Identifying Tweets Written by Russian Troll Accounts

Ethan Brown, Brendan Edelson, and Elijah Taylor
esbrown@stanford.edu, bedelson@stanford.edu, elijaht@stanford.edu
Department of Computer Science
Stanford University

June 10, 2018

Abstract

As the trend toward technological security has been brought to the forefront of news media scrutiny in the past 6 months, there has been heightened focus on the infiltration of fake news into our social media technologies. The ability to classify and filter fake content will be a critical task for social media and other tech companies over the coming years. Most notably, the impact of Russian Troll Accounts on the 2016 U.S. Presidential Election has led to a great deal of discussion in the public sector. We investigated the use of neural networks to predict whether or not a politically-themed tweet had been posted by a Russian Troll Account, utilizing 1.2 million scraped Tweets for our dataset. Ultimately, our LSTM model greatly outperformed our classification expectations. We conclude our paper by discussing future potential projects in this area.

1 Introduction

For our project, we are investigating a number of tweets written by fake accounts attributed to Russian hackers regarding the 2016 Presidential election. We hope to be able to discern some of the critical factors that separate this content cultivated by Russian hackers from tweets written by the American populace. This project is important because we hope to use concepts of deep learning to glean information with implications of national security and preservation of democracy.

For our Russian tweets, we are using the dataset "Russian Troll Tweets" from Kaggle. The dataset includes 200,000 tweets identified as Russian bot accounts. Each entry in the set contains fields such as Twitter handle, date and time tweeted, tweet text, and hashtags used. For the non-Russian tweets, we are using a dataset of over 1 million politically-themed tweets from the 2016 election season collected by a Harvard research project. We then use the tweet text as input to a neural network to output a prediction of the likelihood that the tweet was published by a Russian Troll.

2 Related work

1) Jianqiang, Zhao, Gui Xiaolin, and Zhang Xuejun. "Deep Convolution Neural Networks for Twitter Sentiment Analysis." IEEE Access 6 (2018): 23253-23260
2) Bayot, Roy Khristopher, and Teresa Gonçalves. "Age and Gender Classification of Tweets Using Convolutional Neural Networks." International Workshop on Machine Learning, Optimization, and Big Data. Springer, Cham, 2017.
3) Zhang, Xiang, Junbo Zhao, and Yann LeCun. "Character-level convolutional networks for text classification." Advances in neural information processing systems. 2015.
4) Zhao, Luda, and Connie Zeng. "Using Neural Networks to Predict Emoji Usage from Twitter Data."

5) Badjatiya, Pinkesh, et al. "Deep learning for hate speech detection in tweets." International World Wide Web Conferences Steering Committee, 2017.

Listed above are five studies we used when researching our project that similarly used deep learning in order to classify text, most of them using tweets. Study 1 used a ConvNet with Twitter's GloVe embeddings for sentiment analysis. Study 2 used Word2Vec embeddings and a ConvNet for age and gender classification. Study 3 tested various deep learning frameworks for text classification, finding a ConvNet with characters as input as successful. Study 4 compared a ConvNet and LSTM with GloVe vector for emoji prediction. Study 5 used learned word embeddings, gradient boosted decision trees, and LSTMs to detect hate speech in Tweets. The results from Study 3 in 2015 were very cutting-edge, and it was sited by hundreds of other studies for its findings, as they counter-intuitively discovered that character inputs to a ConvNet work much better than word-features for text classification.

Overall, these studies helped us decide on using GloVe vectors in our implementation. They seemed to be the best way that we could convert the text of the tweets into an input the LSTM could use effectively. Most of the studies used ConvNets, LSTMs, or both models. Overall, we decided on using the LSTM after speaking with TAs at office hours, although we are interested in experimenting with ConvNets in the future to see they compare to our results.

3 Dataset and Features

In our dataset, we had over 1 million non-Russian tweets and just over 200,000 tweets published by Russian Troll Accounts. Due to the large amount of data we had, we decided to split the the train, dev, and test sets up into 98%, 1%, and 1% of the data respectively. This means we had roughly 1,176,000 examples in our training set, and 12,000 examples in our dev and test sets respectively.

We did quite a bit of pre-processing. For the non-Russian tweets, the dataset contained a list of tweet id's, and we used a hydrator to scrape the actual tweet from twitter using the tweet id. Next, for both the Russian tweets and the non-Russian tweets, the data we acquired was not in a directly usable form. We built R scripts that took in the data and exported it into a csv/json file, which we eventually read into another pre-processing python file. In this file, we retrieved the text of the tweet, removed extraneous characters, and used the nltk tokenizer to split the text into words. Using this tokenizer was much more accurate than the built in python tokenizer and proved to be a critical step in getting our model to work with the GloVe vectors input into our model. Additionally, the nltk tokenizer allowed us to tokenize emojis, which were included in the GloVe corpus. We then wrote the tokenized text out to another csv file. These csv files were read into our LSTM.py file, at which point we used the Keras pre-processing tokenizer to convert each tweet into a sequence of word-indexes that could be fed into the Embedding layer in our LSTM. Finally, we padded the series of word-indexes with the maximum number of words that could be in a tweet, which we set to 60 (using 140 character tweets). We also loaded each of the GloVe vectors into a dictionary mapping each word in the corpus to its corresponding 200-dimensional GloVe vector, which we passed into the Embedding layer.

An example of the text from a Russian tweet in the dataset is (the tweet in the dataset contained other fields): "#IslamKills Are you trying to say that there were no terrorist attacks in Europe before refugees were let in?".

In the processing step, it was output as follows: ['#', 'IslamKills', 'Are', 'you', 'trying', 'to', 'say', 'that', 'there', 'were', 'no', 'terrorist', 'attacks', 'in', 'Europe', 'before', 'refugees', 'were', 'let', 'in', '?'].

Finally, the tweet was fed to the embedding layer as a series of 60 word indexes, with each number corresponding to the word index of the word in the vector output by the processing step. Additionally,

each word index after the length of the tweet was set equal to 0 by the padding step.

The citations for our datasets (and the GloVe vectors) are in the References section as follows: Kaggle dataset of Russian Troll tweets: [4], Harvard Dataset of Politically Themed Tweet IDs: [5], GloVe vectors: [7].

4 Methods

The first method we used, which was our baseline, was a vanilla 3-layer neural net with a sigmoid output layer. The baseline's performance wasn't all that impressive, so we quickly moved on to find a better model that would better classify tweets.

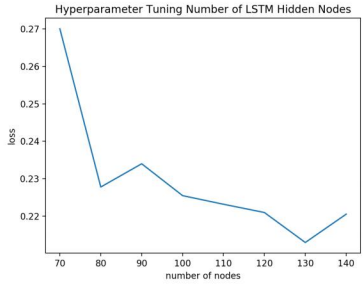
In order to solve our problem more effectively, we trained a Long-Term Short-Term memory network (LSTM), a special type of recurrent neural network. The key idea behind recurrent neural nets is that the output of a layer is fed back into itself, making them effective in the context of sequential data. An LSTM is a type of recurrent neural net that is able to "remember" information for a long time. For example, the LSTM is excellent at processing sentences, as it is able to use important words from earlier in the phrase to provide context for words that occur later, a concept called a long-term dependency. This is made possible by the LSTM's "gates" that selectively remember and forget information, determining what information should pass through and be output to the next layer. We chose to use an LSTM for our problem because of its effectiveness in text analysis.

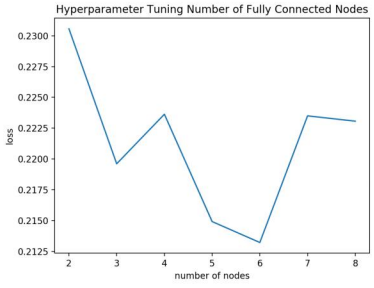
Since we were solving a binary classification problem, we decided to use a binary cross-entropy loss function, as seen below:

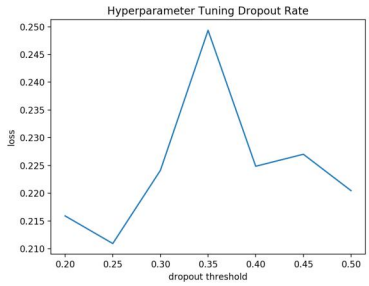
$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

5 Experiments/Results/Discussion

Our primary metric for our model was accuracy, as we were trying to find the model that could most accurately predict whether or not a tweet was posted by a Russian Troll account. Initially for our hyperparameter tuning, we used a uniform search over a logarithmic scale to find the hyperparameter combination that maximized the accuracy of our model. After, we kept all other hyperparameters at their optimal value from the first step and, sampled around the optimal value to find the hyperparameter we used in our model. The second step is visualized in the graphs below for our dropout rate, number of LSTM hidden nodes, and number of nodes in our fully connected layer. We also tuned the batch size used in our mini-batch gradient descent.





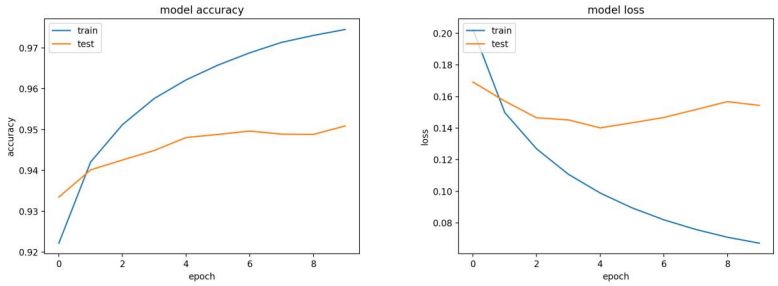


Ultimately, our model worked even better than our expectations. The results were as follows:

Model	# Train Examples	# Test Examples	Train Accuracy	Test Accuracy
3-Layer NN w/ 25D Glove vectors	35,000	5,000	68.12%	62.15%
LSTM w/o Fully Connected Layer	1.1 Million	12,000	97.45%	94.93%
LSTM with Fully Connected Layer*	1.1 Million	12,000	96.69%	95.07%

*Final model

Our LSTM model with a Fully Connected Layer was able to correctly classify the tweets with over 95% accuracy, which we attributed to our use of the nltk tokenizer, the use of GloVe vectors, as well as the aptitude of the LSTM model for this problem. The training loss and accuracy graphs of the LSTM model are seen below. Due to the large size of our dataset, the model trained quite quickly, reaching over 92% accuracy after just one epoch. We used early stopping to decide to train the model for 10 epochs, as that was the point at which the dev loss had stopped decreasing previous times we ran the model:



The confusion matrix of our model, seen below, sheds light on where the model struggled. The model was able to classify non-Russian tweets with 98.5% accuracy and Russian tweets with 91.97% accuracy, meaning that the model was better at classifying human tweets than Russian Troll tweets. This was not surprising, as many of the Russian tweets used more traditional (and oftentimes formal) language, whereas some human tweets used slang, which would have been difficult for the Russian Troll accounts to accurately include in their tweets.

Confusion Matrix

$$\begin{bmatrix} 9843 & 153 \\ 161 & 1843 \end{bmatrix}$$

Qualitatively, we were able to notice by examining the misclassified examples that a majority of them came from retweeted content from other users. Oftentimes, when a tweet was retweeted, the user didn't add any content of their own. Since we weren't using any other input factors besides the text of the tweet itself, it would be impossible for our model to learn this distinction. For example, one Russian tweet that was misclassified as non-Russian was the following, which was a tweet posted by a human that had been retweeted by a russian troll account: ['rt', '@1nataliemaines', ':', 'i', 'get', 'banned', 'for', 'not', 'liking', 'bush', 'and', 'now', 'trump', 'can', 'practically', 'put', 'a', 'hit', 'out', 'on', 'hillary', 'and', 'hes', 'still', 'all', 'over', 'country', radio', '!', 'hypocrites', '!']. In fact, our model would have no way of distinguishing this retweeted content from content posted by a human who had retweeted the same tweet. We noticed that the model usually ended up classifying conservative retweeted content as Russian Troll tweets and liberal retweeted content as human tweets. This factor alone contributed greatly to the tweets that our model misclassified.

Finally, we noticed that in our second model, the LSTM without a Fully Connected Layer did overfit to the training data slightly. Therefore, we added a Fully Connected Layer with dropout to our model which reduced the overfitting of our model and slightly increased the test accuracy of our model as well.

6 Conclusion/Future Work

Overall, we were extremely pleased with the results of our project. Our LSTM with a fully connected layer performed the best out of the models we tried. This model worked the best because of an LSTM's ability to consider long-term dependencies, while the fully connected layer helped to reduce overfitting. It's very interesting that the LSTM was able to learn language patterns well enough that it could predict whether tweets composed of original content were posted by Russian Troll Accounts with extremely high accuracy. This gives us great hope in future efforts to maintain the integrity of content on social media platforms.

If we had more time to work on the project, there are a few things we would look into pursuing. Firstly, we would add features to our input beyond solely the text of the tweet, as including metadata like username, time posted, and location could help our neural net learn more subtle indicators and trends. Additionally, as discussed above, this could help with the issue of misclassifying retweets, which seemed to be our main problem. We also thought about how our work could be applied to slightly broader scenarios, like using a softmax output to determine the probabilities of a tweet having various political biases.

Finally, with extra time, we would like to look into datasets of more current tweets, to see if our model generalizes to content that has been posted recently, or if language use has changed enough in the past 18 months that our model is no longer applicable to current content on Twitter. We would also like more time to specifically analyze hidden nodes in the LSTM to see what words have large impacts on the model's end prediction. This would allow us to see what words were used disproportionately by Russian Troll Accounts and would give us great insight into how generalize our model to other applications.

7 Contributions

The majority of the time our group was working on the project, all three of us were together working collaboratively. All three of us worked together on the main part, which was implementing the LSTM in Keras. Naturally, some members took initiative on and ownership of specific parts of the process. For example, Brendan implemented the baseline and spearheaded a lot of the data preprocessing, Ethan worked to tune the hyperparameters of the LSTM once it was built, and Elijah took initiative in error analysis once we ran the LSTM on the test set. Overall, our group dynamic was excellent, and we are all excited about the results of our teamwork!

References

[1] Bird, S., Loper E. and Klein E. (2009), Natural Language Processing with Python. O'Reilly Media Inc.

[2] François, C. (2015). keras. [online] Available at: https://github.com/fchollet/keras [Accessed 9 Jun. 2018].

[3] Hunter, J. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), pp.90-95.

[4] Kaggle.com. (2018). Russian Troll Tweets | Kaggle. [online] Available at: https://www.kaggle.com/vikasg/russian-troll-tweets [Accessed 8 Jun. 2018].

[5] Littman, J., Wrubel, L. and Kerchner, D. (2016). 2016 United States Presidential Election Tweet Ids. [online] Dataverse.harvard.edu. Available at: https://doi.org/10.7910/DVN/PDI7IN [Accessed 7 Jun. 2018].

[6] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), pp.2825-2830.

[7] Pennington, J., Socher, R. and Manning, C. (2014). GloVe: Global Vectors for Word Representation. [online] Nlp.stanford.edu. Available at: https://nlp.stanford.edu/projects/glove/ [Accessed 8 Jun. 2018].

[8] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.