**Overview**

This project aims to **automate content moderation** to identify hate speech using **machine learning binary classification algorithms.** Baseline models included **Logistic Regression** model that used TFIDF Vectorization for feature engineering. It produced an F1 of 0.33481 and Recall (TPR) of 0.4073. This performance can be attributed to the massive class imbalance and the model's inability to "understand" the nuances of English slang and slurs. Ultimately, automating hate speech detection is an extremely difficult task. And although this project was able to get that process started, there is more work to be done in order to keep this content off of public-facing forums such as Twitter.

**SOURCE CODE LINK -**

**Business Problem**

Human content moderation exploits people by consistently traumatizing and underpaying them. In 2019, an [article](https://www.theverge.com/2019/6/19/18681845/facebook-moderator-interviews-video-trauma-ptsd-cognizant-tampa) on The Verge exposed the extensive list of horrific working conditions that employees faced at Cognizant, Facebook’s former moderation contractor. Unfortunately, every major tech company, including Twitter, uses human moderators to some extent, both domestically and overseas.

Hate speech is defined as abusive or threatening speech that expresses prejudice against a particular group, especially on the basis of race, religion or sexual orientation. Usually, the difference between hate speech and offensive language comes down to subtle context or diction.

Any company with an online forum where users post content could benefit from automating as much as the moderation process as possible. Ultimately, human content moderation is not only detrimental to workers, but also presents a liability to companies that use them.

# Data & Methods

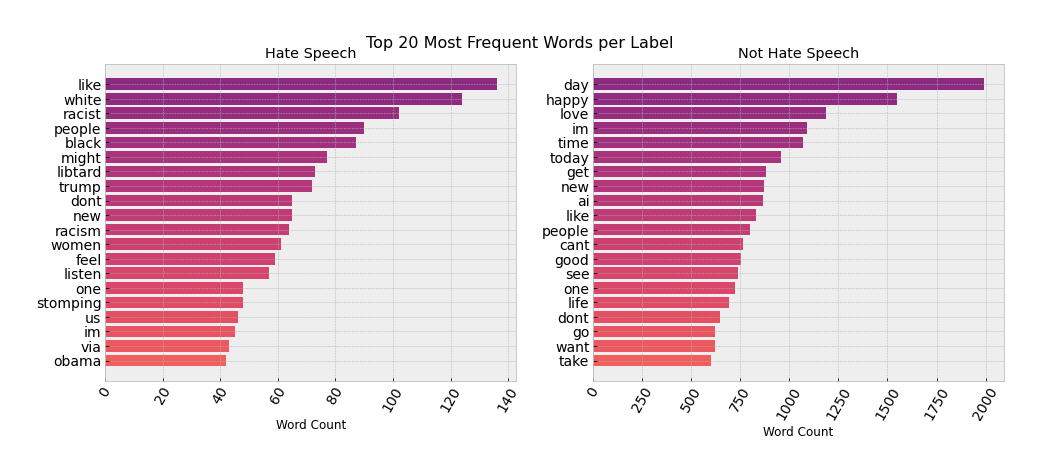
The dataset for this capstone project was sourced from a study called Automated Hate Speech Detection and the Problem of Offensive Language conducted by Thomas Davidson and a team at Cornell University in 2017. The dataset is provided as a .csv file with 24,802 text posts from Twitter where **6% of the tweets were labelled as hate speech**.

Since content moderation is so subjective, the labels on this dataset were voted on by crowdsource and determined by majority-rules. The “class” column labels each tweet as 0 for hate speech, 1 for offensive language or 2 for neither. In order to create a different project and adapt the data to my specific business context, I will be treating the data as a binary classification problem.

Therefore, the final model will be **predicting whether a tweet is hate speech or not.** To prepare the data for this, I will be manually replacing existing 1 and 2 values as 0, and replacing 0 as 1 to indicate hate speech.

# Data Understanding

### 1. What are the linguistic differences between hate speech and offensive language?



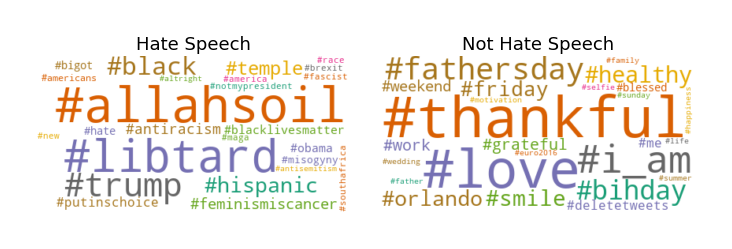
The code for this graph can be found in [EDA.ipynb](https://github.com/sidneykung/twitter_hate_speech_detection/blob/master/preprocessing/eda_notebook.ipynb).

Linguistically, it's important to note that the difference between hate speech and offensive language often comes down to how it targets marginalized communities, often in threatening ways.

Although these graphs have very similar frequently occurring words, there are a few that stand out. For instance, we can notice from this figure that Hate Speech typically contains the N-word with the hard 'R'. **The use of this slur could indicate malicious intent, which goes beyond possibly using the word as slang.**

Examples like that one demonstrate the nuances of English slang and the fine line between Hate Speech and offensive language. **Because of the similarities of each label’s vocabulary, it could be difficult for machine learning algorithms to differentiate between them and determine what counts as Hate Speech.**

### 2. What are the most popular hashtags of each tweet type?



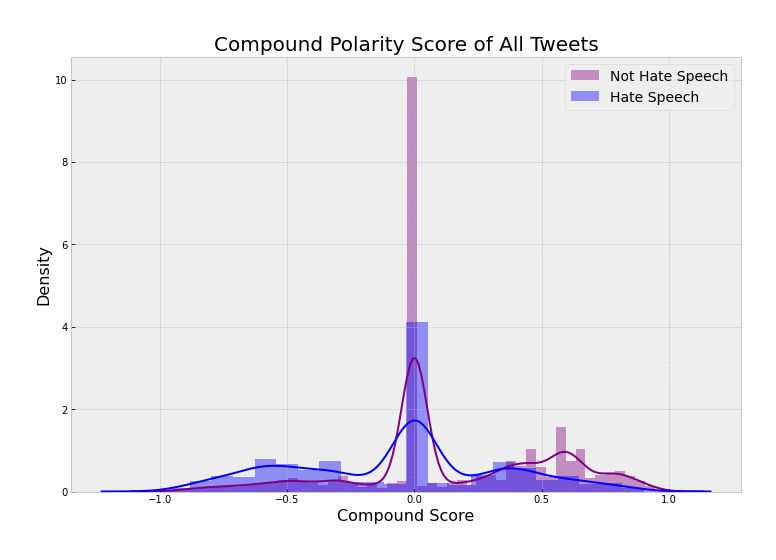
he code for this graph can be found in [EDA.ipynb](https://github.com/sidneykung/twitter_hate_speech_detection/blob/master/preprocessing/censored_hashtags.ipynb).

From these word clouds, we can see some more parallels and differences between what is classified as hate speech or not. For instance, #tcot stands for "Top Conservatives On Twitter” and it appears in both groups. However, #teabagger, which refers to those who identify with the Tea Party, that is primarily (but not exclusively) associated with the Republican Party, appears in only the “Not Hate Speech” cloud. Both hashtags are used among Alt-Right communities.

Additionally, the #r\*\*skins hashtag appears in only the "Not Hate Speech" cloud. This was the former name of the Washington NFL team. Knowing the context, we know that hashtag could certainly include text that constitutes as hate speech. With this, and other hashtags that appear in the “Not Hate Speech” cloud, we can clearly see the very slight differences between the two labels.

Besides that, others are simply pop culture references, such as #Scandal the TV show or #vote5sos referring to the boy band. It’s interesting that those contain a lot of offensive language, probably from fan reactions and community conflicts. Ultimately, we can recommend that **Twitter should closely monitor those top hashtags for potential posts containing hate speech** or even regular offensive language.

### 3. What is the overall polarity of the tweets?



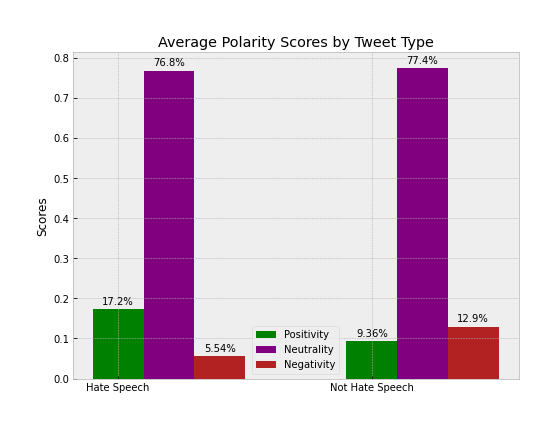
The code for this graph can be found in [EDA.ipynb](https://github.com/sidneykung/twitter_hate_speech_detection/blob/master/preprocessing/VADER_sentiment.ipynb)

The compound polarity score is a metric that calculates the sum of all the [lexicon ratings](https://github.com/cjhutto/vaderSentiment/blob/master/vaderSentiment/vader_lexicon.txt) which have been normalized between -1 and +1. With -1 being extreme negative and +1 being extreme positive. **This score encompasses the overall sentiment of this corpus.**

* Hate Speech tweets on average have a compound score of -0.363
* Non Hate Speech tweets on average have a compound score of -0.263

According to this metric, both classes of tweets have pretty negative sentiments because their normalized compound scores are less than the threshold of -0.05.

Additionally from the graph above, we can see that tweets classified as Hate Speech are especially negative. This further emphasizes how slim the difference between the two labels are. Although both classes contain negative and offensive language, Hate Speech is much more negative on average.

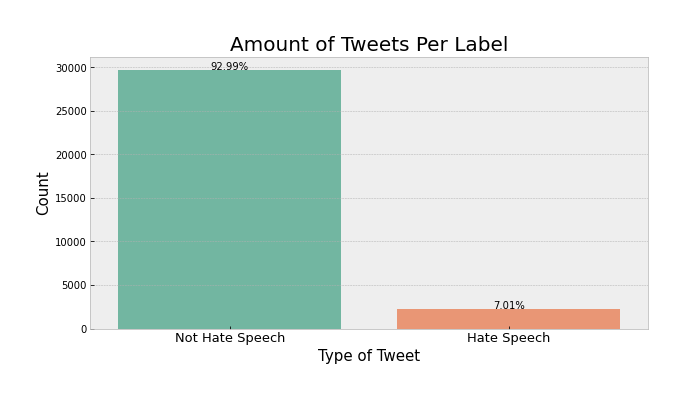


The code for this graph can be found in EDA[.ipynb](https://github.com/sidneykung/twitter_hate_speech_detection/blob/master/preprocessing/VADER_sentiment.ipynb).

To reiterate, this graph shows the average polarity scores for each label. Which are positive, neutral or negative. We can see that a majority were scored as neutral. However, of those that were scored as negative, it seems like "Not Hate Speech" had more on average. This is probably because of the class imbalance.

## Class Imbalance

The main roadblock of this dataset is the extreme class imbalance. We can see that only 5.77% of the data is labeled as hate speech. This could present challenges during the modeling process.



The code for this graph can be found in [EDA.ipynb](https://github.com/sidneykung/twitter_hate_speech_detection/blob/master/preprocessing/data_cleaning.ipynb).

STEPS:

**Pre-processing Text Data**

The original data from twitter\_data.csv was cleaned using RegEx in the NLP\_PROJECT.ipynb notebook.

**Cleaning Steps:**

* Reassigning labels
* Lowercasing tweet text
* Removing hashtags, mentions, quotes and punctuation from tweet text
* Checking for missing values

## Tokenizing & Removing Stop Words

When working with text data, one of the first steps is to remove stop words from the corpus. Although text would be grammatically incorrect without these stop words, they provide little value to models and typically hinder performance.

We can use NLTK's built-in library of stop words to remove them in a tokenizing function.

Top Words

Lemmatization

# Feature Engineering

With Natural Language Processing, the purpose of feature engineering is to transform the tokenized text data into numerical vectors that the machine learning algorithm can "understand."

In this notebook, we'll be iterating through three different feature engineering techniques: **Count Vectorization, TF-IDF Vectorization and Doc2Vec**. Trying out these techniques could yield vastly different metrics on the same four baseline models.

## Train-Test Split

First, let's perform a train-test split of the dataset, where 20% is reserved as unseen testing data.

## TF-IDF Vectorization

First, we’ll be trying one of the most popular methods, TF-IDF Vectorization.

This is an acronym that stands for “Term Frequency — Inverse Document” Frequency which are the components of the resulting scores assigned to each word.

* Term Frequency: This summarizes how often a given word appears within a document.
* Inverse Document Frequency: This down scales words that appear a lot across documents.

Without going into the math, TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across other documents.

## Baseline Logistic Regression

Logistic Regression is another common model used for classification tasks. Additionally, this model tends to work better with larger datasets.

# Dealing with Class Imbalance

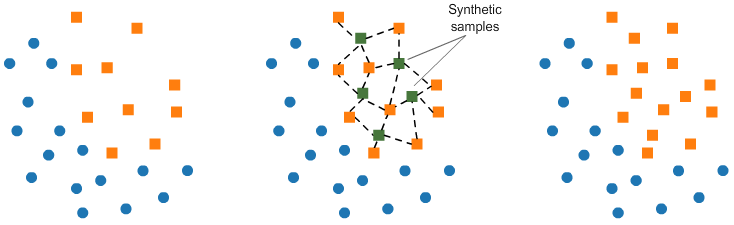
The Logistic Regression model dealt with class imbalance with the parameter class\_weight='balanced'. Let's try other class imbalance remedy methods to improve the baseline, before tuning hyperparameters with grid search.

## Over-Sampling with SMOTE

This method over-samples the minority class, "Hate Speech".

Rather than simply oversampling the minority class with replacement (which adds duplicate cases to the dataset), the algorithm generates new sample data by creating ‘synthetic’ examples that are combinations of the closest minority class cases.

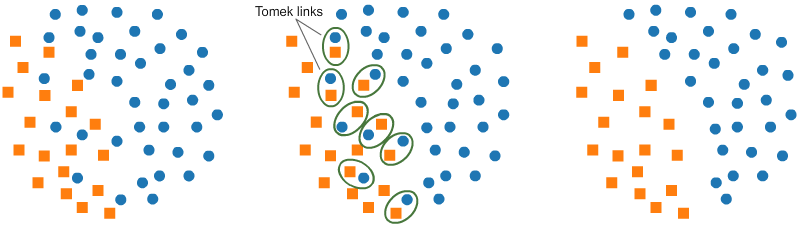
After synthetically resampling our data, we no longer need to lean on penalized class weights in order to improve our model tuning.



## Under-Sampling with Tomek Links

This method under-samples the majority class, "Not Hate Speech."

Tomek links are pairs of very close instances, but of opposite classes. Removing the instances of the majority class of each pair increases the space between the two classes, facilitating the classification process.



**Grid Search**

The sklearn library provides an easy way to tune model parameters through an exhaustive search using GridSearchCV. It combines K-Fold Cross Validation with a grid search of hyperparameters.

A full list of a Logistic Regression model's hyperparameters can be found in the documentation [here](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html).

With Logistic Regression models, however, there are not many hyperparameters to tune. A grid search is implemented to optimize the following hyperparameters:

* Penalty: Used to specify the norm used in the penalization. The default is L2.
* Solver: Algorithm to use in the optimization problem.
* C: Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

Ultimately, the grid search tuned the hyperparameters as follows:

* {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}

However, this model generated the exact same evaluation metrics as the initial Logistic Regression model that used CountVectorizer. Therefore, I decided not to include that grid search code or model in this notebook. As it didn't yield helpful results. Again, this could be attributed to the fact that there just aren't many hyperparameters to tune for this kind of model, so things usually don’t change much.