# Hate Speech Recognizer

Yashita Vajpayee
*Masters in Data science*
*Stevens Institute of Technology*
Hoboken, USA
yvajpaye@stevens.edu

Sharven Subhash Rane
*Masters in Data science*
*Stevens Institute of Technology*
Hoboken, USA
srane8@stevens.edu

Kalyan Varma Patchamatla
*Masters in Data science*
*Stevens Institute of Technology*
Hoboken, USA
kpatcham@stevens.edu

***Abstract*—The aim of this project is to detect hate speech from twitter data using logistic regression, Random forest and Naive Bayes methods and report a comparison between results. In this mid-term report we have done pre-processing of the data and implemented Logistic regression**

## I. Introduction

In recent years, the advent of social media platforms has led users to freely express their opinions on various subjects, including politics, society, health, education, finance, and even business-related issues. However, this widespread usage of social media has also increased the risk of its misuse by some groups resulting in spreading hate speeches or offensive language. This new problem of hate speech on social media has been addressed by recent studies that utilized a number of feature engineering techniques and machine learning algorithms.

## II. Our Solution

In this project, we are going to do sentiment analysis and hate speech recognition. The data from Twitter, a popular microblogging social media platform for sharing short digital content, was used for this project. Preprocessing of data involves basic regex operations, data cleaning and lemmatization. Three machine learning methods will be investigated: logistic regression (LR), random forest (RF), naive Bayes (NB). We use hyperparameter tuning for optimum learning rate and increase accuracy of the model

## III. Description of Dataset

Dataset comprises of three columns : id, label, tweet. Id is the id assigned to a particular tweet and label. Given a training sample of tweets and labels, where label '1' denotes the tweet is racist/sexist and label '0' denotes the tweet is not racist/sexist. We say a tweet contains hate speech if it has a racist or sexist sentiment associated with it. Full tweet texts are provided with their labels for training data. Mentioned users' username is replaced with @user.

Data pre-processing is done as following: Firstly we converted upper case to lower case. Following this we removed URL and hashtags and punctuations. We also removed all the stop words from english language stop words. Then we proceeded to remove any duplicate data by dropping one of them. Finally we perform lemmatization on the processed data.

```
In [18]: #creating a function to process the data
def data_processing(tweet):
    tweet = tweet.lower()
    tweet = re.sub(r"https\S+|www\S+http\S+", '', tweet, flags = re.MULTILINE)
    tweet = re.sub(r'\@w+|\#','', tweet)
    tweet = re.sub(r'[^\w\s]','',tweet)
    tweet = re.sub(r'ð','',tweet)
    tweet_tokens = word_tokenize(tweet)
    filtered_tweets = [w for w in tweet_tokens if not w in stop_words]
    return " ".join(filtered_tweets)

In [19]: tweet_df.tweet = tweet_df['tweet'].apply(data_processing)

In [20]: tweet_df = tweet_df.drop_duplicates('tweet')

In [21]: lemmatizer = WordNetLemmatizer()
def lemmatizing(data):
    tweet = [lemmarizer.lemmatize(word) for word in data]
    return data

In [22]: tweet_df['tweet'] = tweet_df['tweet'].apply(lambda x: lemmatizing(x))
```

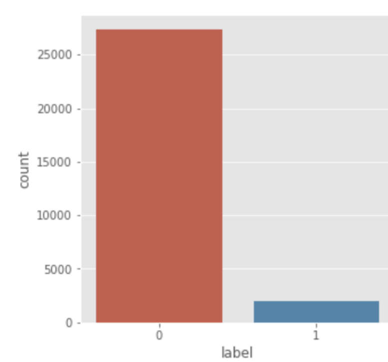
## IV. Data Visualization

The processed data after lemmatization is visualized using count plot and pie-chart as following:

### A. *Count plot:*

Tells the count of both the labels in the form of bar graph

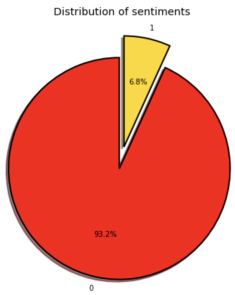
```
In [26]: fig = plt.figure(figsize=(5,5))
sns.countplot(x='label', data = tweet_df)
Out[26]: <AxesSubplot:xlabel='label', ylabel='count'>
```




### B. *Pie-chart:*

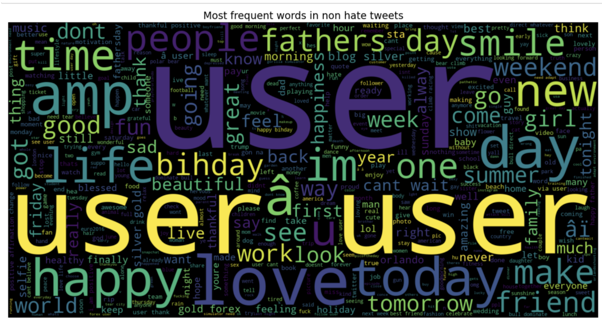
Pie-chart showing both the labels. It can be seen that label 1 i.e. hate speech is 6.8 per cent and non-hate speech is 93.2 per cent.

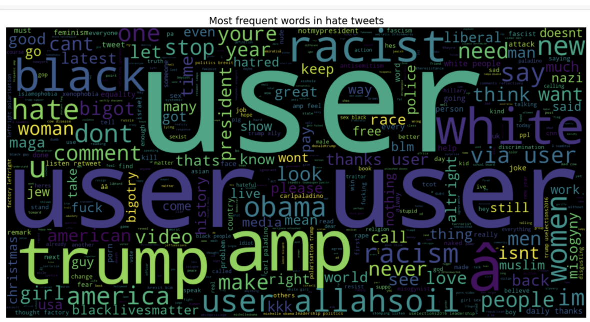
```
In [27]: fig = plt.figure(figsize=(7,7))
colors = ("red", "gold")
wp = {'linewidth':2, 'edgecolor':"black"}
tags = tweet_df['label'].value_counts()
explode = (0.1, 0.1)
tags.plot(kind='pie',autopct = '%1.1f%%', shadow=True, colors = colors, startangle =90,
        wedgeprops = wp, explode = explode, label='')
plt.title('Distribution of sentiments')

Out[27]: Text(0.5, 1.0, 'Distribution of sentiments')
```




### C. *Word cloud:*

Shows the most used words in non-hate tweets and hate tweets respectively




## V. Machine Learning Algorithms

Since our focus is on separating hate tweets and non-hate tweets, going with categorical classification is the best idea. Algorithms like Logistic regression, Naive-Bayes classifier and Random forest can work for such classification.

### A. *Logistic Regression :*

Logistic Regression comes under Supervised Learning. Classification is about predicting a label, by identifying which category an object belongs to base on different parameters. Logistic Regression is a statistical approach and a Machine Learning algorithm that is used for classification problems and is based on the concept of probability. It is used when the dependent variable (target) is categorical. It is widely used when the classification problem at hand is binary; true or false, yes, or no, etc. For example, it can be used to predict whether an email is spam (1) or not (0). Logistics regression uses the sigmoid function to return the probability of a label. In logistic regression models, we extract four types of features, word-level, and character-level n-gram features as well as two types of lexicon derived features. We extract these four types of features from the target comment first.

## VI. Implementation Details

### A. *Building model*

After pre-processing of the data and visualizing it, we vectorized the data using tf-idf vectorizer. The bi-gram and tri-gram language model was created and the features were extracted.

```
]: vect = TfidfVectorizer(ngram_range=(1,2)).fit(tweet_df['tweet'])

]: feature_names = vect.get_feature_names()
print("Number of features: {}\n".format(len(feature_names)))
print("First 20 features: \n{}".format(feature_names[:20]))

Number of features: 208579

First 20 features:
['0000001', '0000001 polluting', '00027', '00027 photooftheday', '001', '0035', '00h30', '01', '01 4995', '01 7900'
, '01 blog', '01 croatia', '01 may', '01 shopalyssas', '0115', '0115 8599968', '0161', '0161 manny', '019', '019 pr
evious']

]: vect = TfidfVectorizer(ngram_range=(1,3)).fit(tweet_df['tweet'])

]: feature_names = vect.get_feature_names()
print("Number of features: {}\n".format(len(feature_names)))
print("First 20 features: \n{}".format(feature_names[:20]))

Number of features: 380305

First 20 features:
['0000001', '0000001 polluting', '0000001 polluting niger', '00027', '00027 photooftheday', '00027 photooftheday mu
sic', '001', '0035', '00h30', '01', '01 4995', '01 4995 rustic', '01 7900', '01 7900 shopalyssas', '01 blog', '01 b
log silver', '01 croatia', '01 croatia happy', '01 may', '01 may actual']
```

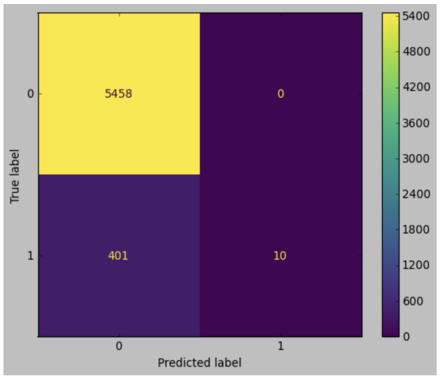
We next proceeded to build the logistic regression model. Firstly we splitted the data into test and training data by test data being 20 per cent and rest to be training. After dataset splitting we fitted the logistic regression model and checked our model accuracy which came out be 93.17 per cent as shown:

```
: X = tweet_df['tweet']
Y = tweet_df['label']
X = vect.transform(X)

: x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

: print("Size of x_train:", (x_train.shape))
print("Size of y_train:", (y_train.shape))
print("Size of x_test: ", (x_test.shape))
print("Size of y_test: ", (y_test.shape))

Size of x_train: (23476, 380305)
Size of y_train: (23476,)
Size of x_test:  (5869, 380305)
Size of y_test:  (5869,)

: logreg = LogisticRegression()
logreg.fit(x_train, y_train)
logreg_predict = logreg.predict(x_test)
logreg_acc = accuracy_score(logreg_predict, y_test)
print("Test accuarcy: {:.2f}%".format(logreg_acc*100))

Test accuarcy: 93.17%
```

We also printed confusion matrix for performance analysis of our model as below:

```
style.use('classic')
cm = confusion_matrix(y_test, logreg_predict, labels=logreg.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=logreg.classes_)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f17be5f2e0>
```




### *B. Hyperparameter Tuning*

Next we performed hyperparameter tuning to improve the performance of our model. We used GridSearchCV for this task and for best cross-validation score and best parameter. Following this we predicted our test values. This can be seen as below:

```
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')
```

```
param_grid = {'C':[100, 10, 1.0, 0.1, 0.01], 'solver' :['newton-cg', 'lbfgs','liblinear']}
grid = GridSearchCV(LogisticRegression(), param_grid, cv = 5)
grid.fit(x_train, y_train)
print("Best Cross validation score: {:.2f}".format(grid.best_score_))
print("Best parameters: ", grid.best_params_)
```

```
Best Cross validation score: 0.95
Best parameters:  {'C': 100, 'solver': 'newton-cg'}
```

```
y_pred = grid.predict(x_test)
```

Lastly we checked the accuracy of the model after hyperparameter tuning and printed the confusion matrix for performance analysis

```
logreg_acc = accuracy_score(y_pred, y_test)
print("Test accuracy: {:.2f}%".format(logreg_acc*100))
```

```
Test accuracy: 94.89%
```

```
print(confusion_matrix(y_test, y_pred))
print("\n")
print(classification_report(y_test, y_pred))
```

```
[[5450    8]
 [ 292  119]]


              precision    recall  f1-score   support

           0       0.95      1.00      0.97      5458
           1       0.94      0.29      0.44       411

    accuracy                           0.95      5869
   macro avg       0.94      0.64      0.71      5869
weighted avg       0.95      0.95      0.94      5869
```

It can be seen that the model accuracy improved to 94.89 per cent.

## VII. Future Directions

Results will be compared with two other machine learning methods: Random Forest (RF) and Naive Bayes (NB), trying to obtain the most accurate classification. In the future, this can be implemented for larger datasets and our final goal is to implement this project in any social media platform and confront the current hate speech issues.

## VIII. Conclusion

The extensive experiments and analysis show logistic regression to be a good algorithm for hate speech detection. More efficient algorithms can be used alongside deep learning and neural networks techniques to further enhance the model. This is to provide important insights into their detection accuracy, computational efficiency, capability in using pre-trained models, and domain generalizability for their deployment in real-world applications.

## References

[1] https://www.kaggle.com/datasets/arkhoshghalb/twitter-sentiment-analysis-hatred-speech?select=train.csv

[2] https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

[3] https://realpython.com/logistic-regression-python/