NLP - Authorship Attribution

Omer Keidar

Abstract

A majority of the social media activity of public figures is handled by assistants and media staffers. In some cases, posts are signed with the names of public figures who wrote them (e.g., Hillary Clinton often signed tweets she wrote with '-H'), but in other cases, readers remain unaware about who wrote the posts. Trump is a particularly interesting public figure, as he takes pride in his untamed use of Twitter, but he does not actually distinguish his own tweets from those of his staff. The goal of this project was to classify whether tweets published by Donald Trump's account were written by him or by his staff. We analyzed 3528 tweets, and implemented 5 machine learning algorithms (i.e., Logistic Regression, SVM, ANN, LSTM, and Random Forest) for the classification task, by using both engineered features and word embeddings. We found that the best model is ANN with engineered features as inputs, with 88% accuracy. During our work, we saw that there was not much difference in the accuracy of the models, but that non-linear models preformed better. Moreover, we saw that the model we speculated to deliver the best results (LSTM), was actually least of all.

1 Data

We have been provided with a dataset of 3528 tweets published between 2015 and 2017 on Donald Trump's and presidential Twitter accounts. The data contains the tweet ID, user handle, tweet text, time stamp, and device.

User handles can be one of the following three: realDonaldTrump (Donald Trump's account), POTUS (the official presidential account), and PressSec (the official Twitter account for the president's Press Secretary). As far as attribution is concerned, only tweets by realDonaldTrump can be considered to come from Trump. We automatically labeled every other handle as not from Trump. The device field can take various values ranging from 'android', 'iPhone', 'Instagram', 'web client', and other possibilities. The device determines whether the tweet was written by trump or not since we know Trump used an android phone and we assume the other devices were used by his staff.

We were also provided with a test dataset, with 390 samples, which is missing the tweet ID and device. At the end of this project, we submitted our authorship attribution classification on this data.

2 Pre-Processing

In this section, we describe our pre-process, which consisted of four steps: data analysis, feature extraction, feature selection, and data preparation. To begin with, we checked for duplicate samples and missing values in the data and confirmed there were none. Following that, we created labels based on user handle and devices, as described in section 1, where Trump's tweets are labelled as '0', and staffers as '1'. Although the data is not evenly distributed between the two classes (**'0':** 62.56%, **'1':** 37.44%), we did not consider it as highly unbalanced since it is not to the extent of involving a bias on our results (Ali et al., 2013), and we did not want to remove samples given the relatively small size of the training datasets.

As we used both engineered features and word embeddings, the next steps of the process are divided into two sections, where the first describes the preprocessing of Logistic Regression, SVM, ANN, and Random Forest models, while the second discusses LSTM preprocessing.

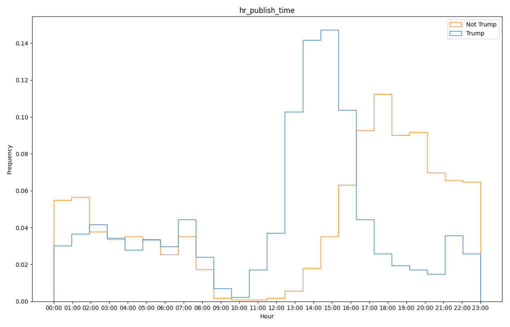
2.1 Engineered Features

We extracted features from the text based on an analysis that was made on Donlad Tump's tweeting habits[1] and other common text features. The features are presented in Table 1. For feature selection,

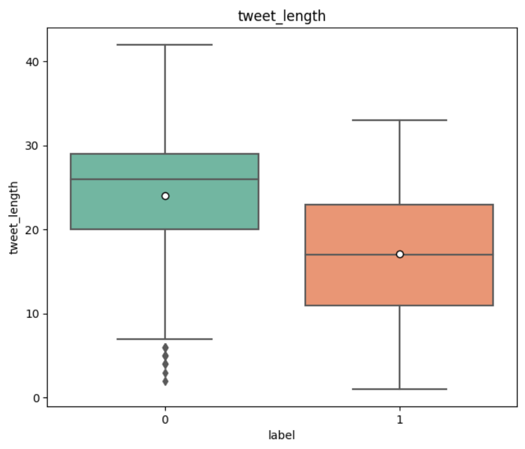
[1]https://www.theatlantic.com/politics/archive/2016/08/donald-trump-twitter-iphone-android/495239/

Table 1: Feature extraction

Feature	Type	Description
Positive score	Float	How positive the tweet is
Number of fully capital lettered words		The number of fully capital lettered words normalized by the tweet length
Beginning in capital letter		The number of words that beginning in capital letter normalized by the tweet length
Pos tag		Percentage of use of a particular tag in a tweet (NN, DT, IN, JJ, NNS, VBZ, VBD)
Exclamation mark	Int	Count the number of time that ! appear in the tweet (count only if there is at least 2)
Tweet length		The length of the tweet, how many tokens in it
Number of hashtags		Count the number of hashtags in the tweet
Tagging other user		Number of times that there was tagging in the tweet (pattern of @XXXX)
URL	Boolean	Check if there is a URL in the tweet or not
Tagging himself		Check if there is a tagging of Trump in himself (patters of @realDonaldTrump)
Event time		Check if there is an announcement of an event time in the tweet
Quotes		Check if there is a use of quotes in the tweet or not
Publish hour	Category	The hour that the tweet publish at (morning, noon, evening-night)
Publish day		The day in the week that the tweet was published (sunday-monday, rest of days)



(a) Tweet publish hour



(b) Tweet length (word count)

Figure 1: Examples for data visualisations we preformed on our extracted features

we created data visualizations to represent differences between the classes (an example is shown in Fig. 1) and estimated correlations of the features with the target (and between themselves). We removed every feature with a low correlation with our target and features with high correlation to other features to control multicollinearity. Categorical features were one-hot-encoded and all features were normalized to 0-1 by min-max standard scaler. The selected features are presented in Table 2.

2.2 Word Embedding

In text analysis, word embedding is used to represent words. Language modeling and feature learning techniques can be used to convert words or phrases from the vocabulary into vectors of real numbers. Our process of generating word embedding vectors for our text data was divided into three main steps: pre-processing the text data, making word embedding vectors, and representing each tweet using the trained embedding vectors we generated.

The pre-processing step contained:

1. Lower-case the text.
2. Remove URLs, hashtags, punctuation, stop words, and stock-market symbols (e.g., $)
3. Lemmatization.

We used GloVe and in particular GloVe.Twitter[2] because we found it to be the best fit for our classification task that includes tweets (Pennington et al., 2014). Lastly, we represented our data using a list of numbers in which each number represents a word index in the dictionary we constructed from our data. Each tweet was then represented by the word embedding vectors and used as input to the LSTM algorithm.

3 Models

In this section, each of the models we used is reviewed with respect to the input, architecture, and

hyper-parameters that have been chosen after tuning. We used 10-fold cross-validation to tune and evaluate all models. The accuracy score was used as the metric for comparing models. Even though the data were unbalanced, we chose this

[2]glove.twitter.27B.100d

Table 2: The selected features for the models.

Model	Features Used
Logistic Regression	• Tags, hashtags, capital words, words starting with capital letter counts. • The appearance of URLs, quotes, and tags of realDonaldTrump. • Tweet length. • POS count. • Tweet publish time (morning / noon / evening-night).
SVM	
ANN	
Random Forest	
LSTM	• Word Embeddings.

metric over AUC since the imbalance is reasonable and from the information we know, it doesn't seem that it was important to obtain a more accurate prediction of a specific class over the other. Thus, we decided that we want to examine the accuracy of predictions for each class. The imbalance may influence the model results, but not in a way that would justify further manipulation by us. All the models except the LSTM received the 15 engineered features that were extracted at the pre-processing (Table 2) as inputs, and the LSTM received GloVe pre-trained embeddings vectors. Each model's hyper-parameters were tuned together to optimize results.

3.1 Logistic Regression

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of the target or dependent variable is dichotomous, which means there would be only two possible classes. As mentioned, we used our engineered features as inputs for the model, and the tuning was made on the following parameters: C, solver, penalty ('l2' for all solvers and 'l1' only for 'liblinear' and 'saga' solvers) The optimal Logistic Regression model we found was with the following settings: *maximum number of iterations = 1000, C = 2, penalty = l2, solver = liblinear*.

3.2 SVM

SVM is a generic classification model that historically performed well on text classification problems (Joachims, 1998; Li et al., 2020). SVM can deal with both linear and nonlinear problems by using different kernels. We used engineered features as the inputs for the model, and fitted and fine-tuned an SVM model for linear and non-linear kernels (RBF, polynomial) from the sk-learn library. The tuning was made on the following parameters: kernel, C, gamma (for non-linear kernels). The optimal SVM model we found was with the following settings: *kernel = rbf, C = 1, gamma = 10*.

3.3 ANN

We built a feed-forward artificial neural network model using PyTorch open source library. Our network contains 2 fully connected hidden layers and an output layer of one neuron, with a ReLU activation function between the layers. The outputs from the output layer are squashed by a sigmoid function for classification by probability threshold (0.5). The hyper-parameters tuning for this model contained the size and number of the hidden layer, the training batch size and the number of epochs, and the optimizer's learning rate. The optimal ANN model we found has the following settings: *hidden_layers = (32, 32), batch_size = 128, epochs = 64, learning_rate = 0.01*.

3.4 LSTM

Recurrent Neural Networks (RNN) have the advantage of looking at a sentence as a sequence of words. Despite this advantage, RNN has a vanishing gradient problem, and hence, we chose to use LSTM that is known to deal with this problem and known to have better results than the RNN network. We used the PyTorch library like at ANN to build the LSTM network. Our network contains an embedding layer, 2 bidirectional LSTM layers, a fully connected linear layer as the output layer, and the outputs from the linear layer squashed by sigmoid function. The hyper-parameters tuning for this model contained the size and number of the hidden LSTM layers, batch size, epochs, learning rate, and the dropout in each LSTM layer. The optimal LSTM hyper-parameters and settings are as follows: *hidden_layers = 64, layers_num (number of lstm layers) = 2, batch_size = 16, epochs=2, learning_rate = 0.01, dropout = 0.2, bidirectional = True, loss function = Binary Cross Entropy loss (BCELoss)*.

3.5 Random Forest

The last model we implemented was Random Forest (RF), which is an engineered feature-based model. Although we found the highest validation accuracy score model to be one with a full depth trees, we did not pick it and rather preferred choosing a model with slightly less validation accuracy score (<.5%) which was better generalized (with a training accuracy score of 90% over 97%).

Thus, the optimal RF model we found has the following settings: *n_estimators = 600, criterion = gini, mex_depth = 8, min_samples_split = 2, min_samples_lead = 4, max_features = sqrt, bootstrap = False*.

4 Results

Presented here are the results for each of our models. The metrics scores of all the tested models can be found in Table 3, with accuracy as our primary metric for evaluating and comparing the models.

Table 3: Cross-Validation results of all implemented models.

Model	Precision	Recall	AUC	F1	Accuracy
Logistic Regression	0.875	0.733	0.921	0.797	0.865
SVM (Linear)	0.949	0.611	0.905	0.742	0.848
SVM (rbf)	0.912	0.73	0.91	0.81	0.877
ANN	**0.907**	**0.745**	**0.851**	**0.817**	**0.88**
LSTM	0.74	0.786	0.813	0.76	0.82
Random Foret	0.923	0.723	0.937	0.81	0.878

As far as the models that received the engineered features as inputs, we can see that the non-linear models (i.e., SVM with rbf kernel, ANN, Random Forest) performed better than the linear models (i.e., Logistic Regression and Linear SVM). The ANN model achieved slightly higher results with 88% accuracy while the Random Forest and SVM achieved 87.8% and 87.7% respectively. These models also achieved similar results on the train set (approximately 90% for all models). A confusion matrix from one of the ANN cross-validation folds is presented in Figure 2. The LSTM model was the worst out of all with only 82%. This model was also the most overfitted one, with a training accuracy of 99%.

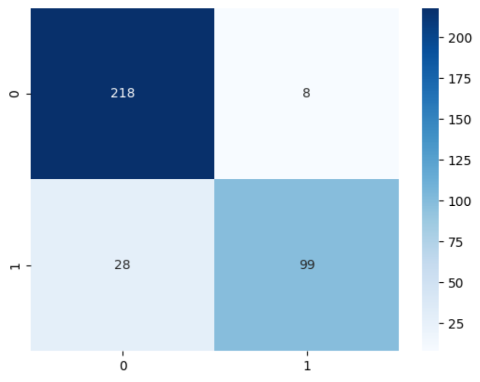


Figure 2: One fold confusion matrix (out of 10 folds cross validation) of the ANN model.

5 Discussion

The results showed that non-linear models are better for the authorship attribution classification task. This did not come as a surprise to us, since we already saw in the data understanding stage that although most of the features we extracted had a certain tendency for one class over the other, there was still a lot of overlapping. Thus, separating the classes linearly might not be ideal. With that being said, the linear models still achieved decent results which were not far from the best results (only 2% difference from the ANN model and Logistic Regression). This suggests, that although it is better to use non-linear models, our engineered features can do a good job of separating the classes even in the most basic model. Regarding the LSTM model, it is an algorithm that knows how to handle the input of sequences and is known as a good algorithm for classification in the field of NLP. Thus, we expected it to deliver out the best results for our task. However, the results were about 6% lower than our ANN model. We believe this could be due to several factors: the relatively small amount of data that did not allow the model to learn enough, the data clearing that may have caused some features to be lost from the text of each class because of the special style of tweets and perhaps the network architecture, that due to lack of time and resources, only a few architectures were tested.

6 Conclusions

We saw that Trump has a very particular style of writing, that we were able to capture at our features extraction and we have seen the impact of this on the results of the models. We found that the conditions of this specific assignment, caused that the neural networks had no significant advantage over the other basic models, but in larger data sets, we will expect to see better results for ANN and for sure in the LSTM algorithm.

Finally, we examined if Trump was kept away from his Twitter account during the election campaign at 2016 and we found that it really seems like that. We have seen from the data, that in times when there was no election, the amount of Trump himself tweeted from his account was almost 80%, compared to only 50% at the time of the election, so this reinforces the claim that he was kept away from his account.

References

Aida Ali, Siti Mariyam Shamsuddin, and Anca L Ralescu. 2013. Classification with class imbalance problem. *Int. J. Advance Soft Compu. Appl*, 5(3).

Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer.

Qian Li, Hao Peng, Jianxin Li, Congying Xia, Renyu Yang, Lichao Sun, Philip S Yu, and Lifang He. 2020. A survey on text classification: From shallow to deep learning. *arXiv preprint arXiv:2008.00364*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.