Word embeddings for predicting political affiliation based on Twitter data

Ibrahim Abdelaziz[1], Oliver Berg[1], Angjela Davitkova[1], Venkatesh Iyer[1], Shriram Selvakumar[1], Kumar Shridhar[1], and Saurabh Varshneya[1]

[1]Technische Universität Kaiserslautern

January 31, 2018

1 Abstract

Twitter as one of today's biggest social media platforms allows political figures to express their thinking easily and with a concise message. We propose a generic way of classifying political affiliation based on Twitter posts. This involves Word2Vec vector representations of the input data and utilizes pre-trained embeddings for the German language. With this we have shown to be capable of insightfully position German political figures in the political spectrum.

2 Introduction

Social media platforms like *Twitter* allows people of interest to communicate their personal opinion, and as such e.g. indicating political alignment, through a comprised message being only a few hundred character long. This yields broad potential to characterize personality traits such as political affiliation on.

Generally speaking, political motives were shown to be consistently predictable with an accuracy better than chance already [Biessmann et al., 2017]. This paper therefore proposes a *deep learning* based classification model together with *word embeddings* [Pelevina1a et al., 2016]. This allows a later analysis to find interesting constellations within the (German) political spectrum. We leverage word embeddings to represent words in context. We thereby restrict ourselves to pre-trained models. Subsequently, a convolutional neural network (CNN) holistically classifies the Twitter profile by assigning each Twitter message a separate party label and combining these into a complete class score.

3 Related Work

Today, sentiment classification is mostly done using recurrent- or convolutional neural networks as described in [Kim, 2014]. The presented approach uses a basic CNN trained on pre-trained word vector representations and does only little hyperparameter tuning to already achieve compelling results in question classification.

Additionally, [Misra and Basak, 2016] introduces Recurrent Neural Networks (RNNs) for political bias analysis. Intuitively, RNNs operate more similar to how humans tend to process language: word by word, forming sentences. RNNs do train slower when compared to CNNs though, and CNNs generally produce more efficient representations of the data.

In connection to the given focus of working on Twitter data, [Cohen and Ruths, 2013] further introduces interesting questions concerning applicability of classification onto real data outside the training samples. It depicts the validation process as being prone to optimistic

interpretations of the result when overlooking problems in latent attribute inference. This also suggests a critical view on this paper's final analysis results.

In addition, [Sug, 2007] motivates context- and time dependencies within political data, which again makes analyzing an overall corpus of Twitter data a broadly connected issue.

4 Methodology

Formally, this paper proposes work on classifying Twitter data of political figures and picturing tendencies and dependencies within the data. As such, the following methodology is being applied.

4.1 Dataset

The dataset used in this approach was constructed of German politicians' Tweets posted on their Twitter accounts. 1000 German politicians' profiles with attributes "person name", "political party" and "twitter username" were retrieved from the website "https://www.wahl.de/". To reduce noise, only politicians belonging to the seven major political parties with respect to parliament activities, namely "CDU", "CSU", SPD", "FDP", "GRÜNE", "LINKE" and "AFD", are considered.

After the filtering stage this leaves a list of around 800 politicians, 125 of them are separated for the testing dataset. For each one up to 1000 of his/her most recent Tweets are gathered using Twitter API.

To prepare the Tweets for the training, and testing steps, each Tweet was preprocessed by first removing URLs, special characters, user names, and mentions. Then output empty Tweets are removed, and all characters are masked and put to lowercase.

Given the difference in the number of Tweets for each party, and in order to create a balanced training dataset, each party is restricted to a total of 12000 Tweets for training dataset, where each politician contributes approximately the same number of Tweets.

To represent the preprocessed Tweets text into numerical values that can be used to train, and test our model a pre-trained Word2Vec model is used [Mikolov et al., 2013] - pre-trained on 200 million German Tweets [Cieliebak et al., 2017] - which represent each word of the Tweet as a 200-dimensional vector.

4.2 Classification

The 200-dimensional word vectors are fed to a CNN model for classification. The employed CNN architecture, shown in Figure 1, is based on [Kim, 2014] which uses its model for sentiment analysis specifically.

The first layers embeds words into low-dimensional vectors. Subsequent layers performs convolutions over the embedded word vectors using multiple filter sizes; sliding over 3, 4 and 5 words at a time. Max-pooling transforms the result of the constitutional layer into a long feature vector, dropout regularization is added and the result is classified using a final softmax-layer.

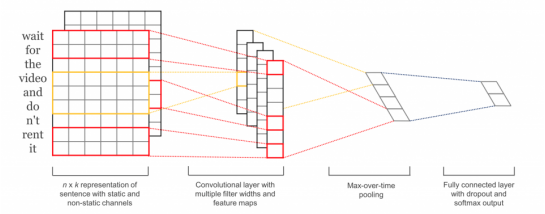

Figure 1: Model Architecture [Britz, 2015]

The overall classification process can be sub-divided into two elementary parts: First we **feed input data** to the neural network to then **classifying the (test-)data** for their respective classes.

In order to feed input data: For each party a raw text collection of all Tweets is considered. For each line - containing a single Tweet - a <PAD> tokens is appended to achieve a fixed size length of size 35 which is the maximum word-length in a tweet for our dataset. A vocabulary built on the complete corpus of existing words within the data maps each word to an integer between 0 and 109933 (number of existing words) for faster indexing. Note that

each Tweet is now represented as a vector of integers only. Using the pre-trained Word2Vec model, each Tweet can then be represented as a Matrix $M \in \mathbb{R}^{35\times200}$.

For classifying the user Tweets to the correct political party, we then feed the batches of twitter data along with their correct political party one-hot-encoded labels and train the above defined CNN. To optimize the network we use cross entropy loss defined as

$$H_{y'}(y) = -\sum_i y'_i \log(y_i)$$

where y is our predicted probability distribution, and y' is the true distribution (the one-hot vector with the true-class party labels).

5 Qualitative Analysis

For a quantitative analysis of the results, every Tweets are taken as sentences and a sentence vector of the same dimension as the word vectors used for training (i.e. 200 dimensional vector) is constructed. In doing so, the Tweet is tokenized into words and every word's corresponding token vectors are taken from the word embedding space. A sentence embedding vector is constructed by adding the word vectors and dividing them by the number of words. Note that only those Tweets for which all its words were present in the word embedding space are considered and that any Tweets with *UNK* token were ignored for a cleaner analysis. This effectively removes any Tweets that were not completely related to political topics such as simple link- or hashtag-forwarding.

5.1 Visualization

For the visualization of results, sentence vectors were visualized using a Tensorflow built-in visualizer called the Embedding Projector that allows interactive visualization and analysis of high-dimensional data. The high-dimensional vectors is reduced to 2 dimensional space using TSNE and then the model is further visualized. However, the visualizations over word-embeddings of tweets lacked clear distinction between political party clusters.

We found several reasons for this: Since a user Re-Tweets other parties Tweets with his opinions, it could be that subjects get mixed rather quickly across parties. Another reason for such mixed clusters could be the fact that any collection of party-members is typically tweeting about similar issues with some in favor of and some against some specific proposition, which may lead to strictly disconnected sentiment groups within the same party.

As such, a direct embedding as described earlier did not prove to be a good measure to visualize the result. This motivates our usage of the "political compass" [Pol, 2017] for visualizing the result further.

For the political compass representation, we use a four way graph plot with attributes of "Left-wing" and "Right-wing" on the X-axis (Economic Scale), as well as "Authoritarian" and "Libertarian" on the Y-axis (Social Scale). Each party is positioned on the compass by a given X,Y coordinates.

To visualize the results and to compare whether the user has been classified to it's correct class, we need to plot each user's position on the compass. This was achieved through following steps:

1. Passing all of a user's Tweets to the classification model as specified above in the model architecture section
2. Obtaining the party classification for each Tweet by getting the party with highest probability
3. For each party count the number of Tweets classified to this party
4. Averaging all Tweets counters to get one vector representing the distribution over all classes
5. Compute x,y coordinates as a weighted sum of the vector distribution multiplied by X,Y vectors coordinates of the seven parties

5.2 Inference

The subsequent visualization was created as described in above section.

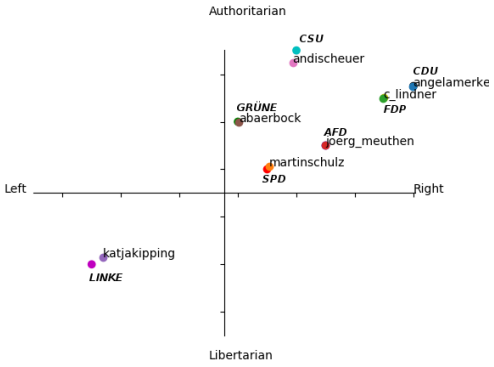

Figure 2: Plotting on Political Compass

From this depiction within the political compass, we can derive the following:

1. Users generally appear close to their respective political party. As an example, we have depicted the leader of each party on political compass in Figure 2. Position of all users is provided in the implementation [1] and here a subset is plotted to avoid clutter.

2. The FDP-class is plotted close to the CDU-class. Both party's ideology are traditionally quite similar, which matches the visualization.

3. Few users were misclassified near other party clusters. Upon analyzing the data, we observed a common occurrence of Tweets in the English language, which led to a huge number of <UNK> token generation, hence the classification may not be correct. Also, total number of tweets for those users were low. After manually cleaning the data, and thresholding the minimum number of tweets to 100, we were able to see the user's plot moving to the correct classes.

[1] The plots of all the users are available at `https://github.com/rngShard/ML1_Team02/tree/saurabh-classification/src/text-classification-on-embedding/plots`

6 Results

Classification to a specific party based on a single tweet is a difficult task, even for humans. We, through our experiments comprehend the same as our results on individual tweets gave a low classification accuracy of 55 percent. However, taking majority voting over a set of tweets for a specific user led to a considerable increase in the classification accuracy depicted below:

party	users	max tweets/user	accuracy
AFD	25	200	86.67
CSU	25	200	87.50
FDP	25	200	81.81
Grüne	25	200	66.67
Linke	25	200	89.28
SPD	25	200	69.40
CDU	25	200	76.20

7 Conclusion

As we have shown, a comparably simple convolutional neural network is able to nicely separate political figures concerning party affiliation. Interesting findings like single entities more closely connecting to general party-affiliated language or specifically different language have been pointed out.

As the underlying word embeddings are taken from the German-language Wikipedia dump, we are currently restricted to German-language Tweets as well as to overall German-language features. This does suffice for general classification purposes, but poses the additional question of how the analysis would be affected if the embeddings were to be taken from *intrinsically political* data samples. Also, our approach primarily focuses on CNNs and proves them to be efficient already. For future work, it would be intriguing to compare the capabilities of RNNs or other topologically different architectures.

References

[Sug, 2007] (2007). Ideology classifiers for political speech.

[Pol, 2017] (2017). German general election 2017.

[Biessmann et al., 2017] Biessmann, F., Lehmann, P., Kirsch, D., and Schelter, S. (2017). Predicting political party affiliation from text.

[Britz, 2015] Britz, D. (2015). Implementing a cnn for text classification in tensorflow.

[Cieliebak et al., 2017] Cieliebak, M., Deriu, J., Egger, D., and Uzdilli, F. (2017). A twitter corpus and benchmark resources for german sentiment analysis. *SocialNLP 2017*, page 45.

[Cohen and Ruths, 2013] Cohen, R. and Ruths, D. (2013). Classifying political orientation on twitter: It's not easy!

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[Misra and Basak, 2016] Misra, A. and Basak, S. (2016). Political bias analysis.

[Pelevina1a et al., 2016] Pelevina1a, M., Arefyev, N., Biemann, C., and Panchenko, A. (2016). Making sense of word embeddings.