1. a. PROBLEM DEFINITION: Sentiment Analysis with 55.000 Tweets

Text can be presented in two different styles: formal and informal writing style. A text may contain a lot of emotion, ideas or feelings. Twitter texts are generally informal and depending on the topic, they contain emotions of the writers. Social media users can freely express their feelings, views, opinions and emotion on social networking site. Short messages like SMS, tweets or Facebook messages are also useful source for sentiment analysis. Because of length constraints in these messages, Opinion Mining might be a bit difficult.

The scope of this project is to predict the sentiment of a short twitter text and categorize them over 3 sentiments (positive, negative, neutral). Since the twitter texts are simple, short, informal and contain creative spelling and punctuation, misspellings, slang, new words or URLs ; handling with that type of text would be harder to classify in terms of sentiments. Thus, analyzing and categorizing the emotional content of twitter text data is an important solution to create a corpus of tweets and texts with sentiment expressions.

a. Client:

a. **Companies/ Organizations using social media:** Sentiment analysis is crucial for the companies that use the feedback of their customers about their products or service. That gives valuable insights to the businesses regarding how people feel about their product brand or service.

b. **Social Media Influencers:** It helps to identify when potential negative threads are emerging online regarding a business, thereby allowing to be proactive in dealing with the source of negativities more quickly.

c. **Ecommerce traders** whose trade mostly depend on the reviews and ratings.

b. Data Set:

Dataset is acquired from https://data.world/crowdflower/sentiment-analysis-in-text

Data Set includes 4 features and 55.000 data points/samples. Features are: Tweet ID, Sentiment, Author, and Content.

This dataset contains labels for the emotional content (such as happiness, sadness, and anger) of texts.

Target Feature is ‘sentiment’

Since it is a supervised problem and there are 13 different sentiments in the target features, this will be a multi-label classification problem.

I will use NLP methods for data preprocessing (stemming and lemmatization, removing special characters, punctuations or removing stopwords). After data visualization and EDA, i will apply several algorithms for modeling (such as logistic regression, naive bayes, linear SVC) and apply deep learning algorithms using Keras.

2. Approach: Our target feature is 'sentiment' and it has 13 emotion labels. After evaluating among those emotions, we planned to limit the emotions to 3 sentiments positive: neutral and negative.

Anger, boredom, hate, worry, sadness: Negative

Happiness, fun, love, surprise, enthusiasm, relief: Positive

Empty, neutral : Neutral

3. Data Understanding and Preprocessing:

Cleaning the tweet texts (Removing special characters, punctuations, accented chracters, html tags): Special characters and symbols which are usually non alphanumeric characters often add to the extra noise in unstructured text. In our data set, which is an twitter text, there is a big amount of special characters.

In any text corpus, especially if you are dealing with the English language, often you might be dealing with accented characters\letters. Hence we need to make sure that these characters are converted and standardized into ASCII characters. A simple example would be converting é to e.

We defined a tweet cleaner function in order to clean the unnecessary characters.

Cleaning the tweet texts

```
import string

def tweet_cleaner(tweet):

    # To lowercase
    tweet = tweet.lower()

    # Remove HTML special entities (e.g. &amp;)
    tweet = re.sub(r'\&\w*;', '', tweet)

    #Convert @username to "@user"
    tweet = re.sub('@[^\s]+','@user',tweet)

    # Remove whitespace (including new line characters)
    tweet = re.sub(r'\s\s+', ' ', tweet)

    # Remove single space remaining at the front of the tweet.
    tweet = tweet.lstrip(' ')

    # Remove characters beyond Basic Multilingual Plane (BMP) of Unicode:
    tweet = ''.join(c for c in tweet if c <= '\uFFFF')

    # Remove hyperlinks
    tweet = re.sub(r'https?:\/\/.*\/\w*', 'http', tweet)

    # Remove tickers such as USD ($)
    tweet = re.sub(r'\$\w*', '', tweet)

    # Remove hashtags
    tweet = re.sub(r'#\w*', '', tweet)
```

Removing Stopwords: Words which have little or no significance especially when constructing meaningful features from text are known as stopwords or stop words. These are usually words that end up having the maximum frequency if you do a simple term or word frequency in a corpus. Words like a, an, the, and so on are considered to be stopwords. There is no universal stopword list but we use a standard English language stopwords list from NLTK.

Removing the stopwords

In [22]:
```
stop = stopwords.words('english')
df['Cleaned'] = df['Cleaned'].apply(lambda x: ' '.join([item for item in x.split() if item not i
df.head(10)
```

Stemming and lemmatization: Word stems are usually the base form of possible words that can be created by attaching affixes like prefixes and suffixes to the stem to create new words. This is known as inflection. The reverse process of obtaining the base form of a word is known as stemming. However the base form in this case is known as the root word but not the root stem. The difference being that the root word is always a lexicographically correct word (present in the dictionary) but the root stem may not be so. We used NLTK lemmatization in order to get the lemmatized text.

In [23]:
```
wordnet_lemmatizer=WordNetLemmatizer()

df.groupby('sentiment').content.apply(lambda x: ' '.join(x)).\
apply(lambda x: Counter([wordnet_lemmatizer.lemmatize(a) for a in
                         [k for k in [l for l in [t.lower() for t in word_tokenize(x)]
                                      if l.isalpha()] if k not in stop]]).most_common(5))
```

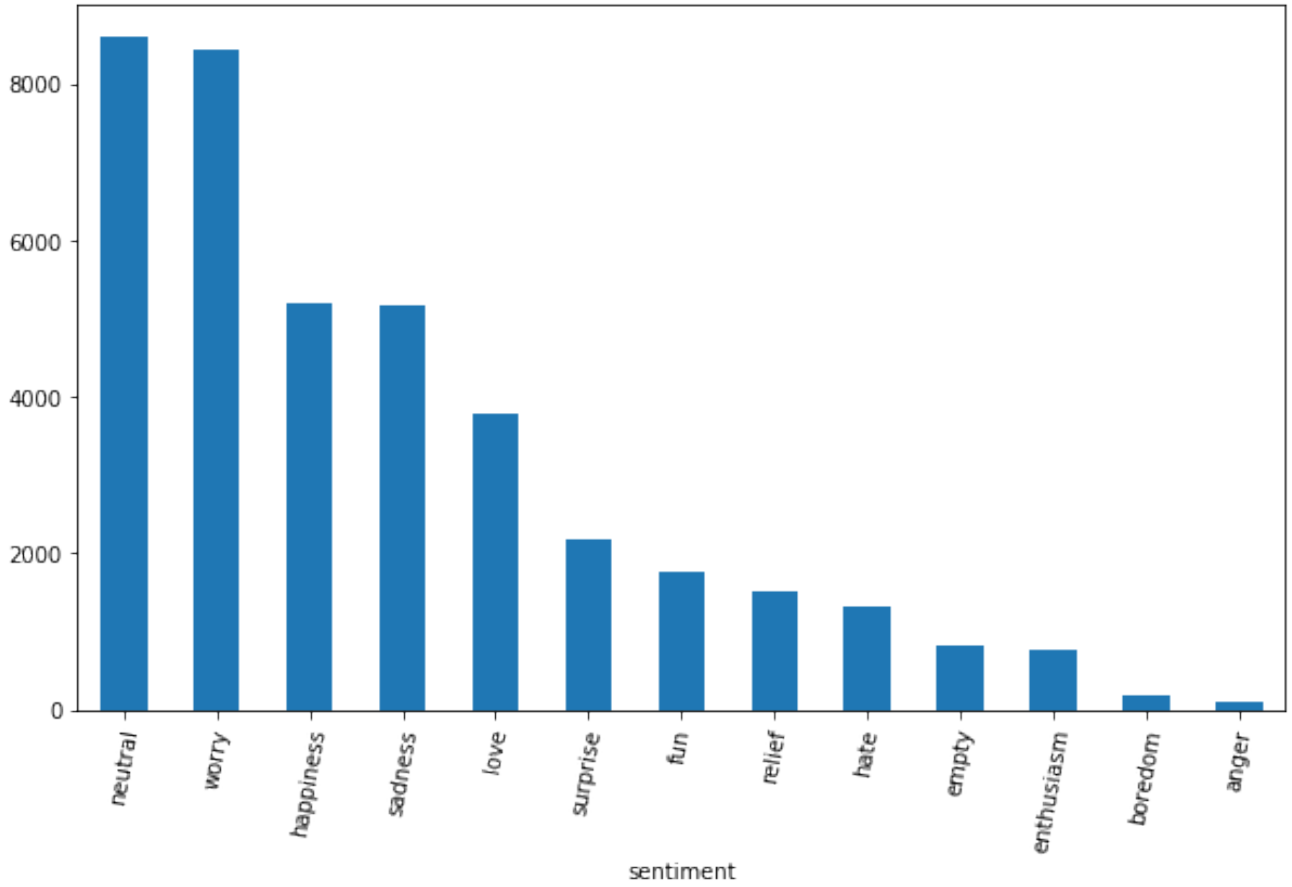
Missing Values, outliers, duplicates: There were no missing values in our data set. On the other hand, there were 173 repeating duplicates, which were removed, and only 1 left in the data set.

Bag of Words (Plus n-grams) (CountVectorizing in ScikitLearn): Bag of Words is the vector space representational model for unstructured text. It is simply a mathematical model to represent unstructured text (or any other data) as numeric vectors, such that each dimension of the vector is a specific feature/attribute. The bag of words model represents each text document as a numeric vector where each dimension is a specific word from the corpus and the value could be its frequency in the document, occurrence (denoted by 1 or 0) or even weighted values.

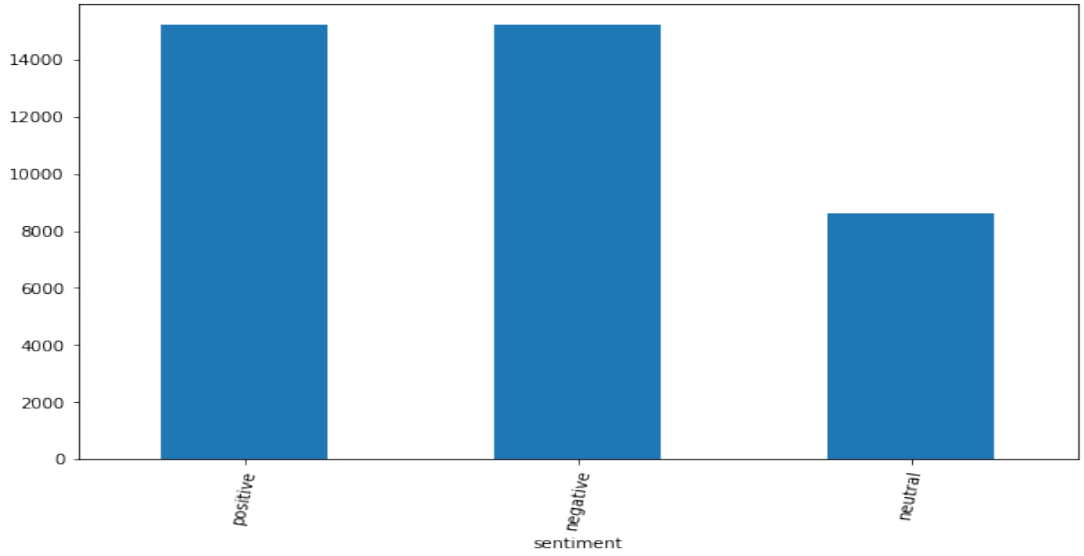
you	youll	your	youre	youtube	youve	yr	yum	yummy	ûï
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

4. Data Visualization:

The distribution of emotions in the data set is seen below.

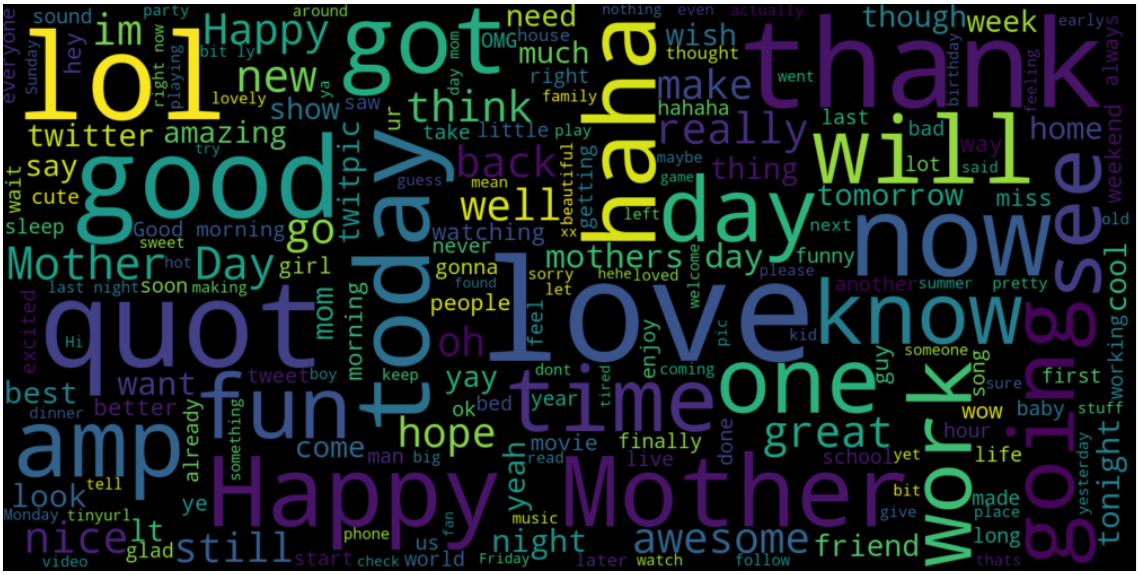


As can be seen from the bar plot (even before statistical analyses), the distribution of the emotions are not balanced. This will affect our analysis and model. In order to balance the data, we combined some certain emotions under 3 sentiments, and add more sample data to our data set.

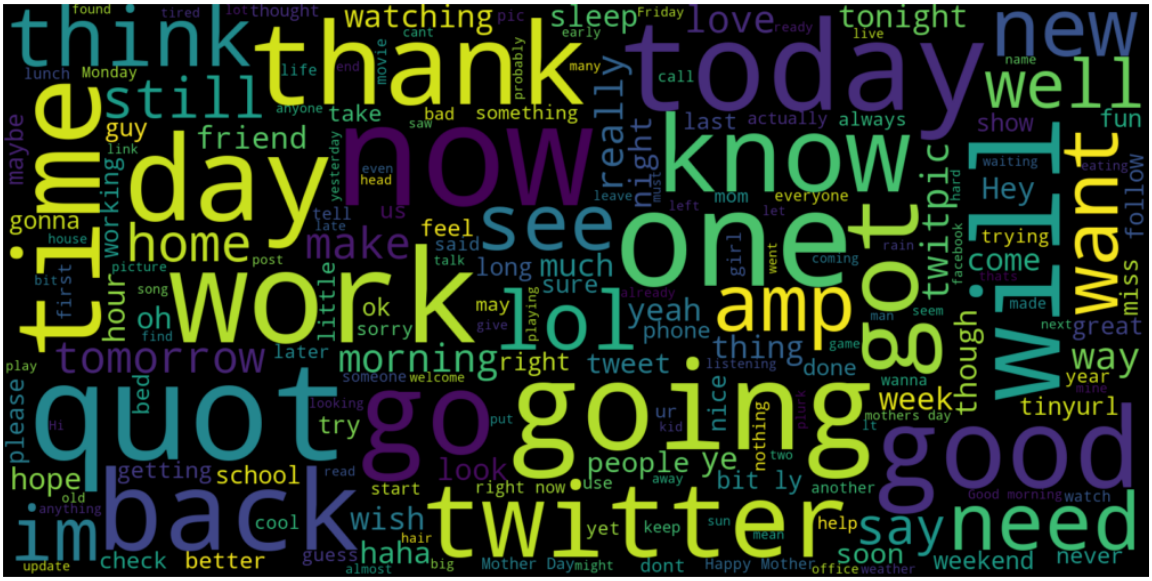


Regarding the 3 sentiments, the distribution of tokens on a word cloud is below:

Positive sentiment:



Neutral Sentiment:



Negative Sentiment

4. **NLTK Vader for sentiment comparison:**

NLTK Vader is a parsimonious rule-based model for sentiment analysis of social media text. With Vader, we can compare our dataset's classification with the Vader classification.

```
sentiments = []
compounds = []
sid = SentimentIntensityAnalyzer()
for i in dfnew.content:
    sentiment = sid.polarity_scores(i)
    if sentiment['compound']< -0.05:
        sentiments.append('negative')
    elif sentiment['compound']> 0.05:
        sentiments.append('positive')
    else:
        sentiments.append('neutral')
    compounds.append(sentiment)

dfnew['vader'] = pd.DataFrame(sentiments)
dfnew['compound'] = compounds
```

If the Vader compound result is lower than -0.05, the text is categorized as negative sentiment. Higher than 0.05 is a positive sentiment. Vader detects the sentiment regarding the tokens, bag of words or characters/emojis. Since our data set consists of informal, slang and short language, the Vader result and our data set categorization comparison fits 0.54. The main reason for this difference is that topics of our text are very wide-spread and it is difficult to understand authors' emotions. Thus, even though you read manually, especially between neutral and negative sentiments, it is difficult to distinguish a certain type of sentiment as seen below.

	tweet_id	author	content	Cleaned	sentiment	vader	compound
0	1.956968e+09	wannamama	Layin n bed with a headache ughhhh...waitin on your call...	layin n bed headache ughhhhwaitin call	negative	neutral	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
3	1.956968e+09	xkilljoyx	@dannycastillo We want to trade with someone who has Houston tickets, but no one will.	trade someone houston tickets	neutral	negative	{'neg': 0.165, 'neu': 0.767, 'pos': 0.068, 'compound': -0.3919}

As a result of Vader comparison, we realized that our dataset's classification is better than Vader's classification.

5. Modeling:

We applied logistic regression, naive bayes, linear svm, random forest, gradient boosting, xgboost and deep learning algorithms on bag of words and tfidf vectorization. We first applied our models on 13 emotions classification. Because of the imbalance in the distribution of features it was obvious that the accuracy results would not be very high.

a. Logistic Regression with Count Vectorizing

For 13 emotions

```
logreg_CV1 = LogisticRegression(multi_class='multinomial', solver='newton-cg',class_wei
                                C=1.0,n_jobs=-1, random_state=5)
logreg_CV1.fit(count_vect_train1, y_train1)
y_pred_lr_CV1 = logreg_CV1.predict(count_vect_test1)
print('Accuracy :', metrics.accuracy_score(y_test1, y_pred_lr_CV1))
```

```
Accuracy : 0.2208691193436739
```

For 3 sentiments

```
logreg_CV = LogisticRegression(random_state=0)
logreg_CV.fit(count_vect_train, y_train)
y_pred_lr_CV = logreg_CV.predict(count_vect_test)
print('Accuracy :', metrics.accuracy_score(y_test, y_pred_lr_CV))
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarni
ng: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this
warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:469: FutureWarni
ng: Default multi_class will be changed to 'auto' in 0.22. Specify the multi_class optio
n to silence this warning.
  "this warning.", FutureWarning)
Accuracy : 0.6158856116185057
```

b. Linear SVC with Count Vectorizing

```
Lsvc = LinearSVC()

Lsvc.fit(count_vect_train, y_train)

pred= Lsvc.predict(count_vect_test)

metrics.accuracy_score(y_test, pred)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:929: ConvergenceWarning: Libl
inear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
0.6156610270998653
```

c. Naïve Bayes with Count Vectorizing

```
from sklearn.naive_bayes import MultinomialNB

nb_classifier = MultinomialNB()

nb_classifier.fit(count_vect_train, y_train)

pred = nb_classifier.predict(count_vect_test)

metrics.accuracy_score(y_test, pred)
```

```
0.566701602036233
```

d. Random Forest with Count Vectorizing

```
from sklearn.ensemble import RandomForestClassifier
rf_CV = RandomForestClassifier(random_state=0, n_jobs=-1, class_weight="balanced", n_est
rf_CV.fit(count_vect_train, y_train)
y_pred_rf_CV = rf_CV.predict(count_vect_test)
print('Accuracy :', metrics.accuracy_score(y_test, y_pred_rf_CV))
```

```
Accuracy : 0.5810001497230124
```

e. Gradient Boosting with Count Vectorizing

```
xg_boost_CV = XGBClassifier()
xg_boost_CV.fit(count_vect_train, y_train)
y_pred_xg_boost = xg_boost_CV.predict(count_vect_test)
print('Accuracy :', metrics.accuracy_score(y_test, y_pred_xg_boost))
```

```
Accuracy : 0.5488097020512053
```

f. Xgboost with Count Vectorizing

```
xg_boost_CV = XGBClassifier()
xg_boost_CV.fit(count_vect_train, y_train)
y_pred_xg_boost = xg_boost_CV.predict(count_vect_test)
print('Accuracy :', metrics.accuracy_score(y_test, y_pred_xg_boost))
```

```
Accuracy : 0.5488097020512053
```

g. Logistic Regression with Tf-idf

```
logreg_TV = LogisticRegression(class_weight='balanced', random_state=5)
logreg_TV.fit(tfidf_vect_train, y_train)
y_pred_lr_TV = logreg_TV.predict(tfidf_vect_test)
print('Accuracy :', metrics.accuracy_score(y_test, y_pred_lr_TV))
print (classification_report(y_test, y_pred_lr_TV))
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarni
ng: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this
warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:469: FutureWarni
ng: Default multi_class will be changed to 'auto' in 0.22. Specify the multi_class optio
n to silence this warning.
  "this warning.", FutureWarning)
Accuracy : 0.618580625842192
              precision    recall  f1-score   support

    negative       0.71      0.72      0.72      6124
     neutral       0.43      0.41      0.42      2917
    positive       0.61      0.62      0.61      4317

    accuracy                           0.62     13358
   macro avg       0.58      0.58      0.58     13358
weighted avg       0.62      0.62      0.62     13358
```

h. Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer

text_clf = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB()),
])

text_clf.fit(X_train, y_train)

print(text_clf.score(X_test, y_test))
```

```
0.5841443329839796
```

i. Grid Search

```
grid_search.fit(X_train, y_train)

print('score', grid_search.score(X_test, y_test))
print('-----')

print('GridSearchCV:')
y_pred = grid_search.predict(X_test)

print("Best score: %0.3f" % grid_search.best_score_)
print("Best parameters set:")
best_parameters = grid_search.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

```
Fitting 3 folds for each of 64 candidates, totalling 192 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    7.4s
[Parallel(n_jobs=-1)]: Done 192 out of 192 | elapsed:   28.8s finished
score 0.5946998053600838
-----
GridSearchCV:
Best score: 0.593
Best parameters set:
        clf__alpha: 1.0
        vect__max_df: 0.7
        vect__min_df: 10
        vect__stop_words: 'english'
```

j. Deep Learning Algorithm

```
import keras
from keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(num_words=2500,split=' ')
tokenizer.fit_on_texts(dfnew.Cleaned)
```

```
from keras.preprocessing.sequence import pad_sequences

X = tokenizer.texts_to_sequences(dfnew.Cleaned)
X = pad_sequences(X)
print(X)
```

```
[[   0    0    0 ...  106  538   52]
 [   0    0    0 ...    0 1637  207]
 [   0    0    0 ...  669  147  111]
 ...
 [   0    0    0 ...  603  944  479]
 [   0    0    0 ... 1702    2   29]
 [   0    0    0 ...   42   55    4]]
```

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Embedding, LSTM

model = Sequential()
model.add(Embedding(2500,128,input_length=X.shape[1],dropout=0.2))
model.add(LSTM(300, dropout_U=0.2,dropout_W=0.2))
model.add(Dense(3,activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,optimizer='adam',metrics=['accu
```

```
Epoch 1/5
 - 336s - loss: 0.8706 - acc: 0.5974
Epoch 2/5
 - 333s - loss: 0.7893 - acc: 0.6472
Epoch 3/5
 - 333s - loss: 0.7533 - acc: 0.6618
Epoch 4/5
 - 333s - loss: 0.7174 - acc: 0.6812
Epoch 5/5
 - 333s - loss: 0.6860 - acc: 0.6964
```

Our best score is 0.68 with 3 sentiments and additional data samples. There are many reasons for not reaching to higher scores:

- We have chosen a difficult and informal text in order to make it harder to analyze the sentiment. If the data were about a review of a product or a company, it would be easier to apply sentiment analysis techniques. On the other hand, our data is a general text with a wide variety of twitter text. For some of the text, it is very difficult to categorize in a certain type of emotion or sentiment.

- The distribution of the features are not balanced. In order to deal with this problem, we concatenated additional 15 thousands row data set, we applied random forest model and hyper-tuning and feature engineering.

- Twitter texts are very short and there are lots of abbreviations or social media slangs that are difficult to be categorized. In order to deal with this, we applied nltk methods for data preprocessing

5. CONCLUSION:

Our project is based on a multi-class classification problem consisting of emotions and sentiments. The aim of the project is to predict the sentiment of a certain twitter text. We worked on the preprocessing part of the project by means of several tools such as NLTK, Spacy, Beautifulsoup, Vader, tokenizers and many python functions.

Since the distribution of the sentiments are not balanced, we decided to categorize the emotions into 3 sentiments and add more samples in order to prevent the imbalance. We used random forest model to balance the distribution and Grid Search with 5-fold cross validation technique to deal with the overfitting problem. Because of the reasons mentioned above, our best score is 0.68 after applying Tf-idf and deep learning algorithm.

It is clearly understood that most important things for an effective sentiment analysis of short social media texts are data preprocessing, feature engineering and choosing the best model.

6. FUTURE WORK:

For increasing the accuracy of our model, it is very important to find additional balanced data sets, and applying effective feature engineering techniques. Applying Word2vec or Phrase modeling could also improves the model. Categorization of sentiments in 2 classes (such as bad or not bad) could also give higher results. Run time for these kinds of data with decision tree models and deep learning models is relatively long. Decreasing the run time with more efficient computers or cloud systems could be used for increasing the effectiveness.

Improving our deep learning algorithm with professional deep learning techniques and applying advanced hyper parameter tuning could also contribute much for future studies of our project.