# Machine Learning Engineer Nanodegree Capstone Project Report

## Depression Sentiment Prediction

Tanay Mehta

October 2019

# 1. Definition

## 1.1 Project Overview

Depression has been known to be a major problem for humans since times not known. Sometimes Physical Problems can cause depression, but most of the times, it was found that Symptoms of Depression are a part of more complex psychiatric problem, often related to mental tension and extreme stress-buildup.

There are many types of depression, but the one that is the heaviest on its victims is, **Major Depression** [1]. Also known as Clinical Depression; an individual diagnosed with Major Depression feels a constant sense of hopelessness and despair.

And this is where, this project jumps in.

About 24% of all Male Internet users and about 21% of all female internet users use Twitter. Nearly 500 Million tweets are sent every day on Twitter [2]. This makes Twitter a major target platform for detecting and analyzing depression among its users.

In layman terms, we have a given sentence, and we give two predictions;

1. Is the tweet of positive sentiment or not? (answer in %)
2. Is the person depressed? (True or False)

Dataset used in this project is freely available on Kaggle known as *“Sentiment140 dataset with 1.6 million tweets”* [3].

## 1.2 Problem Statement

The Final goal of this project is to train a model on the given training data (after splitting the total data into train and test set). Important task-checkpoints that I need to pass to ensure project success:

- **Proper Data Importing and Cleaning:** Since the data contains raw text, it must be imported into proper format with right encoding. The Data also includes a lot of useless columns. Deleting those columns along with renaming the important columns and dropping the nan values will do this task-checkpoint.
- ***Mild* Data Visualization:** Since the important part of data (that will be used for training) only contains 2 significant features, 1. Raw Text and 2. Target Values (0 or 1). So, the overall EDA process is very limited now since, we can’t *humanely* visualize 1.6 Million tweets into a 1.6 Million-Dimensional

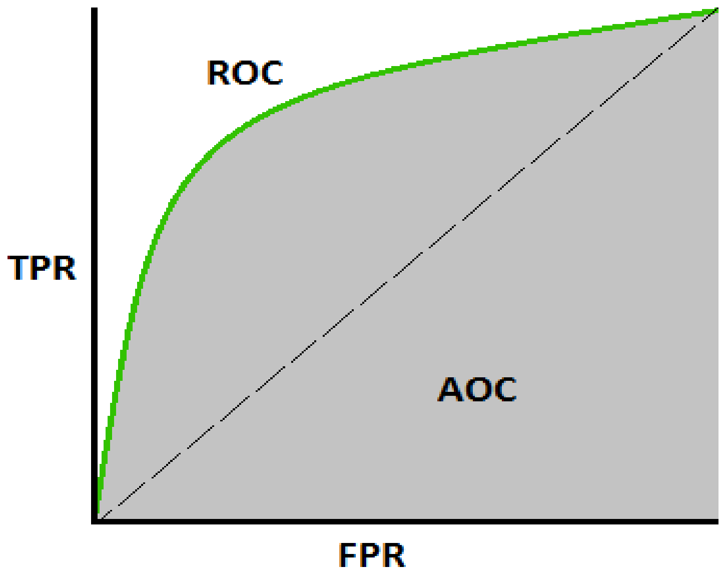
vector using an ordinary machine! And for that reason, I have visualized data using WordCloud only.

- **Data Preprocessing and Feature Extraction:** In this step, I have preprocessed the data using several Natural Language Processing techniques and algorithms along with extracting important features using feature extraction techniques.
- **Model Training, Testing and Hyperparameter Search:** In this stage I have trained 6-different Machine Learning models on my data (as I was only going to do this the "machine learning way" [4]). After training, I have tested each and every model on test data and reported the corresponding scores. At the end, I have searched for Optimal Hyperparamter for the existing models that performed well.
- **Real world-model evaluation:** Finally, at the end, I have evaluated the best performing model on real world sentences and have reported the correct results.

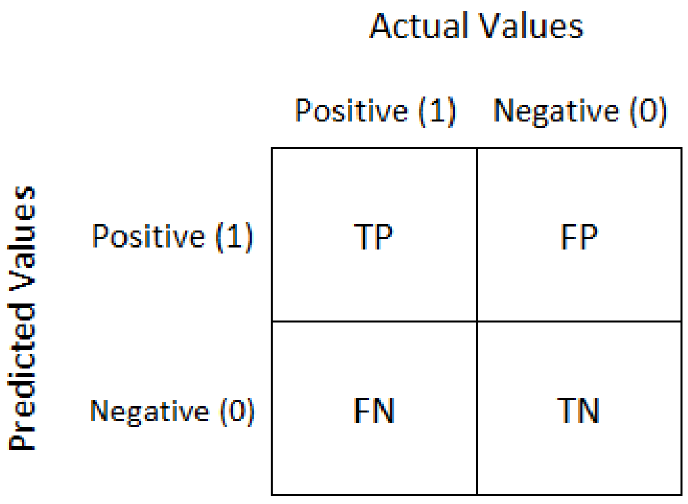
## 1.3 Metrics

I have used two metrics for this project;

I. **ROC-AUC Score:** ROC stands for Receiver Operating Characteristic. It is a graph showing performance of a classification model at all Classification thresholds. The curve plots 2 parameters: **True Positive Rate (TPR)** and **False Positive Rate (FPR)**. AUC stands for Area Under the ROC Curve. It provides and aggregate measure of performance across all possible classification thresholds. Image below shows the Curve precisely [5].



**II. Confusion Metric:** This metric is used to display the summary of prediction results on a classification problem. The confusion matrix shows precisely not only the performance of model, but also number of **True Positives (TP), True Negatives (TN), False Positives (FP)** and **False Negatives (FN).** Below is a typical Confusion Matrix [6].



| | | Actual Values | |
|---|---|---|---|
| | | Positive (1) | Negative (0) |
| Predicted Values | Positive (1) | TP | FP |
| | Negative (0) | FN | TN |

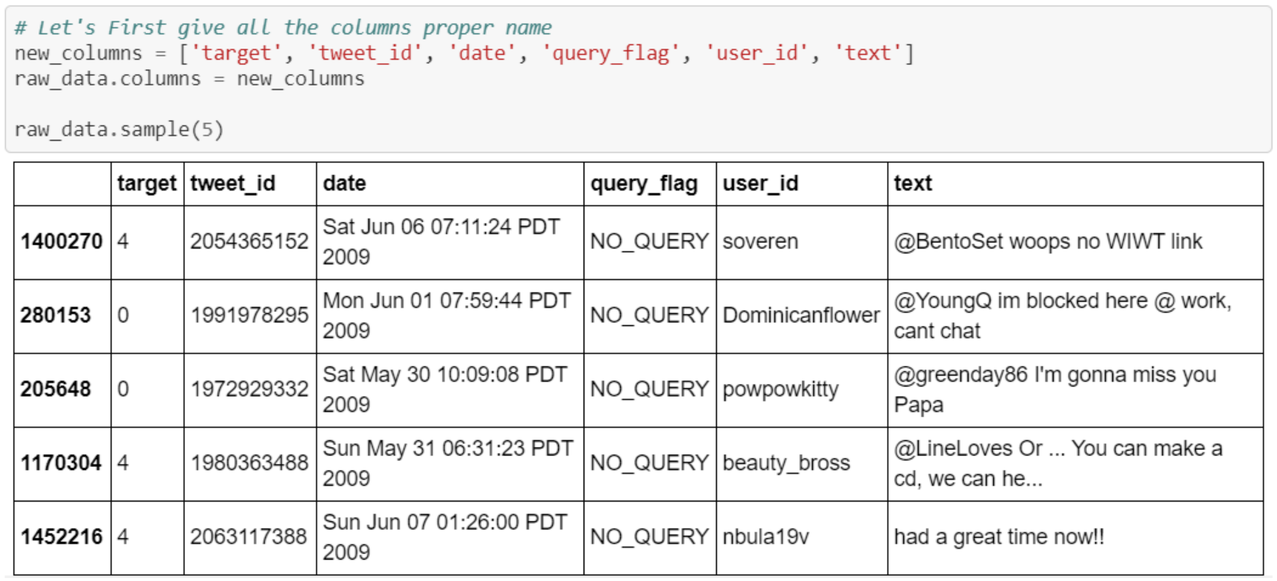
# 2. Analysis

## 2.1 Data Exploration

The raw data consists of 5 columns:

1. **target:** Polarity of the tweet. Either 4 (Positive) or 0 (Negative)
2. **id:** Id of the tweet
3. **date:** Date of the tweet
4. **flag:** If any query was entered when extracting the tweet
5. **user:** Twitter handle Id of the user that tweeted
6. **text:** The main text of the tweet

Obviously, the raw data wasn't in this format. Instead, it didn't have the right column names. The following code solved the problem as shown below:



```
# Let's First give all the columns proper name
new_columns = ['target', 'tweet_id', 'date', 'query_flag', 'user_id', 'text']
raw_data.columns = new_columns

raw_data.sample(5)
```

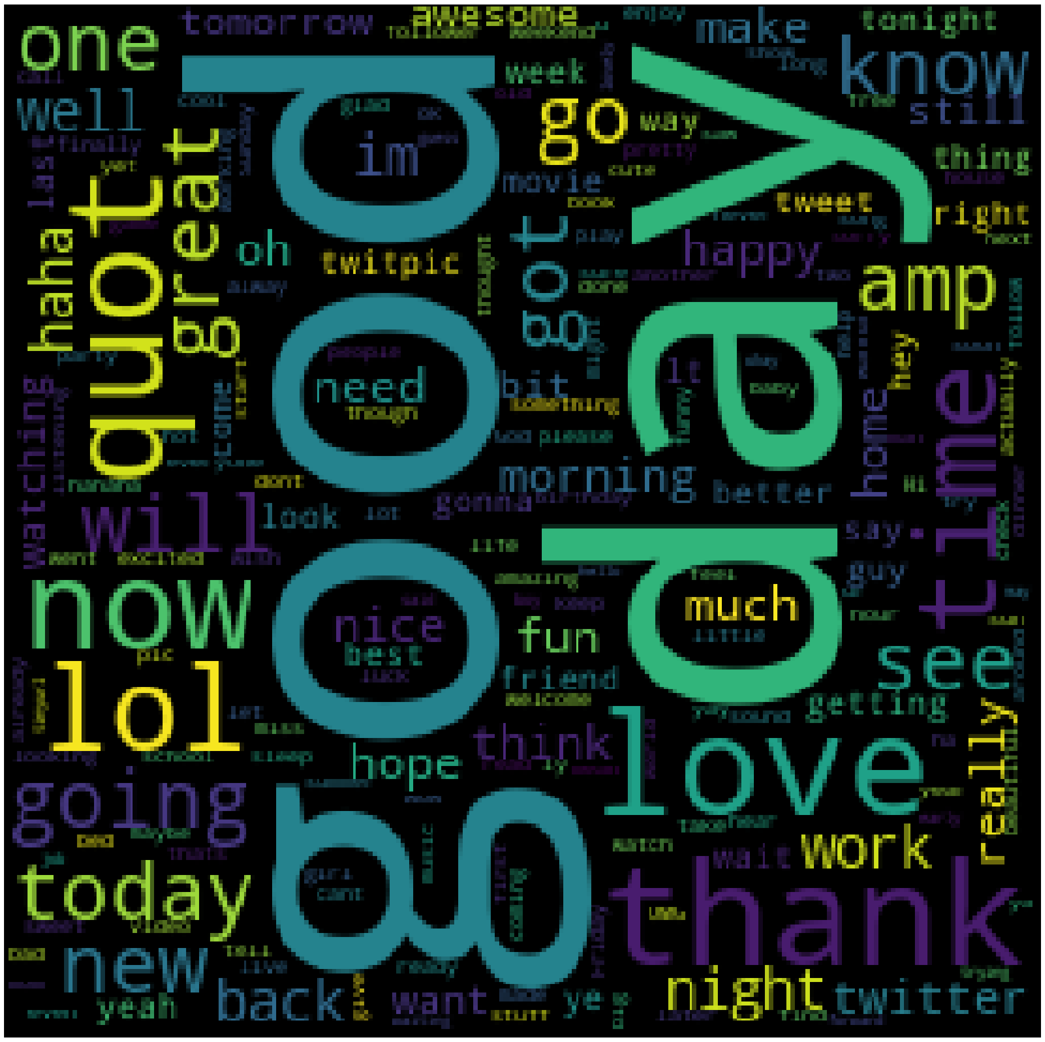
| | target | tweet_id | date | query_flag | user_id | text |
|---|---|---|---|---|---|---|
| **1400270** | 4 | 2054365152 | Sat Jun 06 07:11:24 PDT 2009 | NO_QUERY | soveren | @BentoSet woops no WIWT link |
| **280153** | 0 | 1991978295 | Mon Jun 01 07:59:44 PDT 2009 | NO_QUERY | Dominicanflower | @YoungQ im blocked here @ work, cant chat |
| **205648** | 0 | 1972929332 | Sat May 30 10:09:08 PDT 2009 | NO_QUERY | powpowkitty | @greenday86 I'm gonna miss you Papa |
| **1170304** | 4 | 1980363488 | Sun May 31 06:31:23 PDT 2009 | NO_QUERY | beauty_bross | @LineLoves Or ... You can make a cd, we can he... |
| **1452216** | 4 | 2063117388 | Sun Jun 07 01:26:00 PDT 2009 | NO_QUERY | nbula19v | had a great time now!! |

## 2.2 Exploratory Visualization

As noted above as well as in the proposal, there is not much scope of EDA in the particular project due to the type of data and limited computational power we humans possess.

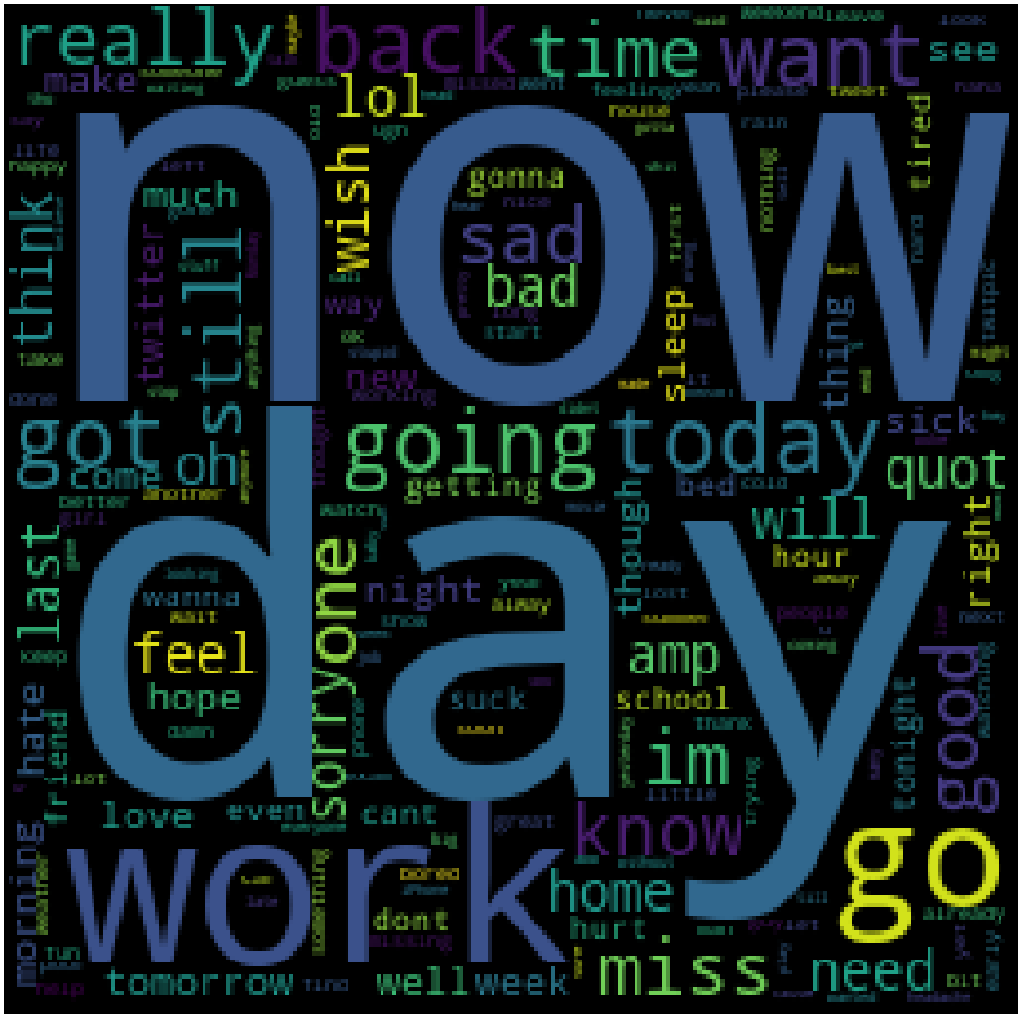
But I have managed to make 2 different Wordclouds specifically to get an idea of the type of words positive and negative (depressive) words in the dataset.

Below is the wordcloud of Positive Words in the dataset.



As very much obvious, this specific Wordcloud consists of words that are generally linked with positive sentiment. This wordcloud also shows that the data isn’t vague and has right words for the polarity of corresponding text.

Below now, is the Wordcloud of Negative (Depressive) words.



One interesting thing to note here, is that the negative words wordcloud have very vague words in majority (such as “now” or “day” instead of “sad” or “bad” in majority). This specific phenomena caught me in doubt for a long time and only after a substantial research, I found out that, Most people who are depressed don’t often express their emotions using words like (“sad” or “depressed”). Instead, most of them express this using their day-to-day feelings and thoughts of different aspects of their life.

The above wordclouds weren’t directly generated from raw data, instead the data was cleaned of things such “HTTP links”, “user handle references”, etc.

## 2.3 Algorithms and Techniques

Given the purpose of that the purpose of this project is to predict based on a given text sentence, whether the sentence is Depressive (1) or Not (0). For this specific purpose, I decided to use 6-different Machine Learning algorithms that have been previously observed to do well on similar Sentiment prediction task.

I have first used the most basic Logistic Regression Model (which has surprisingly worked better than all other models!). Since the problem is basically a prediction one, Logistic Regression seemed worth to a give a try.

After that, I have used Multinomial Naïve Bayes, which is more specifically an application of Bayes Law in Statistics. This model also did fine on the test-set achieving better than Benchmark set before.

Then, I used Random Forest Classification model, which has been previously observed to do much-much better in sentiment prediction and related tasks. However, in this case, this model came third in terms of Performance; behind Logistic Regression and Naïve Bayes.

After that, I used Decision Tree Classifier which too is among those models which have previously worked on such similar data.

Then, I used K-Nearest Neighbor Classifier on the training data. Although we are dealing with a Supervised learning problem, I still went ahead and used this Unsupervised learning technique.

Finally, I used an XGBoost Classifier on the data and calculated test-set ROC-AUC values as well as the Confusion Matrix.

Individual Performance was calculated on each and every model and was reported in proper form.

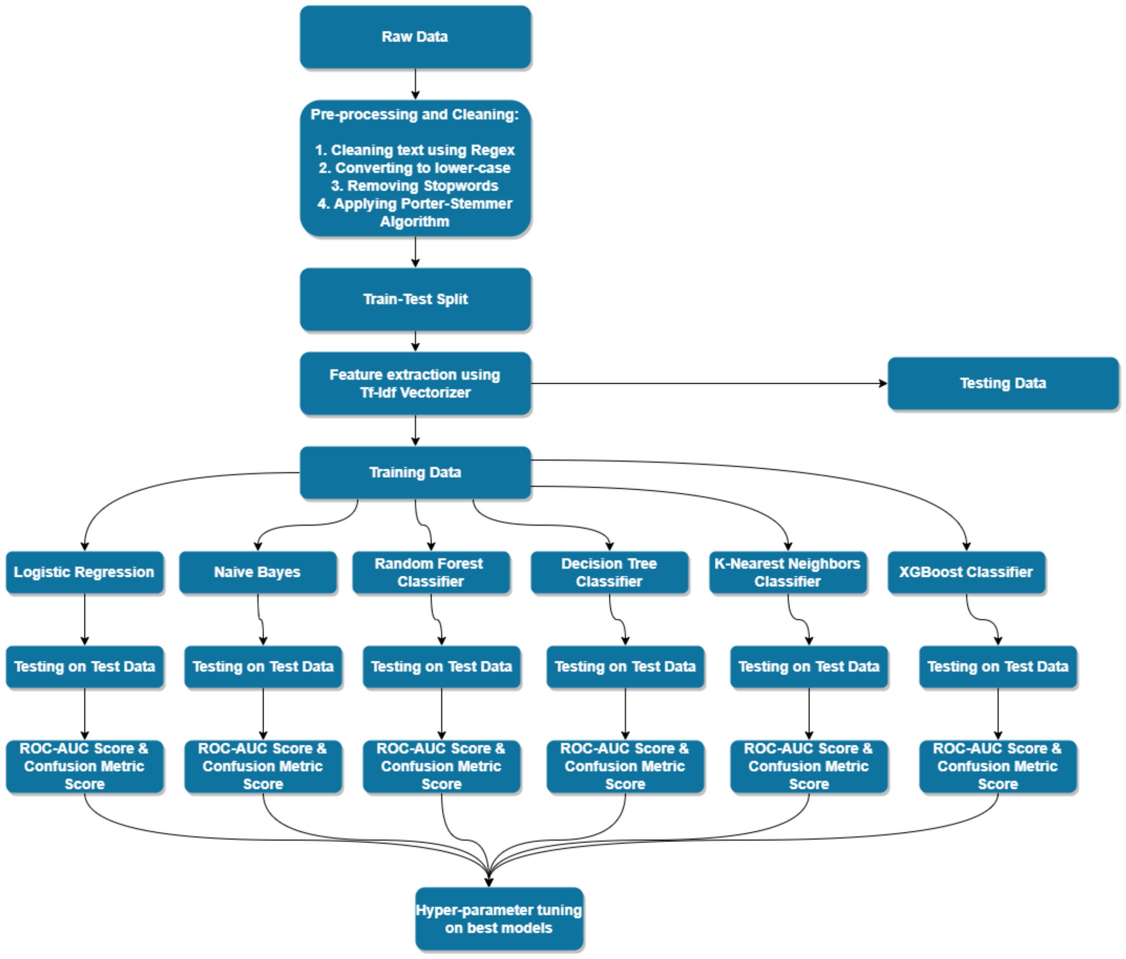
## 2.4 Benchmark

I decided to use either Random Forest Classifier (should it perform better than all other models) or alternatively a test-set value of **0.7** or above.

The reason for setting the benchmark value on test-set so low, is because, I was relatively less powerful Feature Extraction (such as using Tf-Idf Vectorizer, instead of Word2Vec, etc). Also, the fact I set the standard of this project to only use Machine Learning techniques and not deep learning account for the benchmark. Reason for such decisions was that the data is humongous, and training such models would take forever.

# 3. Methodology

The Overall methodology and plan of attack can be described from the flowchart below;


Raw Data
Pre-processing and Cleaning:
1. Cleaning text using Regex
2. Converting to lower-case
3. Removing Stopwords
4. Applying Porter-Stemmer Algorithm
Train-Test Split
Feature extraction using Tf-Idf Vectorizer
Testing Data
Training Data
Logistic Regression
Naive Bayes
Random Forest Classifier
Decision Tree Classifier
K-Nearest Neighbors Classifier
XGBoost Classifier
Testing on Test Data
ROC-AUC Score & Confusion Metric Score
Hyper-parameter tuning on best models

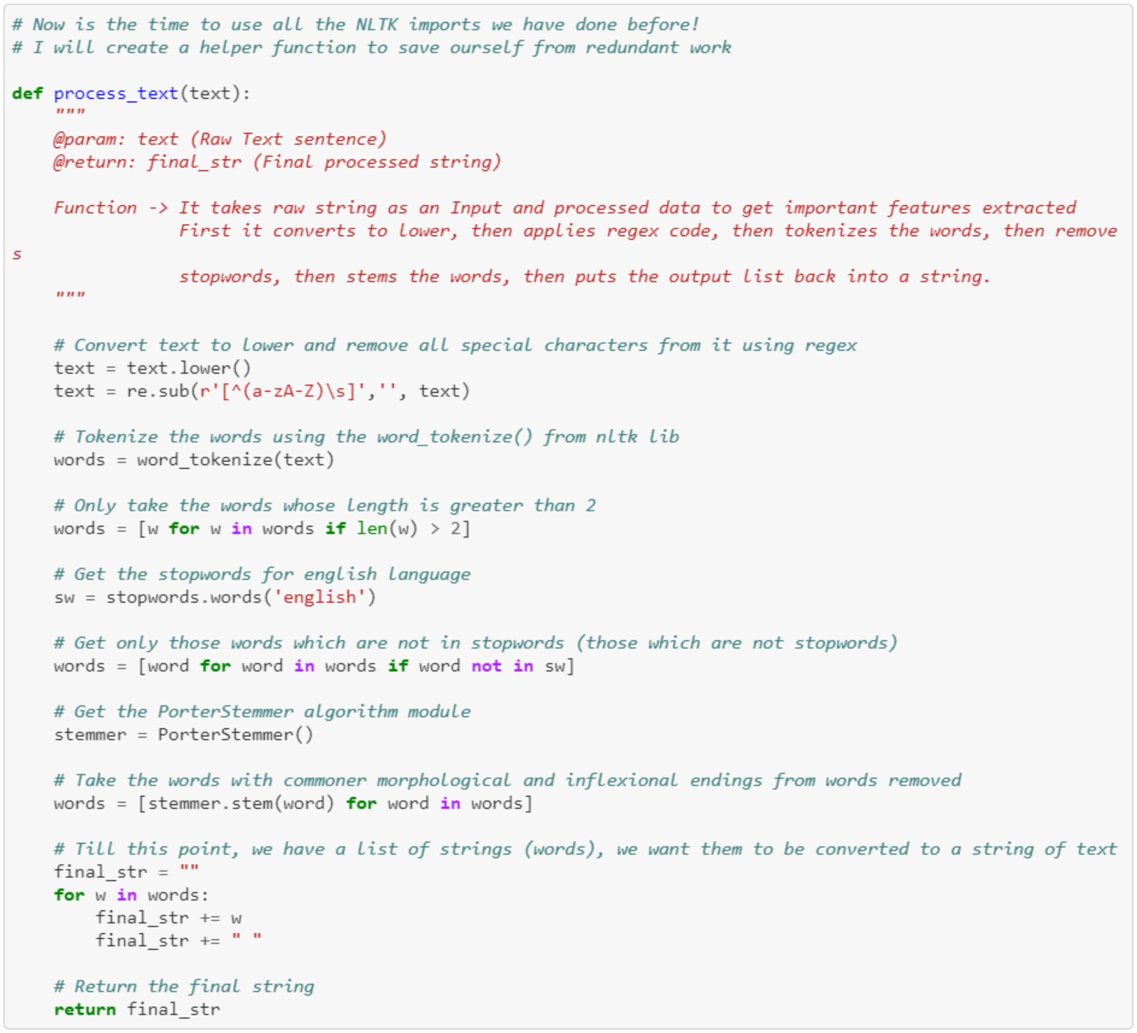
## 3.1 Data Preprocessing

Following steps were followed for overall data preprocessing:

- Data was first and foremost imported with proper encoding.
- Null Values were dropped from the data, poste-haste. As there was no feasible way to replace them, since data is text.
- **Cleaning:** Data was cleaned off HTTP links, twitter user references and any non-alpha numeric characters using regular expressions.
- **Processing:** Preprocessing steps to extract important features were applied, these included:
    - First, the Columns were renamed to make the data more approachable.

- Unneeded features were removed. These included 'tweet_id', 'date', 'query_flag', 'user_id' columns.
- The target values were changed to be more intuitive and less confusing. Initially, (4 --> Positive and 0 --> Negative), after changing, (1 --> Negative and 0 --> Positive).
- **Tokenizing:** Words were tokenized and only the words more than 2 characters long, were used, others were discarded.
- **Stopwords:** Stopwords were removed from the text.
- **Stemming:** Porter Stemmer algorithm was applied to the text

Following is the snapshot of one of the functions I used to pre-process the data:



```python
# Now is the time to use all the NLTK imports we have done before!
# I will create a helper function to save ourself from redundant work

def process_text(text):
    """
    @param: text (Raw Text sentence)
    @return: final_str (Final processed string)

    Function -> It takes raw string as an Input and processed data to get important features extracted
                First it converts to lower, then applies regex code, then tokenizes the words, then remove
s
                stopwords, then stems the words, then puts the output list back into a string.
    """

    # Convert text to lower and remove all special characters from it using regex
    text = text.lower()
    text = re.sub(r'[^(a-zA-Z)\s]','', text)

    # Tokenize the words using the word_tokenize() from nltk lib
    words = word_tokenize(text)

    # Only take the words whose length is greater than 2
    words = [w for w in words if len(w) > 2]

    # Get the stopwords for english language
    sw = stopwords.words('english')

    # Get only those words which are not in stopwords (those which are not stopwords)
    words = [word for word in words if word not in sw]

    # Get the PorterStemmer algorithm module
    stemmer = PorterStemmer()

    # Take the words with commoner morphological and inflexional endings from words removed
    words = [stemmer.stem(word) for word in words]

    # Till this point, we have a list of strings (words), we want them to be converted to a string of text
    final_str = ""
    for w in words:
        final_str += w
        final_str += " "

    # Return the final string
    return final_str
```

## 3.2 Implementation

At the beginning the plan was to use only 4 models on the data, but I later extended it to 6.

The first model I trained was Logistic Regression model. Surprisingly it did the best of all models and reported an overall ROC-AUC Score of **0.85** on the test-set. I used the default parameters and trained the model on it.

Second, I used the Multinomial Naïve Bayes and trained it on the training_data. The ROC-AUC Value was just behind the previous Logit Regression Mode of about **0.83** on the test set.

The 3rd model I used was the Random Forest Model. This was the model, I had most expectations with, but it didn’t perform that well. Same as the above 2, I used the default parameters and the score on the test-set came out to be **0.80**.

Note: Random Forest took a lot of time to train (a whole-whole lot!). This caused notebook session timeouts when running the notebook on Time-metered sessions (such as Google Colab, Kaggle Kernels, etc).

The 4th Model I used was Decision Tree Classifier. This model scored the lowest of all models I trained. Achieving a test-set ROC-AUC Score of **0.70** which is just the one I set as Benchmark.

The 5th Model was K-Nearest Neighbors. I first changed the "n_neighbors” parameters to 2, but for some reason, the performance was low. Turns out, the best choice was to leave it at default setting. The model achieved a test-set ROC-AUC score of **0.72**.

The 6th and final model was XGBoost Classifier. Leaving out all the parameters to default, the model achieved a test-set ROC-AUC score of **0.76**.

## 3.3 Refinement

I selected 3 of the above models for refinement:

I. Logistic Regression
II. Naïve Bayes
III. XGBoost

I used GridSearchCV to search for Optimal Hyperparamter. I provided the parameters to start the search on and it took on an average **3x** more time to do the parameter searching on each and every algorithm than it took to train them in the first place.

Unfortunately, after hours upon hours of refinement, I didn't manage to achieve any significant improve and the test-set ROC-AUC values remained all the same.

I have provided a table of performance comparison in the next section.

# 4. Results

## 4.1 Model Evaluation and Validation

The final model selected after all the process of training, testing and hyper-paramter search was found out to be Logistic Regression model. Not just because it showed better performance on test ROC-AUC values, but also because when tested on real data (real positive and negative sentence, I copied from twitter), it performed just as a right model should have performed.

Below is a tabular representation of all the models and their performance:

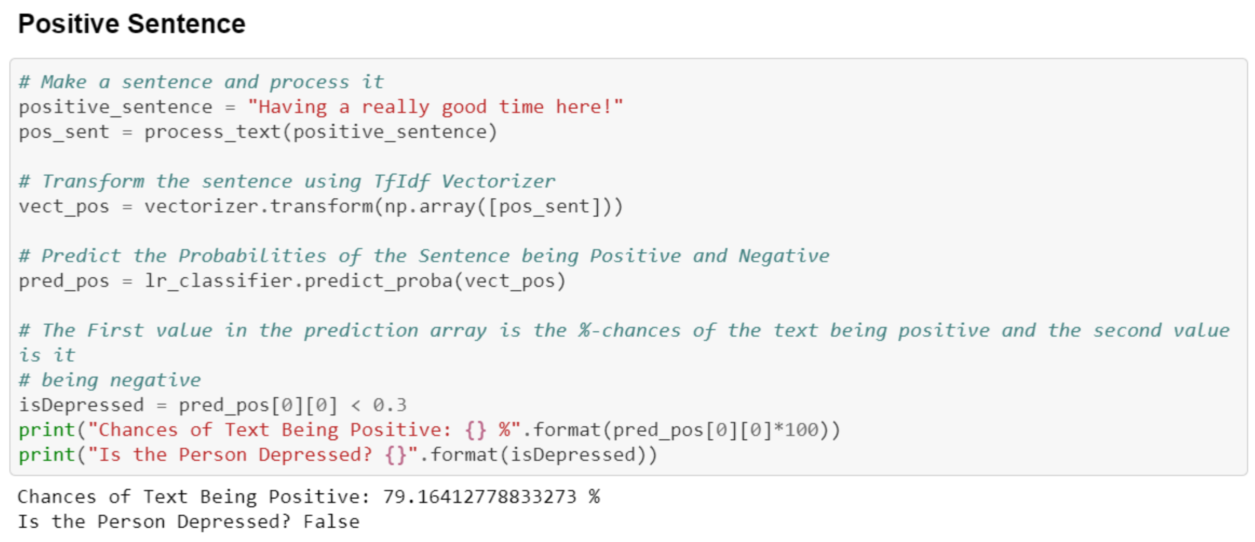
| **Models / Performance** | **Test-set ROC-AUC Score** |
|---|---|
| **Logistic Regression** | 0.8506 |
| **Multinomial Naïve Bayes** | 0.8328 |
| **Random Forest Classifier** | 0.8065 |
| **Decision tree Classifier** | 0.7042 |
| **K-Nearest Neighbor Classifier** | 0.7263 |

|  |  |
|---|---|
| **XGBoost Classifier** | 0.7625 |

## 4.2 Justification

Summarizing, the Logistic Regression approach proved out to be really good and the model achieved a descent score on the test-data. The benchmark that I proposed (0.7 on test-set) was met successfully by all models (except Decision tree classifier, which was on the edge).

I test the model on some real-world sentences and the result was fair-enough.



**Positive Sentence**

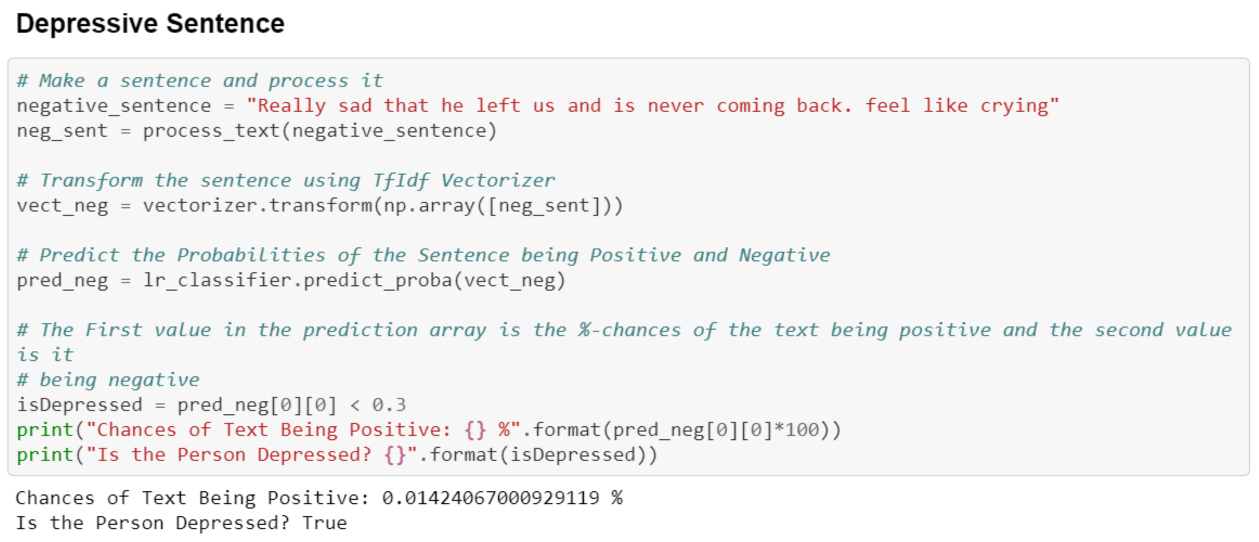
```
# Make a sentence and process it
positive_sentence = "Having a really good time here!"
pos_sent = process_text(positive_sentence)

# Transform the sentence using TfIdf Vectorizer
vect_pos = vectorizer.transform(np.array([pos_sent]))

# Predict the Probabilities of the Sentence being Positive and Negative
pred_pos = lr_classifier.predict_proba(vect_pos)

# The First value in the prediction array is the %-chances of the text being positive and the second value is it
# being negative
isDepressed = pred_pos[0][0] < 0.3
print("Chances of Text Being Positive: {} %".format(pred_pos[0][0]*100))
print("Is the Person Depressed? {}".format(isDepressed))
```

```
Chances of Text Being Positive: 79.16412778833273 %
Is the Person Depressed? False
```



**Depressive Sentence**

```
# Make a sentence and process it
negative_sentence = "Really sad that he left us and is never coming back. feel like crying"
neg_sent = process_text(negative_sentence)

# Transform the sentence using TfIdf Vectorizer
vect_neg = vectorizer.transform(np.array([neg_sent]))

# Predict the Probabilities of the Sentence being Positive and Negative
pred_neg = lr_classifier.predict_proba(vect_neg)

# The First value in the prediction array is the %-chances of the text being positive and the second value is it
# being negative
isDepressed = pred_neg[0][0] < 0.3
print("Chances of Text Being Positive: {} %".format(pred_neg[0][0]*100))
print("Is the Person Depressed? {}".format(isDepressed))
```

```
Chances of Text Being Positive: 0.01424067000929119 %
Is the Person Depressed? True
```

## 5. Improvement

The scope of improvement in the project is very much high. For instance, with big-enough compute power, One can extend the model to have Word-embeddings and train Sequence Models such as LSTMs and RNNs. These would highly increase the performance and make it more precise in prediction. We can also extend it to incorporating it into a Web-app and have on-the-go predictions.

## References and extra points

[1]: https://www.webmd.com/depression/guide/major-depression#1

[2]: https://www.omnicoreagency.com/twitter-statistics/

[3]: https://www.kaggle.com/kazanova/sentiment140

[4]: For this project, I decided to not use any Deep Learning algorithm, partly because of the data size and partly because I wanted to do this the Machine Learning way! I even mentioned it in my proposal

[5]: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

[6]: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62