Bots in the Net: Applying Machine Learning to Identify Social Media Trolls

Jack Cable (cablej) Grant Hugh (ghugh)

June 12, 2019

1 Introduction

Following the 2016 elections, the United States and the world at large noticed a disturbing phenomenon of the use of social media for malicious political influence. It was later discovered that the Internet Research Agency, Russia's social media troll farm, spread disinformation to tens of millions of American voters using targeted tweets and Facebook posts (US, 2018).

Social media platforms and citizens alike would benefit from additional tools to detect whether posts were created by bots. In this paper, we apply Naive Bayes, Logistic Regression, Kernel SVM, Random Forest, and LSTM neural networks to identify political trolls across social media and compare their accuracies. Using these trained models, we build a classifier for Twitter content that predicts the likelihood a given tweet was created by a troll bot in real time. This tool can aid moderation for social media companies by flagging suspicious tweets in a live feed.

2 Related Work

The field of social media bot detection is still emerging and thus the current body of literature is relatively shallow. Initial studies made a broad overview detailing the percentage of twitter users that are bots and the percentage of tweets that are made by bots on Twitter. These studies found that around 5% of US political Twitter users are bots (Badawy et al., 2018) and over 50% of tweets on Russian politics are made by bots (Stukal et al., 2017), thus motivating further study of bots in social media, and pinpointing the area of study to US and Russian politics and their intersection. Additionally, the current body of literature also features analysis of tweets made by identified bots using natural language processing techniques (Griffin and Bickel, 2018).

Surprisingly, before the 2016 election, there were studies utilizing a variety of methods from Random Forest classifiers (Chu et al., 2012) to SVMs (Ratkiewicz et al., 2012) to classify bots. However, with the renewed importance of the field in the aftermath of the 2016 election and drastic improvements in machine learning technology, there should be a push for more research in political bot manipulation on social media. There was an attempt to classify bots in 2017 using logistic regression and decision trees, with statistically significant precision of 78.5% (Im et al., 2019), yet there is a notable lack of more sophisticated machine learning techniques such as deep learning. Furthermore, there is also a lack of direct applicability from the current work in the field to bot detection on social media today. Therefore, our aim of performing direct live detection of trolls on Twitter using a variety of machine learning techniques from Naive Bayes to Neural Nets, is a logical and important next step.

3 Dataset and Features

We utilized two primary resources to gather a large number of political tweets made by trolls (positive labels), and by real people (negative labels). For the positive labels, we used a subset of FiveThirtyEight's 3 million Russian troll tweets (Roeder, 2018). For the negative labels, we wanted to have real tweets that would have a similar political context to the Russian troll tweets. Otherwise, we would run the risk of simply discriminating whether a tweet was about the 2016 election or not rather than whether the tweet was made by a bot. Therefore, we used George Washington University's Tweet Sets database of tweet ids related to the 2016 elections (Littman, 2018), which we then obtain tweet contents for by scraping the Twitter API.

To preprocess the data, we filtered for only English tweets (using available metadata in the datasets) and eliminated tweets with no text content (i.e., only an image). For our algorithm, we used a total of 142,560 unique tweets, of which 61,665 were positive and 80,895 were negative. Among these tweets, we split the data 0.8/0.2 train/test for all algorithms.

We extracted features using several methods:

- Word count: we experimented with two bag of words approaches, first representing each tweet as a binary indicator vector, where a 1 denotes that particular word was used, and second as a count vector, with the number of times each word appears in a tweet. In both instances, the size of the vocabulary was capped at 5000 words.

- TF-IDF: We extracted word, ngram, and character level TF-IDF scores, which weight words based on how common they appear in a document. This allows better identification of uncommon words, which may be strong indicators of troll accounts.

- Word embeddings: For the LSTM neural network, we utilized word embeddings to capture more complexity by mapping words to higher-dimensional feature vectors.

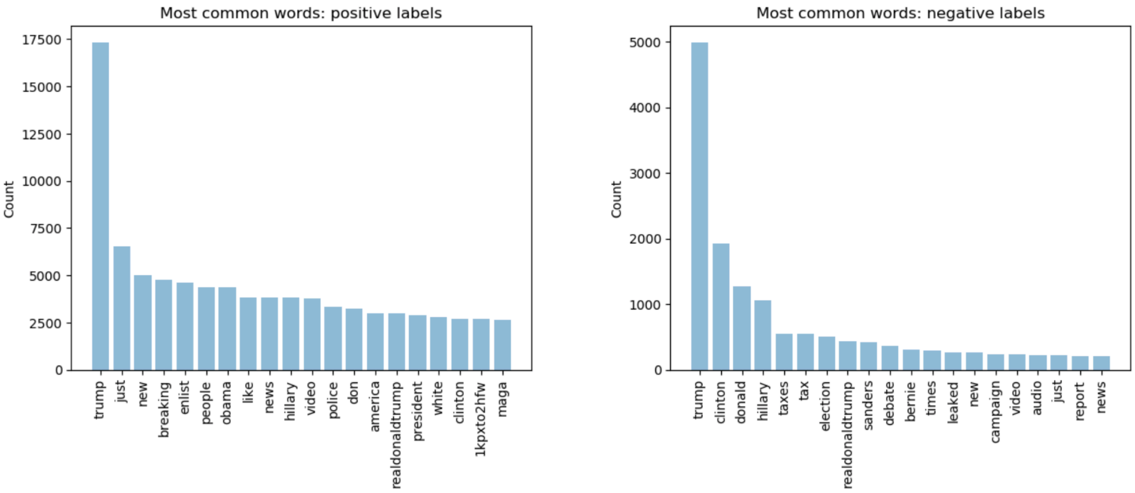

Figure 1: Most common words across positive and negative datasets

4 Methods

4.1 Naive Bayes

Naive Bayes is a simple classification algorithm which treats each each input feature independently. Naive Bayes computes the likelihood of each class given an input vector according to the following formula:

$$p(y=1|x) = \frac{(\Pi_{j=1}^{d} p(x_j|y=1))p(y=1)}{(\Pi_{j=1}^{d} p(x_j|y=1))p(y=1) + (\Pi_{j=0}^{d} p(x_j|y=1))p(y=0)}$$

As is evident from the formula, Naive Bayes only utilizes raw word usage and cannot capture the relationship between words, which makes it a rather simplistic method. Furthermore, it makes the strong assumption that different features are independent; in our case, this means that not only does it not capture the relationship between words but it also assumes that the usage of any particular word is independent of the usage of all other words, which is probably incorrect. However, Naive Bayes serves as a good baseline to compare more complicated methods against.

4.2 Logistic Regression

Logistic regression is a classification algorithm that maximizes the likelihood of labels given input data and tuned weights. In particular, logistic regression uses the sigmoid function, given by $g(z) = \frac{1}{1+e^{-z}}$, to express $p(y = 1|x)$. Logistic regression then maximizes the following log likelihood, using gradient ascent:

$$\ell(\theta) = \sum_{i=1}^{n} y^{(i)} \log g(\theta^T x^{(i)}) + (1 - y^{(i)}) \log(1 - g(\theta^T x^{(i)}))$$

4.3 Kernel SVM

Support Vector Machines attempt to compute a decision boundary that maximizes the margin of data, i.e. data is the most separated possible. We use the RBF kernel, defined by $K(x, x') = \exp(\frac{-||x-x'||^2}{2\sigma^2})$. SVM performs optimization according to the following constraints:

$$\min_{w,b} \frac{1}{2}||w||^2 \qquad \text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1, i = 1, \ldots, n$$

4.4 Random Forest

The Random Forest algorithm utilizes multiple decision trees where only a random subset of features are considered. The random forest approach allows us to capture a greater range of correlations between different features for each of our featurizations, wherein the algorithm can capture correlations between words. We utilize the Gini impurity criterion, which is represented by the equation GINI$(t) = 1 - \sum_{i=1}^{j} P(i|t)^2$. (Breiman and Cutler, 2004).

4.5 LSTM Neural Network

An LSTM neural network is a form of recurrent neural network (RNN) which operates by learning long-term dependencies alongside short-term dependencies. The recurrent structure allows the output of the previous LSTM cell (the context) to be incorporated in the input of the next LSTM cell. Each LSTM cell can drop and add information, as well as change the weighting of information, to form a new context that will be passed to the next cell.

To change the context, each LSTM cell contains gates that operate much like normal cells in a standard neural network. First we have a "forget-gate", which weights different attributes of the context on a 0-1 scale using the equation $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ where f_t represents the new weightings for all of the old information in the context, W_f is all the previous weightings, h_{t-1} is the previous output (probability that the tweet is made by a bot or not), x_t is the new input (new feature of the tweet), and b_f is the bias for the weights. The sigmoid activation is chosen to put values between 0 and 1. The final context is simply an aggregation of the new weightings for the old context attributes and the weighting for the new context attributes and the new output h_t, is a function of the output of the standard neural network equation $o_t = W_t \cdot [h_{t-1}, x_t] + b_t$ and the context (Olah, 2015).

LSTMs are effective compared to other forms of RNNs precisely because they can incorporate past data as context into present operations. This allows LSTMs to capture the relationship between words in a tweet like with random forests, but it can also start to form an idea of the meaning of a tweet from context, which results in more informed judgments on whether a tweet is made by a bot or not.

5 Experiments

Our experiments were conducted using the train/test datasets described above. We tested each of Naive Bayes, Logistic Regression, Kernel SVM, and Random Forest using binary indicator vectorization, word count vectorization, word TF-IDF, Ngram TF-IDF, and character TF-IDF. In addition, we trained a LSTM neural network with custom word embeddings of length 50. In total, we conducted 21 tests, and for each, we measured the accuracy on the training and test set.

For our Naive Bayes implementation, we utilized the Multinomial Naive Bayes model from the `sklearn` library (Pedregosa et al., 2011). The Multinomial model assumes that all words in a document are generated independently and from the same Multinomail distribution, depending on the label of the document. Although this is a strong assumption, it proves effective in classifying with our dataset. We again utilized the `sklearn` library for our implementation of logistic regression. In the context of parsing tweets, logistic regression assigns each word featurization a weighting which then forms a linear boundary that classifies whether or not the tweet was made by a bot.

Both Kernel SVM and Random Forest were unable to efficiently train on all 142,560 datapoints. Instead, we instead trained on a subset of 12,514 of these datapoints.

The SVM and neural network models required hyperparameter tuning, as at first the SVM algorithm performed no better than a random classifier. In particular, we attempted values $0.001, 0.01, 0.1, 1$, and 10 for the penalty C and values $0.001, 0.01, 0.1$, and 1 for the kernel coefficient γ. After tuning, we obtained $C = 10$ and $\gamma = 0.1$ to be the optimal hyperparameters, which yielded much higher accuracies.

Our LSTM neural network was made using the `Keras` library (Chollet and others, 2015). We experimented with different values for the number of LSTM layers, dropout rates, batch size for training, activation/loss functions, and optimizer algorithms. In our final model, we used a three layer architecture consisting of one LSTM layer followed by a conventional densely connected layer and an output layer. For our LSTM layer, we used rectified linear unit (ReLU) activation function and we set the dropout rate to 0.2. On the densely connected layer, we again used the ReLU activation function, and finally, we utilized the sigmoid activation to put the output to be a probability between 0 and 1. We trained this model using binary crossentropy loss, $-(y\log(p) + (1-y)\log(1-p))$, and optimized it using the RMSProp algorithm to create our final neural net.

6 Results

After training, we obtained the following accuracies:

Method	Binary Count		Word Count		Word TF-IDF		Ngram TF-IDF		Char TF-IDF	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Naive Bayes	0.953	0.898	0.951	0.901	0.883	0.874	0.814	0.801	0.884	0.882
Logistic Regression	0.977	0.939	0.977	0.936	0.934	0.926	0.835	0.818	0.948	0.940
Kernel SVM	1.00	0.947	1.00	0.944	0.987	0.959	0.917	0.861	0.995	0.964
Random Forest	0.993	0.915	0.993	0.906	0.991	0.930	0.930	0.826	0.998	0.955

The LSTM neural network obtains a train accuracy of 0.981 and a test accuracy of 0.957, slightly outperforming most other methods. The one exception was Kernel SVM with character TF-IDF featurization, which obtained the highest test accuracy of 0.964 out of all tests. However, simple methods such as Logistic Regression with character TF-IDF featurization also did surprisingly well, with an accuracy of 0.948 on the test dataset, demonstrating that tweets made by bots may be easier to distinguish than expected, and that the methodology of detecting tweets made by bots need not be overly complex. It also appears that character TF-IDF featurization yields the best results out of all featurizations for all methods except Naive Bayes, where word count vectorization gives a higher accuracy.

From the above plots, we see that the LSTM neural network begins to overfit the training data

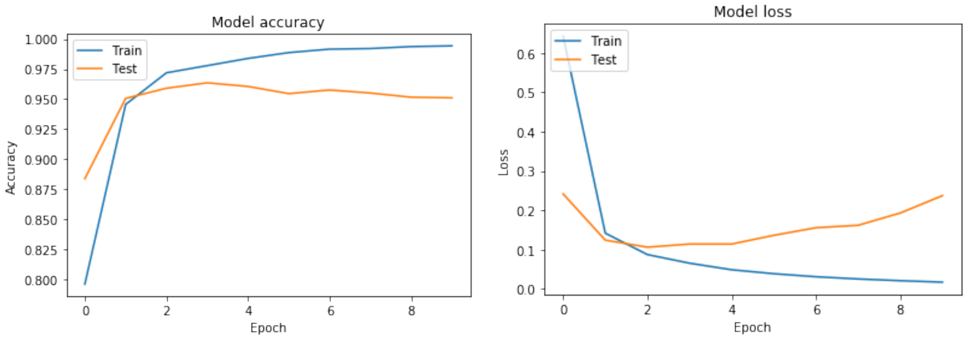

Figure 2: LSTM accuracy and loss over successive epoches

after a few epoches. This is seen by the model loss continually decreasing for the training data but increasing for the test data. In order to mitigate this, we applied the Keras `EarlyStopping` monitor with a delta of 0.0001, which reduced overfitting on the training dataset.

In addition, we obtained confusion matrices for each model and featurization. Shown are the confusion matrices for the highest-performing featurization for each model under test data:

Method	Naive Bayes		Logistic Regression		Kernel SVM		Random Forest	
	Pred. 0	Pred. 1	Pred. 0	Pred. 1	Pred. 0	Pred. 1	Pred. 0	Pred. 1
Actual 0	14770	1365	15433	702	1581	27	1575	33
Actual 1	1453	10924	927	11450	52	843	69	826

Across all methods we see that the number of false negatives is slightly higher than the number of false positives. Depending on the application of this algorithm, it may be desirable to focus more on decreasing either the number of false positives or false negatives. For example, if being flagged as a troll resulted in an automatic ban for the user, it would be desirable to have less false positives. Alternatively, if the algorithm is used in conjunction with human moderation, having more false positives may be acceptable in order to better filter content. This could be accomplished by either increasing or decreasing the ratio of positive to negative labels.

6.1 Applications

In order to demonstrate an immediately applicable tool, we built a website that classifies tweets live using the Twitter streaming data API (twi, 2019). Such a tool can be utilized by social media companies such as Twitter to moderate suspicious tweets as they are posted.

7 Conclusion and Future Work

In this paper we demonstrated the need for more applicable research on the presence of bots in social media in the aftermath of the 2016 election and presented implementations and evaluations for machine learning algorithms detecting political bots on Twitter. We culminated this work with a real-time bot detection system for tweets. Our survey of machine learning algorithms concluded that Kernel SVM with character TF-IDF featurization gave the highest accuracy, but almost all the other algorithms gave comparable results. However, we faced limitations in the scope of our dataset; in the future, we hope to expand our dataset to incorporate political tweets made by bots and real people from a variety of online databases. Additionally, we may attempt using other forms of deep learning such as C-RNN-GAN models to adapt to possible novel attacks by adversaries. While the field of bot detection in social media remains in its infancy, we believe that our analysis of different machine learning techniques and live Twitter bot detection application serves to provide a guideline for future attempts to use machine learning for bot detection and sets an important norm for applicable research in the field.

Contributions

Grant - Initial data processing and Naive Bayes implementation, implementation of LSTM neural network and hyper-parameter tuning

Jack - Collection of both datasets and preprocessing, Logistic Regression, Kernel SVM, and Random Forest implementations, SVM hyper-parameter tuning, live classification tool website

Link to project code: `https://github.com/cablej/CS229-Final-Project`

References

A. Badawy, E. Ferrara, and K. Lerman. 2018. Analyzing the Digital Traces of Political Manipulation: The 2016 Russian Interference Twitter Campaign. In 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pages 258–265, Aug.

Leo Breiman and Adele Cutler. 2004. Random Forests.

François Chollet et al. 2015. Keras. https://keras.io.

Z. Chu, S. Gianvecchio, H. Wang, and S. Jajodia. 2012. Detecting Automation of Twitter Accounts: Are You a Human, Bot, or Cyborg? IEEE Transactions on Dependable and Secure Computing, 9(6):811–824, Nov.

Christopher Griffin and Brady Bickel. 2018. Unsupervised Machine Learning of Open Source Russian Twitter Data Reveals Global Scope and Operational Characteristics. arXiv, Oct.

Jane Im, Eshwar Chandrasekharan, Jackson Sargent, and Paige Lighthammer. 2019. Still Out There: Modeling and Identifying Russian Troll Accounts on Twitter. AAAI, Jan.

Justin Littman. 2018. justinlittman/TweetSets: Version 1.0.0, June.

Christopher Olah. 2015. Understanding LSTM Networks, Aug.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python.

J. Ratkiewicz, M D Conover, and M Meiss. 2012. Detecting and Tracking Political Abuse in Social Media. Fifth International AAAI Conference on Weblogs and Social Media, page 297–304.

Oliver Roeder. 2018. Why We're Sharing 3 Million Russian Troll Tweets, Jul.

Denis Stukal, Sergey Sanovich, Richard Bonneau, and Joshua A. Tucker. 2017. Detecting Bots on Russian Political Twitter. Big Data, 5(4):310–324.

2019. Consuming streaming data - Twitter Developers.

2018. Exposing Russia's Effort to Sow Discord Online: The Internet Research Agency and Advertisements.