CS230

Detecting Russian Troll Tweets from the 2016 U.S. Elections

Javier Echevarría Cuesta
Department of Mathematics
Stanford University
javierec@stanford.edu

Talal Rishani
Department of Computer Science
Stanford University
trishani@stanford.edu

James Schull
Department of Computer Science
Stanford University
jschull@stanford.edu

Abstract

Revelations of attempted Russian interference in the 2016 presidential election via social media have had significant implications both for individuals and for American democracy. We used a bidirectional LSTM trained on 2.5 million tweets, compiled from known Russian accounts and anonymous accounts posting during the same time period, to classify Russian propaganda tweets. We achieved a test accuracy of 91.7%.

1 Introduction

During the 2016 election, 677,775 people were exposed to Twitter posts from more than 50,000 automated accounts with links to the Russian government [1]. Such a significant effort to covertly mediate discussion on American social media platforms may have had considerable political consequences; it is in the clear interest of states, companies, and individuals to prevent such interference in the future. It is in this context that our task is defined: can we use an algorithm to classify tweets as Russian propaganda? We thus have a binary classification problem at hand, where we try to predict whether a tweet was generated by a Russian troll account based on the text of the tweet alone.

The consequent input to our algorithm is the text of a tweet. We then use word embeddings to obtain a sequence of word vectors that are fed into a bidirectional LSTM to output a predicted label for the nature of the tweet. We hope that the results of our investigation may shed light on potential approaches for preventing malicious behavior on social media in the future.

2 Related work

Before jumping into the project, we did a short survey of existing papers on the topic. We found that most spam-detection algorithms focus on the different strategies spammers engage with users, which is usually bypassing any requirement of sharing a social connection (i.e. follow/ following relationship) with a victim. For instance, in Almaatouq et al.'s paper, spam accounts are detected with respect to three categories of features, which are content attributes, social interactions, and profile properties [2]. Based on prior knowledge on how Twitter users interact, they are able to detect malicious behavior and report them as such. This process differs from ours, as it uses prior knowledge of social behavior between Twitter users, and analyses each individual account that is suspected to be corrupt.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

More aligned with the objective of our project is the paper by Xiaoling Chen et al., which combines self-learning and clustering analysis [3]. Their approach uses substrings of any length, which is beneficial for Twitter's informal style of writing, which can be grammatically incorrect. The self-training, however, does not use labeled data, which makes it vulnerable to training bias. Their model is interesting because it is useful for twitter data analysis, and is very efficient at updating as new tweets are collected. Their model's accuracy in detection scam tweets is 87%. Our approach in comparison to theirs focuses on more specific scams which in turn makes it more useful to have labeled data during training, but makes it more difficult to generalize to general scams.

In the year 2010, Alex Hai Wang used user-based as well as content-based features to detect spam profiles [4]. These are used to facilitate spam detection with a Bayesian classification algorithm. The Bayesian classifier has been judged to have the best performance with an accuracy of 93.5%. This percentage is higher than that of our model which has 91.7% accuracy. Their method, however, searches for classic scam posts that are malicious to the user, whereas our model tries to detect an even more specific type of spam. Our intuition that we should use word embeddings was confirmed by Xiao Yang et al.'s paper which describes the use of word embeddings for Twitter election classification, although they use a CNN instead of an LSTM [5]. Finally, our baseline model bags-of-words was inspired by Ankush Khandelwal paper in which he tries to classify Spanish elections tweets [6].

3 Dataset and Features

3.1 Dataset collection and cleaning

Our dataset is compiled from two primary sources. In February 2018, NBC released a dataset containing 200,000 Russian troll tweets which serves as our primary resource for training examples of tweets that should be classified as malicious [7]. To augment this dataset with non-malicious tweets from a similar period, we needed to find data whose content and context was appropriate; the topics mentioned in the Russian tweets are sufficiently context-sensitive that it was clear we could not use random tweets scraped from Twitter. In attempting to capture a distribution of tweets (with regard to content) that is sufficiently general so as to mimic the distribution of Russian propaganda tweets, we downloaded roughly 2.5 million tweet IDs from a Harvard Dataverse dataset containing the tweet IDs of approximately 280 million tweets related to the 2016 United States presidential election, collected between July 13, 2016 and November 10, 2016 from the Twitter API using Social Feed Manager [8].

Since the Harvard Dataverse datasets only contain tweet ids, we collect the whole tweets and their associated metadata by 'hydrating' the tweet ids. To do this, we use Hydrator, a desktop application that converts tweet IDs into JSON entries [9]. We extracted each tweet's text from these entries, before annotating them, merging them, and removing any duplicates. Finally we divide them into training, development, and test datasets with respective splits 90%, 5%, and 5%. Below are two examples of such inputs, where the first one is an example of a tweet from a Russian propaganda account and the second is an example of a regular tweet (hence labeled 1 and 0 respectively):

```
"RT @MikeElChingon:  Mexico also created the contraceptive pill, but yeah, white
people totally created everything."
```

```
‘‘@clairecmc Does Hillary Clinton have Alzheimers?  She had an IPad in her Podium
during the debate Unnamed sources swore they could read it.’’
```

3.2 Word embeddings

Given that our task at hand is relatively hard, even for humans, the idea of using one-hot vectors to represent each word is not feasible; we need to learn relationships between words. Our first intuition was to use pre-trained GloVe embeddings (trained either on Twitter or Wikipedia) [10]. The primary advantage of using pre-trained embeddings is that they prevent us from having to train them ourselves. Moreover, using GloVe embeddings is an example of transfer learning because they have been trained on large datasets, so they capture much more intricate relationships between words than what we would likely be able to capture with our relatively smaller dataset.

After initial implementation, however, we realized that GloVe embeddings were not wholly applicable to our task. Our dataset contains hashtags, account names, and contextual information relevant to a very specific timeframe (for instance, `#ImVotingBecause`, `BeingKimmieTwo`, `BENGHAZI`, and `#wakeupandwin`). As a result, even when using the embeddings trained on Twitter, the GloVe embeddings did not contain many of the relevant words that proved to be key to our problem. As a result, we built our own vocabulary based on the tokens in our training set along with a token for padding – '⟨pad⟩' – and a token for unknown words – 'UNK'. We also defined two mappings: word-to-index and index-to-word, which help us retrieve the information later (similar to the functions that come with GloVe).

4 Model

The learning algorithm that we use is a many-to-one bidirectional LSTM recurrent neural network, implemented using PyTorch. As described above, to transform our tweets into real valued vector inputs, we use word embeddings that we train ourselves as part of the model, based on a vocabulary built on the training set. Our model hence consists of a bidirectional LSTM whose neuron outputs are each fed through a logistic regression unit, ultimately resulting in a vector of activations (whose dimension is equal to the length of a padded tweet); we take the mean of this vector to generate our final prediction.

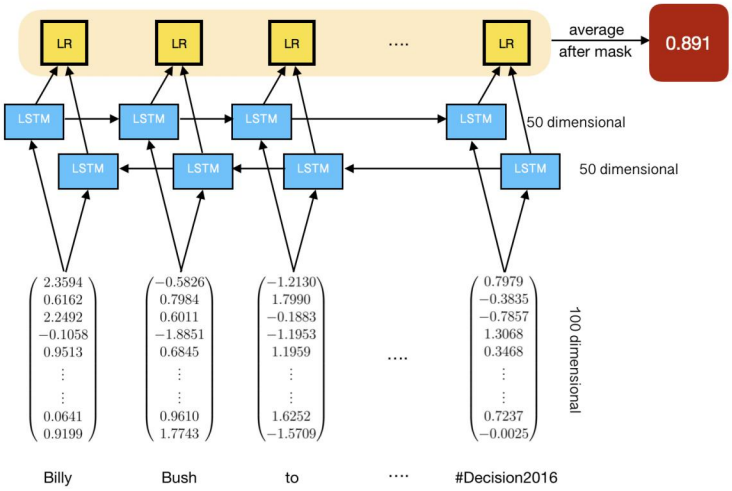


Figure 1. Diagram of our model

For a given tweet, if A is the resulting vector of activations and Y is a vector of the same dimension with all 1's or 0's depending on the ground truth of the specific vector, the loss function that we use is

$$||(A - Y) * \text{mask}||_2,$$

where $*$ indicates element-wise multiplication. The necessity for the padding and the mask comes from the way our data loader feeds the mini-batches into our model. Indeed, our data loader takes `batch_size` examples from our dataset and replaces each word with its index in the vocabulary. It then computes the length of the longest tweet in the batch and pads the rest of the tweets so as to make them all of the same length. Once fed into the model, each tweet in the batch gets converted into a sequence of embeddings which constitutes the input for the bidirectional LSTM.

By using an LSTM, as opposed to a regular recurrent neural network, our model is able to better 'remember' what is has already seen. For instance, imagine our recurrent network is checking for grammatical errors and looking at the sentence "The cat, which already ate ..., was full." Of course, it would be useful to remember whether the noun was "cat" or "cats" to determine whether the verb should be "was" or "were." Nevertheless, there might be a relatively long lapse between the time it encounters the noun "cat" and the time it sees the verb "was." We notice, however, that it would be inefficient to remember that information until the end of the sentence. First developed to deal with the exploding and vanishing gradient problem, an LSTM solves this problem by including memory cells that allow us to store a value for either short of long time periods as

necessary (hence the name long-short term memory). A bidirectional LSTM works just like a regular LSTM except it splits the neurons into two directions, one from left to right and another from right to left. This enables the model to have information from both before and after a certain word when it looks at its word embedding in the sequence.

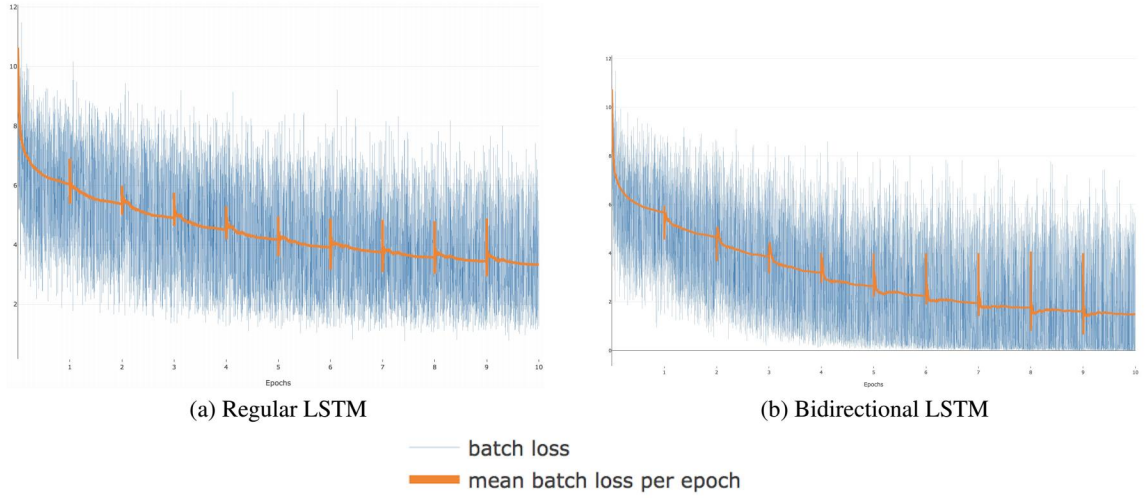
5 Experiments and Results

5.1 Experiments and hyperparameter tuning

The single-number metric that we decided to look at was accuracy. Throughout our search for the best model to use, we implemented the bag-of-words model which uses logistic regression, a regular LSTM and a bidirectional LSTM. After conducting hyperparameter search using our development set, we found the following to be optimal for a bidirectional LSTM, which is the model that performed best as described in the subsection below:

Hyperparameter	Value
Learning Rate	0.001
Batch Size	32
Epochs	8
LSTM Hidden Units	50
Embedding dimension	100

To illustrate how the bidirectional model outperforms the regular LSTM (at least during training), we can see how mini-batch gradient descent performs for the same hyperparameters but different models:



(a) Regular LSTM (b) Bidirectional LSTM

Although the graph shows that the loss keeps on decreasing for each epoch, in fact after 8 epochs our model began to overfit to the training set, causing accuracy on the development set to decrease. This is what instructed our choice of 8 epochs as a hyperparameter, to give an example.

5.2 Results and error analysis

Thanks to our hyperparameter tuning, our final results on the test set were

Model	Train Accuracy	Test Accuracy
Bag-of-words (logistic regression)	63.2%	54.8%
LSTM	94.3 %	89.8%
Bidirectional LSTM	96.2 %	91.7%

We conducted error analysis to attempt to understand what factors most heavily influenced our model's predictions, as well as what factors might be causing it to make erroneous predictions. A number of interesting

observations arose. First and foremost, it appears that our model most successfully classifies Russian troll tweets that contain pertinent hashtags and account names. This confirmed our prior suspicions: the content of the tweets in the dataset is highly similar, so the most useful features in each tweet would be names and hashtags. Indeed it seems likely that prolific troll accounts would exhibit some repetition in terms of the accounts they would re-tweet and the hashtags they would refer to. Below is an example of one such case, a Russian troll tweet that was correctly classified by our model:

```
"RT @NickAPappas:  Trumpers, explain how, and more UNK WHERE Hillary will "rig"
this election.''
```

The activations for the first three words were [0.2266, 0.8832, 0.4905], indicating that the key term in the prediction was the account name "@NickAPappas." Indeed, looking at the training set, we found that the account NickAPappas was re-tweeted in numerous other troll tweets. Looking at false positives and false negatives yielded similarly interesting observations. We found that many of the false positives appeared to be misclassified for predictable reasons. Below is one example of a non-Russian tweet that was classified as malicious:

```
``Hop on the Trump train my friends!!!  This man can do great things for our
country.  MAGA!!!  #Trump2016''
```

While none of the individual activations were so overwhelming as to generate a surefire 1 prediction, the overall content of the tweet was sufficient for the model to mis-classify the tweet. This tweet appears to be indicative of the problem above: when a vocally pro-Trump tweet contains no 'giveaway' features, like commonly re-tweeted accounts, it is very difficult to discern whether it is an attempt at inflammatory political interference or merely a reflection of authentic support. This appears to be one of the most intractable difficulties with the problem we chose to take on: it is not clear whether such ambiguous tweets *can* be reliably classified based on their text alone. The false negatives generated by our model contain many such ambiguous cases. However, looking at them also revealed some other difficulties:

```
``RT UNK Trump is NOT a racist!  UNK''
```

The tweets in our dataset are rife with spelling errors, grammatical inconsistencies, and other such idiosyncrasies; thus, 'hello,' 'Hello,' and 'helo,' though semantically the same, are recorded as three distinct words in our vocabulary. Combined with a necessary limit on the size of our vocabulary, this means that many tweets are riddled with unknown words. For short tweets, like the one above, this can remove much of the information in a tweet. Potential solutions to this problem may be to increase the vocabulary size, to use n-gram features, or to lower all the characters in the training set; all these solutions, however, have potentially significant trade-offs.

6 Conclusion and Future Work

In summary, by using a bidirectional LSTM we were able to classify Russian troll tweets with an accuracy of 91.7%, an accuracy that we were pleased with given the innate difficulties of the task. Various changes that we made –hyperparameter tuning, training our own embeddings, and altering our architecture– all made progressive improvements to the model's performance. We believe that our model's performance demonstrates that with the right dataset, classification of malicious tweets is a task in which deep learning can indeed prove very useful.

Were we to have more time with this project, we have a number of ideas for further steps. Firstly, we would retrain GloVe embeddings on a larger dataset of tweets from the 2016 election period. More broadly, and more significantly, we would be very interested to see if we could train our model to generalize to tweets *not* from the 2016 election. This would likely be much more difficult, and would likely require incorporation of vastly more data, as well as the metadata associated to each tweet: however, the successful development of a model that could accurately classify tweets from troll accounts (not just Russian accounts) in a varying political and social media environment could have valuable implications for the way in which social media companies maintain the integrity of their platforms.

7 Contributions

Summary of what each team member worked on and contributed to the project:

- Topic research: Echevarría, Rishani, Schull
- Data collection and cleaning: Schull
- Literary review and related work: Rishani
- Code
  - `build_tweet_dataset.py`: Rishani, Schull
  - `build_vocab.py`: Echevarría
  - `data_loader.py`: Echevarría
  - `net.py`: Echevarría, Rishani, Schull
  - `train.py`: Echevarría
  - `evaluate.py`: Schull
- Hyperparameter tuning: Rishani
- Milestone report and poster presentation: Echevarría, Rishani, Schull
- Final report: Echevarría, Rishani, Schull

References

[1] Twitter Public Policy (Jan. 19. 2018) *Update on Twitter's Review of the 2016 U.S. Election*, `blog.twitter.com/official/en_us/topics/company/2018/2016-election-update.html`.

[2] Abdullah Almaatouq et al. (Oct 2016) *If it looks like a spammer and behaves like a spammer, it must be a spammer*, International Journal of Information Security, Volume 15, Issue 5, pp 475-491.

[3] Xiaoling Chen, R. Chandramouli, K. P. Subbalakshmi (2015) *Scam Detection in Twitter*, Stevens Institute of Technology.

[4] Monika Verma, Sanjeev Sofat (Jan. 2014) *Techniques to Detect Spammers in Twitter- A Survey*, International Journal of Computer Applications, Volume 85, Issue 10.

[5] Xiao Yang, Craig Macdonald, Iadh Ounis (2016) *Using Word Embeddings in Twitter Election Classification*, University of Glasgow.

[6] Ankush Khandelwal, Sahil Swami (2017) *Classification Of Spanish Election Tweets (COSET) 2017 : Classifying Tweets using Character and Word Level Features*, International Institute of Information Technology.

[7] Ben Popken (Feb. 14. 2018) *Twitter deleted 200,000 Russian troll tweets. Read them here.*, `nbcnews.com/tech/social-media/now-available-more-200-000-deleted-russian-troll-tweets-n844731`.

[8] Justin Littman; Laura Wrubel; Daniel Kerchner (Nov. 23. 2016) *2016 United States Presidential Election Tweet Ids*, `https://doi.org/10.7910/DVN/PDI7IN`, Harvard Dataverse.

[9] MIT, *Turn Tweet IDs into Twitter JSON from your desktop*, `https://github.com/DocNow/hydrator`.

[10] Jeffrey Pennington, Richard Socher, Christopher D. Manning (2014) *GloVe: Global Vectors for Word Representation*.