Milestone Report 2: Classifying Hate Speech on Twitter

Data Processing for Machine Learning

While a tokenized wordspace is convenient for ensuring the language in each tweet is comparable, this form of the data is not yet ready to be used as features for machine learning. Using every word in every tweet as a potential feature creates difficulties, both in terms of computation speed but also in terms of model accuracy. In this project, four approaches will be taken to generate meaningful features from the processed tokens.

A **bag of words vectorizer** selects a fixed number of random feature words specified by the user to act as label predictors. In this project, a bag of 1000 feature words was generated for this purpose. Future explorations of the bag of words approach may include a larger number of feature words to improve accuracy.

A **tf-idf vectorizer** is designed like the bag of words vectorizer, except that it weights feature words that are unique to each data set more heavily in its computations. So for instance, if the following was passed to a tf-idf vectorizer with with feature words “love,” “shoes,” and “protestors”

“I love these shoes from the clearance sale” and “I’d love to stomp those protestors”.

The words “shoes” and “protestors” would be considered more predictive than the word “love.” In this project, 1000 feature words were also chosen to provide the predictive vectors for ML.

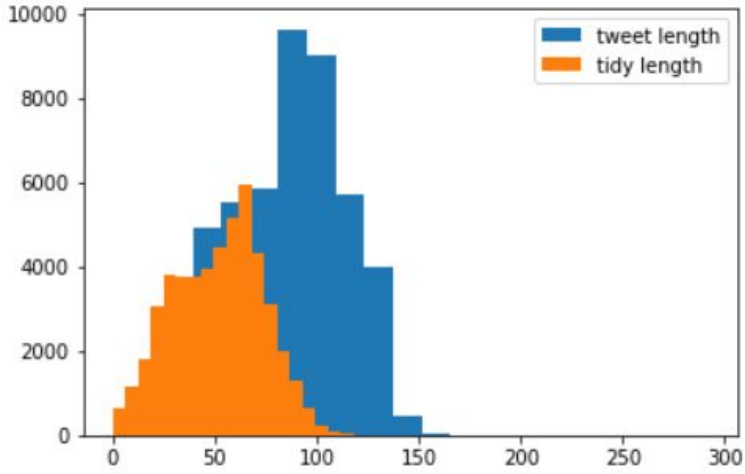
If a tf-idf vectorizer can be said to infer the likelihood of an individual word’s connotation, a **Word to Vector (or Skip-Gram) Embedding** goes a step farther by training a small neural network to infer contexts around groupings of words found in the training set. This method creates some surprisingly clever results, and can even identify words that don’t belong in a certain context (like comparing the words to the context “tool”, seeing “staple, hammer, drill, saw,” and identifying that staple shouldn’t be included). 200 words were selected to form the context features for label association.

Tokens that most often occur with the stated context:

“Trump”	Donald, hillari, phoni, unstabl, unfit, melo, nomine, #delegaterevolt, potu, unfavorable
“breakfast”	#feelingdiet, melani, avocado, #tacotuesday, #deadlift, #pushyourself, cookout, #yogu, cuppa, sushi

Here, it should again be noted that the a different random seed and a larger number of context words might provide improved future improved performance. We can see from the breakfast example that a motivated group of exercise enthusiasts have taught our Word to Vector algorithm that deadlifting and "#pushyourself are important features of breakfast which is only really true for a small subset of breakfast-goers. As such, it may also be helpful to incorporate non-twitter data in future iterations of this project.

A **Document to Vector Embedding** works the same way as a Word to Vector, but it trains its neural nets to guess a context from an entire tweet, and not just the most closely correlated contexts for each word. While Document to Vector Embedding may sound like a more comprehensive version of Word to Vector, it runs the risk of being less effective when the documents themselves are fairly small. When analyzed, just over half of the tidied tweets contained 55 characters or less!



After completing this feature engineering, the bag of words, tf-idf, word2vec, and doc2vec results were exported to .npz and .csv files to be called in the ML notebook.

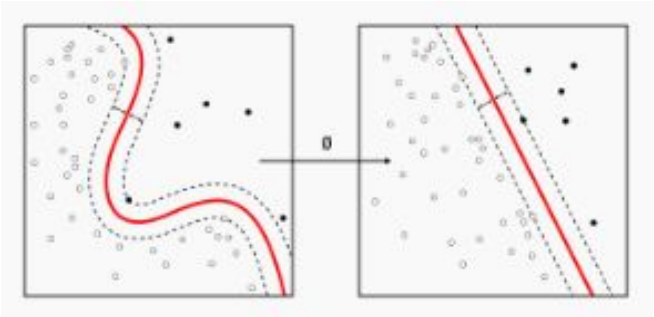
Machine Learning Component

In this Data Science experiment, all four feature sets were used to train and test four different machine learning algorithms: Logistic Regression, Support Vector Machine, Random Forest, and Microsoft's LightGBM (an up-and-coming gradient boosting machine).

Logistic Regression utilizes a classic statistical method of the same name for evaluating the likelihood data belongs to one of two categories. Under the hood, Logistic Regression estimates the log odds of an event by using a multiple linear regression function. While using LogReg as a first step is very common in data science experiments, it is not expected to produce the best results in this experiment because of some of the base assumptions underlying the Logistic Regression method. Specifically,

1) Observations are required to be independent of one another. The observations should not come from repeated measurements or matched data. In this example, the raw data did not allow for ways to determine if a tweet had been retweeted.

2) There should be little or no multicollinearity between independent variables. This means that independent variables should not be closely correlated. From the context of our example, if the features like “hillari” and “bengazi” were selected, they might have a stronger correlation than is desirable.



A **Support Vector Machine** approached classification with tools from a branch of mathematics called linear algebra. If all of the data values can be thought of as points in space, a SVM’s job is to find the best possible boundary to divide those points by category, and with the widest gap between them possible. It often does so by warping the space contained in the points, as seen in the illustration above. Typically, the results aren’t as clean as in the above illustration, and even after the best possible warping and boundary are determined, elements from each data set may still be found on both sides of the boundary. Additionally, the feature sets our SVM will evaluate in this experiment contain 200 to 1000 independent variables, filling points in a space which is hundreds of dimensions! While this is impossible to visualize, the SVM performs this feat computationally, and returns its best guesses given the boundary it has been programmed to find.

A **Random Forest Classifier** is made by combining the results of several “decision trees.” The way a decision tree works is very much like a game of twenty questions. You examine the data and get positive or negative feedback based on the questions asked, and at the end of the game you make a guess. In this experiment, the decision tree has to guess “hate speech or not?” after examining the features provided with a set number of questions. A **random forest** combines the guesses of dozens, hundreds, or thousands of decision trees and summarizes the majority opinion.

A **Gradient Boosting Algorithm** is an ensemble method, similar to random forest. What makes gradient boosting algorithms interesting is that they are tuned by cross-validation. That is, portions of the data are held out during training, and evaluated by the forest trained on the remaining data. Those trees which did the best job of accurately predicting the data in the hold-out set receive a heavier weight. This is done multiple times, with different hold-out data sets. **LightGBM** is a fairly new gradient boosting machine that develops the individual trees in a novel way which makes the whole algorithm faster to train and implement.

Results:

The F1 score was chosen for the evaluation criterion. An F1 score is a calculation that represents the model accuracy by examining the proportion of false positives and false negatives in the prediction of the test data. An F1 score of 1 means every case was correctly classified, where an F1 score of 0 means none were correctly classified. The models with default and tuned parameters performed as follows.

Classifier w/Feature Set	F1 Score
LogReg with Bag of Words	.53
LogReg with TFIDF	.544
LogReg with Word to Vector	.529
LogReg with Doc to Vector	.129
SVM with Bag of Words	.51
SVM with TFIDF	.511
SVM with Word to Vector	.571
SVM with Doc to Vector	.22
Random Forest with Bag of Words	.553
Random Forest with TFIDF	.562
Random Forest with Word to Vector	.492
Random Forest with Doc to Vector	.053
Light GBM with Bag of Words	.504
Light GBM with TFIDF	.521
Light GBM with Word to Vector	.603
Light GBM with Doc to Vector	.259