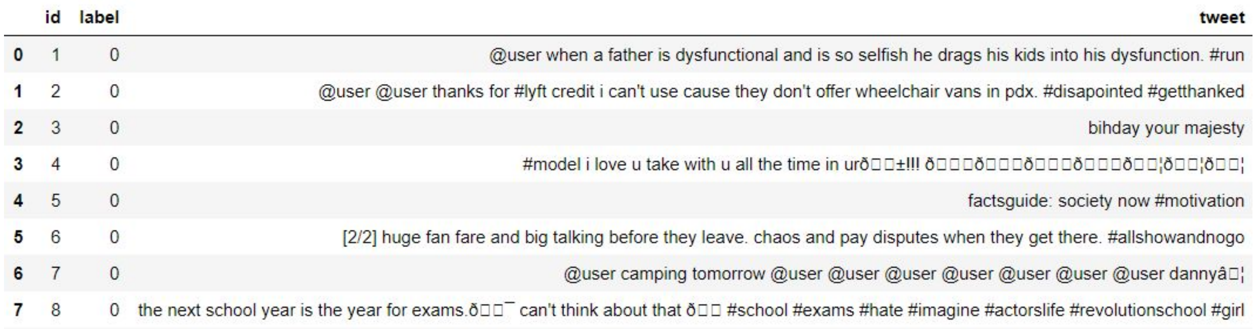
<u>Classifying Hate Speech on Twitter: Final Report</u>

Steve Donahue

Hate speech, trolling, and bullying on social media have become a serious problem in the last few years. From extra-national actors carrying out cyber campaigns exacerbate political differences to teenagers viciously abusing their peers, the internet has become a hostile place. The goal of this project is to develop a model which classifies racist and sexist language, using a previously labeled data set provided in a Vidhya Analytics challenge located here.

The data provided consists of unique index values, binary labels for hate speech vs normal speech, and the text of each tweet. All tweets have been represented as originally posted, except that the usernames have been removed to keep the posters anonymous. The raw data consists of about 32,000 training tweets and another 17,000 which serve as a validation set. The format for the raw data appears as follows.



	id	label	tweet
0	1	0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run
1	2	0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in urð±!!! ðððð¦ð¦ð¦
4	5	0	factsguide: society now #motivation
5	6	0	[2/2] huge fan fare and big talking before they leave. chaos and pay disputes when they get there. #allshowandnogo
6	7	0	@user camping tomorrow @user @user @user @user @user @user @user dannyâ¦
7	8	0	the next school year is the year for exams.ð¯ can't think about that ð #school #exams #hate #imagine #actorslife #revolutionschool #girl

This project will clean, tokenize, and normalize the tweets using standard string processing methods and NLP procedures from the nltk module. The resulting processed tweets will be engineered into various features using the bag of words, ti-idf, word to vector, and document to vector methods. Then, SVM, Logistic Regression, Random Forest, and Light GBM classifiers will be trained on the data set and evaluated using the F1 metric. The top performing classifiers will then be further tuned for optimal classification.

The ultimate goal of this exercise is to produce a classifier which accurately determined which social media posts are racist or sexist. With a highly accurate model, it may be integrated into social platforms as an optional filter for a user's feed or a way to voluntarily prohibit inflammatory posts from being left on an individual user's wall.

It is the goal of this project to allow social media users the ability to customize their experience by electing not to entertain hate speech where they share their personal stories, images, and experiences.

Data Processing for EDA:

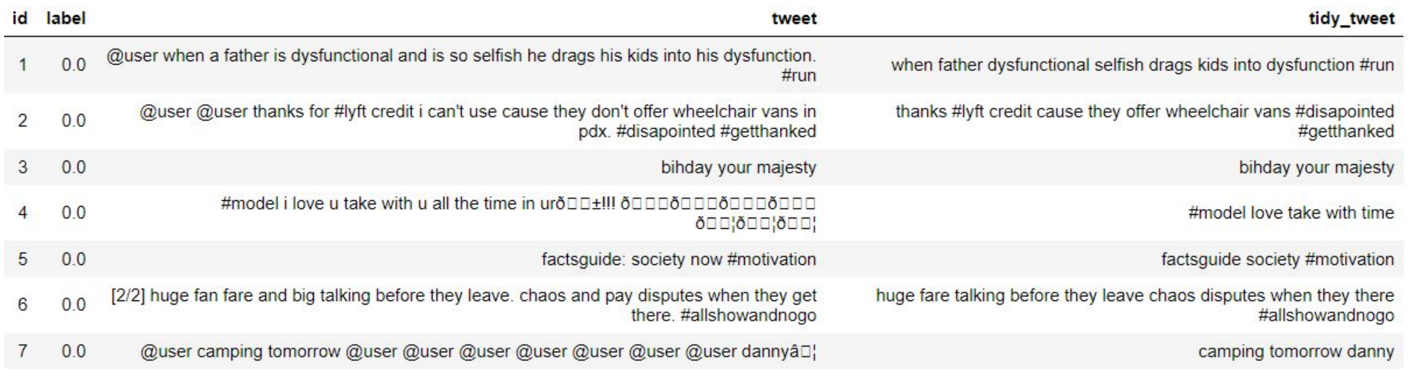
Several operations are applied to the raw tweets to turn them into usable, normalized, “tidy” tokens. First, we’ll define a function to edit the strings directly, based on boolean logic matching patterns we can specify.

```
def remove_pattern(input_txt, pattern):
   r = re.findall(pattern, input_txt)
   for i in r:
      input_txt = re.sub(i, '', input_txt)
   return input_txt
```

This function is then applied to a combined df of both test and train data to remove the @username strings, special characters, and all words of length 3 or less, using the following calls respectively.

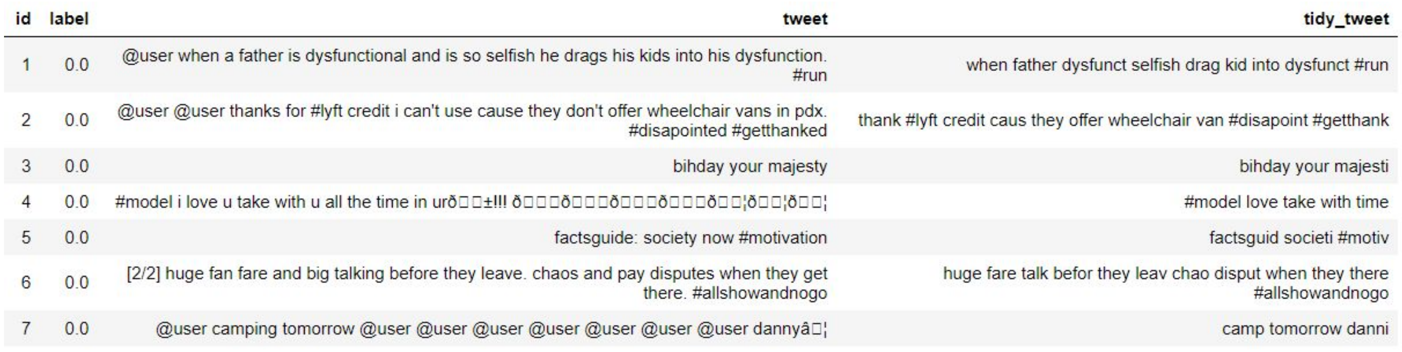
```
combined['tidy_tweet'] = np.vectorize(remove_pattern)(combined['tweet'], "@[\w]*")

combined['tidy_tweet'] = combined['tidy_tweet'].str.replace("[^a-zA-Z#]", " ")

combined['tidy_tweet'] = combined['tidy_tweet'].apply(lambda x: ' '.join([w for w in  x.split() if len(w)>3]))
```

We obtain the following “tidy” dataset:



id	label	tweet	tidy_tweet
1	0.0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run	when father dysfunctional selfish drags kids into dysfunction #run
2	0.0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked	thanks #lyft credit cause they offer wheelchair vans #disapointed #getthanked
3	0.0	bihday your majesty	bihday your majesty
4	0.0	#model i love u take with u all the time in urð±!!! ððððððð¦ð¦ð¦	#model love take with time
5	0.0	factsguide: society now #motivation	factsguide society #motivation
6	0.0	[2/2] huge fan fare and big talking before they leave. chaos and pay disputes when they get there. #allshowandnogo	huge fare talking before they leave chaos disputes when they there #allshowandnogo
7	0.0	@user camping tomorrow @user @user @user @user @user @user @user dannyâ¦	camping tomorrow danny

Next, we normalize and tokenize the words in each tweet. This means transforming various grammatical forms of a word to a single version. For instance, in tweet number 6 above, the word “talking” is transformed to “talk.” “This,” “those,” “these”, etc are transformed to “thi” which represents the normalized form for those words. These normalized values can now be counted and may contribute to feature generation for ML.



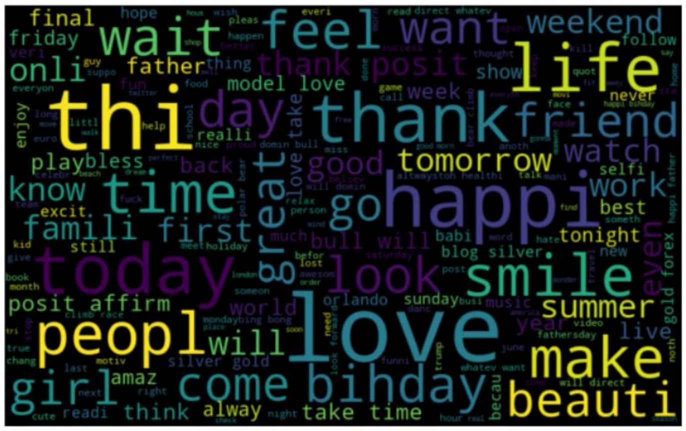
id	label	tweet	tidy_tweet
1	0.0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run	when father dysfunct selfish drag kid into dysfunct #run
2	0.0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked	thank #lyft credit caus they offer wheelchair van #disapoint #getthank
3	0.0	bihday your majesty	bihday your majesti
4	0.0	#model i love u take with u all the time in urð□□±!!! ð□□□ð□□□ð□□□ð□□□ð□□¦ð□□¦ð□□¦	#model love take with time
5	0.0	factsguide: society now #motivation	factsguid societi #motiv
6	0.0	[2/2] huge fan fare and big talking before they leave. chaos and pay disputes when they get there. #allshowandnogo	huge fare talk befor they leav chao disput when they there #allshowandnogo
7	0.0	@user camping tomorrow @user @user @user @user @user @user @user dannyâ□¦	camp tomorrow danni

The tidy data is then exported to a .csv file “processed_tweets” to be called by subsequent notebooks.

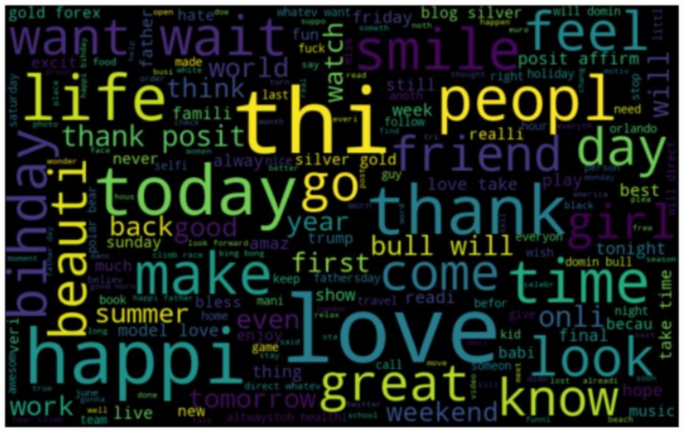
EDA and Statistics

A Wordcloud was chosen to illustrate the differences between normal tweets and those flagged as containing racist or sexist sentiments. A Wordcloud will show the word tokens most frequently occuring in the data fed to it.

The Wordcloud for the whole dataset:

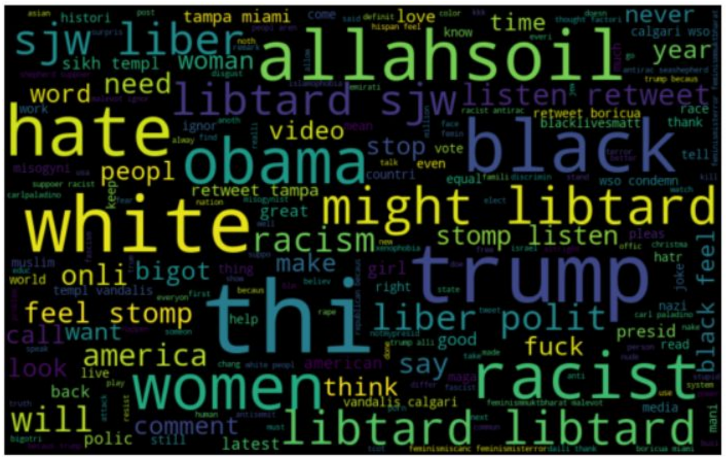


The Wordcloud for normal tweets:



There are few differences, due to the relatively small amount of hate speech overall.

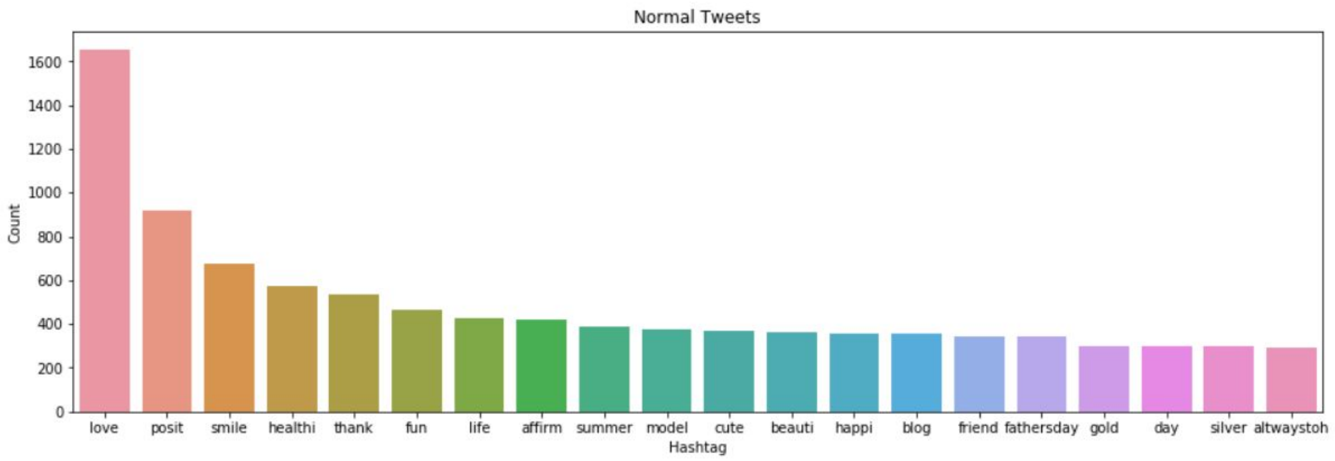
The Wordcloud for hate speech takes on a much more political context.



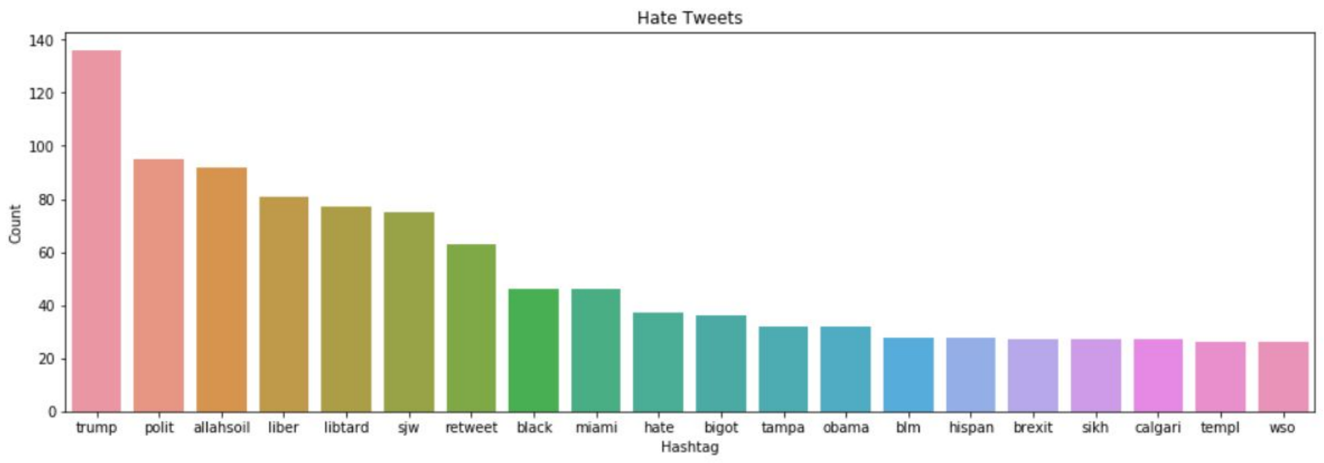
We can also develop some understanding of the difference between normal and abnormal tweets by examining the hashtag content across each. To do so, we define a function to extract the hashtags from each group and create a hashtag array.

```
def hashtag_extract(x):
    hashtags = []

    # Loop over the words in the tweet
    for i in x:
        ht = re.findall(r"#(\w+)", i)
        hashtags.append(ht)
    return hashtags
```

Applied to each group and ranked, we can obtain bar graphs of the most frequently occurring hashtags in each sentiment group. Normal tweets are celebratory in nature.



Tweets in the hate speech group are predominantly political.



Observations:

There are significant differences between normal online speech and hate speech, measurable by the words appearing most frequently in each. Even a casual observer can see the difference clearly from the Wordclouds and predominant hashtags. While this difference might be a starting point to develop a filter, we cannot use keywords and hashtags alone. Censoring all tweets about Trump, Hillary, Miami, politics, and the color black would be too crude a tool and would stifle honest conversations. However, because a noticeable difference exists, it should be possible to develop classification tools via machine learning.

Part 2: Applying ML

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.

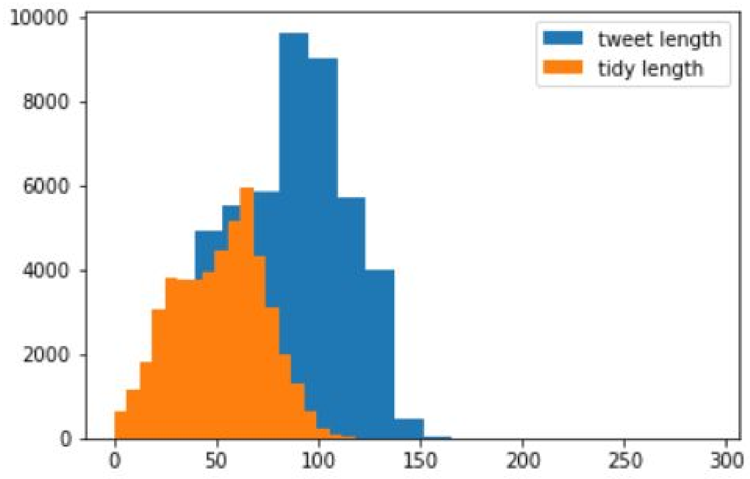
If a tf-idf vectorizer can be said to infer the likelihood of an individual word’s connotation, a **Word to Vector (or Skip-Gram) Embedding** goes a step farther by training a small neural network to infer contexts around groupings of words found in the training set. This method creates some surprisingly 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.

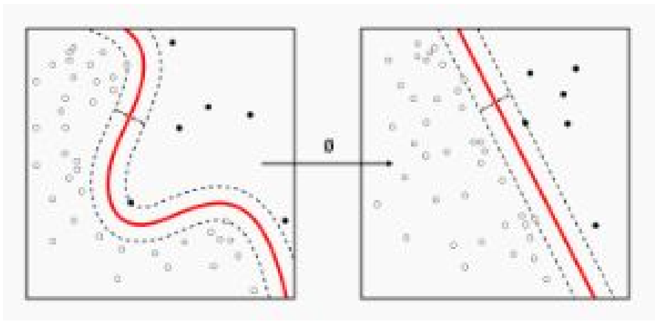
Machine Learning Component

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.

Preliminary 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

Developing a Stacked Classifier

Stacking is a natural extension of machine learning after multiple models have been trained. A stacked model uses the prediction results from trained models, along with the training data set, as features in a new model. The benefit of this approach is that where one model misclassifies a data point, the other model might have the correct classification. By training a model with the original features and those predictions, the stacked model can assign higher weight to those models which classify a data point correctly, given the features contained in that data point.

The Light GBM model (with the Word to Vector features) showed the most potential, so all of the classifiers which the word to vector features were tuned. The following F1 score improvements were made (Random Forest did not respond well to tuning, and SVM was too time intensive to be useful).

LogReg with Word to Vector: + .004	New F1 Score:	.533
Light GBM with Word to Vector: + .035	New F1 Score:	.639

A new dataframe was constructed to hold the original 200 feature words, and each data point was assigned to one of five random folds. To generate the model predictions, the following had to occur:

1. A fold was selected
2. The remaining folds were used to train the tuned model
3. The trained model made predictions for the fold from step 1.
4. A new fold is selected and the process repeats.

Finally, all the folds with their two new features are concatenated into a new training dataframe, which can then be fit to a new Light GBM model, which can then be tuned.

The effects of stacking on this project were dramatic, creating a classifier with a tuned F1 score of .956. To be more precise, of the 9,589 tweets in the holdout set, there were 0 false negatives and only 57 false positives!

Next Steps:

While the classifier is a little conservative to be used as a filter on social media, the lack of false negatives suggest it will be effective. Additional stacking may be possible using other classifiers, or by incorporating the Random Forest TF-IDF or SVM results. Automation and implementation are also areas for further development.