Hate speech detection with deep-learning

Apolline Foucher
a.foucher@mpp.hertie-school.org

Augustine Malija
a.malija@mpp.hertie-school.org

Vlad Surdea-Hernea
v.surdea-hernea@mpp.hertie-school.org

Abstract

The following project[1] *uses a series of deep-learning tools to design a method of automatically detected hate speech across Twitter. We contrast the results of our proposed LSTM-based RNN architecture with the more traditional Support Vector Machines approach, trying to highlight the improved accuracy offered by deep-learning algorithms. In terms of novelty, we intend to do two things: a) offer a comprehensive analysis of how different pre-processing methods differ, b) offer a regional-orientations, trying to see if there are geographical particularities that could assist in detecting hate speech in a given country. If the accuracy improvements are not significant, we will make appeal to other deep-learning architectures such as BERT.*

1. Proposed Method

1.1. Baseline method

We use Support Vector Machines (SVMs) as a type of supervised learning classification algorithm. The main reason we use this is because of its reputation of having a high prediction power when operating with data from Twitter. The advantages of SVMs include:

- effectiveness in high dimensional spaces, where the number of dimensions is greater than the number of samples;
- memory efficacy, because SVM uses a subset of training points in the decision function.
- versatility, in the sense that different common kernels can be specified in the decision function but also allows for specifying custom kernels

Nevertheless, operating with SVMs has significant disadvantages:

- the risk of over-fitting if the number of features is greater than the number of samples
- they also do not directly provide probability estimates - they have to be calculated using cross validation and probabilities.

1.2. Proposed deep learning method

Our paper's primary goal is to experiment with various tweet semantic embeddings like word Term Frequency Inverse Document Frequency (TF-IDF) values, Bag of Words Vectors (BoWV), and other potential different pre-trained word embeddings such fastText, Word2Vec, and GloVe.

Recent studies focus on the word embeddings and processings to differentiate between hate speech and non-hate speech. For our study, we would define task-specific embeddings learned using deep learning architectures such as the LSTM-based RNN. In contrast to the standard RNN architecture, the hidden layers of a LSTM-based RNN have a more complex structure capable of learning long-term dependencies, which is useful given the scope of this project. The LSTM-extension introduces the concepts of gate and memory cell in each hidden layer in the network. As a consequence, a memory block in the LSTM-based RNN is composed of four structural parts: an input gate , a forget gate , an output gate , and the self-connected memory cells:

- The input gate manages the entry of the activations to the memory cells of the neural network.
- The output gate is designed to learn what cell activations to filter and, therefore, send as output to the successive network.
- The forget gate assists the neutral network to disregarding the past input data and reset the memory cells for the new inputs.

In addition to this tripartite structure, LSTM applies multiplicative gates to make it possible for the memory cells to access and store the information over a long time interval.

[1] GitHub account https://github.com/ApollineFo/NLP---Hate-Speech

The first step in the process is for the forget gate to be applied in order to decide what part of the information to discard from the cell state.The activation of the forget gate is computed via the use of a sigmoid function:

$$f_t = \sigma(W_{ix}x_t + W_{fw}h_{t-1} + W_{fc}C_{t-1} + b_f)$$

f_t has values between 0 and 1, where 0 would mean discarding all the information from the last cell state, and 1 retaining all the information in the last cell state. Nevertheless, usually the value does not attain this extreme values.

After deciding what information to forget, the LSTM model decides what new information to store in the new cell state. Once again, a sigmoid function is used to determine the input gate layer. This second step is the reverse, symmetrical operation to the first step.

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}C_{t-1} + b_i)$$

Potential values of the new cell states are included in a new vector, U_t, that is computed by using the standard sigmoid layer present in the RNN.

$$U_t = g(W_{cx}x_t + W_{ch}h_{t-1} + W_{ic}C_{t-1} + b_i)$$

The old cell state C_{t1} is updated to a new cell state C_t, with the support of the previously-estimated f_t and U_t. f_t is being used in order to deduce what information to forget from the last state C_{t1} , while U_t is used to understand how much new information to retain:

$$C_t = U_t i_t + C_{t-1} f_t$$

Based on this, the output gate is produced using a different sigmoid layer of the network:

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + W_{oc}C_{t-1} + b_o$$

A cell output sigmoid activation function ℓ is applied over the cell state, which is then multiplied by the output o_t to generate the desired information about h_t:

$$h_t = o_t \ell(C_t)$$

An output activation function $k$is used to produce information about the output of the memory block:

$$y_t = k(W_{yh}h_t + b_y)$$

The full methodological setup will be exploited in the final project, and the clear comparative, in terms of the theoretical foundations between LSTM and SVM will be detailed.

2. Experiments

Data: For our study's scope, a hateful and offensive dataset provided by Davidson et al. (2017) is used. The tweets are characterized as hateful, offensive, and neither. The authors conducted manual annotation coded by CrowdFlower (CF) workers. Three or more people coded each tweet where the intercoder-agreement score provided by CF is 92%. The authors used the majority decision to assign a label to each tweet. Some tweets were not labeled as no majority class was reached. The final dataset we used is composed of a sample of 24,802 labeled tweets. We derive our features from these tweets and use them for training a classifier. Since our focus is on hateful speech, we removed the tweets labeled as "offensive" speech. After dropping all tweets considered "offensive," we obtain a dataset of 1,430 tweets as hate speech and 4,163 tweets as neither offensive nor hateful.

In our study, data preprocessing is a necessary process to reduce the number of unwanted features. Such removal leads to clean textual data that can easily be transformed to numeric arrays ready to be subjected to machine learning techniques. Thus, we end up with results from which we can deduce the extent to which hate speech was detected. Corollary text preprocessing offers the opportunity for the model's optimality to be tested. We used two different preprocessing techniques without handling stemming/misspelling to see how preprocessing can affect the results. This step is crucial to simplify and improve feature extraction. This entailed extracting tweets separately from the whole data frame, ready to be cleaned. To explain the process, we provide an original tweet as follows:

"!!! RT @mayasolovely: As a woman you shouldn't complain about cleaning up your house. amp; as a man you should always take the trash out..."

To clean the tweets, we used two different pre-processing methods:

First method

The first step was removing all unnecessary white spaces in tweet sentences, special characters, substituting multiple spaces with a single systematic space, removing prefixes, removing single characters from the start of tweets, hashtags, punctuations, numbers (e.g.,@, \$, %, *).

"rt mayasolovely as woman you shouldn complain about cleaning up your house amp as man you should always take the trash out"

All single characters that would jeopardize the transformation of features to arrays were removed. This also included ones that were found at the beginning of tweets. There was more than one white space between words. They were cleared to remain with a single standard space between words in the tweets. To have our features much clearer, we also removed all prefixes found in the tweets.

Second method

In the second pre-processing, we removed all retweets (annotated with an ”RT” at the beginning of the tweets), removed all users’ name (started with @username), removed all punctuation from the tweet, deleted all numbers, lowered the cases of the tweets, removed all hashtags (i.e., word) and finally deleted all URLs (i.e., https://). After that, all tweets were lowercase, and all stop words were removed.

After all the second pre-processing, we ended up with an updated tweet as follows:

“as a woman you shouldnt complain about cleaning up your house amp as a man you should always take the trash out”

We can now visualize the most hundred common terms based on our pre-processing tweets:

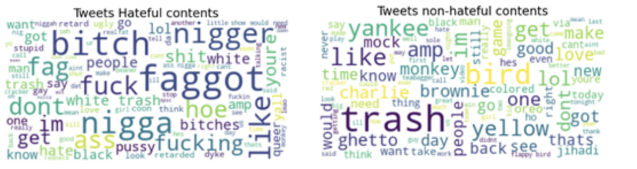

Figure 1. Post-cleaning dataset structure

Evaluation method: Getting a clear idea of how our model works is an assuring part of our analysis. We used two methods for evaluating our model. These are calculating the ROC AUC score and using the cross-validation evaluation score. These evaluation techniques are helpful for our model since it is a multiclass model. Unlike precision, recall, and F1 scores, they do not form part of the training and testing process. This is because if they were, they would return a perfect evaluation score suggesting that the model, both optimal and sub-optimal, does a good job. In the event of a sub-optimal model, this would lead to model over-fitting. Thus, it was essential to have evaluation techniques that did not form part of the training and testing process.

With the ROC-AUC score, we are computing the area under the receiver operating characteristics from the prediction scores. This served as our first evaluation technique because it is easy to interpret, with a single score being above 0.5 means that our model is performing suitably well. As it turned out, our model performed well with this technique since the ROC-AUC score was 0.882, meaning that the model is doing a very good job.

Cross-validation was used to determine the level of model precision. By running cross-validation, we can obtain the score for standard deviation—the lower the standard deviation, the more precise the model. Our model turned out to be exact enough to form a baseline since the standard deviation score we obtained was 0.004, which is relatively low, suggesting that the model’s precision is too high for standards. With such a precise standard deviation, the model can apply to an even larger dataset.

Experimental details: We evaluate our model by 5 fold cross validation using our processed tweet. To have our TF-IDF pre-processor working efficiently, we specified the following parameters. We first ignored words with frequencies higher than 90 percent from a cluster of our pre-processed document through a min_df function. In the same respect, we missed records with lower frequencies than 2. Our lower and upper boundaries of the range of n-values for different word grams were unigrams and bigrams. This is because having both of them offered a high enough accuracy, a perfect ROC_AUC score, and a precise standard error from our model’s cross-validation test. We also specified the strength of our regularization parameter to be ”C” to be 1.

We decided to specify both Gaussian and Linear kernels to take care of irregularities that may have arisen from the dataset as we run our machine learning model. With a Gaussian kernel, we specified a kernel coefficient ”gamma” of zero. We set the class weight to ”balanced” to automatically adjust weights/probabilities inversely proportional to class frequencies. We did not have to specify further for a linear kernel since it regards the dataset as a regular one with no imbalances.

Since we have more than one class of tweets categorization - hate speech and neutral, we built a pipeline that applies a TF-IDF vectorizer taking into account the parameters we specified. It also uses an OneVsRest classier of SVC that takes into account probabilities/weights. With these in check, we separated our dataset to train and test sets where we have 80 percent of the data as a training set while 20 percent of it as a test set. We then use a Grid Search to do an exhaustive search over parameter values where all processors are used. To be concise, we control the verbosity of messages by only allowing computation time and parameter candidates for each fold to be displayed. The use of Grid search is deemed paramount since it picks the best performing kernel out of the Gaussian and linear ones we specified earlier. The output it produces is one for the best-performing kernel. In this regard, we did not have to worry about being explicit of a single kernel among the two. We then built a classifier by fitting our training sets from which we were able to predict our test set.

Results In our normalized confusion matrix shown in Figure 2, zeros represent tweets classified as hate speech while ones represent tweets classified as neutral. These are only preliminary results, and their interpretation and comparison with the literature and with the baseline model will be properly addressed in the second half of the project.

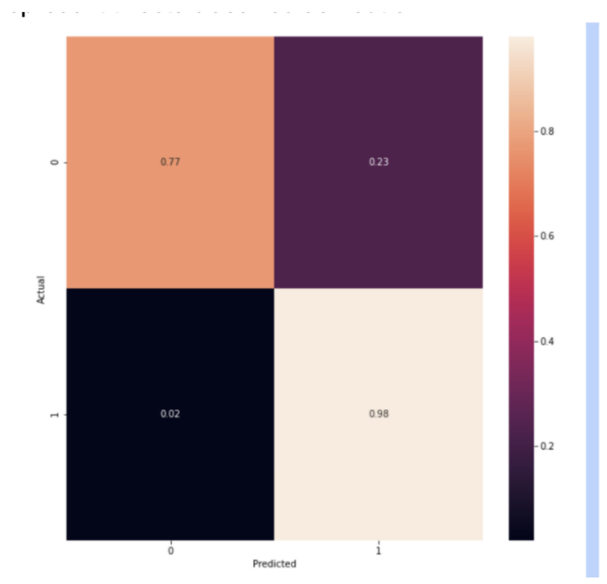

Figure 2. Normalization matrix

Comment on your quantitative results. We can see that our model correctly predicted about 77 percent of tweets that related to hate speech. On the other hand, 98 percent of tweets that did not relate to hate speech (classified as neutral) were predicted correctly. No model is perfect, and this applies to our model too. In this light, about 2 percent of tweets that were related to hate speech were classified as neutral, while about 23 percent of neutral tweets were classified as hate speech. Several reasons could explain such a minor flaw, but an important one is tweeting that is hard for a human to code. Nevertheless, our baseline model performs pretty well since it achieves an accuracy score of 93 percent. The following table provides the results for both pre-processing methods.

Classifier	Accuracy scores
SVM Pre-processing 1	0.935
SVM Pre-processingnc2	0.9316

It ensures that our external evaluation metrics of ROC-AUC and cross-validation scores achieve high enough and precise scores and standard deviations, respectively.

3. Future work

- The first step will be to finish the implementation of the LTSM.

- The second step is to develop new pre-processing methods or classifiers to explore how pre-processing can influence the models. These results will be evaluated both through the set of indicators described.

References

[1] Badjatiya et al. (2017) *D*eep Learning for Hate Speech Detection in Tweets WWW ’17 Companion: Proceedings of the 26th International Conference on World Wide Web Companion

[2] Davidson et al. (2017). *A*utomated Hate Speech Detection and the Problem of Offensive Language roceed- ings of ICWSM 2017

[3] Devlin et al. (2019). *P*re-training of Deep Bidi- rectional Transformers for Language Understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics.

[4] Kwok and Wang (2013) . *L*ocate the hate: detecting tweets against blacks. Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

[5] Mozafari et al. (2019). *A* BERT-Based Transfer Learning Approach for Hate Speech Detection in Online Social Media. 8th International Conference on Complex Net- works and their Applications

Sahlgren et al. (2018). *L*earning Representations for Detecting Abusive Language Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)