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 17, 2018

1 Abstract

We propose a generic way of classifying political affiliation based on Twitter posts. This involves word2vec vector representations of the input data and utilizes pretrained 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 political alignment through a comprised message being only a few hundred charcters long. This yields broad potential to characterize political affiliation on.

Generally speaking, political motives were shown to be consistently predictable with an accuracy better than chance already [Biessmann et al., 2017]. Overall, sentiment classification is mostly covered using recurrent- or convolutional neural networks [Kim, 2014]. In connection to the given focus of working on Twitter data, [Cohen and Ruths, 2013] here introduces interesting questions concerning applicability of classification onto real data outside the training samples.

This paper therefore proposes a *deep learning* based classification model together with *word embeddings* [Pelevina1a et al., 2016]. This allows a later *Quantitative Analysis* to find interesting constelations within the (Gemran) political spectrum. We leverage word embeddings to represent words in context. We thereby restrict ourselves to pretrained models. Subsequently, a convolutional neural network (CNN)holistically classifies the Twitter profile by assigning each Twitter message a separate party label and combining this into a complete class score.

3 Data Sets and Feature Extraction

4 Classification

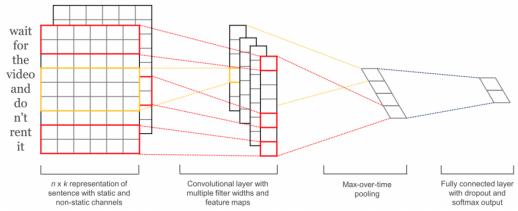
While deciding upon the architecture, RNN(Recurrent Neural Network) looked like an intuitive solution as it is more similar to how humans think and process language: one word at a time and then forming sentence using those words. But we wanted to experiment things in terms of making the training process faster and less computational intensive, so we used a CNN model for our case as convolutions are a central part of computer graphics and implemented on a hardware level on GPUs which make them very fast for training.

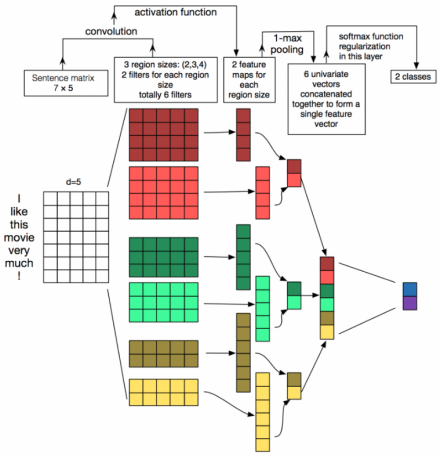
Another point in favor of CNN is being very efficient in representations. With a large vocabulary, computing anything more than 3-

grams can quickly become expensive. Convolutional Filters learn good representations automatically, without needing to represent the whole vocabulary and we can put a bigger filter size of 7 or more depending on the use cases as different filter sizes learn different representations.

4.1 Model Architecture

Our model architecture is taken from [[]] which is used for sentiment analysis. The first layers embeds words into low-dimensional vectors. The next layer performs convolutions over the embedded word vectors using multiple filter sizes. For example, sliding over 3, 4 or 5 words at a time. Next, we max-pool the result of the constitutional layer into a long feature vector, add dropout regularization, and classify the result using a softmax layer.





4.2 Classification Process

The whole classification process can be subdivided into two elementary parts

1. Feeding input data to Neural Network
2. Classifying data to correct class

The following steps were taken in order to feed data to the Neural Network:

1. 7 raw text files were prepared each containing the tweets for respective parties collected from Twitter.
2. The dataset was cleaned and tokenized for building a vocabulary.
3. Each sentence was padded to a max length. We append special <PAD> tokens to all other sentences to make them equal of 35 words. Padding sentences to the same length is useful because it allows us to efficiently batch our data since each example in a batch must be of the same length.
4. For feature extraction from twitter data we needed a pretrained word-embeddings to represent every word in vector form. We use [[cite here]] word-embeddings pretrained on [[]] million tweets and having a 200 dimension vector representation of each word.
5. Finally we built a vocabulary based on our collected data and map each word to an integer between 0 and 109933 (the vocabulary size). Each sentence becomes a vector of integers.

For classifying the user tweets to the correct political party, the following two steps were performed:

1. We feed the batches of twitter data along with their correct political party one-hot-encoded labels and trained above defined CNN
2. To optimize the network we used the cross entropy loss defined as:

$$H_{y'}(y) = -\sum_i y'_i \log(y_i) \qquad (1)$$

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

5 Quantitative Analysis

For quantitative analysis of the results, every tweets were taken as sentences and a sentence vector was constructed of the same dimension as the word vectors used for training (i.e. 200 dimensional vector). In doing so, the tweet was tokenized into words and every word corresponding token vectors were taken from the word embedding space and a sentence embedding vector was constructed by adding the word vectors and dividing them with the number of words. Only those tweets were considered whose all words were present in the word embedding space and any tweets with UNK token were ignored for better analysis. This was done to remove any tweets that were not completely related to politics as a user tweets on a variety of backgrounds and we only want the tweets related to political spectrum.

The sentence embedding was used to visualize the vectors based on their 200-dimensional vector representations. The results were pretty mixed up with no clear distinct clusters for any specific party. There could be several reasons for the mixing up of results and no clear clusters. Since a user re-tweets other parties tweets with his opinions, it could be one of the reason for mixing up of clusters.

Another reason for such mixed clusters is the fact every party is tweeting about the same issues with some in favor of it and some against of it and this leads to creation of same tokens for different party supporting user/group tweets. Further, these tweets with some/very less distinct tokens like together in the cluster prohibiting a distinct cluster for each class.

So a direct embedding as above did not proved to be a good measure to visualize the result. So we used a political compass for visualizing the result further.

For the political compass representation, we use a four way graph plot with [[]] and [[]] on both the axis. On the four axis, we used an one hot encoding to plot all the seven classes on the compass. This way we can see all the political classes plotted.

5.1 Visualization

Now, we need to plot all the users on the compass to see which users corresponds to what cluster and following steps were taken in order to do so:

1. We take a user and scrape all the tweets of the user from twitter.

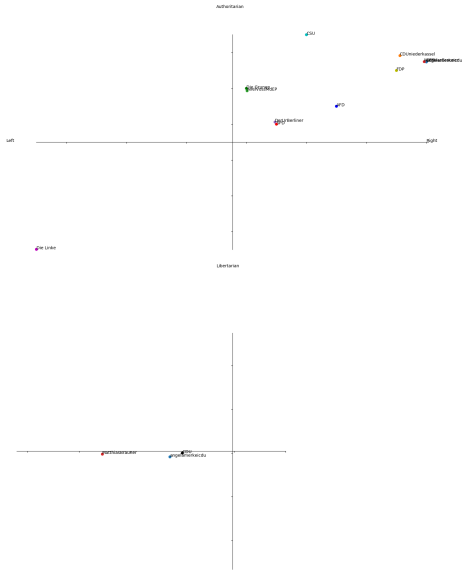
2. Next we pass the tweets of the user to the classification model as specified above in the model architecture.

3. We get the class classification and probability score of all the tweets for all the seven classes.

4. We average all the probability distribution to get one vector consisting of a probability distribution for all seven classes.

5. We plot the user on the political compass to visualize the results and to see if the user has been classified to it's correct class.

5.2 Inference



The following can be inferred from the above political compass visualization:

1. Most users have been plotted very near to it's political party representation. E.q. Angela Merkel plot is very near to CDU which shows all her tweets were pretty biased for the CDU Party.

2. Some plots like [[]] is very near to [[]] class. This means the ideology of these parties are quite similar. (Not exactly similar but some similarity is there.) This could also be seen from the usage of similar vocabulary used be both the parties.

3. User [[]] from party [[]] is being classified to wrong party. [[]] in this case. Upon analyzing the data, it can be inferred that there are a lot of English language based tweets present in the scraped tweets which led to a lot of <UNK> token generation and hence the classification was not correct. Upon data cleaning we were able to see the user plot moving to the correct class.

6 Conclusion

As we have shown, a comparably simple convolutional neural network is able to separate political figures concerning party affiliation. Interesting findings like single entities can be found to more closely connect to general party-affiliated language or to hold a different position regarding general expression.

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 topological architectures.

References

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

[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.

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