TrollSpotting:
Identifying Tweets from Russian Trolls

Benjamin Anderson
Stanford Dept. of Computer Science
banders9@stanford.edu

Background
Since the 2016 election, Twitter has identified thousands of accounts belonging to employees of the Internet Research Agency (IRA), a "troll factory" operating out of St. Petersburg, Russia, which engages in online influence operations on behalf of Russian intelligence services and business interests. Millions of posts from these accounts have been removed from the platform—but fake news and information warfare continue to be a pervasive problem on Twitter and other social media networks. Democracy relies on the ability of citizens to access accurate information—thus, it is imperative that social media companies find ways to stop these disinformation operations at the source. At the same time, social media companies can come under fire for unfairly censoring and suppressing content written by real users, including journalists, activists, and public figures. Therefore, developing a method for accurately identifying and removing content posted by trolls, while leaving ordinary users untouched, is extremely important. My objective in this project is to develop a machine learning model which accurately distinguishes tweets written by ordinary users from tweets written by trolls based on semantic content.

Related Work
My work relates to two well-studied problems in machine learning: (1) Applying machine learning to identify spam; and (2) Applying machine learning to classify tweets. Modern e-mail products, such as Gmail, rely on very accurate spam filters that depend on machine learning to identify and segregate junk mail in a separate inbox. There are countless papers discussing this problem; Bhowmick and Hazarika (2016) review some of the most common modern supervised techniques for text-classification on emails.[1] They identify the Naïve Bayes algorithm (and sophisticated extensions), support vector machines, and "bagged" and "boosted" ensemble models as common successful approaches (I discuss several of these in more detail in the "Classification Methods" section). They also discuss methods for extracting meaningful features from the text of an email, including "bag-of-words" features (discussed in the "Feature Extraction Methods" section), which treats individual words as features, and *n*-gram features, which are taken from length-*n* sequences of consecutive words.[1] It is plausible that techniques that succeed in the domain of classifying email spam would also be useful for identifying junk or "troll" tweets.

When it comes to supervised learning on tweets, one of the most well-studied tasks is sentiment classification: labeling a tweet as "positive" or "negative." A group of Stanford graduate students (Go, Bhayani, and Huang) use distant supervision to train a sentiment classification model on tweets, applying similar techniques to those mentioned previously for email spam.[2] They also published the labeled training data, which I use as part of the corpus of negative examples for this project. There has also been a significant amount of research aimed at distinguishing tweets written by bots from tweets written by humans, which is similar (though not identical) to my objective. Chu et al. (2012) design a multi-part system, including text classification and features based on metadata, to distinguish posts written by humans, bots, and "cyborgs" (human-assisted bots or bot-assisted humans). They use the Naïve Bayes algorithm for text classification, and to combine the results of this text classification with other information, they leverage random forests (a type of decision tree ensemble). The full classification pipeline achieves 96% accuracy, doing the best at identifying humans and bots (98%) and much worse at identifying cyborgs (92%).[3] Chen et al. (2016) develop a system for streaming classification of tweets from bots, which differs from some classification schemes because it *only* leverages features in the tweet, not of the larger social graph (which can be too time consuming to deploy in a streaming setting). They tested several machine learning algorithms, including K-nearest neighbors, decision trees and random forests, SVMs, and Naïve Bayes. They achieve the best results with decision tree ensembles, with TPR (accuracy on "spam" examples) of 92.9%, and FPR (error rate on "non-spam" examples) of 5.6%.[4] Though the machine learning methods are relevant to the current study, these authors focus more on meta-data rather than semantic content (which is my focus). Ideal classification systems might leverage both, but I limit my attention to features based on words in the tweet.

The tweet-classification systems mentioned so far also differ from my objective because they focus on spam tweets sent by *bots*, whereas the type of spam and disinformation disseminated by the Internet Research Agency is *written by humans* for nefarious purposes. Moreover, traditional spammers can have different objectives than the IRA, e.g. getting someone to click on a malicious link, whereas the IRA's goal is to exert political influence. Thus, building a system to classify these tweets is a unique problem to which not much attention has yet been devoted. A 2019 thesis by Kannan Neten Dharan tackles this exact problem, using the same dataset of troll tweets I explore in this study as positive examples, and randomly sampling new tweets from Twitter for negative examples. He tests both bag-of-words features and Word2Vec embeddings, and applies a variety of machine learning algorithms, from linear models to neural networks trained end-to-end.[5] Dharan's work on distinguishing IRA tweets from normal tweets provides a good starting-point for my project, including benchmarks to compare: 84% accuracy with support vector machines and bag-of-words features, and 99% accuracy with a convolutional BERT (transformer) model trained with word2Vec embeddings.

In the present study, I build on this work in several ways. First, I compile a dataset of negative examples that include some "ordinary" tweets, but others that are explicitly political (as many IRA tweets are), posing a more direct challenge to distinguish troll tweets from non-troll tweets. Second, I develop a machine learning pipeline that includes feature extraction using a *pretrained* transformer, which then feeds into a linear model or decision tree (as opposed to the costly step of training a neural network end-to-end on the Twitter data). This can be benchmarked against the simpler approach of using bag-of-words features, and also against Dharan's more expensive and time-consuming approach of training a neural network end-to-end. Finally, I experiment with decision tree ensembles, which can serve as be a happy medium between the simplicity and speed of linear models, and expressive but hard-to-train neural networks.

Data

Darren Linvill and Patrick Warren, researchers at the University of Clemson, archived and catalogued a large collection of tweets from accounts that Twitter has identified as belonging to Internet Research Agency employees—"troll tweets". In their qualitative analysis of millions of these tweets, they sorted the tweets into distinct categories, such as "Left Troll", "Right Troll", and "Fearmonger".[6] For the purposes of this project—which is restricted to binary classification—I treat these all as positive examples of troll tweets. For negative examples, I used a combination of tweets from several sources. My objective was to compile tweets that are "normal"; the type of tweet one might want to distinguish as "not a troll tweet". Once source was the Sentiment140 dataset[2], a dataset of millions of tweets collected by Stanford graduate students, and used to train sentiment classification models. One problem with this dataset is that it might be *too* easily distinguishable from tweets by the troll accounts, simply because trolls are more likely to tweet about politics and news, while tweets in the Sentiment 140 dataset can be far more banal. For example: "`Just arrived on campus. Still dark and dreary outside. Oh well, time for breakfast.`" In order to account for this, and make the task a bit more challenging, I supplemented these negative examples with another dataset[7], which contains over one million tweets from U.S. politicians. Though politicians' posts surely differ in some ways from those of the average Twitter user, this addition will require the model to do more than distinguish "political" tweets from "non-political" tweets, and hopefully discover the important markers that identify tweets written by trolls. Altogether, my dataset contains around 3.9 million tweets, though most models did not need nearly this many to train. 1 million of these were sampled from Sentiment 140, 1 million from politicians' tweets, and the remaining 1.9 million from trolls. I split this data into training, dev, and testing sets (70%, 15%, 15%), but used subsets of each of these for most experiments (as I will show in the "Experiments and Results" section, learning curves showed that 100,000 examples was plenty for most models).

Feature Extraction Methods

Deep learning models are commonly used for text classification. However, they are time-consuming and difficult to train end-to-end. For this project, I tested two methods of extracting features from the text of tweets, which were then fed into several relatively simple and quick-to-train models, including support vector machines, logistic regression, and decision tree ensembles. I compared the more straightforward "bag-of-words" approach to the more complex neural feature extraction, to determine how much performance is gained from the neural approach. I also tested dimensionality reduction on bag-of-words features. In this section, I explain each of the feature extraction methods in detail.

Bag-of-Words Feature Extraction

The first approach for extracting features from tweets is the classical "bag-of-words" method, which begins by creating a vocabulary V from all tweets in the training data, then represents each tweet as a vector of size | V |, with a 1 in a given position if the tweet contains that word, and a 0 if not. The advantages of this approach are simplicity interpretability. It is possible—when using logistic regression or Naïve Bayes—to analyze coefficients or probabilities associated with a word, which can provide direct insight into the differences in identifying words between troll tweets and ordinary tweets.

Dimensionality Reduction

After creating a vocabulary from the millions of tweets in the training dataset, and selecting, out of the hundreds of thousands of words in the vocabulary, a subset that were common enough, but not so common as to be in nearly every tweet (i.e. good candidates to identify unique features of tweets), I was still left with several thousand features. In order to create a smaller, easier-to-work-with dataset from these sparse and numerous features, I applied singular value decomposition to reduce the dimensionality of the dataset (using the *scikit-learn* function `TruncatedSVD`). Singular value decomposition, or SVD, computes a decomposition of data matrix A into three matrices: $A = V \Sigma U^T$. Σ is a diagonal matrix of singular values (sorted from largest to smallest), and V and U are orthonormal matrices. This is an exact factorization that exists for any matrix A.[8] Then, to represent the dataset in fewer dimensions, i.e. "truncated SVD", we can keep the k largest singular values, and use the (truncated) $V \Sigma$ as the new representation of the data. I retained the original bag-of-words dataset with 3,808 features, and used SVD to create a new dataset with 500 features.

Neural Feature Extraction

The second, more modern approach to feature extraction is to feed tweets to a deep neural network (pretrained on a task such as sentence classification), and use the hidden state(s) of that model as the input features to the classification algorithm. I used a pretrained DistilBERT model from HuggingFace.[9] The math behind transformers like DistilBERT is too complicated to explain in detail here, but in brief, DistilBERT relies on contextual word embeddings (representations of words that depend on context) and "attention" layers (which identify the most important words in a sentence to focus on) in order to extract sophisticated representations of a sentence.[10] The advantage of this approach is that the resulting "sentence embeddings" are richer than a bag of words—they capture information about context and word order, not just the presence and absence of words. We would expect models using these features to have higher performance. On the other hand, neural feature extraction is much less interpretable than bag-of-words, since it's not possible to open the black box and assign meaning to each of the 768 components of DistilBERT's hidden state.

Classification Models

Logistic Regression

Logistic regression is an algorithm used for binary classification, i.e. datasets where labels can be 0 or 1.. It considers the class of hypothesis functions of the form $h_\theta(x) = g(\theta^\top x)$, where g(z) is the logistic function $1 / (1 + e^{-z})$. The logistic function "squashes" the output of $\theta^\top x$ to the range (0, 1), so predictions can be treated like as the "probability" of belonging to class 1, or discretized by rounding to 0 or 1. To choose the parameters θ, we maximize the objective function:

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

...which is the maximum likelihood estimator, given the probabilistic assumption that we want to maximize the likelihood that h_θ corresponds to the probability $p(y \mid x)$.[11] The decision boundary is linear in the inputs and parameters.

Support Vector Machines

Like logistic regression, the "support vector machine" (SVM) is an algorithm to classify data using a linear decision boundary. However, the intuition for drawing the boundary is different. The basic SVM algorithm assumes that data is linearly separable, and selects the decision boundary that maximizes the *margin* between the data and the line, where the margin is (loosely speaking) the distance between the decision boundary and the closest data points of each class (which are called the support vectors). This decision boundary is chosen by picking parameters ω and b that maximize the margin, subject to the constraint that all points are correctly classified.[12] Then, at prediction time, we assign label 1 (positive class) if $\omega^\top x + b > 0$, and −1 (negative class) otherwise. When the data is not linearly separable, the SVM algorithm is extended with a soft penalty rather than a hard constraint for misclassified points. We minimize:

$$f(\mathbf{w}, b) = \frac{1}{2} \sum_{j=1}^{d} w_j^2 + C \sum_{i=1}^{n} \max\Big\{0, \quad 1 - y_i \big(\sum_{j=1}^{d} w_j x_{ij} + b\big)\Big\}$$

...where the first term encourages "small" weights, and the second term penalizes misclassified points.[13]

Naïve Bayes

The Naïve Bayes algorithm is a generative model used for classification when the input features are discrete—as they are when we use bag-of-words features. Naïve Bayes models the conditional probability of each feature given the label as a Bernoulli random variable, and in our case, since a tweet can either be "troll" or "not troll", the prior class probability is also modeled as a Bernoulli random variable. Each of these random variables corresponds to a parameter ϕ, which is estimated with maximum likelihood. To simplify the calculation of the joint likelihood, the Naïve Bayes algorithm employs a strong independence assumption: each of the entries of x, the feature vector, is conditionally independent given y, the label. This means that, given *n* training examples, the joint likelihood of the parameters can be decomposed as follows:

$$\mathcal{L}(\phi_y, \phi_{j|y=0}, \phi_{j|y=1}) = \prod_{i=1}^{n} p(x^{(i)}, y^{(i)}) = \prod_{i=1}^{n} p(x^{(i)} \mid y^{(i)}) p(y^{(i)}) = \prod_{i=1}^{n} \left(p(y^{(i)}; \phi_y) \prod_{j=1}^{d} p(x_j^{(i)} \mid y^{(i)}; \phi_{j|y^{(i)}}) \right)$$

Each of the parameters ϕ_y, $\phi_{j|y=0}$, and $\phi_{j|y=1}$ has a closed-form solution. After estimating the parameters, we predict the class y given the features x by using Bayes' Rule to compute p(y = 1 | x) and p(y = 0 | x), and choosing the y that maximizes the probability.[14] In practice, parameter estimation and prediction use log-probabilities to avoid underflow.

Gradient-Boosted Decision Tree Ensembles

A decision tree is a non-linear classifier that can be visualized as a collection of nodes, arranged like a binary tree. Each node corresponds to a condition that "splits" the data into two groups, which then proceed down the tree to be further subdivided. The leaves of the tree render decisions (in our case, "troll" or "not troll"). To classify an item, we start at the root of the tree, and then proceed down the tree, applying each condition until a leaf is reached, which gives the label. There are many ways to fit a decision tree, but in general, the goal is for nodes to maximally separate the two classes.[15] Decision tree *ensembles* are models that combine many decision trees, rendering a decision that is either an average or a majority vote of the constituent trees. There are many methods to intelligently add new trees added to the ensemble. One such method is "gradient-boosting", in which new trees try to fit the residual errors of the previous trees..[16]

Experiments, Results, and Discussion

For each of the three feature sets (bag-of-words, bag-of-words with SVD, and DistilBERT), I tested logistic regression, support vector machines, and gradient-boosted decision tree models. Linear SVMs and logistic regression did not require hyperparameter tuning, since I did not employ regularization. For the decision tree approach, I tuned the learning rate and L2 regularization parameters using a grid search. For the full-dimensionality bag-of-words features, I also tested Bernoulli Naïve Bayes, since it is appropriate for features that can take on values 0 or 1.

I began by producing learning curves up to 100,000 training examples for each model and dataset. There is not space to reproduce all of them here, but I noticed that in most cases, the learning curve appeared to roughly "converge" at or before 100,000 training examples (see below). On the other hand, I found that the gradient-boosted decision tree algorithm (with certain settings of hyperparameters) often showed significant variance, as evidenced by a gap between the training error and test error (see below). Training on the full dataset is very time-consuming, so I only did it for the decision tree models, which seemed that they would benefit from more data to reduce variance.

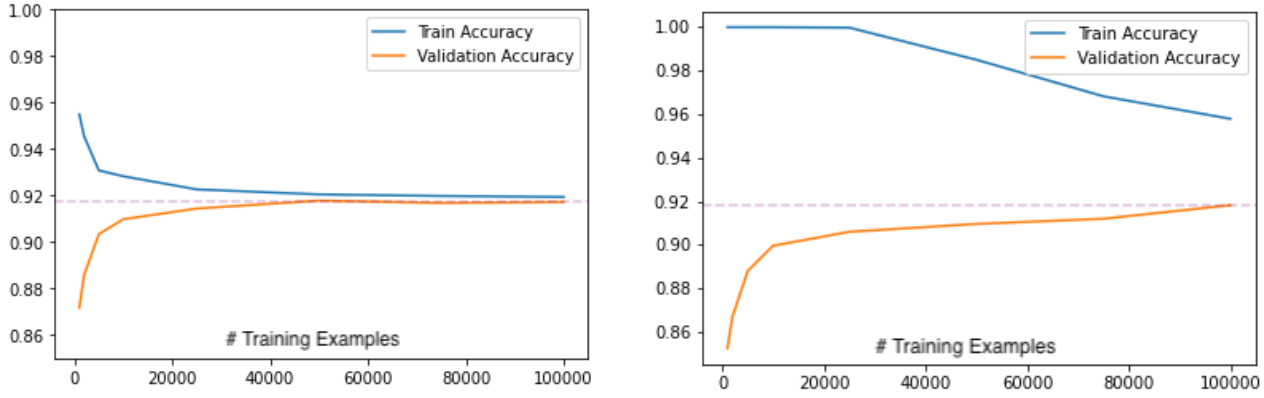

Figure 1: Learning curves (accuracy vs. number of examples) for logistic regression (left) and decision trees (right), using DistilBERT features

Evaluation

To evaluate the performance of each model and dataset, I use a few metrics (computed on test data): *accuracy*, the rate of correct classification on *all* examples; *precision*, the fraction of examples *classified* as positive that are *actually* positive; *recall*, the fraction of positive examples that are correctly classified as such; and the *F1 score*, the harmonic mean of precision and recall. I compare the classifiers on these metrics below (there's not enough space for confusion matrices). I also show training accuracy, which provides insight into overfitting/underfitting when compared with test accuracy.

Model	**# Examples**	**Train Acc.**	**Test Acc.**	**Precision**	**Recall**	**F1 Score**
Bag-of-Words + Logistic Regression	100,000	0.865	0.846	0.847	0.847	0.847
Bag-of-Words + SVM	100,000	0.860	0.842	0.830	0.866	0.847
Bag-of-Words + Naïve Bayes	100,000	0.832	0.833	0.835	0.834	0.835
Bag-of-Words + Decision Trees	~2.5M	0.864	0.866	0.869	0.865	0.867
Bag-of-Words + SVD + Log. Regression	100,000	0.807	0.799	0.794	0.811	0.803
Bag-of-Words + SVD + SVM	100,000	0.804	0.799	0.780	0.837	0.808
Bag-of-Words + SVD + Decision Trees	~2.5M	0.842	0.826	0.825	0.832	0.829
DistilBERT + Logistic Regression	100,000	0.919	0.916	0.919	0.915	0.917
DistilBERT + SVM	100,000	0.918	0.914	0.923	0.905	0.914
DistilBERT + Decision Trees	~2.5M	**0.935**	**0.928**	**0.928**	**0.930**	**0.929**

Parameter Analysis

By analyzing the probabilities associated with Naïve Bayes, and coefficients for logistic regression, we can identify words in the vocabulary that are more likely to be associated with troll tweets or ordinary tweets. Some notable examples:

- **Troll:** "#top", "#sports", "#news", "#blacktwitter", "#igetdepressedwhen", "#maga", "trump"
- **Non-Troll:** "awww", "yay", "haha", "hehe", "xx", "#smallbiz", "#sotu"

For both categories, there were also a number of stop words with high Naïve Bayes probabilities, which means that the heuristics I used to narrow down the vocabulary didn't work well. Stop words can make classification worse—I would do explicit stop-word removal during preprocessing if I were to repeat this experiment. These results also suggest that my leaving in the "#" symbol made sense for Twitter data—common "hashtags" are distinguished from words that aren't hashtags, and are evidently informative for classification. Finally, it is interesting that one of the ways trolls are distinguished from "normal" people is by terms like "haha" and "awww"—casual positive emotion sets them apart.

Discussion

These results show the clear benefit of neural feature extraction over bag-of-words: 93% vs. 87% accuracy for the best models. Comparing to Dharan's baseline,[5] I achieve 86% accuracy with SVM and bag-of-words (compared to his 84%). My models with neural feature extraction did not outperform Dharan's end-to-end BERT (93% vs. 99% accuracy), but it's a decent middle ground—in a setting where some error is acceptable, this might be enough, and you could skip the time and cost of training a neural network. When it comes to SVD, we also see that there is a significant cost to dimensionality reduction. This might not be the case with a more intelligent choice of vocabulary. Finally, whether we use the already-expressive DistilBERT features, or the simpler bag-of-words features, my experiments show clear benefits to the non-linearity of decision trees, compared to algorithms like support vector machines and logistic regression, which have linear decision boundaries. None of the models appear to overfit, as training and test loss are similar. Decision trees appear to underfit when given only 100,000 training examples, but once given more data, training and test loss are also very close. In future experiments, I would improve the bag-of-words approach with better preprocessing. With more time and resources, I would also train or fine-tune state-of-the-art NLP models for this specific task.

References

[1] Bhowmick, Alexy and Shyamanta M. Hazarika. "Machine Learning for E-mail Spam Filtering: Review, Techniques and Trends." *arXiv* (2016). https://arxiv.org/pdf/1606.01042.pdf.

[2] Go, Alec, Richa Bhayani, and Lei Huang. "Twitter Sentiment Classification using Distant Supervision." n.d. https://cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf

[3] Chu, Z., Gianvecchio, S., Wang, H., & Jajodia, S. "Detecting Automation of Twitter Accounts: Are You a Human, Bot, or Cyborg?" *IEEE Transactions on Dependable and Secure Computing*, 9(6), 811–824 (2012). https://doi.org/10.1109/tdsc.2012.75.

[4] Chen, C., Zhang, J., Xie, Y., Xiang, Y., Zhou, W., Hassan, Al-Elaiwi, A., Alrubaian, M. "A Performance Evaluation of Machine Learning-Based Streaming Spam Tweets Detection." *IEEE Transactions on Computational Social Systems, 2(3), 65–76* (2015). https://doi.org/10.1109/tcss.2016.2516039.

[5] Dharan, Kannan Neten. "A comparative study of russian trolls using several machine learning models on twitter data learning models on twitter data." Master's thesis, New Jersey Institute of Technology (2019). https://digitalcommons.njit.edu/cgi/viewcontent.cgi?article=2662&context=theses

[6] Linvill, Darren and Patrick Warren, "Troll Factories: The Internet Research Agency and State-Sponsored Agenda Building." n.d. http://pwarren.people.clemson.edu/Linvill_Warren_TrollFactory.pdf

[7] "Over one million tweets collected from US Politicians (President, Congress and Governors)." *Reddit.* https://www.reddit.com/r/datasets/comments/6fniik/over_one_million_tweets_collected_from_us/.

[8] Leskovec, Jure, Anand Rajaraman, and Jeff Ullman. "Chapter 11: Dimensionality Reduction." *Mining of Massive Datasets.* Cambridge University Press (2020), pp. 418–425. http://infolab.stanford.edu/~ullman/mmds/ch11.pdf.

[9] "DistilBERT." *Hugging Face*. https://huggingface.co/transformers/model_doc/distilbert.html.

[10] Rush, Alexander, Vincent Nguyen and Guillaume Klein. "The Annotated Transformer." *Harvard NLP.* http://nlp.seas.harvard.edu/2018/04/03/attention.html

[11] Ng, Andrew and Tengyu Ma. "Logistic Regression." *CS229 Lecture Notes.* http://cs229.stanford.edu/notes2020fall/notes2020fall/cs229-notes1.pdf.

[12] Ng, Andrew. "Support Vector Machines." *CS229 Lecture Notes.* http://cs229.stanford.edu/notes2020fall/notes2020fall/cs229-notes3.pdf.

[13] Leskovec, Jure, Anand Rajaraman, and Jeff Ullman. "Chapter 12: Large-Scale Machine Learning." *Mining of Massive Datasets.* Cambridge University Press (2020), pp. 490. http://infolab.stanford.edu/~ullman/mmds/ch12n.pdf.

[14] Ng, Andrew. "Naïve Bayes." *CS229 Lecture Notes.* http://cs229.stanford.edu/notes2020fall/notes2020fall/cs229-notes2.pdf.

[15] Leskovec, Jure, Anand Rajaraman, and Jeff Ullman. "Chapter 9: Recommendation Systems." *Mining of Massive Datasets.* Cambridge University Press (2020), pp. 330–331. http://infolab.stanford.edu/~ullman/mmds/ch9.pdf.

[16] Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree." Presented at NIPS (2017). https://papers.nips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

Libraries and Packages

Python

- `scikit-learn`
- `pandas`
- `numpy`
- `transformers`
- `matplotlib`

R

- `tidyverse` (used for data preparation)