	128	0.44	0.7940	0.44	0.7926
BASIC 11	1024	0.5	64	0.45	0.7913	0.44	0.7906
BASIC 12	1024	0.5	128	0.44	0.7941	0.44	0.7933

Table 6: Advanced Preprocessing

Preprocessing	Batch Size	Dropout Rate	LSTM Size	Loss	Accuracy	Val. Loss	Val. Accuracy
ADVANCED 1	256	0.3	64	0.42	0.8046	0.42	0.8028
ADVANCED 2	256	0.3	128	0.42	0.8055	0.42	0.8044
ADVANCED 3	256	0.5	64	0.43	0.8040	0.42	0.8030
ADVANCED 4	256	0.5	128	0.42	0.8038	0.42	0.8029
ADVANCED 5	512	0.3	64	0.43	0.8032	0.42	0.8028
ADVANCED 6	512	0.3	128	0.42	0.8053	0.42	0.8044
ADVANCED 7	512	0.5	64	0.43	0.8033	0.43	0.8027
ADVANCED 8	512	0.5	128	0.42	0.8039	0.42	0.8037
ADVANCED 9	1024	0.3	64	0.42	0.8038	0.42	0.8030
ADVANCED 10	1024	0.3	128	0.42	0.8038	0.42	0.8031
ADVANCED 11	1024	0.5	64	0.43	0.8040	0.42	0.8029
ADVANCED 12	1024	0.5	128	0.42	0.8056	0.42	0.8047

Table 7: Stemmed Preprocessing

Preprocessing	Batch Size	Dropout Rate	LSTM Size	Loss	Accuracy	Val. Loss	Val. Accuracy
STEMMING 1	256	0.3	64	0.42	0.8042	0.42	0.8036
STEMMING 2	256	0.3	128	0.42	0.8058	0.42	0.8051
STEMMING 3	256	0.5	64	0.43	0.8023	0.43	0.8013
STEMMING 4	256	0.5	128	0.42	0.8054	0.42	0.8044
STEMMING 5	512	0.3	64	0.43	0.8014	0.43	0.7984
STEMMING 6	512	0.3	128	0.42	0.8052	0.42	0.8047
STEMMING 7	512	0.5	64	0.43	0.8037	0.43	0.8022
STEMMING 8	512	0.5	128	0.42	0.8067	0.42	0.8054
STEMMING 9	1024	0.3	64	0.42	0.8044	0.42	0.8036
STEMMING 10	1024	0.3	128	0.42	0.8033	0.42	0.8028
STEMMING 11	1024	0.5	64	0.42	0.8047	0.42	0.8035
STEMMING 12	1024	0.5	128	0.42	0.8051	0.42	0.8032

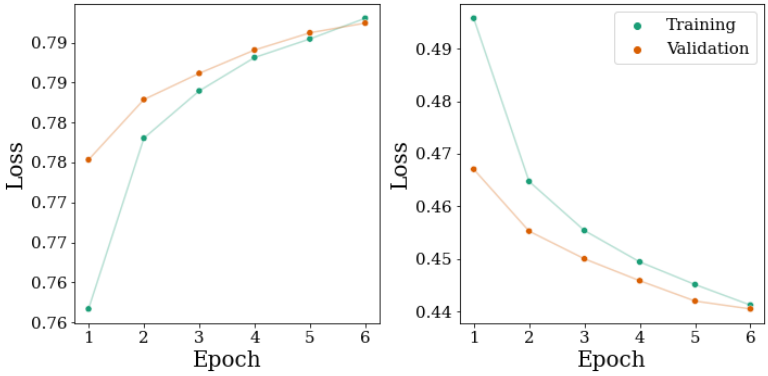


Figure 10: Basic Model 1 CV-Fold 3

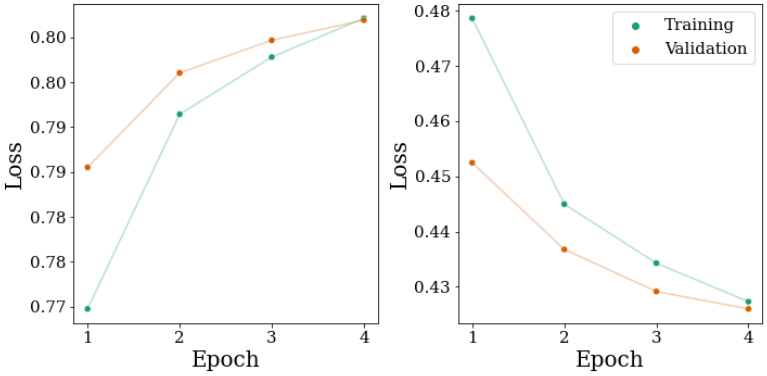


Figure 11: Advanced Model 12 CV-Fold 2

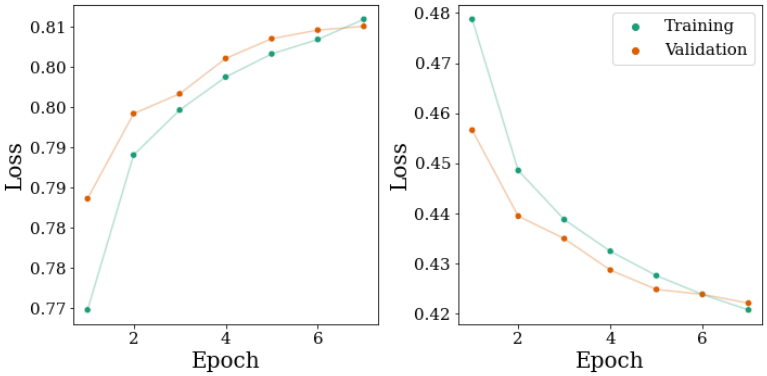


Figure 12: Stemming Model 8 CV-Fold 2

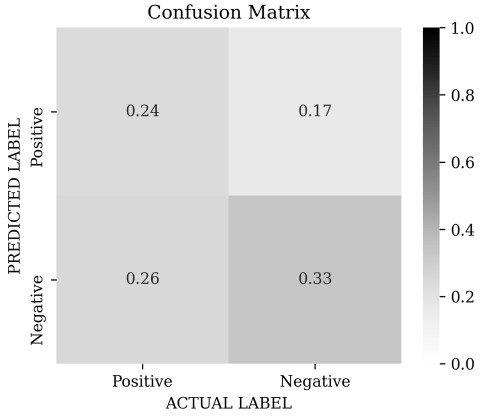


Figure 13: Confusion Matrix for the Baseline Model

References

Agarwal, A., Xie, B., Vovsha, I., Rambow, O., & Passonneau, R. 2011, Proceedings of the Workshop on Language in Social Media (LSM 2011), 30–38
Angiani, G., Ferrari, L., Fontanini, T., et al. 2016, in KDWeb
Bengio, Y., Goodfellow, I., & Courville, A. 2015, Deep Learning
Bird, S., Klein, E., & Loper, E. 2009, Natural Language Processing with Python
Buduma, N. & Locascio, N. 2017, Fundamentals of deep learning: Designing next-generation machine intelligence algorithms (" O'Reilly Media, Inc.")
Chandra, Y. & Jana, A. 2020, in 2020 7th International Conference on Computing for Sustainable Global Development (INDIACom), 1–4
Chollet, F. 2018, Deep Learning with Python (Manning Publications Co.)
Chollet, F. & Others. 2015, Keras (Version 2.4.3) [Computer software], `https://keras.io`
Effrosynidis, D., Symeonidis, S., & Arampatzis, A. 2017
Feldman, R. 2013, Communications of the ACM, 56, 82
Go, A., Bhayani, R., & Huang, L. 2009, CS224N project report, Stanford, 1, 2009
Goularas, D. & Kamis, S. 2019, in 2019 International Conference on Deep Learning and Machine Learning in Emerging Applications (Deep-ML), 12–17
Gulli, A. & Pal, S. 2017, Deep learning with Keras (Packt Publishing Ltd)
Hochreiter, S. & Schmidhuber, J. 1997, Neural computation, 9, 1735
Jalaj, T. 2017, Python Natural Language Processing (Packt Publishing, Limited)
Kharde, V. A. & Sonawane, S. 2016, International Journal of Computer Applications, 5
Kingma, D. P. & Ba, J. 2014, arXiv preprint arXiv:1412.6980
Mikolov, T., Chen, K., Corrado, G., & Dean, J. 2013, arXiv preprint arXiv:1301.3781
Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, Journal of Machine Learning Research, 12, 2825
Pennington, J., Socher, R., & Manning, C. D. 2014, in Empirical Methods in Natural Language Processing (EMNLP), 1532–1543
Porter, M. F. 1980, Program
Řehůřek, R. & Sojka, P. 2010, in Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks (Valletta, Malta: ELRA), 45–50
Wazery, Y. M., Mohammed, H. S., & Houssein, E. H. 2018, in 2018 14th International Computer Engineering Conference (ICENCO), 177–182

Coronavirus tweets NLP - Text Classification

Malte Kramer, Claire Reiß

1 Data

The dataset contains 41157 tweets used for model training and 3798 tweets for model testing. The tweets were collected over the period of four days (12.03.2020 until 16.03.2020). It is not further specified how the tweets were collected. The data leads us to believe that the tweets have been selected based on coronavirus specific hashtags. This is a very common approach to collect data from Twitter using the Twitter API. The tweets were hand labeled based on their tonality. Five sentiments were used for this: {extremely negative, negative, neutral, positive, extremely positive} we coded as {0, 1, 2, 3, 4}. Containing five sentiments allows to pay attention to the ordinal structure of the data. We decided to additionally recode the sentiments to {negative, neutral, positive} we coded as {0, 1, 2} where the extremes are combined with their matching weaker sentiments. This is done to reduce the complexity of the problem and to find out how the lower sentiment count influences the prediction. Both sentiment-codings are shown in table 1. The observed normalized sentiment counts are shown in table 2. One can see that the sentiments in both cases are not evenly distributed.

Text String	Numerical Values	
	5 Classes	3 Classes
Extremely Negative	0	0
Negative	1	0
Neutral	2	1
Positive	3	2
Extremely Positive	4	2

Table 1: Label Encoding

Taking a closer look at the tweets per sentiment shows that the histogram of the word count in the neutral sentiment is right skewed while positive and negative are left skewed (most neutral tweets tend to be short while emotional tweets are longer). The same holds for the histogram of the character count.

Sentiment	Count
Positive	0.275
Negative	0.244
Neutral	0.185
Extremely Positive	0.161
Extremely Negative	0.135

(a) 5 Sentiments

Sentiment	Count
positive	0.436
negative	0.379
neutral	0.185

(b) 3 Sentiments

Table 2: Normalized Counts

We also must discuss some shortcomings of the dataset. Individuals tend to make errors and have different impressions, thus human error and subjective opinion are problematic when labeling the data. This error and bias can be reduced by using more than one individual to label the data and take the average as the resulting label (Kennedy et al. 2020). As a last point the dataset does not consist of perfectly clean text. Some decoding issues are causing that some symbols are not displayed correctly. This adds complexity to the preprocessing.

2 Introduction

On social media platforms like Twitter, people can share their opinion, search for information and interact with one another. Twitter is one of the biggest social media platforms worldwide (Statista 27.02.2021a). In the first quarter of 2019, Twitter had 330 million monthly active users on average (Statista 27.02.2021b). However, such big social media platforms have drawbacks, too. On drawback is described in the theory of the Filter bubbles by Pariser (2011). "To increase user engagement, social media companies connect users with ideas they are already likely to agree with, thus creating echo chambers of users with very similar beliefs" (Chitra & Musco 2020, p.1). This might also happen in health-related topics like the coronavirus. Tweet classification and sentiment analysis can be a first step to recognizing such dangerous bubbles. Sentiment analysis is defined as "[...] the computational study of opinions, sentiments and emotions expressed in text . The goal of sentiment analysis is to detect subjective information contained in various sources and determine the mind-set of an author towards an issue or the overall disposition of a document" (Kumar & Sebastian 2012, p.2).

The main objective of this paper is sentiment analysis on Twitter data. We use several machine- and deep learning approaches for this classification task. Namely we used Support Vector Machine (SVM)-, Multinomial Naive Bayes (NB)-, Random Forest Classifier(RF), a Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) models. We are going to compare them based on their classification results.

We are also going to compare an ordinal loss function suggested by Cao et al. (2020) to the categorical cross entropy loss function which is the standard approach for categorical classification tasks.

In this paper we use the "Coronavirus tweets NLP - Text Classification" dataset by Aman Miglani available at:
`https://www.kaggle.com/datatattle/covid-19-nlp-text-classification`

This dataset contains tweets with a coronavirus reference and for each tweet a label on how the tonality of tweet is (e.g. positive or negative).

In the following we first describe the data in section 1, cover the methods in section 3, then show our results in section 4 and in the end give a short summary and outlook in section 5.

3 Methods

3.1 Preprocessing

Preprocessing and data cleaning prepares the raw text for use in natural language processing (NLP). Preprocessing is also known to have a high influence on the performance of NLP tasks (Alam & Yao 2019; Samad et al. 2020).

For preprocessing the tweets, we used the steps that are presented in case 3 in the article by Samad et al. (2020, p.9) and additional steps from Alam & Yao (2019).

First, we removed urls and email addresses. Regular expressions and functions of the NLTK package (Bird et al. 2009) are used to remove characters, convert all words to lowercase, remove tagged names, punctuation and digits. In most Twitter text processing applications, hashtags are removed entirely. However, in some tweets hashtags are used as normal part of speech in sentences. We therefore decided to only remove the hashtag symbol. The actual word, that follows the symbol is kept in place.

One issue with Twitter data is the use of informal language and contractions. This issue is dealt with by using a dictionary approach. Informal expressions and contractions are replaced by the corresponding long formal form, by mapping according to a simple dictionary. Stop words are removed with the help of a list of commonly used words. However, common words that indicate negation, for example "no", "not", "nor" are not removed. On the cleaned text we then used stemming or lemmatization, depending on the classification algorithm we used. Stemming and lemmatization are used to match words, that are supposed to have the same or very similar meaning. Stemming reduces words to their stem, by removing prefixes and suffixes, which are added to the word to create a new word. For example "walking" and "walked", would be seen as different words, without stemming. Stemming maps them all to their stem "walk". The stem is not always a genuine word. The PorterStemmer from the NLTK package, that uses Porter's algorithm (Porter 1980) maps the word "goes" as in "he or she goes" to "goe" and not to "go" as is might would have been preferable. Srividhya & Anitha (2010, p.50) . However, this is different when using a lemmatizer. The WordNetLemmatizer in the NLTK package returns "go", if you pass the word "goes" to it. That is because it makes "use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma"(Manning et al. 2009, p.32).

3.2 Baseline Models

To compare the performance of our Deep learning models we used a set of classical machine learning algorithms on our data. As mentioned before, we used SVM, NB and a RF Classifier. We now take a brief look at the used baseline models.

3.2.1 Linear Support Vector Machine

The Support Vector Machine (SVM) uses a linear hyperplane that maximize the width of the gap between the categories to create a classifier. Boser et al. (1992) suggested a way to create nonlinear classifiers. When the data is not separable by a linear function, the data is then mapped into a higher dimension feature space. This can be done by nonlinear mapping functions like: polynomial-, tanh- or Gaussian radial basis-function. In the higher dimension feature space, the SVM again separates the data using a linear hyperplane (but it may be nonlinear in the original input space). Furthermore, SVM does not need an assumption about the probability model and the probability density functions. This is very useful since there is often not enough information about the underlying probability laws and distributions. (Sakr et al. 2016; Hastie et al. 2009))

3.2.2 Multinomial Naive Bayes

Multinomial Naive Bayes (NB) is a supervised learning algorithm that uses Bayes' rule to calculate the probability that a tweet belongs to a sentiment based on the words that it contains, under the assumption that the words are statistically independent conditional on the true sentiment. NB produces the most likely words for each class of tweets. The model can also generate predicted class probabilities to use for classification. While NB models are easy to interpret and quick to train, they have some disadvantages, like the mentioned conditional independence assumption. NB models are often outperformed by discriminative models like the SVM. (Rennie et al. 2003)

3.2.3 Random Forest

Random Forests (RF) are a non-parametric method that builds an ensemble model of Decision Trees (DT) from random subsets of features and bagged samples of the training data. A basic principle behind ensemble learning is that a single strong classifier can be constructed by combining a series of classifiers which perform slightly better than random guessing (weak learners). Often used ensemble tree models are Bagging Trees (BT). The BT model aims to address the problem of overfitting of DTs. The BT model randomly selects a subset of the training data (with replacement) to train each individual DT model in the ensemble. The classification is then based on majority voting of the individual DTs. The RF classifier similar to BT is grown using a random subset of the training data, but also randomly selects a subset of the classification variables for classification in each DT. This leads to higher variance and lower bias of the DTs. (Jozdani et al. 2019)

RF models have two hyper parameters: the number of randomly selected features (M) used for splitting each node and the number of trees (N). According to findings of Breiman (2001) reasonable accuracy was obtained when M was set to $log_2(m) + 1$. Where m is the number of variables. Setting $M = \sqrt{m}$ is also possible. In addition, Lawrence et al. (2006) found that setting $N \geq 500$ produced unbiased estimates. In practical applications N is often set to 100. (Jozdani et al. 2019)

3.2.4 TF-IDF Vectoriser

In order to be able to apply our baseline models one need to convert the sequence of word in each tweet to integers or floats. This can be done by so called Vectorizers. In the literature there are two common approaches namely Count Vectorizer and TF-IDF Vectorizer. The Count Vectorizer naively converts a collection of tweets to a vector of term counts. On the other hand, the TF-IDF (term frequency - inverse document frequency) Vectorizer not only takes into account how many times a term appears in a tweet (term frequency), it also uses the inverse document frequency of the term across a set of tweets.

The inverse document frequency is a way to evaluate how common or rare a term is inside the total set of tweets. It can be calculated by taking the number of all tweets, dividing it by the number of tweets that contain a term, and then take the logarithm. The inverse document frequency will approach 0 if a term is very common and appears in many documents. Otherwise, it will approach 1. To set up the TF-IDF Vectorizer one just multiplies the term frequency and the inverse document frequency. For our Baseline models we decided to use the TF-IDF Vectorizer. (Pedregosa et al. 2011)

3.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are usually known for being used in image processing, where they apply filters on subregions of pictures. However, the same concept can be used on text data. The input text is seen as a sequence of words or other tokens. When the tokens are represented as vectors, each input tweet is a sequence of vectors. This sequence can also be written as a matrix with each row vector representing a word's feature vector. The convolutional layer then applies filters in form of tensor operations to sub regions of the matrix or respectively sub sequences of vectors. Applying the same filters to all sub regions makes it possible to extract local patterns. But these moving window filters are invariant to translation. So, the CNN can recognize if a sequence of words is repeated, but it cannot distinguish between the same sequence at different positions in the text. Using various filter sizes also allows for extracting word patterns of a different length (Chollet (2018, p.225 ff), Zhang & Wallace (2016, p.2 ff), Zhou et al. (2016, p.5)).

Yin et al. (2017) show that CNN have sufficient performance on some text classification tasks if only local features are relevant but are still outperformed by other deep learning structures otherwise. Chollet (2018, p.225) additionally emphasizes the speed advantages of CNNs in certain NLP tasks.

3.4 LSTM

Long Short-Term Memory (LSTM) networks are based on recurrent neural networks (RNN). All RNNs have feedback loops in the recurrent layer. This lets them maintain information over time.

LSTM networks are using special units in addition to standard units. LSTM units include a cell state. It contains the information that can be passed on to the next steps. A set of gates is used to control when information enters the memory, when it is output, or when it is forgotten. This architecture lets them learn longer-term dependencies. In brief, it remembers information that is needed for the result but at the same time omits information during the training period that does not affect the networks performance positively. (Nowak et al. 2017)

A single LSTM layer cannot be enough in some instances. In this case one should think about using a bidirectional LSTM layer. This layer differs from the single layer in that it runs the inputs in two ways - forward and backward - through the network. It therefore has access to past and future information. (Nowak et al. 2017)

3.5 Ordinal Loss Function

The loss function for each output would benefit if the ordinal nature of the labels is considered. Predictions further away from the true label are worse than the prediction of adjoining labels. For instance if a tweet is labeled as "strongly negative" the predicted probability of "extremely positive" should have a higher weight to the loss than "negative". This follows, because the prediction of "negative" is superior to "extremely positive".

Categorical Cross entropy is the standard loss function used for discrete labels. This loss is a good measure of how distinguishable two discrete probability distributions are from each other. In our case every false prediction would be weighted the same. Therefore, it does not take a possible ordinal nature of the labels into account.

A method to account for the ordinal nature of the labels is called consistent rank logits (CORAL). It was suggested by Cao et al. (2020). When used on k possible labels, the CORAL method decomposes an ordinal regression into k-1 binary classification tasks. The loss function is defined as the sum of the cross-entropies of each of these binary classification tasks. It is implemented in the python package coral-ordinal. (Kennedy et al. 2020)

3.6 GloVe for Word Representation

The embedding layer specifies how words get represented numerical in a vector space. In our deep learning models, we used either a trainable embedding-layer or the pre-trained word vector representation GloVe by Pennington et al. (2014). The vector representation can then be used in the sub-sequential layers of the deep learning model. Pennington et al. (2014) use a "global log bi-linear regression model" with some type of weighted least squares loss function and train it on a global word-word co-occurrence matrix (Pennington et al. 2014, p. 1132). They created a word vector representation, that does not only account for similar meaning of words, but it is possible to perform vector transformations that produce meaningful results. So the same transformation

that transforms the vector for "king" into the vector for "queen" may also be the transformation, that returns the vector representation for "woman" when it is applied to the vector for "man" (Pennington et al. 09.04.2020). As a result their model performs well on tasks like finding "word analog[ies], word similarit[ies] and named entity recognition[...]" (Pennington et al. 2014, p. 1541). We use GloVe embeddings, that were trained on twitter data that contained 2 billion tweets (Pennington et al. 09.04.2020)(`https://nlp.stanford.edu/projects/glove/`).

4 Results

4.1 Baseline Models

As mentioned previously, we use the TF-IDF Vectorizer on our cleaned data. We used the functions provided in the scikit-learn package (Pedregosa et al. 2011). For our baseline models we used the default settings from scikit-learn. We therefore used a linear SVM with l2 penalty and squared hinge loss, a RF classifier with N=100, $M = \sqrt{m}$ and a NB with alpha=1 (Additive Laplace/Lidstone smoothing parameter).

In the following subsection we look at the accuracy that is achieved by our baseline models. For a less biased estimate we used 10-fold cross validation. This leads to the box plot in figure 1. We also want to find out if our plain cleaned tweets are significantly different in terms of accuracy compared to the stemmed and lemmatized version.

Figure 1 shows that the linear SVM performs the best for both three and five sentiments. Also, we can see that the RF classifier outperforms the NB classifier as expected. Concerning the stemmed and lemmatized versions there is no clear difference visible. Comparing the estimation of three and five sentiments shows that all classifiers perform significantly worse on five sentiments. This is kind of expected since a larger set of possible sentiments increases the probability to choose wrong.

In figure 2 we take a closer look on the classification result of the SVM on (approximately) the train dataset. For that we use a confusion matrix. It shows how predicted class compares to the actual sentiments. The main classification performance can be seen on the matrix diagonal. In the normalized matrix that we are using in this paper it should be as close to unity as possible. We can find that for both three and five sentiments the diagonal is the largest value in each row. The positive and negative diagonal entry (both $> 80\%$) in the three-sentiment case are larger than the neutral entry (70%). In the five-sentiment case it is the other way around. Here the neutral sentiment is again about 70% but negative and positive are now only $> 50\%$. Both are classified with high percentages ($> 10\%$) in four out of five sentiments. Given the ordinal structure of the data a classification as close to the true value would be superior. An allocation to a class that is not a direct neighbor is therefore less desirable. The extremely positive and extremely negative sentiment are both covered better than their weaker alternatives at around 60%. This is also expected since stronger sentiments tend to be more distinguishable for humans as well. Problematic in the five-sentiment case is that positive and negative are often mistaken for each other ($> 15\%$).

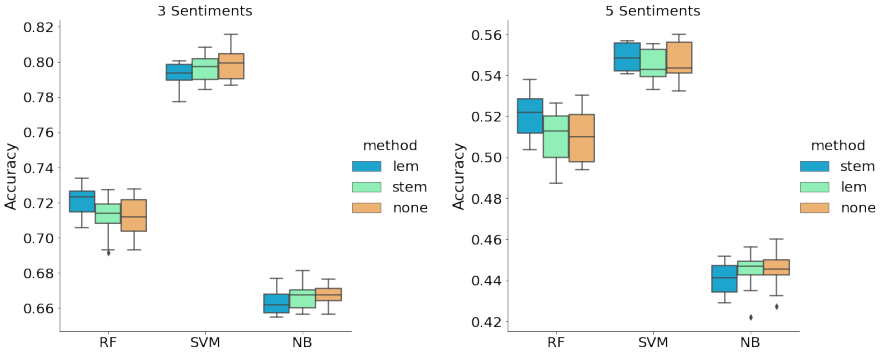


Figure 1: Baseline Model Accuracy

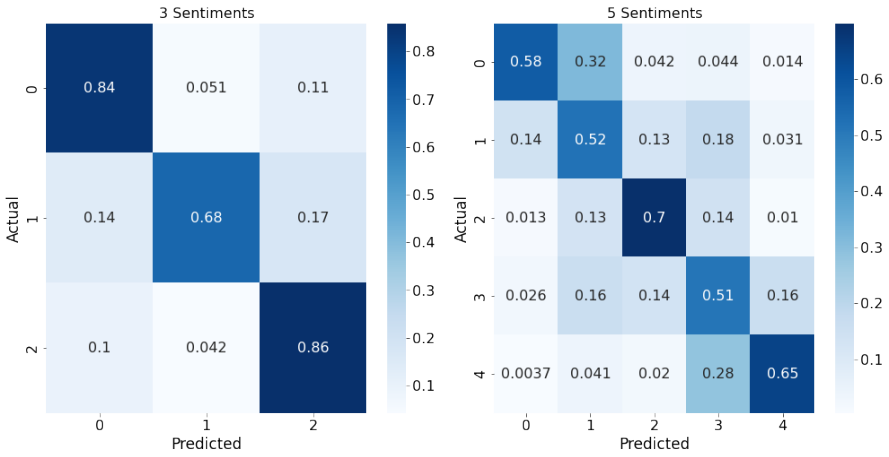


Figure 2: SVM Classifier Confusion Matrix (Sentiment-Coding in Table 1)

4.2 Convolutional Neural Networks

For all deep learning models, we used an early stopping to find the end point of training. The early stopping monitored the validation loss. We used a patience of 5 and a min-delta of 0. In case of the categorical model we use the accuracy as the evaluation metric. Using the accuracy assigns equal cost to false positives and false negatives which can cause issues with highly imbalanced datasets (Chawla 2010). We assume our dataset to be balanced enough that this is not a major issue. As suggested by Cao et al. (2020) we used a mean absolute error (MAE) as the evaluation metric for the ordinal model. The MAE for CORAL is further described in their paper. The resulting accuracy of the optimal ordinal model can be obtained by the classification report on the test data. We used the lemmatized versions of the text in all deep learning models.

The CNN model is constructed as follows:

- one dimensional Convolution layer with 64 filters, a kernel size of 8, and a rectified linear unit (ReLU) activation function
- one dimensional MaxPooling layer
- 20% Dropout layer
- Dense layer with 16 units and a ReLU activation function
- one dimensional GlobalMaxPooling layer
- Dense layer with 'nclass' and activation function 'X'

Where nclass $\in \{3, 5\}$ the activation function X is either 'softmax' or 'ordinal-softmax' based on if the categorical or the ordinal model is estimated. We used the Adam optimization algorithm and found a learning rate of $1 * 10^{-4}$ to be adequate. The convolutional layer is made up of 64 one-dimensional filters of size 8. The dropout layer is supposed to reduce overfitting. The max-pooling layers are reducing the dimensionality. As the embedding layer, we used a non-trainable layer with an embedding matrix that is derived from the pre-trained GloVe Embedding Vectors as described in section 3.6. The word representation vectors are of dimension 200. With the pre-trained vectors we obtained a word vector matrix that contained a vector representation for each unique token. In the graphs 3 we can see that the categorical model model starts overfitting after 6 epochs and the ordinal model after 12 epochs. The resulting categorical model shows an overall accuracy of $\geq 51\%$ and the same for the weighted F1-score, recall and precision on the test dataset. The three-sentiment case is omitted here. Nevertheless it shows an accuracy of 73 %. In the confusion matrices in figures 3 and 4 one can see that the ordinal model better predicts the extreme sentiments, whereas the categorical model is better at classifying neutral and weakly positive or negative sentiments. The predictions are non-consistent since non-adjoining classes are confused more frequently than neighboring classes. Here the positive sentiment is mistaken for the negative sentiment and vice versa. That is the case in both the ordinal and the categorical model.

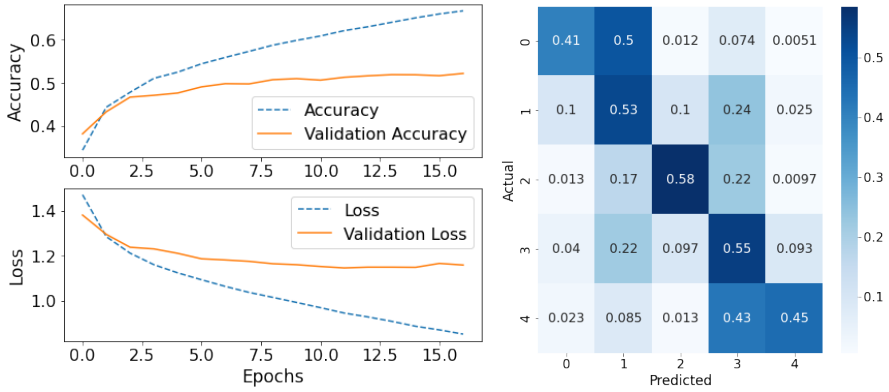


Figure 3: CNN Categorical 5 Sentiments (Sentiment-Coding in Table 1)

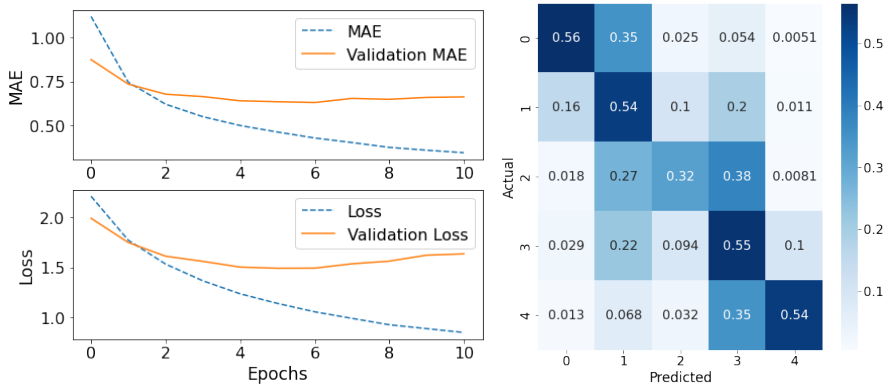


Figure 4: CNN Ordinal 5 Sentiments (Sentiment-Coding in Table 1)

4.3 LSTM

The LSTM models is constructed as follows:

- Bidirectional LSTM layer with 256 units
- one dimensional GlobalMaxPooling layer
- 40% Dropout layer
- Dense layer with 64 units and a ReLU activation function
- 40% Dropout layer
- Dense layer with 'nclass' units activation function 'X'

Where nclass is $\in \{3, 5\}$. The activation function X is either 'softmax' or 'ordinal-softmax' based on if the categorical or the ordinal model is used. We used the Adam optimization algorithm and found a learning rate of $5 * 10^{-4}$ to be adequate. The global-max-pooling layer is used to reduce the dimensionality. The dropout layers are included to further reduce overfitting. The main layer is a bidirectional LSTM layer with 256 units. The used evaluation metrics are analogous to the CNN.

4.3.1 LSTM with Own Trainable Embeddings

The first layer of the model is an embedding layer with 16 output dimensions, tweet length of 50 and a reduced total vocabulary size to reduce overfitting. The total vocabulary size is given by all unique words in the dataset. As described in section 4.2 we used an early stopping to determine the end of training. We are restoring the model weights with the lowest loss in order not to show an overfitted model.

The overfitting in the categorical model starts after 2 epochs, whereas in the ordinal model it can be detected after 7 epochs. As it can be seen in the loss function plots in figures 5 and 6, choosing a model later than that is not optimal. We could have used a learning rate scheduler to compensate that behavior but decided against that (accuracy and MAE converge fairly well). The resulting categorical model shows an overall accuracy of $\geq 70\%$ and the same for the weighted F1-score, recall and precision on the test dataset. The ordinal model however shows only values $\geq 63\%$. The classification results in form of the confusion matrices in figures 5 and 6 show that the ordinal loss function leads to a better classification of the extreme sentiments. The diagonal entries in the confusion matrix of the categorical model are higher for positive neutral and negative but here again as in the SVM model (figure: 2) positive is mistaken for negative and vice versa while both are rarely mistaken to be neutral. The ordinal model shows a more consistent picture. Here no adjoining categories are left out when a category further away is predicted. This comes at the cost that the recall in each positive negative and neutral is lower compared to the categorical model. But it also has the positive effect that positive is no longer largely mistaken for negative. The other way around it at least shrinks the amount of negative that is mistaken for positive. When comparing the confusion matrices in figures 5 and 6 the ordinal model additionally predicts the extreme sentiments better.

The three-sentiment case is not displayed here. It shows in the categorical case an accuracy of $\geq 83\%$. When the ordinal model is estimated the loss function often finds it optimal to not classify any tweet as neutral. It is possible to find models where this is not the case but when a certain number of independent models is estimated it happens to often too be considered viable.

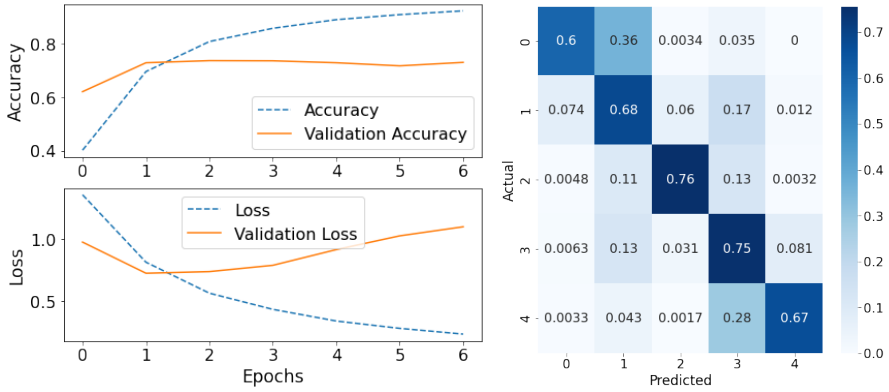


Figure 5: LSTM Categorical 5 Sentiments (Sentiment-Coding in Table 1)

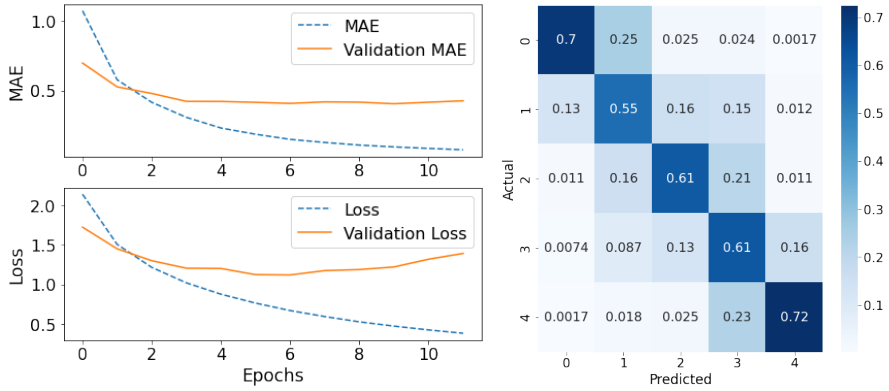


Figure 6: LSTM Ordinal 5 Sentiments (Sentiment-Coding in Table 1)

4.3.2 LSTM with GloVe Embeddings

In this section we are using the pre-trained 200-dimensional GloVe embeddings (non-trainable). The model as well as the evaluation metrics are the same as described in section 4.3. Figure 7 and 8 show metric loss and the confusion matrix of the test set. We observe a similar behavior as described in section 4.3, but the overfitting seems to start at a later point (categorical: epoch 6, ordinal: epoch 9). Additionally, the accuracy of the categorical model is $\geq 70\%$. This also holds for the weighted F1-score, recall and precision. All metrics are slightly better as compared to section 4.3.1. In the ordinal model we observe values of $\geq 66\%$. The ordinal model again leads to the more consistent estimation. In the (not shown) three-sentiment categorical case

the accuracy is $\geq 84\%$. The classification based on the ordinal model tends to be more stable when compared to section 4.3.1 but also suffers from the same problems previously mentioned.

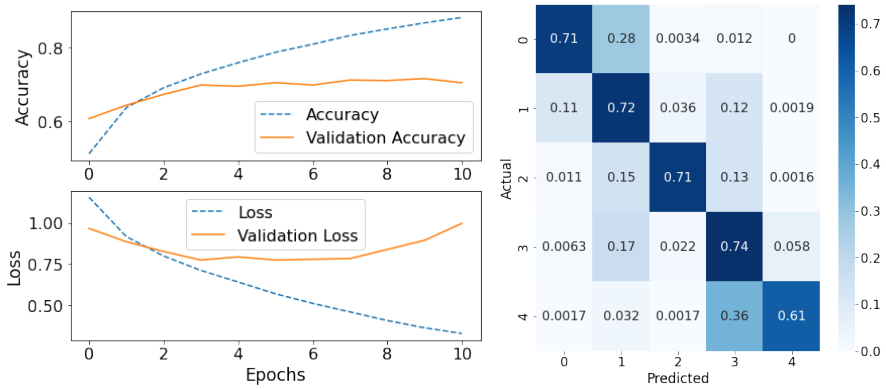


Figure 7: LSTM GloVe Categorical 5 Sentiments (Sentiment-Coding in Table 1)

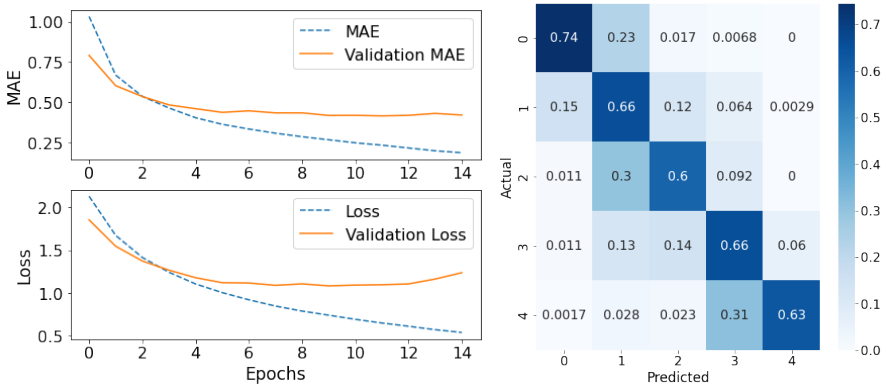


Figure 8: LSTM GloVe Ordinal 5 Sentiments (Sentiment-Coding in Table 1)

5 Conclusion and Outlook

We can conclude that the approach using a CNN (ordinal and categorical) performs worse than the SVM classifier (baseline) for both three and five sentiments. However, both LSTM models are able to beat our baseline. When we look at the three-sentiment categorical case the LSTM outperform the SVM very slightly (2%) in terms of accuracy. In the ordinal case the SVM classifier has actually a higher accuracy but the ordinal

model leads to a more consistent classification result. However, the very high dropout rates in the LSTM models leave the question if the findings are generalizable. In the five-sentiment case both ordinal and categorical outperform the SVM clearly. We use SVM to represent the baseline models since it outperformed RFs and NB. When comparing both LSTM models, it is also notable that the GloVe embeddings seem to outperform the self-trained embeddings.

We also compared an ordinal with a categorical loss function. We found that one needs to decide what is more important: Accuracy or a classification result that takes the ordinal structure of the data into account. In the five-sentiment case we observed that the accuracy in the categorical model is always higher when compared to the ordinal model. On the other hand, the ordinal loss function leads to a more consistent classification result (when it is important that classifications further away are less desirable). This does not hold for the CNN. Also, we found that the recall values of the extreme sentiments are better covered by the ordinal loss. In the three-sentiment case we can only recommend the usage of the categorical loss since the ordinal loss function often ignores the neutral sentiment. A possible explanation for this behavior can be the underrepresentation of the neutral segment in in the three-sentiment case (as can be seen in table 2).

When we compare five and three sentiments, we can observe that the accuracy on the three-sentiment case is higher (by approximately 10%). This is as expected since the prediction tends to be easier with more condensed classes. However, the ordinal structure of the data can only be utilized in the five-sentiment case.

Nevertheless, there is still room to improve the performance of our models. First, we could change the preprocessing. There are decoding issues in the raw files we used. We solved the issue by removing problematic character sequences. However, one should find a way to properly decode them. Samad et al. (2020) evaluate the influence of different preprocessing steps on the performance of several NLP tasks. Their findings suggest that emojis should be encoded rather than removed. They also come to the conclusion that the excessive removal of stopwords, using a predefined list of common words for a specific language has a negative effect on performance. Further they suggest that the dimension of the vectors, representing the tokens in the embedding layer, should not be too high.

We further noticed that the pre-trained GloVe Embedding does not contain a vector representation for about 40 % of or tokens. Thus they are represented as zero vectors. Some of the tokens, that are not recognised by GloVe are compound words. Compound words arise when “two or more words linked together [...] produce a word with a new meaning” Cambridge University Press (2021). Some Hashtags, that are used on Twitter, are also made up of multiple words (e.g. “#stayhome”). One could try to split this string and so retrieve a vector representation for the separate words. Another possibility to increase the share of tokens, that are represented by non-zero vectors is to use the mittens algorithm by Dingwall & Potts (2018). This algorithm can add representations for corpus specific words to a pre-trained GloVe Embedding.

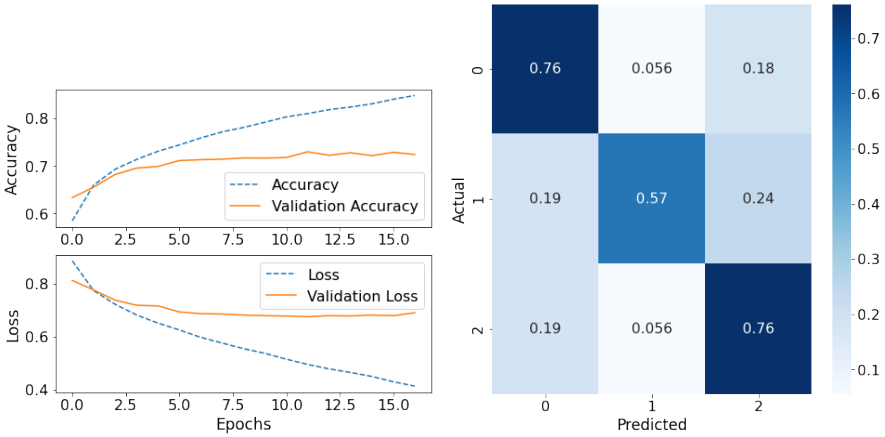


Figure 9: CNN Categorical 3 Sentiments

6 3 Sentiments

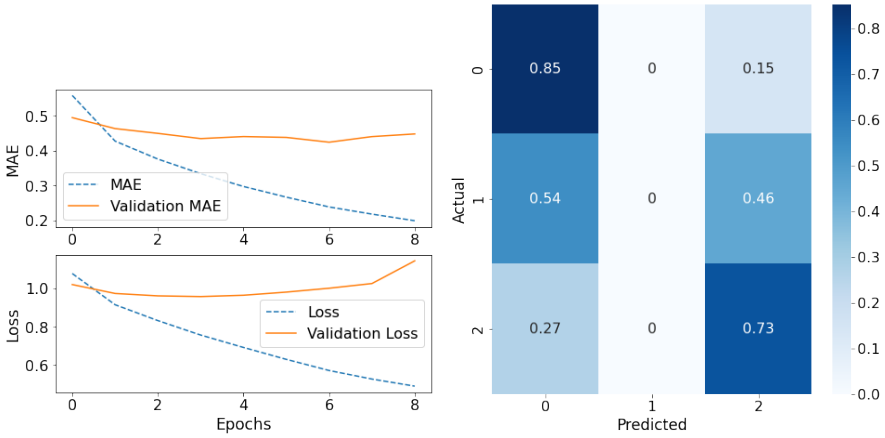


Figure 10: CNN Ordinal 3 Sentiments

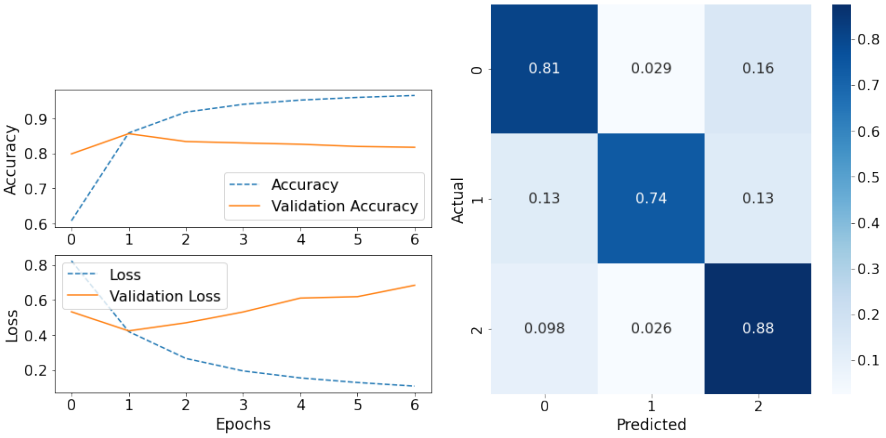


Figure 11: LSTM Categorical 3 Sentiments

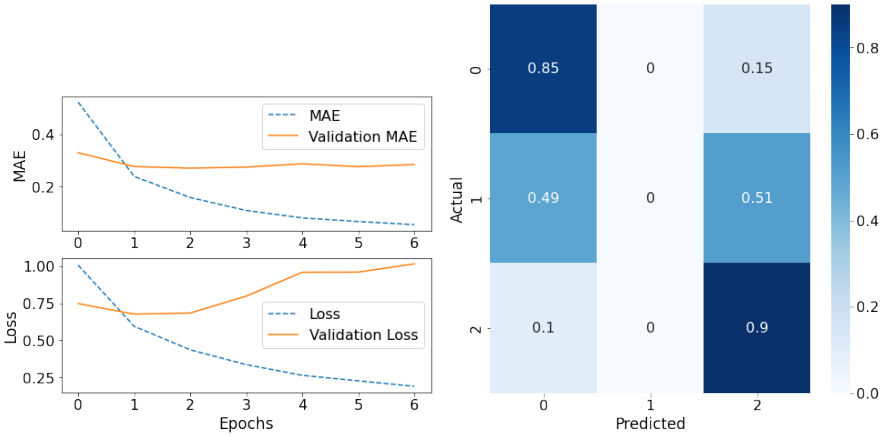


Figure 12: LSTM Ordinal 3 Sentiments

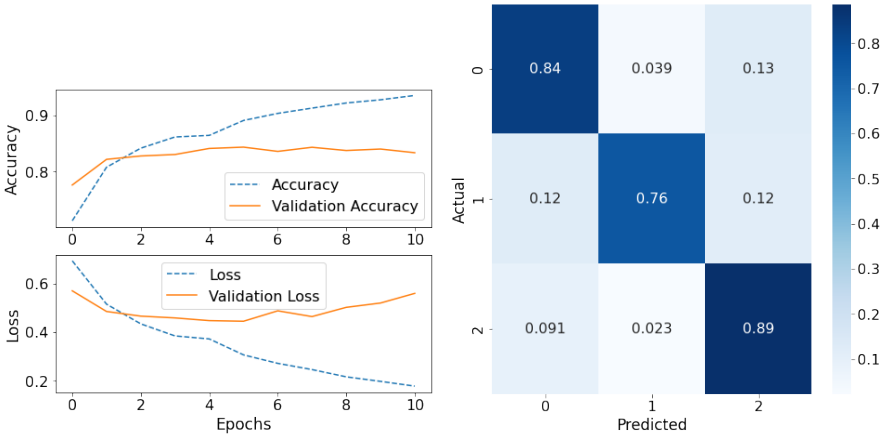


Figure 13: LSTM GloVe Categorical 3 Sentiments

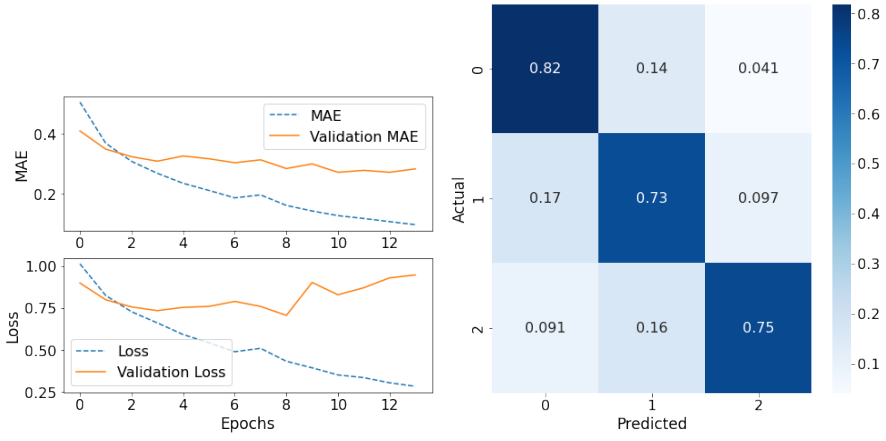


Figure 14: LSTM GloVe Ordinal 3 Sentiments

References

Alam, S. & Yao, N. 2019, Computational and Mathematical Organization Theory, 25, 319
Bird, S., Klein, E., & Loper, E. 2009, Natural language processing with Python: analyzing text with the natural language toolkit (O'Reilly Media, Inc)
Boser, B. E., Guyon, I. M., & Vapnik, V. N. 1992, in Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92 (New York, NY, USA: Association for Computing Machinery), 144–152
Breiman, L. 2001, Machine Learning, 45, 5
Cambridge University Press. 2021, Compounds: Cambridge Dictionary
Cao, W., Mirjalili, V., & Raschka, S. 2020, Pattern Recognition Letters, 140, 325–331
Chawla, N. V. 2010, in Data Mining and Knowledge Discovery Handbook, ed. O. Maimon & L. Rokach (Boston, MA: Springer US), 875–886
Chitra, U. & Musco, C. 2020, in Proceedings of the 13th International Conference on Web Search and Data Mining, ed. J. Caverlee, ACM Digital Library (New York,NY,United States: Association for Computing Machinery), 115–123
Chollet, F. 2018, Deep learning with Python, Safari Tech Books Online (Shelter Island, NY: Manning)
Dingwall, N. & Potts, C. 2018, in Proceedings of the 2018 Conference of the North American, ed. M. Walker, H. Ji, & A. Stent (Stroudsburg, PA, USA: Association for Computational Linguistics), 212–217
Hastie, T., Tibshirani, R., & Friedman, J. 2009, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition, 2nd edn., Springer Series in Statistics (New York: Springer-Verlag)
Jozdani, S. E., Johnson, B. A., & Chen, D. 2019, Remote Sensing, 11
Kennedy, C. J., Bacon, G., Sahn, A., & von Vacano, C. 2020, Constructing interval variables via faceted Rasch measurement and multitask deep learning: a hate speech application
Kumar, A. & Sebastian, T. M. 2012, International Journal of Intelligent Systems and Applications, 4, 1
Lawrence, R. L., Wood, S. D., & Sheley, R. L. 2006, Remote Sensing of Environment, 100, 356
Manning, C. D., Raghavan, P., & Schütze, H. 2009, Introduction to information retrieval, reprinted. edn. (Cambridge: Cambridge Univ. Press)
Nowak, J., Taspinar, A., & Scherer, R. 2017, in ICAISC, 553–562
Pariser, E. 2011, The filter bubble: What the Internet is hiding from you (London: Viking)
Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, Journal of Machine Learning Research, 12, 2825
Pennington, J., Manning, C. D., & Socher, R. 09.04.2020, GloVe: Global Vectors for Word Representation
Pennington, J., Socher, R., & Manning, C. D. 2014, in Empirical Methods in Natural Language Processing (EMNLP), 1532–1543
Porter, M. F. 1980, Program, 14, 130
Rennie, J. D. M., Shih, L., Teevan, J., & Karger, D. R. 2003, in Proceedings of the Twentieth International Conference on International Conference on Machine Learning, ICML'03 (AAAI Press), 616–623
Sakr, G. E., Mokbel, M., Darwich, A., Khneisser, M. N., & Hadi, A. 2016, in 2016 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET), 207–212
Samad, M. D., Khounviengxay, N. D., & Witherow, M. A. 2020, arXiv:2007.13027 [cs], arXiv: 2007.13027

Srividhya, V. & Anitha, R. 2010, International Journal of Computer Science and Application Issue, 2010
Statista. 27.02.2021a, Most used social media 2020 — Statista
Statista. 27.02.2021b, Twitter: monthly active users worldwide — Statista: from 1st quarter 2010 to 1st quarter 2019 (in millions)
Yin, W., Kann, K., Yu, M., & Schütze, H. 2017, Comparative Study of CNN and RNN for Natural Language Processing
Zhang, Y. & Wallace, B. 2016, arXiv:1510.03820 [cs], arXiv: 1510.03820
Zhou, P., Qi, Z., Zheng, S., et al. 2016, arXiv:1611.06639 [cs], arXiv: 1611.06639

Labelling and predicting sentiments in Twitter Data on the US Election 2020

Katharina Linke, Jasmin Wiebke Schilling

1 Introduction

The US Elections 2020 (presidential, house and senate election) took place on 3rd November and were subject to many discussions, arguments and controversies. The election campaigns were largely influenced by recent topics like the Covid-19-pandemic and were emotionally charged. Surely, one of the main reasons for this was the controversial candidate and then incumbent President of the United States, Donald J. Trump. Many of these discussions took place on social media platforms. These platforms enable their users on the one hand to express their opinions and emotions directly, and on the other hand to get in touch with other users to share and discuss different positions. One of the big and important services in this context is Twitter. Twitter is a microblogging site that allows its users to share contents in short texts which are limited to 280 characters. It has become one of the most important social media platforms nowadays, and is used by more than 330 million people worldwide (2018) (Statista 2020). Furthermore, it is used frequently to apply and investigate topics in Natural Language Processing such as sentiment analysis. This is due to the possibility to collect real time discussions, and the convenient way of data sampling that Twitter allows using an API. In this paper, we aim to apply a deep learning model to Twitter data on the US Election 2020, and investigate whether we can classify information in the tweets. A major challenge is to introduce an appropriate way of labelling our data, since we need labels to train our model, and the tweets have not been labelled in advance. To find appropriate labels, we try different approaches such as a sentiment analysis by using the *TextBlob* library or a clustering approach. Moreover, we attempt to gain information on the tweets' sentiments by taking the faces and emojis into account. After evaluating the results of the labelling techniques, we train different neuronal networks on the data to see whether we can predict the sentiments in the tweets. In this paper, we first introduce the applied methods in section 2. Next, we present the data set on the US Election in November 2020 that we use for our analysis and how we process them in section 3.1. Section 4 then shows our results and discusses them while section 5 gives a conclusion and outlook for further research.

2 Methods

The following chapter gives a brief introduction to the field of Deep Learning, and explains the meaning of this term. Afterwards, we will introduce the methods we used. As a relatively new sub field of machine learning, Deep Learning was subject to a huge amount of interest in science and industry lately. Chollet (2018) states this is due to the availability of larger computational power and data sets. Deep Learning-models are able to process various input data next to mere numbers such as pictures, music or text data and gained their name from their structure. By structuring the computations in layers the models become "deep". In this paper, we focus on the analysis of textual data, and thus chapter 2.1 gives an idea of sentiment analysis as a field of Natural Language Processing and how we apply it.

2.1 Sentiment Analysis

In general, Natural Language Processing can be described as „the ability for a computer/ system to truly understand human language and process it in the same way that a human does" (Goyal et al. 2018). Sentiment analysis is a wide field of study which contains the analysis of "people's opinions, sentiments, evaluations, appraisals, attitudes, and emotions towards entities such as products, services, organizations, individuals, issues, events, topics, and their attributes" (Liu 2012) In our analysis, we want to classify tweets according to their sentiment and check whether we can use these classifications as labels. We use *TextBlob* for this purpose (Loria 2020). *TextBlob* is a widely used Python Library for processing textual data and can be used for sentiment analysis. It offers many architectures to classify textual data in various classes, such as positive and negative (Goyal et al. 2018). In our analysis, we use *TextBlob* to compute the tuple 'polarity' and 'subjectivity', which represent the sentiment of a expression. The polarity reflects whether a Tweet is more negative or positive and floats in the rage of $[-1, 1]$. A negative value implies a negative sentiment, a positive value vice versa. Values around zero show to be neutral. This analysis is performed by calculating the polarity score of a text based on the context (Micu et al. 2017).

2.2 Cluster analysis

Another approach to label our data is the use of a clustering algorithm as proposed by Jan (2020). As a cluster analysis method, we use the K-means algorithm (MacQueen 1967) which is a basic, but still widely used method. It is known as an unsupervised learning algorithm that splits the data into K different clusters. In the first step, the data is split into the given number of clusters by randomly assigning the observations to them. Next, the algorithm computes the centroid for every cluster and rearranges the partition. Every observation is put into that cluster whose centroid possesses the shortest distance to the observation. This is repeated until no further changes in the partition occur. In our paper, we intend to use the algorithm on textual data. Hence, we have to vectorize the data first which is done using the TF-IDF (term-frequency times inverse document-frequency) vectorizer from the skleans library (Pedregosa et al.

2011). The vectorizer uses an in-memory vocabulary and computes a word occurrence frequency matrix that is "reweighted using the IDF vector collected feature-wise over the corpus" (Pedregosa et al. 2011). We use different numbers of clusters and methods of preprocessing to find the best partitioning for our data. The aim of this procedure is to use the clusters as labels to build a DL-model.

2.3 Neuronal Networks

This subsection describes neuronal networks (NN), or more precisely known as artificial neuronal network (ANN) (Nwankpa et al. 2018). In more detail, the most commonly used architecture is shortly introduced and the specification of the individual parameters used here are explained. In principle, an ANN is a mathematical function that takes some information as an input and turns them into an output. In order to do so, the ANN uses neurons that are connected like a biological NN, which gave the inspiration for the contraction. As mentioned initially, it is the layers that make the network deep. Each layer processes the information it receives from the previous one in a specific way and passes them on to the next layer. Our model is a rather simple one that consists of an embedding layer, a dropout layer, a LSTM layer and a dense layer:

- Word embeddings are used to transform textual data into a multidimensional representation which connects them to words that are similar, but not equal. Therefore, the NN is capable of "understanding" textual data. We use the pre-trained model GloVe (Pennington et al. 2014).
- Dropout: The dropout layer makes the network randomly "forget" a proportion of parameters by setting them to zero during training (Weidman 2019). The dropout rate is the proportion of dropped neurons. This procedure helps to prevent overfitting, since it can break non significant patterns during training, which otherwise will be memorized by the model.
- LSTM: Recurrent Neuronal Networks (RNNs) can be seen to have a kind of internal loop by "iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far" (Chollet 2018). Furthermore, they can be understood as creating long-term dependencies. The long-short-term-memory (LSTM) is a further development and solves a problem of the RNN, since long-term dependencies are impossible to learn in practice. The LSTM, therefore, drops certain previous information and establishes a connection between newly and previously learned information.
- Dense: This layer is the last layer in the model, and takes as many inputs as it produces outputs and, therefore, is a fully connected layer. It is normally used as an output layer that performs the task of the classifying.

The number of input nodes is set to 140. This stems from the fact that a tweet can not contain more than 280 characters which is the double of the nodes. The number of output nodes depends on the chosen number of labels, this will be specified in subsection 4.1.4. The model-specific parameters, activation function and optimizer,

were chosen based on scientific and practical experience which define the standard choice. We choose ADAM as the underlying optimization algorithm, as the loss function the categorical-crossentropy is picked, and for the activation function softmax is selected.

3 Data

The following chapter introduces the data used, their analysis and shows the application of the algorithms to them. Therefore, we give an overview of the available variables. In the subsequent section, we will employ different methods and algorithms on the presented data set.

3.1 Used data set

This section gives a summary of the data used. Subsequently, the data collection technique, as well as the overall scope of the collected data, are described in more detail. Furthermore, limitations of the given data and a brief overview of the variables included are given.

The two data sets we investigate in this paper contain various variables from Twitter data collected from 15 October until 8 November 2020. The data has been collected using the Twitter API *statuses_lookup* and *snsscrape* for keywords (Hui 2020). It is freely accessible via the platform Kaggle (2020). The first data set contains 810.986 tweets regarding Donald Trump, while the second set provides 551.391 tweets that were posted with the hashtags JoeBiden or Biden. It is important to note that one Tweet can contain both hashtags and, therefore, can occur in both data sets. This affects in total a number of 375.292 tweets. Furthermore, it needs to be considered that the raw data sets contain tweets in 97 different languages, whereas the majority is written in English (see Figure 6).

In a further step, the data set is restricted to tweets that are written in English. The final data set contains 698.330 tweets, 435.943 mentioning Donald Trump and 262.387 with reference to Joe Biden.

The un-edited data set contains 21 variables, which provide information about the content, the user, the time, and the location. After running the preprocessing and the different methods for labeling, the data set contains 51 variables (see table 11).

3.2 Descriptive Analysis

This section describes the descriptive analysis how it is performed. Since our algorithms are not able to process non-English language properly, we exclude those tweets containing any language other than English.

When looking at the size of the data sets, it is easy to recognize that in the observed period the number of tweets mentioning Donald Trump (see Figure 2 exceeds those mentioning his rival.

However, when taking a closer look at the total number of Likes and Retweets for both candidates, a different finding can be obtained. While the tweets referring to Trump received a total of 2.480.305 Likes, which equals 99.212,20 Likes per day

and 643.538 Retweets, which are 25.741,52 per day. On the other hand, the tweets concerning Biden gained 2.804.786 Likes (112.191,44 per day) and 724.400 Retweets (28,976 per day). Thus, the Biden tweets appear to generate a larger range of influence, since even though they are less in total numbers, they have been both liked and retweeted more frequently (see Figure 4 and 5). This and other insights are gained by the variables given in both data sets. The available variables are given in table 11. For some selected variables the observed data are distributed as stated in table 1:

Table 1: Summary of selected variables

Variable	data set	Min	25 %	50 %	Mean	75 %	Max
`likes`	Trump	0	0	0	5,690	1	35.449
	Biden	0	0	0	10,189	1	165.702
`retweet_count`	Trump	0	0	0	1,476	0	12.181
	Biden	0	0	0	2,761	0	63.473
`user_followers_count`	Trump	0	73	415	9.616,215	1.939	13.919.941
	Biden	0	72	420	12.270,518	2.065	82.417.099

The majority of the tweets have neither been liked or retweeted. However, there is a small number of tweets with a large range of influence. It is reasonable to assume that there are some accounts which generate numerous reactions by other users. These accounts might belong to some prominent users. Looking at the correlations between the variables, we can state that there is a high positive correlation between the number of Likes a Tweet receives and the times it gets retweeted ($r_{likes,retweets,Biden} = 80,10\%, r_{likes,retweets,Trump} = 89,18\%$). Intuitively, it is very reasonable to presume that a user who liked a Tweet intends to share it with others and retweets it for this reason. Other studies were able to the find a positive correlation between Likes and Retweets as well (Daga et al. 2020).

Looking at other variables, it becomes visible that there is a multitude of tweets without any information about the location given. This is due to the privacy policy of Twitter that sets the information about the users' location to private by default. As a consequence, in a large number of cases, we have no information about the users' location (Twitter 2020). In fact, this affects 269.658 (61,86 %) tweets in the data concerning Trump for which we do not know the users' state, and 221.711 (50,86 %) for which we do not even know the users' country. In the other data set that provides tweets mentioning Biden, there are 161.001 (61,36 %) tweets without information about the state and 134.6977 (51,34 %) without data on the country. Therefore, it is difficult to make any statement about the number of tweets regarding a candidate with respect to the states in the US or even the users' country. Hence, we will investigate the sentiments in the tweets regardless of where a user is located.

3.3 Preprocessing

In order to clean our data, we use various helper functions. These fulfill numerous different tasks that will be described briefly in the following section. First, we have to eliminate all tweets that are not written in English language. As mentioned in section 3.1, our data sets contain tweets of various languages, as they are not distinguished

by the language in which they are written. Since the algorithms and directories that we use are mainly made for the English language and contain English words, we have to delete all those tweets which are not written in English. In addition, we have to remove breaks and parentheses, urls and links as well as special characters like "@, #, <, >" and such. Moreover, we set all letters to lower case. This is a commonly used procedure as seen in Akshat Bakliwal et al. (2012) as well. Furthermore, we remove stopwords in the data. In our every day language, these words are essential in making the language fluent, but they do not provide any information for the neuronal net. Saif et al. (2014) state that removing stopwords "helps maintaining a high classification performance".

Two other related methods we utilize to prepare the data for analysis are stemming and lemmatization. Stemming reduces words to their stem by removing affixes from words (Goyal et al. 2018). Thus, e.g. "studying" is reduced to "study". Lemmatizing is a technique where one takes into consideration the morphological analysis of the words, and links the word back to its lemma. As a consequence, the result is a real word. Thus, "went" will be transformed into "go". Balakrishnan & Lloyd-Yemoh (2014) compare both techniques and examine whether a model performs better if one of the techniques is applied to the data in advance. They conclude that stemming and lemmatization perform better than the baseline model, and lemmatiziation outperforms stemming. However, the difference between the precision of both techniques is not significant. We use the stopword list, stemmer, and lemmatizer from the *nltk* package (Bird et al. 2009). The algorithms used here are based on the PorterStemmer and WordNetLemmatizer respectively.

In addition, there are inconsistencies in the data as the state name for the United States of America occurs in different expressions like "United States of America" and "United States". To remove this inconsistency, we replace these by "USA". Furthermore, we calculate the number of words in a Tweet as well as the length of a Tweet. A sentiment analysis is performed by calculating the subjectivity and polarity score of a Tweet based on the functionalities of the *TextBlob* package (see section 2.1).

3.4 Labelling

Since our data sets do not come with any labels or categories to train a NN, we try different approaches to find suitable labels for the data. The provided labels are positive (1), negative (2), and neutral (0) or positive (1), slightly positive (3), negative (2), slightly negative (4), and neutral (0). The labels are represented by numbers to allow comparison of the different methods. The labels are chosen as described above, so that the designation for positive, negative and neutral is always the same, regardless of the number of categories. The fact that the numbers of slightly positive and positive as well as slightly negative and negative do not directly follow each other is considered acceptable.

The first approach uses the polarity score of the sentiment analysis that is provided by using the *TextBlob* package. It is performed on the tweets after the cleaning process, removing the stopwords and performing stemming or lemmatization. The classification uses the following cut-off points:

$$analysis3 = \begin{cases} score \leq -0,33, & negative\ (2) \\ -0,33 < score \leq 0,33, & neutral\ (0) \\ score > 0,33, & positive\ (1) \end{cases}$$

$$analysis5 = \begin{cases} score \leq -0,6, & negative\ (2) \\ -0,6 < score \leq -0,2, & slightly\ negative\ (4) \\ -0,2 < score \leq 0,2, & neutral\ (0) \\ 0,2 < score \leq 0,6, & slightly\ positive\ (3) \\ score > 0,33, & positive\ (1) \end{cases}$$

The next approach is to use clusters, identified by applying the k-means-clustering algorithm, as labels. The precise method is explained in section 2.2, and the outcome is discussed in section 4.1.2. The algorithm is used once with tweets that have only been cleaned, once again after removing stopwords, and once more after additional stemming or lemmatization. Furthermore, the number of clusters (k) is set to three or five.

As a third approach, the extraction of the emojis in the raw tweets is taken into consideration. However, only 75.659 of the tweets, which are 10,83 % of the overall number of tweets, contain emojis. Therefore, this approach is not pursued further.

Furthermore, smileys (faces) are extracted from the tweets, and the classification into positive and negative emotions is conducted by using the following (template).

Table 2: Classification Emotions

Positive emotions	$:),\ (:,\ :\ ),\ :D,\ :-D,\ :-),\ :],\ :-],\ =]$
Negative emotions	$:(,\ ):,\ ;(,\ =(,\ :-(,\ :/,\ :c,\ :[,\ =[$

Each time a positive emotion is found, a count variable is increased by one. Similarly, this count variable is decreased by one every time a negative emotion is found. The resulting categorisation is conducted as follows:

$$faces3 = \begin{cases} score < 0, & negative\ (2) \\ score = 0, & neutral\ (0) \\ score > 0, & positive\ (1) \end{cases}$$

$$faces5 = \begin{cases} score < -2, & negative\ (2) \\ -2 \leq score < 0, & slightly\ negative\ (4) \\ score = 0, & neutral\ (0) \\ 0 < score \leq 2, & slightly\ positive\ (3) \\ score > 2, & positive\ (1) \end{cases}$$

In a next step, the results of the different methods are compared, and an attempt to generate a common label is made. Furthermore, when training a NN, a distinction in the labels betwéen the tweets belonging to Biden or Trump has to be made.

4 Findings and discussion

The following chapter presents the different results and discusses them. In a subsequent step, the problems of creating a collective label are highlighted. Subsequently, the different models that have been fitted are evaluated, and special features are illustrated, and the model that performs best is identified.

After having a first look into the data, it is easy to see that there are diverse difficulties to face. First, we have to apply an appropriate strategy of preprocessing. We have to deal with the typical problems that occur in Twitter data. Since the postings are limited to 280 characters, the users utilize abbreviations and directly include hashtags in their text. Since the algorithms we use are not capable of processing special characters, we remove them. Further, we exclude both non-English tweets and stopwords. Excluding the latter can lead to a large improvement in the performance of algorithms, as can be seen in the clusters (see section 4.1.2). These improvements and their relevancy will be discussed in the suitable sections. Another problem mentioned in section 3.2 is that the majority of the tweets do not provide sufficient information on the users' location. In consequence, a limitation to tweets that were posted from inside the US was not possible without excluding a relatively large number of tweets which likely have been posted in the US.

A further idea is to extract tweets that were posted by the candidates themselves and analyse them in certain ways. However, the data sets do not contain tweets that were published by the candidates themselves. Hence, is was not possible to analyse them. A possible explanation is that the candidates simply did not use the relevant hashtags.

4.1 Creating Labels

In this section the results of the sentiment analysis performed are discussed (see section 4.1.1). Secondly, the outcome of the k-means clustering is reviewed (see section 4.1.2). Thirdly, the approach with the extracted faces and the associated emotions are considered (see section 4.1.3). Finally, a comparison of the various results is made and the way of choosing a shared label is described (see section 4.1.4).

As described in section 3.4, the idea to extract and use emojis to create labels was rejected, because only a small share of the tweets (10,83 %) contain at least one emoji. This seems to be a common finding, since Kralj Novak et al. (2015) investigated the sentiment of emojis in Twitter data and found that only 4 % of their data contained emojis.

4.1.1 Discussion Sentiment Analysis

In the following section, the results of the sentiment analysis are discussed. For this purpose, a short summary as well as a presentation of the constraints encountered is given in the first part. In the second part, the results are presented and reviewed in detail.

We use the *TextBlob* library to classify our data as positive, negative, and neutral to perform a sentiment analysis (see section 2.1). The classification is performed as described in section 3.4. This analysis has to be performed on the data set that contains only tweets written in English, because *TextBlob* is designed for the English language.

Figure 1 shows the result of this first classification. The major portion of the tweets were classified to have a neutral sentiment (0). The share of negative tweets (2) is larger in the Trump data set, whereas the share of positives (1) is larger than the share of negatives in both data sets.

(a) Biden (b) Trump

Figure 1: Sentiment Analysis

The data used in this first step was cleaned but not preprocessed further. Hence, the data contain stopwords and potential inconsistencies. Since this first trial led to a large share of neutral tweets, we try to improve the analysis by using various steps of preprocessing. Table 3 shows the result of the classification via *TextBlob* after the different steps of preprocessing into the three categories cases as described in section 3.4. The absolute and relative numbers of tweets that have been classified to the according sentiment are given.

Table 3: Results of the Sentiment Analysis for three categories

	Variable	cleaned Text	removed stopwords	stemming	lemmatization
Biden	positive	41.095 (0,157)	39.747 (0,151)	51.799 (0,197)	44.733 (0,170)
	negative	14.348 (0,055)	16.421 (0,063)	13.854 (0,053)	15.286 (0,058)
	neutral	206.944 (0,789)	206.219 (0,786)	196.734 (0,750)	202.368 (0,771)
Trump	positive	57.737 (0,132)	54.801 (0,126)	75.243 (0,173)	59.421 (0,136)
	negative	31.357 (0,072)	36.392 (0,083)	29.618 (0,068)	34.901 (0,080)
	neutral	346.849 (0,796)	344.750 (0,791)	331.082 (0,759)	341.621 (0,784)
Total	positive	98.832 (0,142)	94.548 (0,135)	127.042 (0,182)	104.154 (0,149)
	negative	45.705 (0,065)	52.813 (0,076)	43.472 (0,062)	50.187 (0,072)
	neutral	553.793 (0,793)	550.969 (0,789)	527.816 (0,756)	543.989 (0,779)

As visible in the relative shares, removing the stopwords only has a small impact on the classification results. This can be found in both the data set regarding Trump and the one regarding Biden. Stemming and lemmatization are able to reduce the number of tweets classified as neutral slightly. However, there are more tweets that are rated positive than negative. The number of positives is even increased by applying stemming or lemmatizing. Furthermore, the number of positives is larger in the Biden data, compared to those in the Trump data, which has a larger share of negatively classified tweets. The share of neutrals dominates the tweets in both data sets, and has approximately the same size. Since we still have a very large share of neutral tweets, it seems to be less sensible to use a more detailed split of the data (see Table 12). Furthermore, the results can also be represented graphically in terms of the candidates' support (see Figure 7), show the polarity score for the different states of the US (see Figure 9, 10 and 12), the average score (see Figure 11) or show relation between the polarity and subjectivity (see Figure 13).

4.1.2 Discussion Clustering

As described in section 2.2, we apply a clustering algorithm to our data. To find a suitable solution that provides appropriate labels, we try different ways of preprocessing before applying K-means and different numbers of clusters. The goal is to find labels to train a model on. Therefore, suitable clusters should reflect specific topics or sentiments in the Tweets. In a first trial, we use the cleaned data for each candidate separately. Subsequently, no stopwords have been removed and no stemming or lemmatizing has been performed. When looking at the main terms in each cluster, it is to mention that these are dominated by meaningless words like "is, for, to, and, it, ... " (see Table 13 and 14). The names of the candidates occur but there is no pattern observable.

Table 4: Three clusters for Biden after stemming

Cluster 1	Cluster 2	Cluster 3
joebiden	joe	biden
vote	biden	vote
kamalaharri	joebiden	trump
presid	vote	win
elect	presid	amp
trump	hunter	elect
america	trump	go
amp	elect	presid
democrat	via	like
hunterbiden	go	get

After removing the stopwords, the clusters appear to be more meaningful. Table 4 shows the ten most frequent terms in the Biden data after stemming and for three clusters. However, when having a closer look into the terms, there is no clear sentiment for each cluster identifiable (see Table 15, 16, 17, 18 and 19). Seemingly, this approach

does not give appropriate labels for our data. The only way to compare the results is to compare the relative sizes of the clusters and evaluate if they are similar or comparable for the candidates (see Table 20 and 21). It is possible to see similarity for some clusters, but, overall, it can be said that the relative numbers for the different clusters and candidates share the same relative numbers.

4.1.3 Discussion Extract Faces

Besides using *TextBlob* and the clustering, we try two further approaches to find suitable labels for our data. These are using faces, since those reflect the feeling a user wanted to express when posting the Tweet. Emojis or faces have been known to be used to express feelings, which are easier to illustrate this way or to enhance the statement's meaning (Ayvaz & Shiha 2017).

We extract faces in the Tweets as described in section 3.4, and divide them according to the emotions they imply. We count the number of positive and negative faces in the Tweets and calculate a score based upon that. At this point, it has to be questioned whether the choice of method for calculating the score is appropriate. Is it possible to simply weigh the different faces against each other?

Table 5: Results of the faces approach for five categories

	positive	**slightly positive**	**negative**	**slightly negative**	**neutral**
Biden	2 (0,000)	195 (0,001)	618 (0,002)	144.804 (0,552)	116.768 (0,445)
Trump	1 (0,000)	223 (0,001)	1.234 (0,003)	251.265 (0,576)	183.220 (0,420)
Total	3 (0,000)	418 (0,001)	1852 (0,003)	396.069 (0,567)	299.988 (0,430)

Here, it is noticeable that the relative numbers for the Tweets that are detected as (slightly) positive are the same for both candidates. However, the relative number of Tweets identified as negative, and as slightly negative Tweets, is higher in the data set for Donald Trump. Thus, the relative number of neutral Tweets in the data set containing Tweets with respect to Joe Biden is higher. However, the trend in both data sets is the same, a low number of slightly positive Tweets, and almost no positive Tweets. The number of Tweets that are identified as slightly negative is the highest, followed by the number of neutral Tweets.

4.1.4 Creating a collective label

Since our data was not labeled in advance, we have to find a sufficient strategy to find labels for our data. In the following paragraph, we describe the process of generating the labels we finally used.

As mentioned in section 4.1.2, the cluster analysis did not lead to any clusters to be associated with a clear sentiment. Furthermore, it was not possible to identify a clear trend when looking at a confusion matrix, to compare the results with the findings of the sentiment analysis or the faces (see Table 6, 23, 24 and 25). Hence, the outcome of the cluster analysis is not taken into account any further.

Moreover, when taking a closer look at the approach with the faces, it needs to be considered that only 57,11 % of the Tweets include this way of sharing emotions. All the other Tweets are automatically labelled as neutral, nevertheless the percentage of negatively labelled Tweets is higher. This is an interesting observation, because when looking at the results of the sentiment analysis, the percentage of negatively labelled Tweets is always the lowest. To analyse if a clear trend between these two methods is identifiable, the confusion matrices are computed. Therefore, only the three-category approach is compared, because no impact is trackable by using five categories as described in subsection 4.1.1.

Table 6: Confusion matrix for sentiment analysis after removing stopwords and faces for three categories

		Faces		
Sentiment analysis		**neutral**	**positive**	**negative**
	neutral	231.163	322	319.484
	positive	43.901	82	50.565
	negative	24.924	17	27.872

The confusion matrix gives an overview of how an algorithm has classified the different inputs and can be understood as a form of truth matrix (Raschka & Mirjalili 2018). The size of the confusion matrix is $n \times n$, where n is the number of possible categories (Visa et al. 2011). In our case, we have $n = 3$. The values for *true positive* (TP), *true negative* (TN), *false positive* (FP) and *false negative* (FN) classifications of a classifier are specified (Raschka & Mirjalili 2018). The values on the main diagonal give the number of the correct specified objects, while the values off the main diagonal are the misclassified objects. Looking at the confusion matrix, it is obvious that almost all Tweets classified as positive by one method are not recognised as such by the other. This phenomenon is also observed when the sentiment analysis is performed on the basis of other preprocessing steps. Furthermore, more Tweets that are identified as neutral in the sentiment analysis are identified as negative in the face analysis. However, more than half of the Tweets classified as negative in one approach are also labelled as negative in the other. Nevertheless, there is a very wide dispersion and no clear trend (see Table 23, 24 and 25).

Based on these findings, the results from the sentiment analysis seem most reasonable to us. As a label, the results based on the cleaned text are chosen, as it is the middle of the frequencies.

4.2 Predicting sentiments

In the following section, the evaluation of the NNs is performed. Two different types of NNs are fitted, in the first one a normal embedding layer is used, and in the other one the pre-trained word embedding GloVe for Twitter data is integrated. These models are compared and the best model is identified by using a variety of evaluation methods. On top of that, problems that occurred are mentioned.

To complete the model specification given in section 2.3, the model has three output nodes, based on the findings in subsection 4.1.

1. Embedding layer (input size (nodes): 140; output size: None, 140, 100)
2. Dropout layer (dropout rate: 0,2; output size: None, 140, 100)
3. Dense layer (activation: 'softmax', output size: None, 140, 100)
4. LSTM layer (dropout rate: 0,2, output size: None, 100)
5. Dense layer (output size (nodes): None, 3)

The preprocessed Tweets after removing stopwords are used as input. Before fitting the model, the Tweets are tokenized and set to the same length, therefore, the input size for all is the same. The outputs are labels identified in subsection 4.1, these are defined as categories.

In order to compare and evaluate the models, different key measures are given which can be calculated utilizing the confusion matrix. Furthermore, the recall, the precision, the accuracy, and the F_1 score (classification score) are calculated (Chicco & Jurman 2020; Raschka & Mirjalili 2018; Visa et al. 2011). The F_1 score is a combination of precision and recall (TPR), which is often used in practice (Raschka & Mirjalili 2018). It is calculated as follows (see equation 8).

$$F_1 = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{8}$$

However, the loss is used as a criterion to measure the model's performance as well.

The split-off size for the test set is 20 %, subsequently, the training set contains 80 % of the data. As validation set 20 % of the training set is considered. As shown in Table 26, the distribution of the labels is reproduced in the test and train set. The batch size is set to 64 and the numbers of epochs to five.

4.2.1 Evaluation of the models without pre-trained embedding

In these models no weights are pre-specified and all weights are trainable. As a first part of the evaluation the loss and accuracy are calculated for the test data set, these values are represented in the following table:

Table 7: Evaluation of the test data set using the model without pre-trained embedding

	Loss	**Accuracy**
Full	0,205	0,931
Biden	0,241	0,918
Trump	0,225	0,920

When taking a look at the loss and the accuracy we can see that the values do not differ much. Furthermore, the model was able to classify the majority of the Tweets correctly. This can be seen in the confusion matrix in table 8 as well. The further analysis is based on the model which is fitted using the data regarding Joe Biden.

Table 8: Confusion Matrix for model without pre-trained embedding

	Neutral	**Positive**	**Negative**
Neutral	39.842	987	583
Positive	1.633	6.550	7
Negative	712	31	2.133

It can be observed, that the precision (0,94), recall (0,96) and F_1 score (0,95) for the label neutral are the highest. The positive label has the second highest scores, which are 0,87 for precision, 0,80 for recall and 0,83 for F_1 score. For the negative category we got 0,78 for precision, 0,74 for recall and 0,76 for F_1 score. The overall F_1 score for the model is 0,92, which is lower than the F_1 score (0,96) when using the training set (see Table 28).

However, when taking a look at the loss (see Figure 14), we can see that the validation loss is not increasing, it remains almost constant over time. The training loss is increasing very fast in the beginning and after the first epoch of training it increases slower. Furthermore, the accuracy of the training set is increasing strongly and reaches after the five epochs of training approximately 0,95. The increase of the validation accuracy is much slower and after the third epoch it is even decreasing slightly. This observation indicates an overfitting problem. We suspect the data structure to be to complex for this relatively simple model. A more complex model with further layers might be able to face this problem.

4.2.2 Evaluation of the models with pre-trained embedding

In this type of model we use an pre-trained embedding matrix as weight matrix. These weight matrix describes the relationship between different words. However, some words of our data set do not occur in this matrix, for these words the weight to other words is set to zero. Therefore, it is important to take into consideration how many of the vocabulary items in relative proportion are not covered by the pre-trained embedding model. In case of the considered data these values are 24,58 % (Biden) and 26,79 %, so almost 75 % of the vocabulary is covered. For the further evaluation of the model the loss and accuracy will be calculated for the test set as well (see Table 9).

Table 9: Evaluation of the test data set using the model with pre-trained embedding

	Loss	**accuracy**
Full	0,638	0,797
Biden	0,637	0,789
Trump	0,637	0,797

When taking a look at this table the first thing that stands out is that the values of the loss are almost the same for all three data sets and the accuracy score for full set and the subset regarding Tweets to Donald Trump are exactly the same (0,797). The accuracy score for Joe Biden is 0,008 points lower, so it is still very close. Hence, we can see that the model performs almost equally well for the different data sets and the general performance is not bad. The further evaluation will be based on the model fitted with the data regarding Donald Trump.

Table 10: Confusion matrix for model with pre-trained embedding

	Neutral	**Positive**	**Negative**
Neutral	69.473	0	0
Positive	11.372	0	0
Negative	6.344	0	0

The confusion matrix shows an interesting result, all the Tweets from the test set are predicted as neutral. Based this result on one the hand the recall, precision, and F_1 score are zero for the labels positive and negative. On the other hand for the category neutral the model performs very well, the precision is 0,80, the recall 1, and the F_1 score 0,89. When looking at the confusion matrix using the training set we can see the same result, all inputs are predicted as neutral (see Table 27). Even when the model performance looks not to bad at the beginning, when taking a closer look one can that the accuracy is only that high because the relative proportion of the label neutral lies by almost 80 %. The accuracy score is almost constant for the over the five epochs of training for the training set as well as for the validation set. Moreover the loss is decreasing after the first epoch of training and stays afterwards almost constant (see Figure 15). These problem occurs probably because of the very basic model and the NN is not able to learn.

4.2.3 Conclusion neural networks

Comparing both models the one without the pre-trained embedding performed better than the other one, since it was able to classify the majority of the labels correctly and achieved better scores. It should be mentioned that the high number of correctly specified number of neutral Tweets has always be considered with the distribution on the labels in mind. Still the same problems occur in both models. As a limitation for the whole structure of the work the selection of the data, or more precisely the distribution of the labels, has to be considered. As already described in subsubsection 4.1.4, the data is highly unbalanced. However, in terms of unbalanced

data the accuracy score can not be considered as a reliable measurement score, because the score of the major class is overoptimistic (Chicco & Jurman 2020). Hence, it should be noted that the reliability of the F_1 score decreases with unbalanced data as well (Chicco & Jurman 2020).

Furthermore, the relatively low complexity of the model causes other problems like the overfitting, but a the low complexity provides a more robust model and leads to shorter computation time. In the model with the pre-trained embedding layer a more sophisticated preproseccing might be able to improve the performance. In order to do so removing hashtags completely should be considered.

5 Conclusion and Outlook

In this paper, we analysed Twitter data on the US election 2020 and intended to train a Deep Learning model on it to see whether we can predict sentiments. After introducing the used methods in section 2 we gave an overview about the used data sets and how we processed it in the sections 3. Section 3.2 presents our results and discusses them. One major question when the data sets were created was if it is possible to predict the outcome of the election by analysing the Tweets (Hui 2020). To make any meaningful statement regarding this issue it is necessary to have representative data which means that the users represented in the data set should mirror the whole population of eligible voters. Unfortunately, we have to assume that this is not the case. Studies on the population of Twitter users found that they tend to be younger, wealthier, healthier and more educated than the average population. But more important to mention is the political orientation of Twitter users. According to the investigation of Wojcik & Hughes (2019), Twitter users are more likely to favour the democratic party. Furthermore, there is a small number of users who create the major number of Tweets. Therefore, their opinions and claims will be overrepresented in our data.

Nevertheless, we were able to get some interesting insights. Due to the mentioned problems of representativeness for the US population we did not try to predict the outcome of the election but rather tried to predict the sentiments in the data sets. We used two data sets, one containing Tweets regarding Donald Trump and one consisting of Tweets regarding his rival Joe Biden. After cleaning the data for duplicates, inconsistencies and non-English language, we were able to see that even though the number of Tweets in the Trump data set was higher, the Tweets in his rival's data set received more likes. Therefore, we concluded that the latter were able to generate more attention (see section 3.1).

Since our data did not come with labels to train a model we first had to find a way to label our data appropriately. In order to do so we tried different approaches such as the sentiment analysis from *TextBlob*, the K-Means-clustering or using the faces and emojis in the Tweets as a access to the feeling behind a Tweet. But before doing so the data had to be prepared by preprocessing. Removing stopwords led to a tremendous improvement in the performance of the applied techniques as it was easy to see in the cluster analysis. Furthermore, we considered stemming and lemmatizing the Tweets. It turned out to be difficult to find an appropriate way of labelling the

data. The sentiment analysis of *TextBlob* led to a large share of neutrally classified Tweets whereas is was not possible to assign meaningful labels based on the outcome of the cluster analysis. After evaluating the number of Tweets that contained emojis this approach was rejected due to the small number of Tweets with emojis. Using faces as a last approach was rejected as well after evaluating it in comparison to the sentiment analysis. Using the results from the sentiment analysis as labels we trained different models on the data and compared the performance and results. We trained a model consisting of the following layers: 1. Embedding layer, 2. Dropout layer, 3. Dense layer, 4. LSTM layer, 5. Dense layer. One model was fitted with an pre-trained embedding layer (using GloVe) whereas in the other one all weights were trainable. Regarding the pre-trained word embeddings is to note that they are restricted by the embedded words. Hence, a problem in using Tweets occur. Since the length of a Tweet is restricted to a relatively short number of 280 characters, users trend to write their contents as short as possible using abbreviations, neglect spaces or such. As a consequence, the pre-trained embedding matrix does not represent such expressions. Concluding, some remarks on the restrictions our methods have to face have to be mentioned. With the data sets at hand we had so face some difficulties that have been solved more or less. The major problem was finding suitable labels. Since our final labels were quite unbalanced regarding the representation of the different categories the model had a hard time to learn in an appropriate way. We used a rather basic model to predict the sentiments in the Tweets. At this job it surely is more constructive to use a more sophisticated model. However, this still will be limited by the quality of the data. On the other hand, the data can still be used to produce fascinating descriptive statistics and gain interesting insights this way.

6 Appendix

Table 11: Variables

Variable	Description	Data type
created_at	Date and time of tweet collection	datetime
tweet_id	Unique ID of the tweet	float
tweet	Full tweet text	object
likes	Number of likes	float
retweet_count	Number of retweets	float
source	Utility used to post tweet	object
user_id	User ID of tweet creator	float
user_name	Username of tweet creator	object
user_screen_name	Screen name of tweet creator	object
user_description	Description of self by tweet creator	object
user_join_date	Join date of tweet creator	object
user_followers_count	Followers count on tweet creator	float
user_location	Location given on tweet creator's profile	object
lat	Latitude parsed from user_location	float
long	Longitude parsed from user_location	float
city	City parsed from user_location	object
country	Country parsed from user_location	object
continent	Continent parsed from user_location	object
state	State parsed from user_location	object
state_code	State code parsed from user_location	object
collected_at	Date and time tweet data was mined from twitter*	object
hashtag	defines from with dataset the Tweet comes originally	object
ClearTweet	Tweet after the first stage of preprocessing	object
weekday	Day of the week (as number)	object
day_name	Day of the week	object
hour	Time of the Tweet	int
day	Day of the month	int
date	The Tweet's date	datetime
subjectivity	subjectivity score from TextBlob	float
polarity	Polarity Score from TextBlob	float
analysis	Stores the results of the Sentiment Analysis	int
cluster	Stores the results of the clustering	int
faces	Stores the results of the faces method	int
norm	Cleaed Tweet without stopwords	object
stem	Stemmed text	object
lem	Lemmatized text	object
language	Defines the languages of the Tweets	object

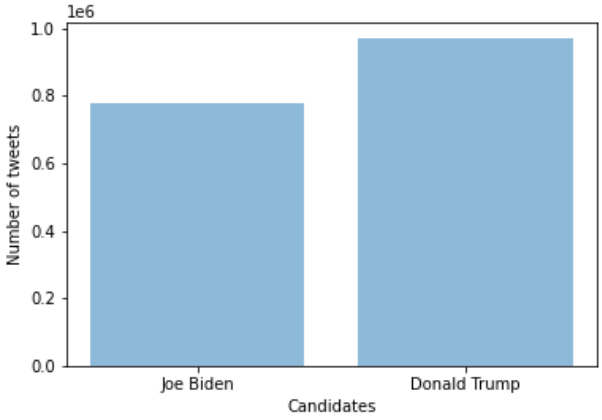


Figure 2: Number of Tweets per candidate

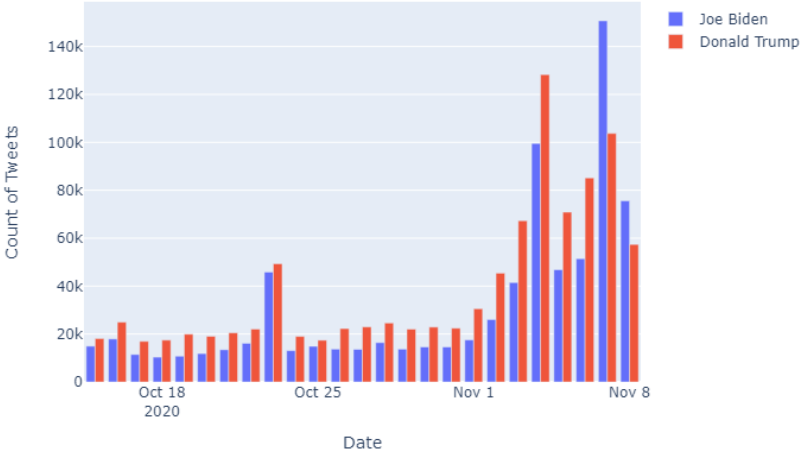


Figure 3: Daily number of Tweets per candidate

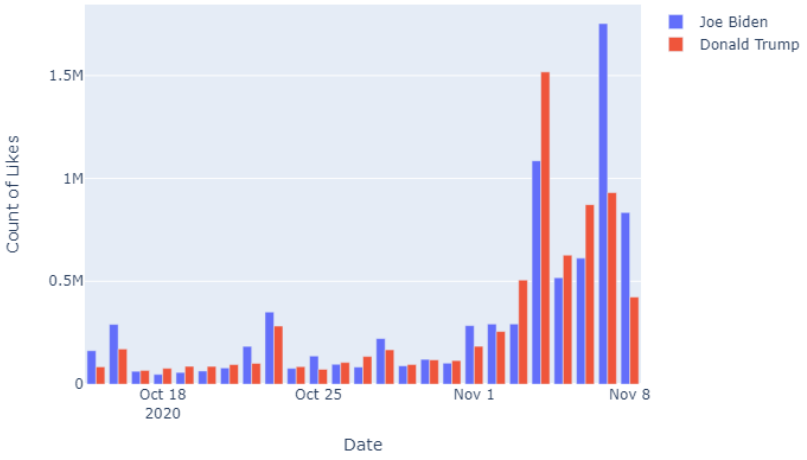


Figure 4: Daily number of Likes per candidate

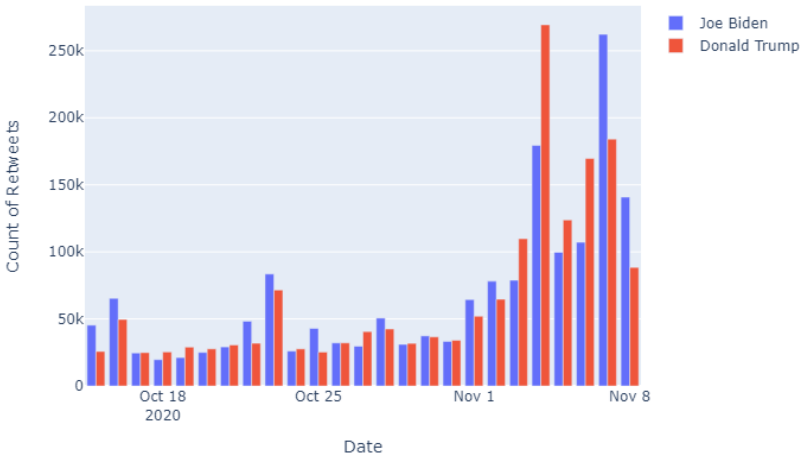


Figure 5: Daily number of Retweets per candidate

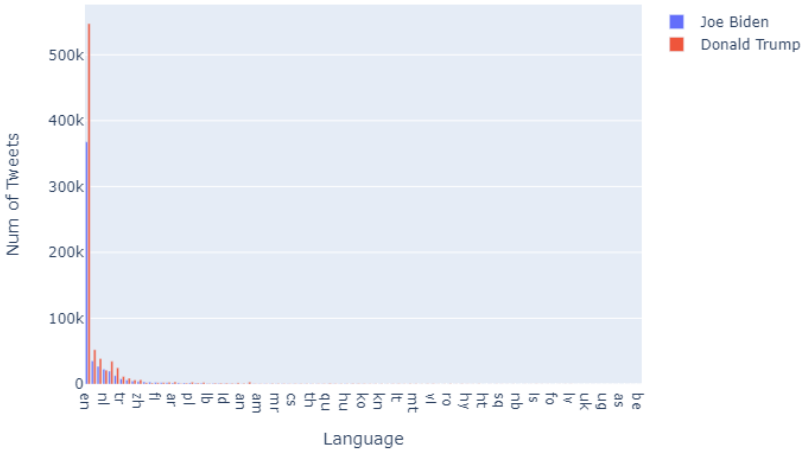


Figure 6: Occurrence of different languages in the full dataset

Table 12: Results of the Sentiment Analysis for three categories

	Variable	Cleaned Text	removed stopwords	stemming	lemmatization
Biden	positive	11.241 (0,042)	12.473 (0,048)	23.136 (0,088)	17.276 (0,066)
	slightly positive	52.971 (0,202)	48.250 (0,184)	45.219 (0,172)	46.351 (0,177)
	negative	3.980 (0,015)	4.705 (0,018)	3.608 (0,014)	4.245 (0,016)
	slightly negative	21.826 (0,083)	23.536 (0,090)	20.284 (0,077)	22.752 (0,087)
	neutral	172.469 (0,657)	173.423 (0,661)	170.140 (0,648)	171.763 (0,655)
Trump	positive	14.383 (0,033)	16.097 (0,037)	33.066 (0,076)	21.256 (0,049)
	slightly positive	77.198 (0,177)	69.328 (0,159)	67.418 (0,155)	66.545 (0,153)
	negative	10.288 (0,024)	12.328 (0,028)	9.594 (0,022)	11.438 (0,026)
	slightly negative	45.737 (0,105)	49.678 (0,114)	40.026 (0,092)	47.590 (0,109)
	neutral	288.337 (0,661)	288.512 (0,662)	285.839 (0,656)	289.114 (0,663)
Total	positive	25.524 (0,037)	28.570 (0,041)	56.202 (0,080)	38.532 (0,055)
	slightly positive	130.169 (0,186)	117.578 (0,168)	112.637 (0,161)	112.896 (0,162)
	negative	14.268 (0,020)	17.033 (0,024)	13.202 (0,019)	15.683 (0,022)
	slightly negative	67.563 (0,098)	73.214 (0,105)	60.310 (0,086)	70.342 (0,101)
	neutral	460.806 (0,660)	461.935 (0,661)	455.979 (0,653)	460.877 (0,660)

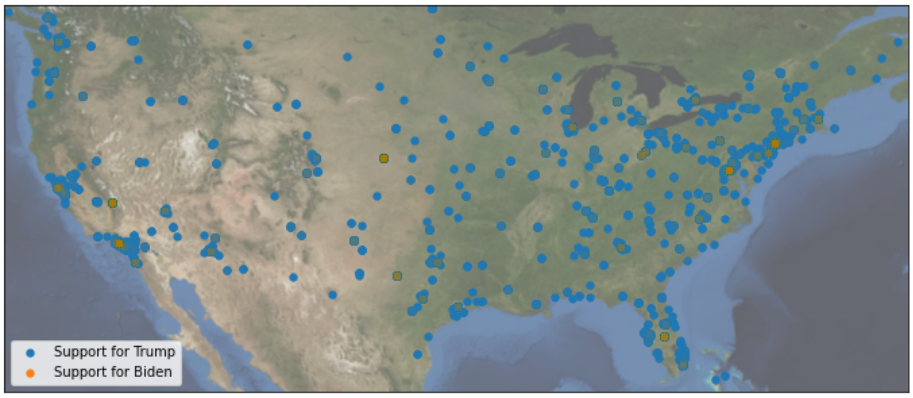


Figure 7: Graphically illustration for the support of the candidates

(a) Biden (b) Trump

Figure 8: Sentiment Analysis

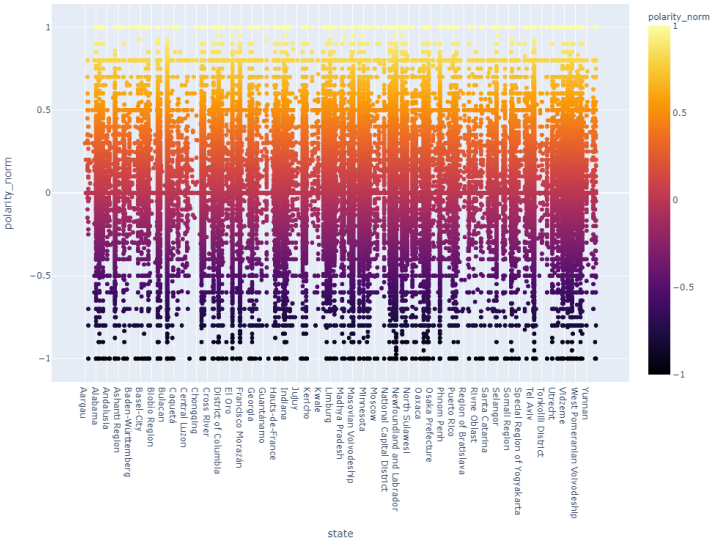


Figure 9: Polarity Score per state performing the Sentiment Analysis after removing stopwords

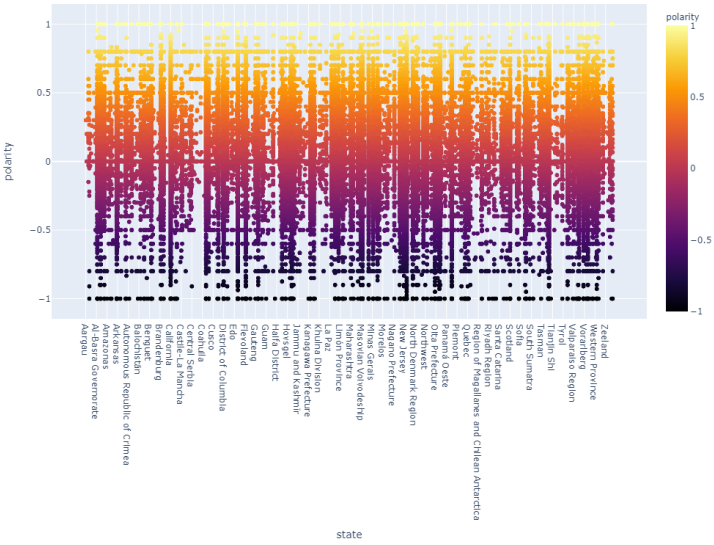
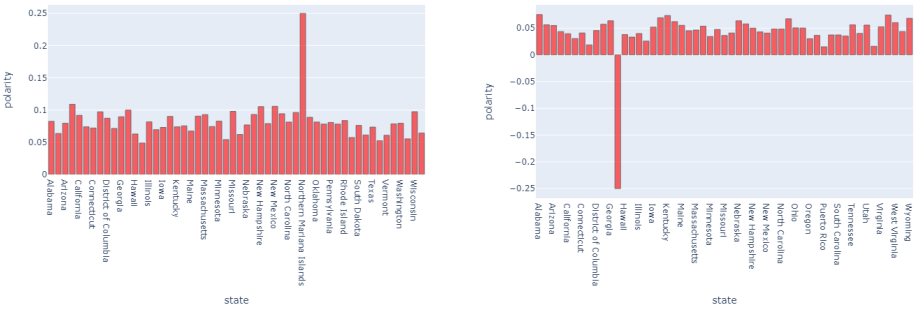


Figure 10: Polarity Score



(a) Biden (b) Trump

Figure 11: Average Polarity Score per state

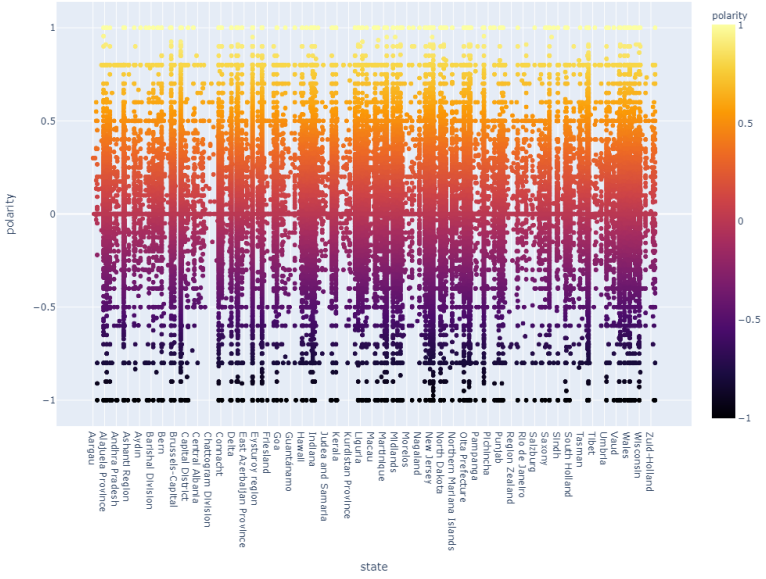


Figure 12: Polarity Score per state in the Trump dataset

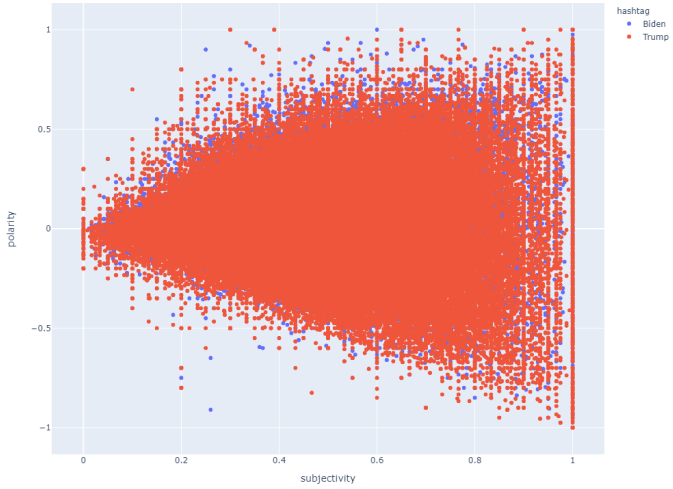


Figure 13: Polarity and Subjectivity score

Table 13: Three clusters for Biden

Cluster 1	Cluster2	Cluster 3
biden	you	the
joebiden	vote	to
is	for	is
for	to	and
to	biden	of
and	joebiden	biden
in	your	in
the	and	he
this	the	that
on	are	it

Table 14: Five clusters for Biden

Cluster 1	Cluster2	Cluster 3	Cluster 4	Cluster 5
joebiden	biden	you	the	he
for	is	to	to	is
is	for	for	of	the
vote	to	your	and	to
to	vote	the	is	biden
and	in	biden	biden	his
the	the	and	in	and
kamalaharris	and	vote	for	joebiden
this	this	are	that	that
in	trump	joebiden	it	in

Table 15: Three clusters for Biden after removing stopwords

Cluster 1	Cluster 2	Cluster 3
vote	joebiden	biden
biden	president	trump
joebiden	joe	amp
trump	kamalaharris	joe
please	trump	win
america	amp	president
kamalaharris	hunterbiden	votes
joe	like	election
voteearly	people	us

Table 16: Five clusters for Biden after removing stopwords

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
going	joebiden	biden	president	win
biden	vote	vote	biden	biden
joebiden	kamalaharris	trump	joebiden	amp
win	joe	amp	states	election
trump	amp	joe	united	joebiden
vote	hunterbiden	like	next	trump
amp	trump	bidenharris	joe	votes
get	america	people	trump	pa
like	people	votes	america	wi
people	like	election	vote	mi

Table 17: Five clusters for Biden after stemming

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
vote	elect	joebiden	people	biden
biden	biden	presid	american	trump
joebiden	joebiden	joe	biden	win
count	win	kamalaharris	joebiden	amp
trump	presed	hunterbiden	vote	joe
joe	vote	trump	amp	go
kamalaharri	trump	amp	trump	presid
go	us	go	presid	like
america	day	democrat	want	get
pleas	amp	america	like	vote

Table 18: Three clusters for Biden after lemmatization

Cluster 1	Cluster 2	Cluster 3
biden	joebiden	trump
vote	vote	biden
win	kamalaharris	vote
go	joe	joebiden
joe	president	amp
get	go	win
amp	get	donald
election	america	go
say	say	election
president	amp	president

Table 19: Five clusters for Biden after lemmatization

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
biden	want	go	joe	joebiden
vote	biden	biden	biden	vote
trump	vote	vote	joebiden	kamalaharris
win	joebiden	joebiden	vote	president
amp	trump	win	president	america
get	amp	trump	hunter	trump
election	president	get	trump	get
say	people	amp	via	amp
like	know	let	say	say
president	win	say	get	hunterbiden

Table 20: Results of the Cluster Analysis for three categories

	Variable	cleaned Text	removed stopwords	stemming	lemmatization
Biden	Cluster 1	131.152 (0,500)	21.934 (0,084)	30.524 (0,116)	147.700 (0,563)
	Cluster 2	20.047 (0,076)	145.503 (0,555)	92.663 (0,353)	96.825 (0,369)
	Cluster 3	111.188 (0,424)	94.950 (0,362)	139.200 (0,531)	17.862 (0,068)
Trump	Cluster 1	206.910 (0,475)	44.544 (0,102)	44.455 (0,102)	44.563 (0,102)
	Cluster 2	53.615 (0,123)	354.727 (0,814)	36.506 (0,084)	28.248 (0,065)
	Cluster 3	175.418 (0,402)	36.672 (0,084)	354.982 (0,814)	363.132 (0,833)
Total	Cluster 1	338.062 (0,484)	66.478 (0,095)	74.979 (0,107)	192.263 (0,275)
	Cluster 2	73.662 (0,105)	500.230 (0,716)	129.169 (0,185)	125.073 (0,179)
	Cluster 3	286.606 (0,410)	131.622 (0,188)	494.182 (0,708)	380.994 (0,546)

Table 21: Results of the Cluster Analysis for five categories

	Variable	cleaned Text	removed stopwords	stemming	lemmatization
Biden	Cluster 1	26.550 (0,101)	11.236 (0,043)	135.115 (0,515)	6.711 (0,026)
	Cluster 2	15.905 (0,061)	18.644 (0,071)	10.103 (0,039)	120.216 (0,458)
	Cluster 3	83.911 (0,320)	83.896 (0,320)	93.701 (0,357)	27.190 (0,104)
	Cluster 4	55.778 (0,213)	19.372 (0,074)	8.098 (0,031)	24.519 (0,093)
	Cluster 5	80.243 (0,306)	129.239 (0,493)	15.370 (0,059)	83.751 (0,319)
Trump	Cluster 1	123.773 (0,284)	40.951 (0,094)	20.115 (0,046)	274.351 (0,629)
	Cluster 2	40.203 (0,092)	25.748 (0,059)	42.844 (0,098)	41.522 (0,095)
	Cluster 3	59.145 (0,136)	26.955 (0,062)	36.977 (0,085)	58.892 (0,135)
	Cluster 4	45.164 (0,104)	33.641 (0,077)	34.217 (0,078)	35.448 (0,081)
	Cluster 5	167.658 (0,385)	308.648 (0,708)	301.790 (0,692)	25.730 (0,059)
Total	Cluster 1	150.323 (0,215)	52.187 (0,075)	155.230 (0,222)	281.062 (0,402)
	Cluster 2	56.108 (0,080)	44.392 (0,064)	52.947 (0,076)	161.738 (0,232)
	Cluster 3	143.056 (0,205)	110.851 (0,159)	130.678 (0,187)	86.082 (0,123)
	Cluster 4	100.942 (0,145)	53.013 (0,076)	42.315 (0,061)	59.967 (0,086)
	Cluster 5	247.901 (0,355)	437.887 (0,627)	317.160 (0,454)	109.481 (0,157)

Table 22: Results of the faces approach for three categories

	positive	**negative**	**neutral**
Biden	197 (0,001)	145.422 (0,554)	116.786 (0,445)
Trump	224 (0,001)	252.499 (0,579)	183.220 (0,420)
Total	421 (0,001)	387.921 (0,570)	299.988 (0,430)

Table 23: Confusion matrix for sentiment analysis with cleaned Tweet and faces for three categories

		Faces		
Sentiment analysis		**neutral**	**positive**	**negative**
	neutral	234.073	326	319.394
	positive	44.645	81	54.106
	negative	21.270	14	24.421

Table 24: Confusion matrix for sentiment analysis after Stemming and faces for three categories

		Faces		
Sentiment analysis		**neutral**	**positive**	**negative**
	neutral	220.568	303	306.945
	positive	58.707	99	68.146
	negative	20.623	19	22.830

Table 25: Confusion matrix for sentiment analysis after Lemmatization and faces for three categories

		Faces		
Sentiment analysis		**neutral**	**positive**	**negative**
	neutral	226.733	312	316.944
	positive	49.810	93	54.251
	negative	23.445	16	26.726

Table 26: Distribution Test and Train set for Biden dataset

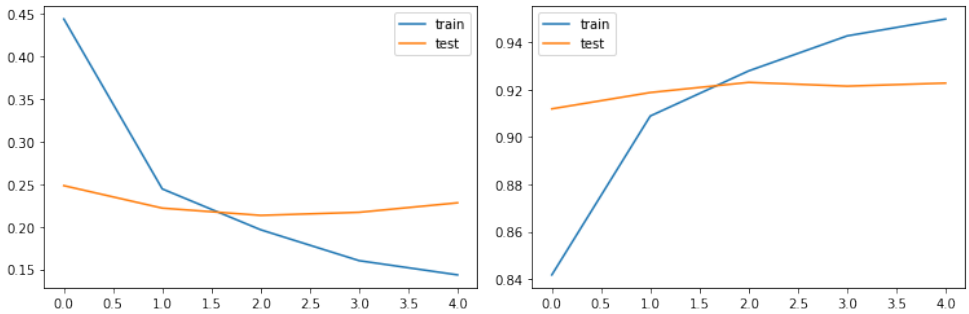
label	**positive**	**negative**	**neutral**
complete	15,676%	5,468%	78,870%
train	15,676%	5,465%	78,859%
test	15,607%	5,480%	78,913%

Table 27: Confusion matrix for model with pre-trained embedding for training set

	Neutral	**Positive**	**Negative**
Neutral	277.376	0	0
Positive	63.365	0	0
Negative	25.013	0	0

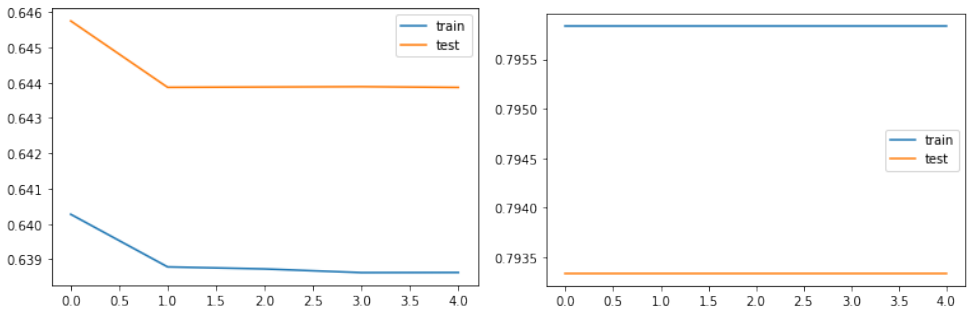
Table 28: Confusion matrix for model without pre-trained embedding for training set

	Neutral	**Positive**	**Negative**
Neutral	162.622	1.902	1.008
Positive	3.622	29.250	33
Negative	1.304	88	10.080



(a) Loss (b) Accuracy

Figure 14: Progress of the loss and accuracy



(a) Loss (b) Accuracy

Figure 15: Progress of the loss and accuracy using the pre-trained embedding

References

Akshat Bakliwal, Piyush Arora, Senthil Madhappan, et al. 2012, in Proceedings of the 3rd Workshop on Computational Approaches to Subjectivity and Sentiment Analy (Association for Computational Linguistic), 11–18

Ayvaz, S. & Shiha, M. O. 2017, International Journal of Computer and Electrical Engineering, 9, 360

Balakrishnan, V. & Lloyd-Yemoh, E. 2014, in Proceedings of SCEI Seoul Conferences, Seoul, Korea

Bird, S., Klein, E., & Loper, E. 2009, Natural Language Processing with Python, Safari Books Online (Sebastopol: O'Reilly Media Inc)

Chicco, D. & Jurman, G. 2020, BMC genomics, 21, 6

Chollet, F. 2018, Deep learning with Python, Safari Tech Books Online (Shelter Island, NY: Manning)

Daga, I., Gupta, A., Vardhan, R., & Mukherjee, P. 2020, Procedia Computer Science, 168, 123

Goyal, P., Pandey, S., & Jain, K. 2018, Deep Learning for Natural Language Processing: Creating Neural Networks with Python (Berkeley, CA: Apress)

Hui, M. 2020, US Election 2020 Tweets, `https://www.kaggle.com/manchunhui/us-election-2020-tweets`, accessed on 11/13/2020

Jan, T. G. 2020, in Lecture Notes in Electrical Engineering, Vol. 597, Proceedings of ICRIC 2019, ed. P. K. Singh, A. K. Kar, Y. Singh, M. H. Kolekar, & S. Tanwar (Cham: Springer International Publishing), 671–685

Kaggle. 2020, Your Machine Learning and Data Science Communtiy, `https://www.kaggle.com/`, accessed on Nov 20, 2020

Kralj Novak, P., Smailović, J., Sluban, B., & Mozetič, I. 2015, PloS one, 10, e0144296

Liu, B. 2012, Synthesis Lectures on Human Language Technologies, 5, 1

Loria, S. 2020

MacQueen, J. 1967, Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, 281

Micu, A., Micu, A. E., Geru, M., & Lixandroiu, R. C. 2017, Psychology & Marketing, 34, 1094

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. 2018, arXiv preprint arXiv:1811.03378

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, Journal of Machine Learning Research, 12, 2825

Pennington, J., Socher, R., & Manning, C. D. 2014, in Empirical Methods in Natural Language Processing (EMNLP), 1532–1543

Raschka, S. & Mirjalili, V. 2018, Machine Learning mit Python und Scikit-Learn und TensorFlow: Das umfassende Praxis-Handbuch für Data Science, Deep Learning und Predictive Analytics, 2nd edn. (Frechen: mitp)

Saif, H., Fernandez, M., He, Y., & Alani, H. 2014, REC 2014, Ninth International Conference on Language Resources and Evaluation, 810

Statista. 2020, Twitter - Monatlich aktive Nutzer weltweit 2019 — Statista, `https://de.statista.com/statistik/daten/studie/232401/umfrage/monatlich-aktive-nutzer-von-twitter-weltweit-zeitreihe/`, accessed on 11/23/2020

Twitter. 2020, Tweet location FAQs, `https://help.twitter.com/en/safety-and-security/tweet-location-settings`, accessed on 11/23/2020

Visa, S., Ramsay, B., Ralescu, A. L., & van der Knaap, E. 2011, MAICS, 710, 120

Weidman, S. 2019, Deep learning from scratch: Building with Python from first principles, first edition edn. (Sebastopol, CA: O'Reilly Media, Inc)
Wojcik, S. & Hughes, A. 2019

Multiclass Classification of German News Articles Using Convolutional Neural Networks

Joanna Simm, Sophie Potts

1 Introduction

The digital revolution has led to an increasing shift of information flows to the Internet. Newspaper agencies are also affected by this change and often offer their services in digital form. In this way, a large amount of data is created. The analysis of these data is of interest not only to the newspaper agencies but can also play a role for research. One of the possible analyses is to classify the newspaper articles into topics they cover. Such analyses can be a basis for news recommendation systems or can enable automatic tagging of articles for online news repositories (Chase et al. 2014).

The large amount of data leads to an increasing popularity of methods from the field of machine learning, such as deep learning approaches for natural language processing (NLP). Since deep learning methods are more complex and computationally demanding compared to traditional machine learning approaches, the question arises whether they are more effective and lead to better results. In the current research, deep learning approaches for text classification are mainly used for binary classification problems on English data (Xhemali et al. 2009; Neppalli et al. 2018). However, a multiclass classification presents a different challenge and needs more attention in the research. Additionally, the language of the data used in this paper is German, which may require different preprocessing approaches and neural network architecture. Our goal is to compare the performance of traditional and deep learning approaches on a classification problem with nine different classes. Therefore we compare the effectiveness of a Complement Naive Bayes (CNB) with a convolutional neural network (CNN) with self-trained and pre-trained word embeddings.

The used data set "Ten Thousand German News Articles Dataset" (10kGNAD) contains 10,273 newspaper articles from an Austrian newspaper and can be found on kaggle.com (Kaggle 2020). It is based on the data set generated by the Austrian Research Institute for Artifical Intelligence (Schabus et al. 2017). Articles were saved between June 2015 and May 2016 and are categorized into nine different categories: media, national territory, international, culture, panorama, sports, web, economy and science. The data set is imbalanced, which means that the distribution of articles over the various classes is not uniform since some categories include more news

articles than others. The largest category contains 1678 news articles whereas the smallest category holds 539 articles. Like Sun et al. (2009) discuss, imbalance issues in classification tasks can be a problem when they occur with different circumstances like a relative small sample size or overlapping patterns, always depending on the degree of imbalance. We will describe the used measures to counteract the problem of imbalance in the method section.

Following, the method section gives a short literature overview and an introduction to CNNs for text classification tasks. We continue with a description of the preprocessing procedure and some descriptive statistics of the data set. Before defining the models some comments on the importance and types of word embeddings are included. Subsequently, the results of the different models are presented.

2 Methods

In the current research, many approaches for text classification are proposed. Minaee et al. (2020) give a structured overview over different machine learning algorithms for text classification tasks. In addition to the classic approaches like Naive Bayes or Random Forest, authors mention deep learning models like recurrent neural networks (RNN) - particularly LSTM-cells, CNNs, attention mechanisms and transformers as useful architectures. Minaee et al. (2020) highlight the importance of the choice of word embedding type for the classification result.

Kim (2014) formulates CNN models for different text classification tasks and compares their performance when using self-trained and pre-trained word embeddings. The author concludes that the use of pre-trained word embeddings leads to better results compared to task-specific word embeddings. The model performance can be further improved when allowing the pre-trained embeddings to be trainable in order to fine-tune them for the specific task. Kim (2014) notes that even a simple one layer CNN model with non-trainable `word2vec` embeddings performs remarkably well.

Elnagar et al. (2019) compare the performance of deep learning models on Arabic news articles from three different newspaper agencies. Authors formulate RNN models, attention models and CNN-based models and conclude that the attention and CNN models perform best, with a top test accuracy of 95.81%.

Singh et al. (2020) formulate three classical models: Support Vector Machines, Naive Bayes and Random Forest and compare their performance with BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al. 2018) on balanced and imbalanced English data sets. Authors conclude that BERT outperforms classical approaches and that all formulated models achieve better results when being trained on a balanced data set. Singh et al. (2020) obtain an accuracy of 87% with BERT on a balanced data set.

Xhemali et al. (2009) compare the performance of a Naive Bayes and Random Forest classifiers with a neural network approach on a classification task with two classes. Authors conclude that Naive Bayes performs better than the other approaches. Ting et al. (2011) also test the performance of those algorithms in order to classify samples into four different classes and get the same results. Contrary to these findings are the results of Neppalli et al. (2018) that investigated the effectiveness of Naive

Bayes, CNN and RNN approaches on a binary classification task. Also Saritas & Yasar (2019) as well as Sunarya et al. (2019) conducted a similar study and show that neural networks perform better than Naive Bayes. Hence, the empirical findings are not homogeneous in their conclusions regarding the effectiveness of Naive Bayes and neural networks depending on the task. Most of authors test the performance of those algorithms on a binary classification problems. When dealing with imbalanced data sets, the use of Complementary Naive Bayes may lead to better results compared to Multinomial Naive Bayes (Rennie et al. 2003). However, the effectiveness of CNB and CNNs for a multiclass classification task on an imbalanced data set has not been compared yet.

To adress this research question, we compare the performance of a CNB with a deep learning approach. Based on the model performance of different deep learning models on our data set (see section 2.3), we decided to concentrate on CNNs, that will be discussed in short in the following. Additionally, in section 3.4, we will briefly describe other deep learning approaches like recurrent convolutional neural networks or distilBERT, that we implemented and report their performance.

2.1 Convolutional neural networks for text classification

CNNs perform very well not only on computer vision problems like image classification but can also produce very good results on sequence data like texts. The main advantage of CNNs are their significant lower computational costs compared to RNNs making them attractive for various projects with large data sets. They consist of several convolutional layers. In contrast to common dense layers convolutional layers do not work on the whole input but rather on a piece of it. This makes convolutional layers interesting for local pattern recognition. A strength of this approach is the translation invariance, i.e. once a pattern was recognized by the layer based on a smaller piece of the input it can be detected elsewhere in the input data without the need of learning it again (Chollet 2018). Pattern recognition is performed in a convolutional layer using filters of a prespecified size (see Figure 1). Those filters extract a subsequence of the input data and multiply it with weights that were learned by the model. One convolutional layer consists of multiple filters, that search for different patterns. The ability of pattern recognition will be used to detect patterns in news articles that allow the network to classify it to one of the nine categories.

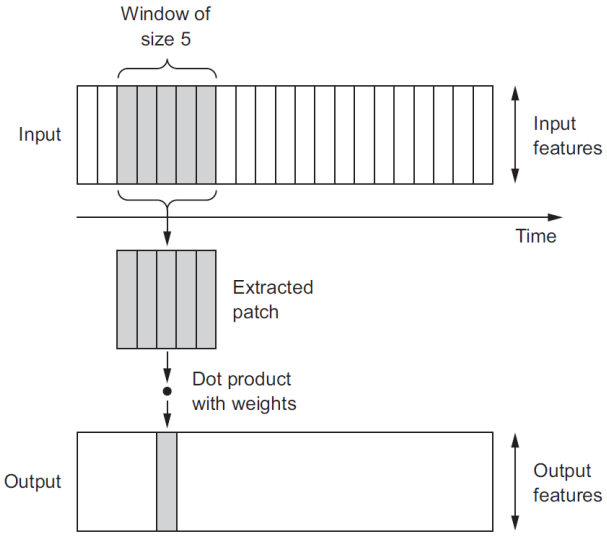


Figure 1: Mode of operation of a convolutional layer. Source: Chollet (2018)

In a CNN architecture the 1D convolutional layer is normally followed by a max pooling layer. Max pooling layers compute the maximal value in a subsequence by moving a pool of a prespecified size along the sentence. As discussed extensively in the literature, the problem of overfitting needs to be addressed. One prominent method to adress this issue to use regularisation techniques like the dropout methodologies (Srivastava et al. 2014). Dropout layers help to prevent the model from perfectly learning the training data. By applying a dropout to a layer, a prespecified share of the output features will be set to zero. This loss of information on training data may improve the degree of generalization on new data.

2.2 Preprocessing

For training the neural network, preprocessing is needed to transform the natural language text into a suitable input, that can be further processed by a neural network. First, punctuation, digits and double white spaces are removed from the text using the `string` library. In a next step, stop words are deleted. Since such words appear in every news article and do not point to category specific content, they have low discriminatory power and thus should not influence the prediction results. Their removal shortens the articles and reduces the vocabulary, leading to a shorter computation time. The list of stop words for German language is provived by the `spacy` library. We decided to use the `spacy` library instead of `nltk`, since the list of the latter is rather short and does not contain many words that should be included in the stop words list, e.g. past participle of verbs or conjunctive adverbs.

After these usual steps of preprocessing there are two different approaches to further transform the input data: lemmatization and stemming. The former is a method that removes the inflectional ending of a word and returns the dictionary form of

it, the so-called lemma. Stemming on the other hand reduces the word to its stem by cutting of the ending (Balakrishnan & Lloyd-Yemoh 2014). Both approaches lead to a reduction of the vocabulary – the list of unique words in the news articles – since originally differing words will be comprised to one. However, in a highly inflectional language as German lemmatization will reduce the vocabulary even more than stemming (see Table 1). Since the inflected forms all refer to the same general meaning of the word we decided to use lemmas. Furthermore, the usage of `Fasttext` word embeddings is only possible with lemmatized words.

Table 1: Stem and lemma of word-forms of "geben" (to give). Source: (Nicolai & Kondrak 2016)

Word form	Stem	Lemma
geben	geb	geben
gibt	gib	geben
gab	gab	geben
gegeben	geb	geben

After the lemmatization, the preprocessed news articles are split into training, validation and test data set. We first use a random split of 25% to obtain training and test data set. In the next step we divide the training set into training and validation set using a split of 20%. This results in three separate data sets, containing 6163, 1541 and 2569 news articles, respectively. The distribution of news articles by categories in each subset can be found in Figure 2.

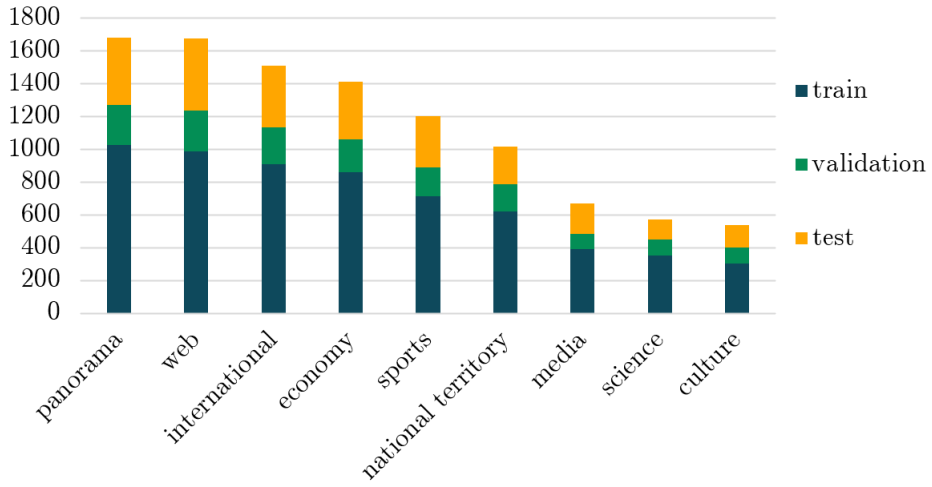


Figure 2: Distribution of news articles by categories and data sets

To enable the processing of the data by the model, the news articles have to be converted into numerical vectors. In a first step a tokenizer from the `keras` library is used. Tokenization describes the process of decomposing a piece of text into smaller parts, the so-called tokens, which are usually words. When tokenizing the news articles we decided to keep all words in the vocabulary, also those which appear only once in the data set. Such words can still have explanatory power, since synonyms can occur in the data set. This would result in similar word embedding vectors for the words and can be a useful information for the classifier.

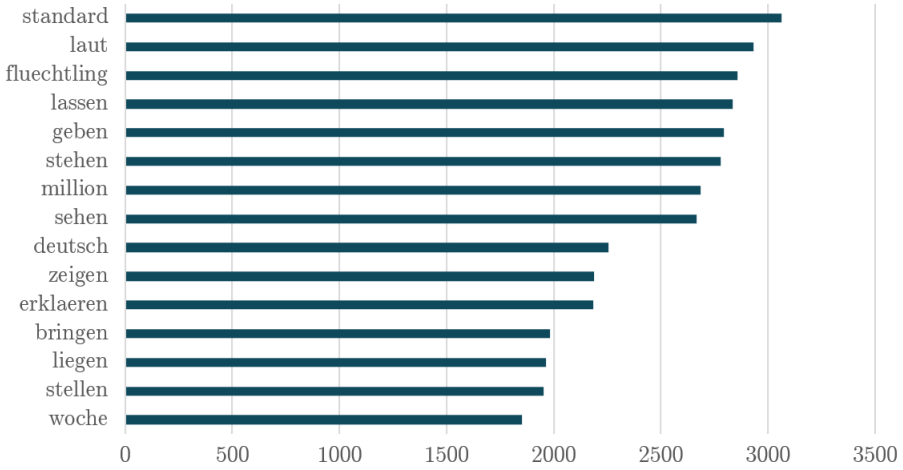


Figure 3: Sample count of the fifteen most frequent words

After the tokenization a vocabulary is created by assigning a unique number to each word. The vocabulary contains all 142,077 unique words appearing in the training and validation data sets. The fifteen most frequent words are visualised in Figure 3. In a next step, the news articles are transformed into numerical vectors by replacing each word with its vocabulary key. After that, the vectors are post padded to obtain the same length for each news article by filling the vectors of the shorter articles with zeros. We did not truncate the articles at a given number of words but used the length of the longest article as the value for padding. This results in a higher dimensional input but since we use CNNs instead of the computationally challenging RNNs we decided in favour of information retention. The final input for the neural network are the training data and validation data of shape (6163, 2136) and (1541, 2136), respectively. In the last step, the transformation of the response variable containing the categories of the articles is needed. On its basis an indicator matrix for all news articles is created.

After the preprocessing steps we also created word clouds for each category in order to study the most common words that occur in each of them. According to theoretical considerations some categories may be more difficult to classify than others. For example media might not have strong identifying words like sports. The word clouds for media, national territory, culture and panorama are displayed in Figure 4.

These categories share some frequently used words, such as Wien (Vienna), Österreich (Austria), Standard (name of the newspaper) or Prozent (percentage). This may lead to higher misclassification within these categories since they might not be well separated from each other. However, since we use CNNs in order to find patterns and not single words in the data, it may still result in good classification if those patterns are present.

Figure 4: Word clouds for media (upper left), national territory (upper right), culture (bottom left) and panorama (bottom right)

2.2.1 Word Embeddings

Word embeddings are a transformation of the language input into a numerical vector, i.e. they map human language into a low dimensional, geometric space. The vectors are chosen according to the semantic similarity and dissimilarity of words. This means that words with the same meaning get similar word representations whereas words with very different meanings get different word representations. Word embeddings are also able to represent the semantic distance of two words in terms of geometric distance (Chollet 2018).

The space of the word embeddings differs depending on the chosen language and the specific task. For example the optimal word embeddings for movie-review sentiment analysis will differ from those obtained for news classification. For this reason it is important to be able to learn word embeddings for a specific task. There are two ways of obtaining such word representation. One can either train them task specific in a neural network or use pre-trained word embeddings. The first option has the advantage that the embeddings are learned for this specific task but in order to obtain meaningful features it is important to have enough training data available. If this is not assured, pre-trained word embeddings can be used. They are usually obtained with large amount of data and trained unlabelled (Akbik et al. 2019; Chollet 2018).

We decided to follow both approaches. First, we rely on word embedding vectors of length 300 trained by the model using train and validation data. Second, we use pre-trained `FastText` word embeddings from the `flair` library. The obtained vectors are also of length 300 in order to be able to compare the performance of both models. The `flair` library makes different types of word embeddings available. We decided to use the classical word embeddings, which assign one vector per word, independent of the context, i.e. regardless of the previous or next words. The vectors were obtained via unsupervised learning on German Wikipedia articles (Akbik et al. 2019).

A peculiarity of the German language are compound words which are written without any separation sign. This can be problematic if the long compound words are not included in the embedding vocabulary. `FastText` solves this problem by considering subword information, i.e. by creating word embedding vectors for n-grams of subwords and summing them together. For example when considering a word 'Bestellbestätigung' (order confirmation), `FastText` creates vector representation for both subwords ('Bestellung' and 'Bestätigung') and sums them together. This is a very important advantage of `FastText` embeddings compared to e.g. `word2vec` embeddings, when working with German language (Gromann & Declerck 2018).

2.3 Defining the model

A classifier that assigns all news articles to the largest class would lead to approximately 16.3% correctly classified samples. As a baseline model (M0) we define a Complement Naive Bayes (CNB) approach as stated by Rennie et al. (2003). It is a transformation of the Multinomial Naive Bayes classifier and is useful when dealing with imbalanced data sets.

$$l_{\mathrm{MNB}}(d) = \mathrm{argmax}_c[\ln\, p(\boldsymbol{\theta}_c) + \sum_i f_i\, \ln\frac{N_{ci} + \alpha_i}{N_c + \alpha}] \tag{9}$$

$$l_{\mathrm{CNB}}(d) = \mathrm{argmax}_c[\ln\, p(\boldsymbol{\theta}_c) - \sum_i f_i\, \ln\frac{N_{\tilde{c}i} + \alpha_i}{N_{\tilde{c}} + \alpha}] \tag{10}$$

Instead of using the normal likelihood for the Multinomial Naive Bayes as given in equation 9, the maximum likelihood estimator of the parameter vector of class c is estimated via all training data except those in class c (see eq. 10). The sample is assigned to the class with the lowest complement weight.

In order to find the best neural network approach, we compare the performance of three very simple models, containing one LSTM, one Bidirectional LSTM and one CNN layer respectively. In all models we used pre-trained word embeddings and a dropout rate of 0.4. All three models consisted of approximately 30,000 trainable parameters. The results of the model preselection are displayed in Table 2. Since the simple CNN model performs by far the best with significantly lower computational costs, we decided to concentrate on this approach and to formulate CNN models that will be described in the following.

The *first* formulated model (M1) consists of an embedding layer, that trains vector representations of length 300, two convolutional layers with ReLU activation function, containing 24 and 48 filters of size four, respectively. Each convolutional layer is

Table 2: Model preselection

model	validation F1 score	test F1 score
LSTM	0.0073	0.0044
BiLSTM	0.5420	0.5378
CNN	**0.7978**	**0.7807**

followed by a max pooling layer with pool size four and a dropout layer with rates zero (no dropout) and 0.4 respectively. Finally, a global pooling layer is implemented to transform the intermediate results into a usable input for the last dense output layer. This output layer consists of nine neurons in line with the number of categories of the news articles. For the output layer a softmax activation function is used, which returns a vector of probabilities for each category.

As loss function we use the categorical cross-entropy which is the best choice for multiclass classification problems with C classes. It sums up the losses for each class label and is calculated as follows

$$\mathrm{CCE} = -\sum_{c=1}^{C} y_c \ln(\pi_c) \tag{11}$$

with y_c denoting the observation value and parameter π_c of a categorical distribution (Gordon-Rodriguez et al. 2020). The number of trainable parameter of this model is equal to 42,657,321. Since this is a very high number, there is a risk of overfitting. The model may be able to quickly memorize the training data, instead of recognizing the patterns in it, which could result in poor performance on validation and test data.

The *second* model (M2) has the same architecture as the first one with the difference that the embedding layer is non-trainable and uses pre-trained `FastText` word embeddings from `flair` library. We decided to compare these two models in order to asses the impact of the pre-trained word embeddings on the model performance. Since the model uses pre-trained word embeddings, this results in a much smaller number of parameters that needs to be trained, which may improve the generalization. This would be in line with findings of Kim (2014). Additionally, the pre-trained word embeddings provide powerful information about the semantic similarity and dissmiliarity of words. The number of parameters is reduced by 42,623,400, resulting in 33,921 trainable parameters.

Since we expect that some categories may be problematic to classify due to the shared common words we decided to look at this problem in detail. The common and not meaningful words that occur in the four mentioned categories (see Figure 4) are the following: Wien (Vienna), Österreich (Austria), Standard (name of the newspaper) and Prozent (percentage). We will remove them from the data and use the better of the two models above to clarify the impact of the most frequent words and whether their removal enables the model to get better classification results (M2a).

According to Kim (2014) the model performance can be further improved when additionally making the embedding layer trainable. This can lead to fine-tuning of the word representations adapted to our news classification task. The model with the trainable, pre-trained word embeddings is our *third* model (M3).

The parameters of the first model were obtained using a random search. The random search is used for hyperparameter optimization and according to Bergstra & Bengio (2012) it is able to find the best model in much shorter time compared to a grid search or a manual search. In the random search the following hyperparameter of the models were able to vary: the number of block of layers containing one convolutional layer, followed by a max pooling and a dropout layer, the number and size of filters in each convolutional layer, pool size of the max pooling layer, dropout rate and learning rate. The maximal number of such layer blocks was equal to five. The random search settings are displayed in Table 3. The models in the random search were trained for ten epochs with batch size of 128. The number of trials was equal to 15, which means that 15 combinations of hyperparameters were tested, with each model being trained three times. In order to reduce the running time of the random search we used early stopping, i.e. if the validation loss does not decrease in the following five epochs the training process is stopped.

Table 3: Optional hyperparameters in the random search

layer	hyperparameters	minimal value	maximal value	steps	choice
	learning rate				0.01, 0.001, 0.0001
Conv1D	number of filters	16	64	8	
	filter size	1	5	1	
MaxPooling1D	pool size	2	5	1	
Dropout	dropout rate	0	0.6	0.1	

We used the Adam optimizer (Kingma & Ba 2014). As a metric for model evaluation we apply the weighted F1 score. Since the data set is imbalanced a metric is needed that accounts for this imbalance. The weighted F1 score is calculated as follows

$$F1 = 2 \sum_{c=1}^{C} \frac{\mathrm{N}_c}{\mathrm{N}_{total}} \frac{\mathrm{precision}_c \cdot \mathrm{recall}_c}{\mathrm{precision}_c + \mathrm{recall}_c} \tag{12}$$

with N_c being the number of samples in class c, N_{total} being the total number of samples and C the number of categories to sum over (Hammerla et al. 2016). Precision of a label is defined as the proportion of true positive values divided by the the sum of true and false positive values. This measure can be interpreted as the relative share of correctly classified articles measured by the number of all articles with the same prediction label. Recall on the other hand gives the proportion of true positive values measured by all actual positive values (Powers 2020). F1 score is therefore a harmonic mean of recall and precision scores (Lipton et al. 2014). Since the data set is imbalanced, we are using a weighted F1 score, which is a weighted sum of the

individual F1 scores for each class according to their size. To further counteract the unbalanced nature of the data set we have added class weights. They ensure that correctly classifying news articles from smaller classes is equally important as correctly classifying news articles from larger classes. The weights are therefore inversely proportional to class sizes.

3 Results

The naive bag-of-words approach evaluated by Complement Naive Bayes on the preprocessed data yields an already high F1 test score of 84.34%. This is the baseline classification score for further models to be compared with.

3.1 Model with self-trained word embeddings

M1 is using self-trained word embeddings and the values of the hyperparameters were obtained with a random search. The model was trained for 15 epochs with batch size of 128. The results are displayed in Figure 5. The F1 training score reaches one after the third epoch, while the F1 validation score data stays around 0.7 and stops improving after the third epoch. Accordingly, the training loss reaches zero after three epochs, however the validation loss stays constant around 0.8. It seems that the model is able to learn to perfectly classify the training data, while the validation loss stops decaying.

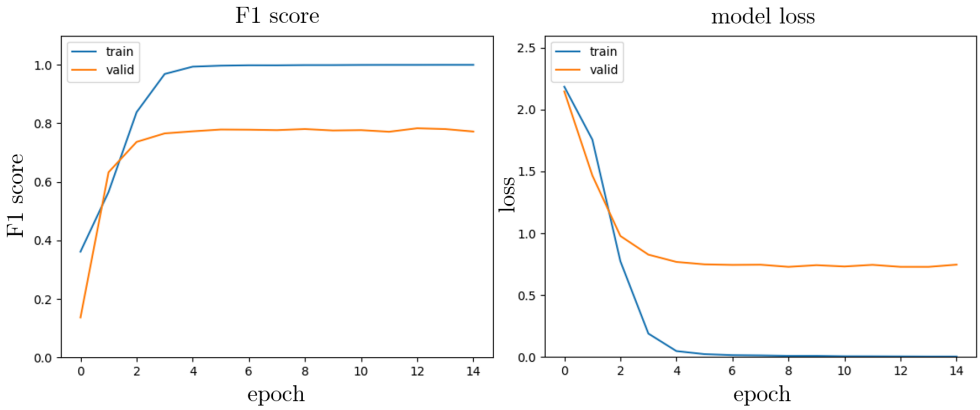


Figure 5: F1 score and model loss of the model with self-trained word embeddings

This is an indicator of overfitting, which can be caused by the high number of trainable parameters. We have also tried a subsequent manual fine-tuning of the hyperparameters of the model, but were not able to overcome the overfitting.

For model evaluation on test data, we used `ModelCheckpoint` from `keras` library. This function saves the best performance of a model according to the F1 validation score and uses it to evaluate the model on the test data. The selected model is able to reach an F1 score on test data of 78.06%. Figure 7 shows that the model

with self-trained word embeddings can classify over 70% of news articles from each category correctly. The best classified category is sports with 91% correctly classified news articles, followed by science with 81% correctly classified samples. As expected, the classes with the worst results are media, culture and panorama, although the outcomes for those classes are nevertheless over 70%. Contrary to our expectations, the percentage of correctly classified articles for the national territory category is high, despite the fact that the most common words in this category were also present in other categories. However, articles labelled as culture often get confused with media and national territory category which points to the problem of overlapping patterns in those categories.

3.2 Model with nontrainable pre-trained word embeddings

In this section the results of M2, the model with nontrainable pre-trained word embeddings, are described and compared with the performance of M1. The F1 score that was achieved for training and validation data is visualised in Figure 6, together with the values for the categorical cross-entropy loss function for both data sets. The F1 score for both training and validation data takes values about 0.8 and is slightly increasing over the epochs. This means that the F1 training score is lower compared to the previous model, while the F1 validation score increased. Furthermore, training and validation loss decay over the epochs, with training loss taking values around 0.4 and the validation loss being about 0.7 after the second epoch. In contrast to the first model with task-specific word embeddings, training loss decays slower and validation loss reaches lower values compared to the previous model.

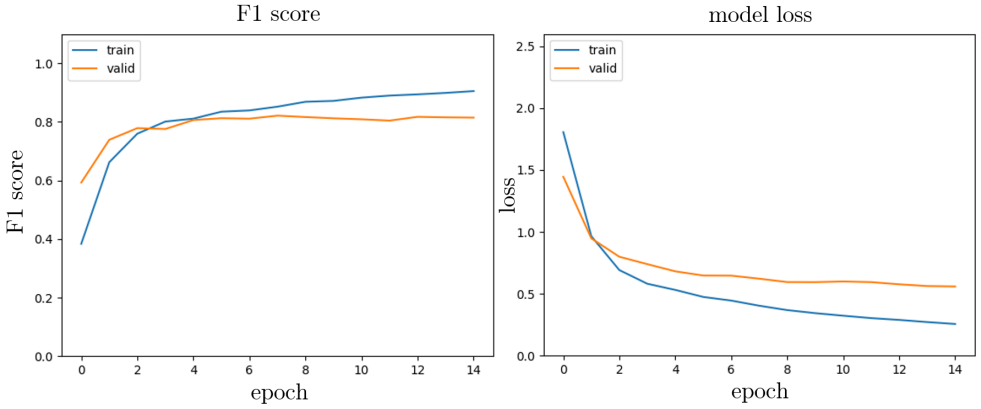


Figure 6: F1 score and model loss of the model with nontrainable pre-trained word embeddings

As expected, the model does not overfit anymore, since the number of trainable parameters was strongly reduced. The F1 score for validation data is slightly lower than for the training data and the losses do not differ much, with validation loss being slightly higher. This is a sign that the model fits the data just right and does not overfit or underfit.

The model with pre-trained word embeddings reaches an F1 test score of 81.15%, which corresponds to an increase of three percentage points compared to the previous model. The share of correctly classified articles from categories science, economy, web, sports, panorama and culture has increased, however, the changes for some categories are only minor. Despite the improvements, the categories media, national territory and panorama are still the worst classified categories. The classification result for the category culture has improved after using the pre-trained word embeddings, but the remaining three problematic categories still exhibit values around 70%. We hoped to see improvements after removing the common words from those categories. However, after excluding those words from the data and fitting the model M2a we were not able to obtain better results for the problematic categories, although the overall performance of this model is slightly better than for the nontrainable pre-trained CNN.

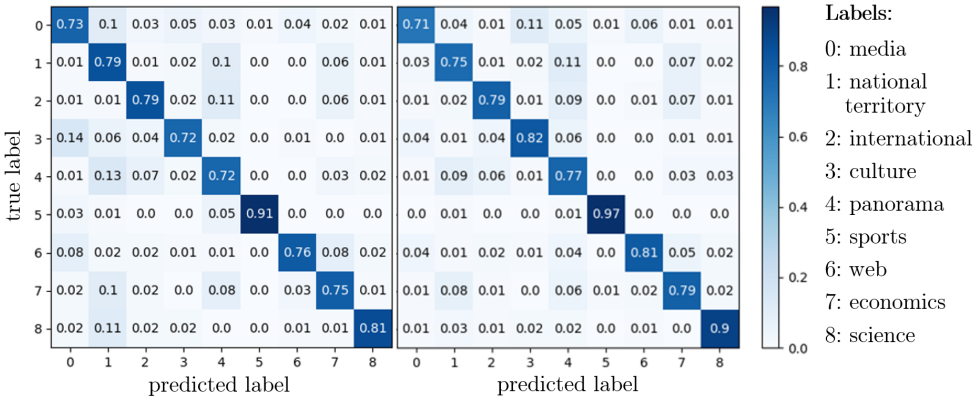


Figure 7: Confusion matrices of models with self-trained M1 (left) and nontrainable pre-trained word embeddings M2 (right)

3.3 Model with trainable pre-trained word embeddings

According to Kim (2014) the fine-tuning of the word embeddings for a specific task can lead to better model performance. In order to investigate this for our classification task, we formulated an additional model M3 that uses pre-trained word embeddings with embedding layer being trainable. Since the number of trainable parameters is again very large, we tried to prevent overfitting by changing the dropout rate of the first dropout layer from 0 to 0.3.

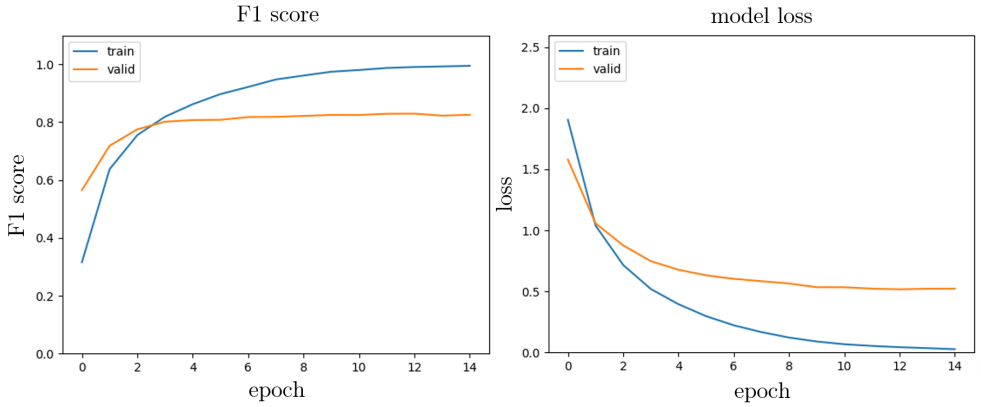


Figure 8: F1 score and model loss of the model with trainable pre-trained word embeddings

As shown in Figure 8 this model yields similar results for the F1 score and model loss for validation data as the previous model. However, the F1 score for the training data is higher than for the model with nontrainable pre-trained word embeddings, whereas the training loss is lower. This is a sign of weak overfitting.

M3 reaches an F1 test score of 83.09%, which is the top performance among all trained CNNs. It corresponds to an improvement of almost 2 percentage points compared to the previous model. As Figure 9 shows, this model leads to strong improvements in categories media, international and web. However it also worsens the results in the panorama and culture category.

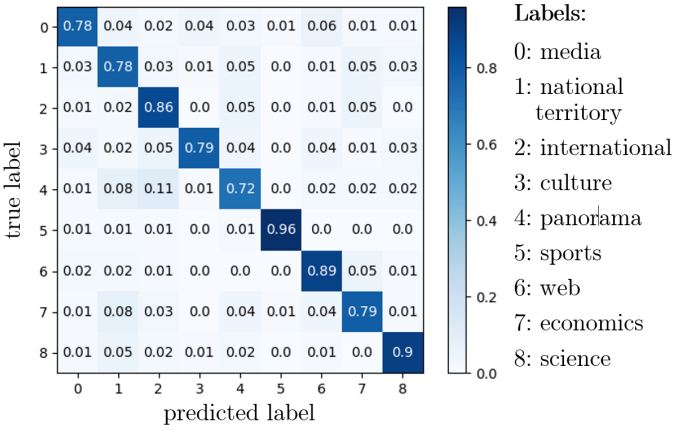


Figure 9: Confusion matrix of the model with trainable pre-trained word embeddings

3.4 Remarks on further models

In addition to the models formulated above, we implemented two further models that will be described in the following. All models and their performance are summarized in Table 4.

Table 4: Results

	model	test F1 score
M0	Complementary Naive Bayes (baseline)	**0.8434**
M1	CNN with self-trained word embeddings	0.7806
M2	CNN with nontrainable pre-trained word embeddings	0.8115
M2a	CNN with nontrainable pre-trained word embeddings w/o freq. words	0.8160
M3	CNN with trainable pre-trained word embeddings	**0.8309**
M4	CLSTM	0.8080
M5	distilBERT	0.6150

According to Elnagar et al. (2019) the combination of CNNs and RNNs, the so called recurrent convolutional neural networks may lead to good classification performance. Therefore we extended the model with nontrainable pre-trained word embeddings by an additional bidirectional LSTM layer with 16 neurons. However, this model (M4) did not outperform our best model and resulted in an F1 test score of 80.8%.

Another approach, that can lead to better classification results is the use of contextual word embeddings. Such word embeddings take the context of the word into account by analysing the previous and following words, leading to different word representations for the same word, depending on its context (Singh et al. 2020). BERT (Devlin et al. 2018) is a Google language model that enables the use of such word embeddings (Denk & Ramallo 2020). We built M5 with the German version of distilBERT, being a simplification of BERT (Sanh et al. 2019), and trained it for seven epochs with batchsize of 64 on Google Colab. This model reveals a surprisingly low F1 test score of 61.5% and does not outperform the model with non-contextual word embeddings. However, due to the very high computational cost, we were not able to further fine-tune this model. The bad performance of distilBERT can also be caused by the imbalance of our data set (Singh et al. 2020).

4 Conclusion

In this paper we investigated the usefulness of deep learning approaches on a multiclass classification problem on a German text data basis. Our goal was to classify news articles into nine different categories. Due to the particularities of the German language a careful preprocessing using lemmatization was needed to counteract the high amount of different inflected word forms. Furthermore, `FastText` pre-trained word embeddings were used in order to tackle the large flexibility of compound words.

A descriptive statistic of our data set revealed the existence of problematic categories, which was also reflected in the results for all formulated models. The categories media, national territory, culture and panorama have the lowest proportion of correctly

classified samples among all categories in almost all formulated models. We believe that those categories are similar in content, since for example media and panorama are more general categories that report on current events and news of all kind, which can lead to overlaps with other categories. If distinguishing patterns for these categories are not present in the data, the classifier will not be able to produce good results for those categories (Chollet 2018).

A comparison between a CNN with task specific, i.e. self-trained word embeddings and pre-trained word embeddings reveals a slightly higher (3 percentage points) weighted F1 score on the test data for the latter one. Furthermore, a task specific fine-tuning of the pre-trained word embeddings has lead to a further improvement of 2 percentage points, resulting in an F1 test score of 83%. Our results are in line with those obtained by Kim (2014). However, none of the CNNs outperform the Complement Naive Bayes bag-of-words approach. The best CNN model achieves a test F1 score of 83.09%, which is close to the performance of the Complement Naive Bayes (84.34%). This can be due to the fact, that CNB computes the probabilities based on simple counts of each word occurring in the news articles, which may already be a sufficient information for the classifier. In contrast, the CNN architecture takes word order and semantic information into account, which may not play an important role for some classification tasks.

Models with contextual word embeddings and appropriate fine-tuning of the parameters may be one of the further research steps to do in order to outperform the baseline model.

References

Akbik, A., Bergmann, T., Blythe, D., et al. 2019, in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations), 54–59

Balakrishnan, V. & Lloyd-Yemoh, E. 2014, Lecture Notes on Software Engineering, 2

Bergstra, J. & Bengio, Y. 2012, Journal of machine learning research, 13

Chase, Z., Genain, N., & Karniol-Tambour, O. 2014

Chollet, F. 2018, Deep learning with Python, Vol. 361 (Manning New York)

Denk, T. I. & Ramallo, A. P. 2020, arXiv preprint arXiv:2010.15778

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. 2018, arXiv preprint arXiv:1810.04805

Elnagar, A., Einea, O., & Al-Debsi, R. 2019, in Proceedings of the 3rd International Conference on Natural Language and Speech Processing, 59–66

Gordon-Rodriguez, E., Loaiza-Ganem, G., Pleiss, G., & Cunningham, J. P. 2020, arXiv preprint arXiv:2011.05231

Gromann, D. & Declerck, T. 2018, in Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)

Hammerla, N. Y., Halloran, S., & Plötz, T. 2016, arXiv preprint arXiv:1604.08880

Kaggle. 2020, Ten Thousand German News Articles Dataset., `https://www.kaggle.com/tblock/10kgnad`, accessed January 31 2020

Kim, Y. 2014, in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (Doha, Qatar: Association for Computational Linguistics), 1746–1751

Kingma, D. P. & Ba, J. 2014, arXiv preprint arXiv:1412.6980

Lipton, Z. C., Elkan, C., & Naryanaswamy, B. 2014, in Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 225–239

Minaee, S., Kalchbrenner, N., Cambria, E., et al. 2020, arXiv preprint arXiv:2004.03705

Neppalli, V. K., Caragea, C., & Caragea, D. 2018, in Proceedings of the 15th Annual Conference for Information Systems for Crisis Response and Management (ISCRAM)

Nicolai, G. & Kondrak, G. 2016, in Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 1138–1147

Powers, D. M. 2020, arXiv preprint arXiv:2010.16061

Rennie, J. D. M., Shih, L., Teevan, J., & Karger, D. R. 2003, in Proceedings of the Twentieth International Conference on International Conference on Machine Learning, ICML'03 (AAAI Press), 616–623

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. 2019, arXiv preprint arXiv:1910.01108

Saritas, M. M. & Yasar, A. 2019, International Journal of Intelligent Systems and Applications in Engineering, 7, 88

Schabus, D., Skowron, M., & Trapp, M. 2017, in Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), Tokyo, Japan, 1241–1244

Singh, R., Chun, S. A., & Atluri, V. 2020, in The 21st Annual International Conference on Digital Government Research, 354–355

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. 2014, The journal of machine learning research, 15, 1929

Sun, Y., Wong, A. K., & Kamel, M. S. 2009, International journal of pattern recognition and artificial intelligence, 23, 687

Sunarya, P., Refianti, R., Mutiara, A. B., & Octaviani, W. 2019, International Journal of Advanced Computer Science and Applications, 10, 77

Ting, S., Ip, W., Tsang, A. H., et al. 2011, International Journal of Software Engineering and Its Applications, 5, 37

Xhemali, D., J HINDE, C., & G STONE, R. 2009, D. XHEMALI, CJ HINDE and Roger G. STONE," Naive Bayes vs. Decision Trees vs. Neural Networks in the Classification of Training Web Pages", International Journal of Computer Science Issues, IJCSI, Volume 4, Issue 1, pp16-23, September 2009, 4

Detecting terrorist hate speech on Twitter

Markus Heidenreich, Nils Muttray

1 Introduction

1.1 Motivation

Terrorism has become a major threat to our societies in the 21st century. It is not only the sheer number of civilians that have fallen victim to a terrorist attack that is frightening, but also the impact that the resulting sense of threat has on our communities. Fear leads people to reinforce their original world views on both sides of the political landscape and to identify more easily with the opinions of extreme and radical parties (Castano et al. 2011; van Prooijen et al. 2015). In this way, terrorist attacks can contribute considerably to the increasing degree of polarization nowadays. The fight against terrorism of all kinds therefore remains of high importance. However, this is no longer a conflict that can be solved with conventional weapons in the affected combat zones solely. Modern communication channels have greatly changed the nature of terrorist warfare.

ISIS, for example, successfully used an innovative social media strategy in order to distribute its propaganda globally and attract additional supporters around the world (Farwell 2014). It is reported that up to 15,000 foreigners, including 2,000 Westerners, from over 80 countries have fought for ISIS in Syria (Jomana Karadsheh & Smith-Spark 2014). Especially Twitter has become a helpful tool for radical groups to share their extremist materials among an international network of followers. The structure of Twitter fits their needs perfectly. A core group of members usually communicates the group's message (tweets), which can then be spread further by other supporters (re-tweets) (Gialampoukidis et al. 2017). Although the number has dropped significantly since ISIS strength peaked in 2015, the amount of jihadist content on Twitter is still concerning. Ctrl-Sec - a group of independent cyber activists that exposes ISIS members on Twitter so they can be suspended - identified 50,883 new ISIS accounts on Twitter in 2020 (CtrlSec 2021).

The objective has changed a little since the demise of ISIS as a territorial power in the Middle East. The focus is now on radicalizing young and similar situated people to engage in terrorist activities on their own. Gill et al. (2013) argue that jihadist terrorist attacks by lone-wolfs are very often the result of an extreme radicalization on the internet. Such attacks are very difficult to prevent. It is almost impossible

to identify possible threats who do not actively belong to the actual terrorist group but are merely inspired by it (Johnston & Weiss 2017). This makes it all the more important to identify extremist material on the internet at an early stage so that it can be deleted or monitored by intelligence services.

The development of statistical models that can reliably classify terrorist content on Twitter thus represents a challenge of high relevance for current research. This work aims to recognize jihadist tweets in an ISIS-related dataset using deep learning models in the context of natural language processing. The structure of this paper is as follows: The next sub-section provides an overview of the existing literature. The second section describes the data and methodology in more detail. The results are then presented in the third section. The final section concludes the paper and discusses the practical implications of our analysis.

1.2 Related Work

With the rise of ISIS in 2014, the use of social media for extremist purposes has become a subject of research for the first time. Most of the existing studies on terrorist content on Twitter can be categorized into one of the following two strands of literature. The descriptive approach, which mainly involves summaries and network analysis, seeks to increase our knowledge of how and to what extent terrorist groups use Twitter (Parekh et al. 2018). By its very nature, this type of research can only be conducted once enough relevant content has been identified. The second strand of the literature therefore specializes in the detection of tweets that either sympathize with terrorist ideas or actively spread them. Since this work contributes to the second strand, the following review focuses on studies that aim to identify radical content and its associated accounts on Twitter. As pointed out by Johnston & Weiss (2017) this is a nontrivial task. One often deals with multilingual, highly imbalanced and noisy data (misspellings, abbreviations, etc.). In order to reliably distinguish extremist tweets from neutral tweets on the same topic, the models need to capture the context of the tweets. The analysis of keywords alone may therefore not be sufficient. In the end, training a powerful classifier requires enough actual terrorist content which is rare and hard to locate.

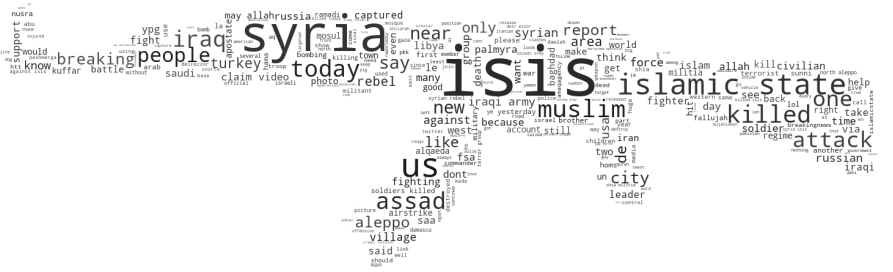


Figure 1: Vocabulary in the radical tweets

The challenge of detecting radical content on Twitter has been addressed in numerous studies already. The majority of the work, though, aims to identify hate speech in general. Pitsilis et al. (2018) and Badjatiya et al. (2017) apply deep learning techniques to classify tweets as sexist, racist, or neither. The language of terrorists, however, consists of more components than pure hate rhetoric. In the context of jihadist terror religious parlance plays a crucial role. When looking at ISIS in detail it is also important to consider its claim to secure its own territory and build its own state. Thus, a tweet from an ISIS supporter does not necessarily have to be offensive, but can contain reports of war, joy over a successful conquest, or slogans of perseverance to its allies.

Rehman et al. (2021) conducted corpus analysis to consider religious features in identifying radical content on Twitter. Their results suggest that using religious text in model training increases the overall performance of the classifier. In addition, the authors consider the complete history of Twitter users, allowing them to observe a change in linguistic patterns over time. Such approaches can be very useful in predicting vulnerability to radicalization. Further work on estimating future radicalization is listed in their study.

Figure 2: Vocabulary in the neutral tweets

An automatic detection system for jihadist hate speech on social media was developed by De Smedt et al. (2018). The authors manually identified terrorist profiles on Twitter and collected tweets in the aftermath of ISIS attacks. Using natural language processing, a preliminary analysis was conducted to obtain information on the distributions of language, demographics and tone. Unlike most studies, they not only applied the trained model to their own test data, but also reported the predictive accuracy of their classifier on related data such as jihadist manifestos as well as unrelated data like Lewis Carroll's Alice in Wonderland. Their final SVM achieves a stable prediction accuracy of 80% (F1 score) on different types of test data.

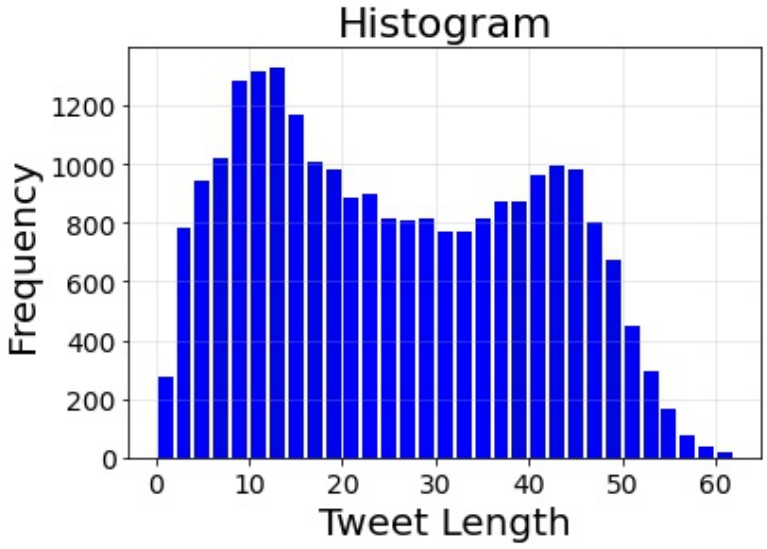
Ashcroft et al. (2015) applied various machine learning techniques to automatically distinguish radical and non-radical tweets. To avoid manual labelling of the training data, they collected tweets with hashtags related to jihadist terror from known ISIS supporters. They used stylometric features such as the occurrence of frequent words (e.g.: *state*, *Islam* or *Allah*), time-based features and sentiment-based features as model inputs. The final classifiers performed quite well, achieving accuracies of 89% with Naïve Bayes, 98% with SVM and 100% with AdaBoost. However, the high accuracy could result from the fact that the hashtags used to find radical content for the dataset are later also included in the classification process. Thus, the hashtags can be a potential "shortcut"for the classifier (Johnston & Weiss 2017). Finally, the authors conclude that the proposed methodology can only support humans in the recognition process but cannot yet completely replace them.

A very flexible and robust framework is provided by Johnston & Weiss (2017). They extended the identification of terrorist content to other websites and social media channels. In an extremely labor-intensive and time-consuming data collection exercise, a multilingual dataset of extremist and non-extremist content was built. The data was then processed using the doc2vec algorithm and prediction was performed using a neural network with six hidden layers. The resulting accuracy of 93% of the binary classifier is considered quite decent (Bedjou et al. 2019).

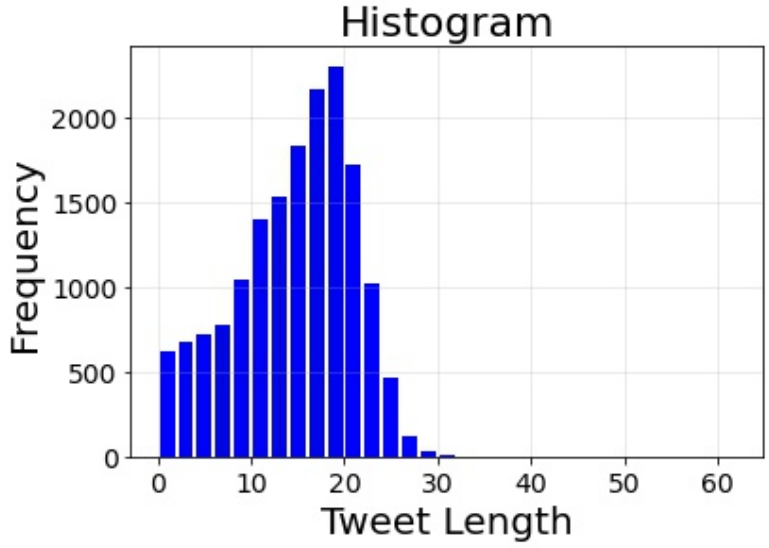
The approach that corresponds most closely to ours is that of Nizzoli et al. (2019). Deep Learning methods are used to distinguish pro-ISIS content on Twitter from content that is neutral about ISIS. The authors add random tweets into their analysis such that the final dataset contains only 1% of pro ISIS tweets. In this way, they account for the imbalance between radical and harmless tweets in reality. The pre-trained FastText word embedding is then used to train a recurrent convolutional neural network classifier which achieves an F1-score of 90%.

2 Methods

2.1 Data



(a) Neutral



(b) Terrorist

Figure 3: Length of tweets

The Pro-ISIS data, hereafter referred to as pro-set, is made available by Fifth Tribe, a digital agency that advises the U.S. government in addition to numerous private sector clients. The data set has been used widely for research purposes as it represents an extensive, reliable and publicly available source of tweets that can be associated to ISIS supporters. The data set contains over 17,000 tweets from over 100 pro ISIS users. It was built in the three months after the terror attacks in Paris in November 2015. The accounts were identified based on keywords like *Amaq*, *Dawla* and *Wilyat* and images of radical content (Tribe 2015). By construction the tweets are thus not independent.

The neutral counterpoise, hereinafter neutral-set, consists of 52,000 tweets collected via the Twitter Streaming API between Jan. 10 and Jan. 30. ISIS related search terms [1] were used to ensure that all tweets are thematically about ISIS. Thus, a simple dictionary approach based on keywords may not be able to sense the difference between the two types of tweets. All considered tweets are written in English.

Table 1: Processing of tweets

State	Tweet
original	@Adrianmccann9 @SameeraKhan @HillaryClinton Drone strikes in Yemen, Pakistan and Afghanistan that murdered hundreds of innocent civilians, arming Islamists in Syria, Libya and the massive troop surge in Afghanistan
processed	<user> <user> <user> drone strikes in yemen , pakistan and afghanistan that murdered hundreds of innocent civilians , arming islamists in syria , libya and the massive troop surge in afghanistan

Figure 1 and Figure 2 give an overview over the vocabulary in the two data sets. The word clouds show the 300 most frequently used words in each data set, with the size of each word representing its frequency and importance. It is noticeable that the two vocabularies are very similar. *syria* and *isis* are the most used words in both data sets. However, upon closer inspection one can notice some differences. Whereas *jihad* appears over 1,900 times in the neutral tweets, it appears only 66 times in the terrorist tweets. While the term is frequently used by people talking about ISIS, it is rarely used among the terrorists themselves. On the other hand, the Arabic word for nonbeliever - *kuffar* or *kafir* - is preferably used by jihadists. The word is less common among neutral Twitter users. Figure 3 displays the absolute frequencies of tweet lengths in the respective data sets. Twitter allows for a maximum of 280 characters only. So the length of tweets is inherently capped. Strikingly, almost no

[1] English tweets containing at least one of the following words were mined (not case sensitive): *isis, daesh, syria, islamicstate, isil, raqqa, mosul, aleppo, jihad*

terrorist tweet exceeds 30 words, while a significant proportion of neutral tweets is longer than 30. With a mean length of 14 words, the pro tweets are substantially shorter than the neutral tweets with a mean length of 25.

The Preprocessing applied to the data at hand is mainly based on a Python version of the Ruby script used by Pennington et al. (2014) to process the Twitter data.[2] This includes shifting all letters to the lower case, removing unnecessary characters, transforming links and mentions to tokens such as *<url>* and *<user>*. Additionally, the module *emoji* is used to transform emoticons not covered by the script to words. Hashtags are completely removed (the actual character as well as the words) since they could be used to link certain pro ISIS tweets directly to their author. Furthermore, some patterns specific to the data set such as *RT* for re-tweet or *English Translation* are removed. Re-tweets and tweets consisting of less than three words are excluded. A total of 15,930 pro tweets and 23,060 neutral tweets then remains for the analysis. Table 1 shows a neutral tweet before and after preprocessing.

Table 2: Missleading Tweets

Tweets	Label
Turkey to issue smoking ban for outdoors	Terrorist
Kurdish FSA militant killed today by #Russian airstrike in northern #Aleppo.	Terrorist
#RuAF hit #ISIS/#Daesh targets in #Homs desert #Syria	Neutral
QUICK! ISIS KILLING TIME! Don't worry!	Neutral

Unfortunately, the data comes with some major flaws. The pro-set contains tweets that are associated with ISIS supporters. However, not every tweet that has been made by an pro ISIS user is of terrorist nature. Table 2 shows four tweets with their respective labels in the underlying data set. The terrorist tweets can hardly be identified as terrorist because they contain neutral reporting solely. Additionally, our neutral-set almost necessarily contains some pro tweets. Since manually labeling the tweets is beyond the scope of this paper, these limitations cannot be fully addressed. For a later evaluation of the classifier it is important to take this aspect into consideration.

After preprocessing and embedding the words of the tweets, two different deep learning classifiers were used, a Convolutional Neural Network as well as a Recurrent Neural Network using long short-term memory cells, were applied in order to classify the tweets. The best models were chosen via gridsearch.

[2] Note that Twitter data can be also streamed and processed with the Python Package TTLocVis (Kant et al. 2020).

2.2 Embedding

In order to make the tweets processible each word was transformed to a vector of 25 dimensions. Due to the limited amount of observations the embedding was not trained on the target corpus or during training. Instead, GloVe (Global Vectors for Word Representation), pretrained on two Billion tweets with a vocabulary of 1.2 Million different tokens, is used. The training objective of GloVe is to obtain vector representations such that their dot product is equivalent to the log-probability of the words' co-appearance (Pennington et al. 2014). Using a pretrained vector representation has multiple advantages - models using pretrained embeddings do generalize better since the embeddings are not trained with regards to the classification target. Additionally, the training corpus of pretrained embeddings is much larger compared to the corpus used for this classification task. Furthermore, unknown tokens in validation and test set can be handled more elegantly. Since GloVe was not trained on lemmatized or stemmed tokens, no such transformations were undertaken - similarities between different tenses or cases are already included in the vector representations.

2.3 Handling Out-Of-Vocabulary words

Even though GloVe for Twitter is trained on a substantial amount of tweets, around 45% of the tokens were unknown. There are several ways to deal with OOV-words ranging from simply passing an extra token up to complex subword- or character-level models. Due to high percentage of unknown tokens the embedding of unknown tokens was created using a character-based model based on Pinter et al. (2017).

Pinter et al. (2017) mimick embeddings of OOV-words using a bidirectional LSTM on character level. The advantage of this approach is that the original corpus, on which the embedding was trained, is not needed. The model is trained on all known tokens and their embeddings. Inspired by the Dynet implementation a Pytorch implementation is used to receive embeddings for OOV-words. Each character is embedded and a word interpreted as a sequence of these characters. Therefore, unknown words receive embeddings similar to words of a similar word root. This approach might lead to misleading embeddings in case of words with a low Levenshtein distance but very different meaning (Levenshtein 1966). Still, Pinter et al. (2017) show an overall reliable performance. Due to significant computational costs similar hyper parameters where chosen: a character embedding dimension of 20, 60 hidden LSTM units as well as two layers.

$$MSE = \frac{1}{n * d} \sum_{i=1}^{n} \sum_{j=1}^{d} (y_{i,j} - y*_{i,j})^2 \tag{13}$$

Reducing the learning rate (starting value: 0.01) on plateau, the training is stopped after 25 epochs. As a loss function the mean squared error averaging over every embedding dimension and batch element was chosen (13). 10% of the GLoVe dictionary was chosen for validation. Figure 4 shows the loss curve of the model training. Due to

the fact that the training is not based on the data set itself or the overall classification task of the paper, the embedding of OOV-words in training, validation and test set can be simulated by this model.

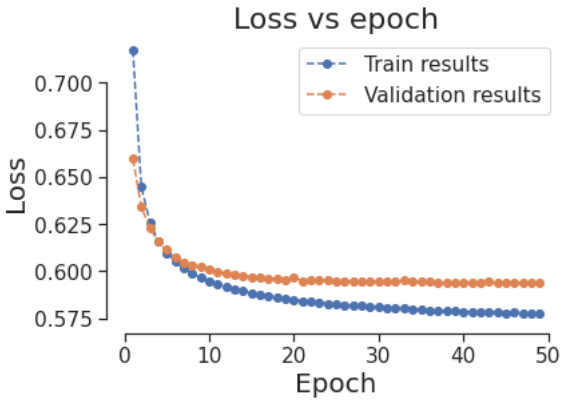


Figure 4: Loss curve of mimick model

2.4 Sequence length, data splitting and batchsize

For handling tokens a vocabulary is created, linking every token to a specific integer. Shorter tweets are padded to a length of 60. Neutral tweets receive the label 0, terrorist tweets the label 1. 10% of tweets were randomly chosen for testing, 80% of the remaining tweets used for training, the remainder for validation. Choosing a batch size can have a significant effect on training and the overall ability of a model to generalize (Keskar et al. 2016). Too large batch sizes can lead to a generalization gap due to local minima. Since the limited data set presents no computational obstacle, a comparatively small batch size of 64 was chosen.

2.5 LSTM

Tweets are sequences of words, therefore a sequential model was chosen. Simple recurrent neural networks tend to suffer from exploding or vanishing gradients. Subsequently, observations with a greater lag can be simply under- or overrated. This issue can be avoided by using LSTM cells. Due to the use of multiple gates and a current as well as a hidden state, which are passed through the cell for each sequence element, LSTM cells are able to capture structures and bridge lags far beyond thousand steps (Hochreiter & Schmidhuber 1997; Säfken 2020).

$$ReLU(x) = max(0, x) \tag{14}$$

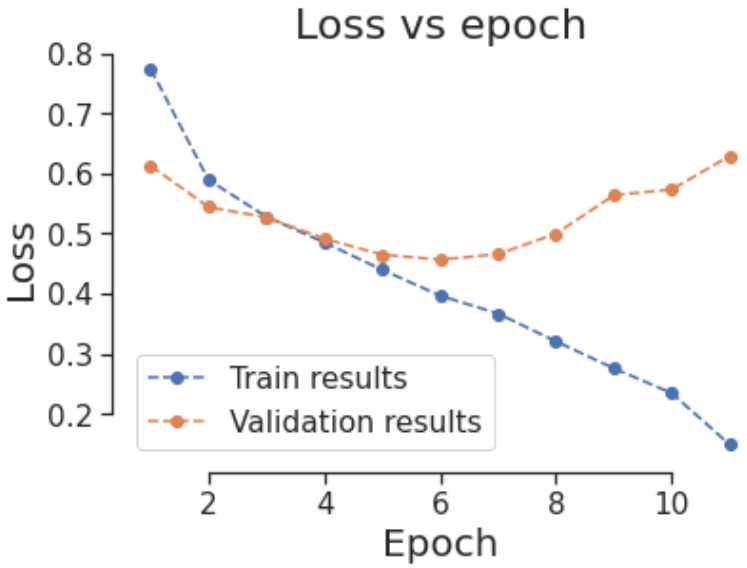
The model consists of a variable number of LSTM-layers stacked upon each other. In order to avoid over-fitting a dropout layer between the LSTM-layer was implemented. The final output of each cell is being passed to a one-layer neural network using a ReLU activation function (14). The output is reduced to one Float and mapped to $(0, 1)$ by a sigmoid function (15). The hidden and current cell states are initialized with zeros before each batch. The embedding is loaded based on GloVe and the embeddings for unknown words generated by the model described above.

$$Sigmoid(x) = \frac{1}{1 + \exp(-x)} \tag{15}$$

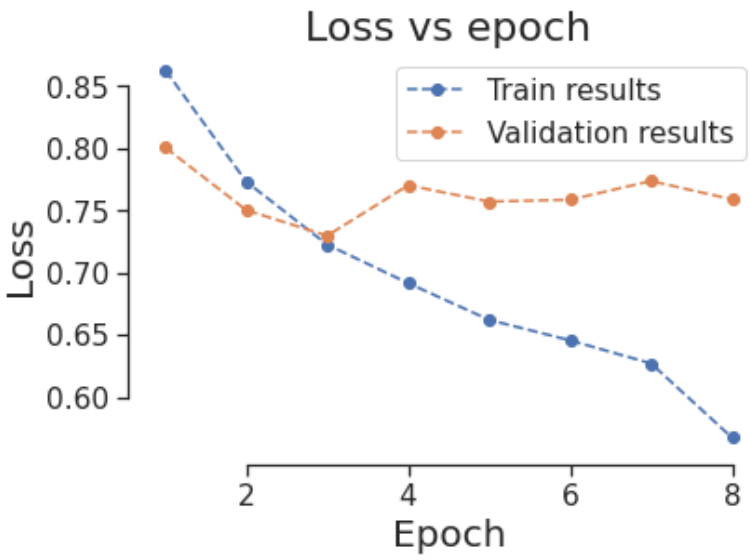
2.6 CNN

Even though Convolutional Neural Networks were originally designed to handle image data, they can also solve to sequential problems such as text classification. Filters of different sizes can capture structures in sequences (Kowsari et al. 2019). The implemented model uses four different kernel sizes - two, three, four and five with a variable number of filters. These convolutions are applied parallel. A ReLU activation function is exerted on each convolution and the output reduced by a maximum pooling layer. The output of the four different pooling layers is then concatenated and passed through a dropout layer as well as a linear layer reducing the output to one Float. Again, a sigmoid function is applied to the output.

2.7 Training



(a) RNN



(b) CNN

Figure 5: Loss curves of the best classifiers

For both models the Adams optimizer is used, which performs very well on both RNNs as well as CNNs (Kingma & Ba 2014). The learning rate is initially set to 0.01 and automatically reduced in case of plateauing training loss. The binary cross entropy was chosen as loss function. Since the data set is not balanced, the loss of each observation in each batch is weighted in order two avoid training in favor of the more frequent label. An early stopper is used to end training in case of stagnating or increasing validation loss and the best model with regard to validation loss loaded.

$$BCE(y, y*) = (l_1, \ldots, l_N)^T,\ l_n = -w_n(y *_n \log(y*_n) + (1 - y_n)\log(1 - y*_n)) \quad (16)$$

$$\text{with} \quad w_n = \begin{cases} \frac{N}{N - \sum_{i=1}^{N}(y_i)} & \text{if } y_n = 0 \\ \\ \frac{N}{\sum_{i=1}^{N}(y_i)} & \text{if } y_n = 1 \end{cases}$$

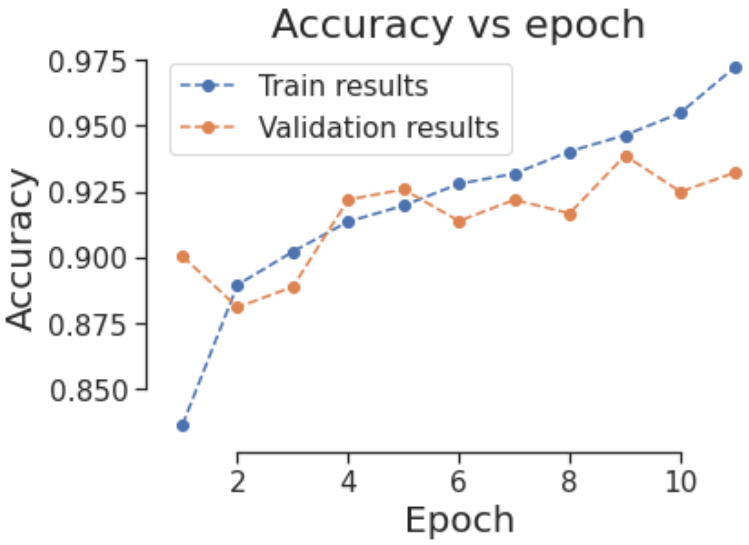
In order to find the models with the best hyperparameters a gridsearch on both models was performed, evaluating different numbers of layers, hidden units and dropout probabilities in case of the LSTM-model[3] and different numbers of filters and dropout probabilities in case of the CNN-model[4]. While no large differences between the different models could be observed during the grid search, the LSTM-model consisting two layers, fifty hidden units and a dropout probability of 0.1 performed best on the validation set. The best performing CNN used 200 filters per convolution and a dropout probability of 0.2.

[3] values used: 1-3 layers; 25, 50, 100 and 200 different filters per convolution; 0.1, 0.2 and 0.3 as drop out probability

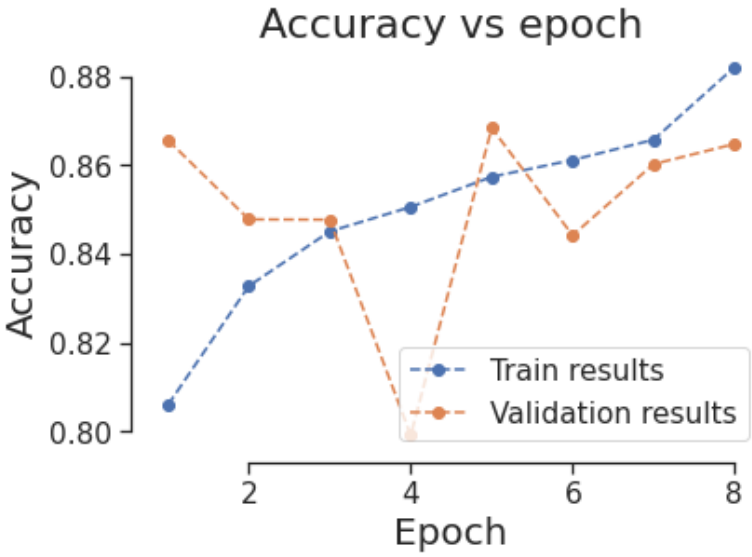
[4] values used: 50, 100, 200 and 400 hidden units; 0.1, 0.2 and 0.3 as drop out probability

3 Results

3.1 Performance on validation and test data



(a) RNN



(b) CNN

Figure 6: Accuracy curves of the best classifiers

Table 3 shows the performance of both models on the validation set. On our data the RNN outperforms the CNN in every performance score. In addition, when tracking the performance across training epochs, the RNN model seems to produce much more stable results with less variation (Appendix Figure 8).

Table 3: Performance on the validation set

Model	Loss	Acc.	Rec.	Spec.	F1-score
RNN	0.36	0.92	0.94	0.91	0.91
CNN	0.69	0.84	0.86	0.82	0.81

Furthermore, the confusion matrix of true and predicted labels shows reasonable results. The main reason for the worse accuracy of the CNN on the validation set is the fact, that the CNN more often classifies neutral tweets as terrorist tweets.

Table 4: RNN confusion table

True Label	Pred. Neutral	Pred. Pro
neutral	3,778	381
pro	174	2,686

Table 5: CNN confusion table

True Label	Pred. Neutral	Pred. Pro
neutral	3,409	750
pro	402	2,458

Based on the validation set the RNN model seems to work better on the task at hand. The final performance on the test data underlines this observation. With an accuracy of around 0.92% and 0.84% for the RNN and CNN there seems to be no significant over-fit of the models trained. Table 6 and Table 7 show the confusion matrix for the test set. Again, the observations match with the performance on the validation set.

Table 6: RNN confusion table (test data)

True Label	Pred. Neutral	Pred. Pro
neutral	2,109	215
pro	101	1,474

Table 7: CNN confusion table (test data)

True Label	Pred. Neutral	Pred. Pro
neutral	1,927	397
pro	222	1,353

3.2 Analysis of wrongly classified tweets

To get a better understanding of how the trained classifier works, one can look at the tweets that it does not label correctly. Table 8 shows six examples of misclassified tweets. The first tweet looks like a tweet that could have come from a jihadist. The true label of the tweet seems to be wrong in this case and the assessment of the model is reasonable. In the second example, however, the model does not capture the irony

of the statement and thus identifies terrorist content. Irony detection is a task that even humans sometimes fail at and will remain a major challenge for all automatic text recognition systems in the future. The following two examples contain mostly neutral reports of wartime activities and are therefore difficult to assign clearly to one of the two categories. The last two examples are terrorist tweets that were labeled as neutral. As opposed to the other examples there is no obvious explanation as to why the tweets were mislabeled here.

Table 8: Misclassified tweets

Tweets	Pred. Label	True Label
You declare jihad. The lions of labbaik will be at the forefront	pro	neutral
That's right women are only women if they get abducted by ISIS	pro	neutral
Iraqi forces have killed the self-proclaimed IS leader abu jaser al-issawi in Iraq	pro	neutral
IS fighters recapture tall battal at the border strip in Aleppo's northern countryside after Syrian oppo factions retreat frm the village	neutral	pro
you kafir your burning aren't you the islamic project is winning in syria you kafir	neutral	pro
Nations of the †, this message is for you. Know that your options are few, either you join islam, or pay tribute, or freeze the war	neutral	pro

3.3 Performance on thematically similar tweets

In order to test whether the model can be used to recognize pro ISIS tweets in general, the model needs to be tested on a a different dataset. Therefore, the model was applied to the available sample of the data used by De Smedt et al. (2018). Both models only achieved an accuracy of around 51% (49% in case of the CNN), overwhelmingly classifying tweets as terrorist hate speech, even though around 50% of tweets are neutral.

3.4 Discussion and outlook

Both models perform reasonably well on the test data, which does imply a good generalization ability concerning in-domain data. Nevertheless, the performance on similar and out-of-domain tweets needs to be analyzed. There can be several reasons for the observed difference of performance.

The first one lies within the data itself. As previously mentioned, the tweets by pro ISIS users are not independent from each other. Instead, there are substantially more tweets than authors. Therefore, the models might learn to detect the style of the different ISIS supporters in contrast to detect terrorist hate speech in general. Additionally, many of the tweets by pro ISIS users do not contain any references to jihadism or terrorism. This could explain the fact, that the model is biased towards terrorist hate speech. Secondly, the model is only trained on thematically related tweets. While a certain amount of neutral tweets regarding the topic of terrorism is needed in order to ensure, that the model does not simply classify based on the vocabulary, it might be helpful, to add completely random tweets to the data set. This way the model might learn a bigger variety of neutral tweets and its performance on tweets in general could be improved. The poor performance on the out-of-domain data leads to the suspicion, that the model rather trains how pro ISIS tweets do *not* look like instead of how they actually present. Thirdly, it cannot be precluded, that there might be better models than CNNs or RNNs for this classification tasks. De Smedt et al. (2018) present a strong out of domain performance for their Support Vector Machine classifier. In addition, modern approaches such as transformers like BERT could achieve better results.

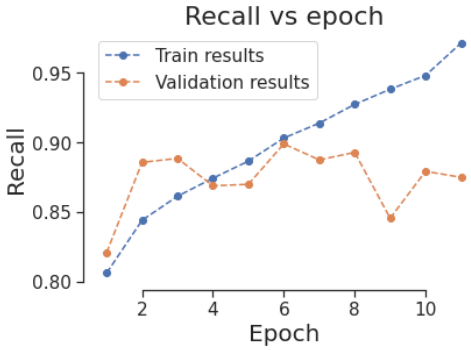
One suspicion that may arise from the different mean tweet lengths in the pro set and in the neutral set is that the length of tweets influences the prediction. A closer look at the mislabeled tweets reveals that both mislabeled groups - terrorist tweets labeled as neutral - as well as neutral tweets labeled as terrorist - have an average length of 20 words. Misclassfied neutral tweets have a length significantly below the average of all neutral tweets and vice versa for terrorist tweets. Therefore, the classifier might partly use the length of tweets as a decision criterion. This could also explain the poor performance on other data sets.

4 Conclusion

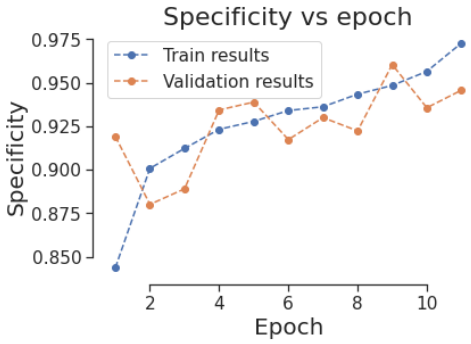
The results of this work and others show, that it is possible to detect terrorist and jihadist hate speech using deep learning as wells as machine learning techniques in general. The accuracy of those classifications are formidable, but highly dependent on the quality and amount of data used to train the respective models. This holds especially for the generalization to different data sets. In addition, in real world applications the data would be less balanced. The ratio of terrorist tweets among all tweets is vanishingly small. Even models with a decent accuracy might produce a very high a number of false positives. Nevertheless, such models can become a helpful tool for researchers, social media platforms as well as law enforcement and security once more data, particular with regards to real terrorist hate speech becomes available. While the current models performance is not sufficient as a universal detection system for tweets, it is capable of recognizing tweets of known ISIS sympathizers. This can still be helpful, since opening new accounts is so far a viable strategy for blocked users.

5 Appendix

5.1 Figures

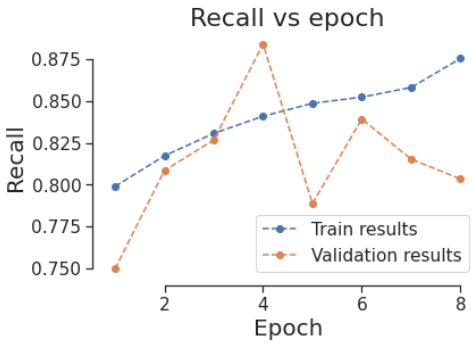


(a) Recall

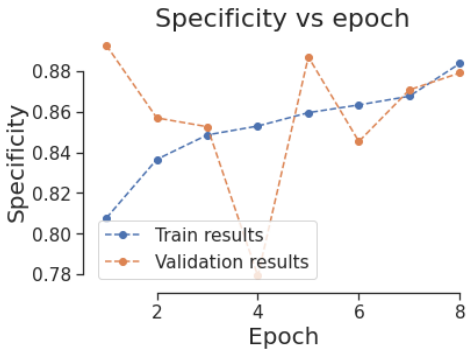


(b) Specificity

Figure 7: Performance curves of RNN



(a) Recall



(b) Specificity

Figure 8: Performance curves of CNN

5.2 STATUTORY DECLARATION

We hereby declare that we have authored this paper independently, that we have not used other than the declared sources and resources, and that we have explicitly marked all material which has been quoted either literally or by content from the used sources.

References

Ashcroft, M., Fisher, A., Kaati, L., Omer, E., & Prucha, N. 2015, in 2015 European Intelligence and Security Informatics Conference (IEEE)

Badjatiya, P., Gupta, S., Gupta, M., & Varma, V. 2017, in Proceedings of the 26th International Conference on World Wide Web Companion, Perth, Australia, April 3-7, 2017 (ACM Press)

Bedjou, K., Azouaou, F., & Aloui, A. 2019, in Proceedings of the 3rd International Conference on Future Networks and Distributed Systems (ACM)

Castano, E., Leidner, B., Bonacossa, A., et al. 2011, Political Psychology, 32, 601

CtrlSec. 2021, https://twitter.com/CtrlSec, [Online; accessed 18-February-2021]

De Smedt, T., De Pauw, G., & Van Ostaeyen, P. 2018, arXiv preprint arXiv:1803.04596

Farwell, J. P. 2014, Survival, 56, 49

Gialampoukidis, I., Kalpakis, G., Tsikrika, T., et al. 2017, in Proceedings of the 2nd International Workshop on Multimedia Forensics and Security (ACM)

Gill, P., Horgan, J., & Deckert, P. 2013, Journal of Forensic Sciences, 59, 425

Hochreiter, S. & Schmidhuber, J. 1997, Neural Computation, 9, 1735

Johnston, A. H. & Weiss, G. M. 2017, in 2017 IEEE Symposium Series on Computational Intelligence (SSCI) (IEEE)

Jomana Karadsheh, J. S. & Smith-Spark, L. 2014, How foreign fighters are swelling ISIS ranks in startling numbers, [Online; accessed 18-February-2021

Kant, G., Weisser, C., & Säfken, B. 2020, Journal of Open Source Software, 5, 2507

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. 2016, arXiv preprint arXiv:1609.04836

Kingma, D. P. & Ba, J. 2014, arXiv preprint arXiv:1412.6980

Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., et al. 2019, Information, 10, 150

Levenshtein, V. I. 1966in , Soviet Union, 707–710

Nizzoli, L., Avvenuti, M., Cresci, S., & Tesconi, M. 2019, in Proceedings of the 11th ACM Conference on Web Science, WebSci 2019, Boston, MA, USA, June 30 - July 03, 2019 (ACM Press)

Parekh, D., Amarasingam, A., Dawson, L., & Ruths, D. 2018, Perspectives on Terrorism, 12, 5

Pennington, J., Socher, R., & Manning, C. D. 2014, in Empirical Methods in Natural Language Processing (EMNLP), 1532–1543

Pinter, Y., Guthrie, R., & Eisenstein, J. 2017, arXiv preprint arXiv:1707.06961

Pitsilis, G. K., Ramampiaro, H., & Langseth, H. 2018, Applied Intelligence, 48, 4730

Rehman, Z. U., Abbas, S., Khan, M. A., et al. 2021, Computers, Materials & Continua, 66, 1075

Säfken, B., ed. 2020, Learning deep (Göttingen: Universitätsverlag Göttingen)

Tribe, F. 2015, How ISIS Uses Twitter, [Online; accessed 18-February-2021]

van Prooijen, J.-W., Krouwel, A. P. M., Boiten, M., & Eendebak, L. 2015, Personality and Social Psychology Bulletin, 41, 485

Artificial intelligence is considered to be one of the most decisive topics in the 21st century. Deep learning algorithms, which are the basis of many artificial intelligence applications, are of central interest for researchers but also for students that strive to build up academic knowledge and practical competencies in this field.
The Deep Learning Seminar at the University of Göttingen follows the central notion of the Humboldtian model of higher education and offers graduate students of applied statistics the opportunity to conduct their own research. The quality of the results motivated us to publish the most promising seminar papers in this volume. For the selected papers a review process was conducted by the lecturers.
The presented contributions focus on applications of deep learning algorithms for text data. Natural language processing methods are for example applied to analyse data from Twitter, Telegram and Newspapers. The research applications allow the reader to gain deep insights into some of the latest developments in the field of artificial intelligence and natural language processing from the perspective of students of whom many will take part in shaping the future research in this field.

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

ISBN: 978-3-86395-501-4

Universitätsdrucke Göttingen