Untitled1

September 19, 2020

0.0.1 NLP : Project 2 - Help Twitter Combat Hate Speech Using NLP and Machine Learning

0.0.2 Importing All Library Packages

```
In [20]: import re
         import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as sns
         import string
         import nltk
         import warnings
         warnings.filterwarnings("ignore", category=DeprecationWarning)
         from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.pipeline import Pipeline
         from sklearn.model_selection import train_test_split, GridSearchCV
         from sklearn.metrics import confusion_matrix,f1_score
         from sklearn.naive_bayes import MultinomialNB
         from sklearn.ensemble import RandomForestClassifier

         %matplotlib inline
```

0.0.3 Load the tweets file using read_csv function from Pandas package.

```
In [2]: df = pd.read_csv(r'E:\study\simpli\NLP_proj\TwitterHate.csv')
```

```
In [3]: df.head()
```

```
Out[3]:    id  label                                              tweet
        0   1      0   @user when a father is dysfunctional and is s...
        1   2      0  @user @user thanks for #lyft credit i can't us...
        2   3      0                                bihday your majesty
        3   4      0  #model   i love u take with u all the time in ...
        4   5      0             factsguide: society now    #motivation
```

```
In [4]: df.shape,df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31962 entries, 0 to 31961
```

```
Data columns (total 3 columns):
id       31962 non-null int64
label    31962 non-null int64
tweet    31962 non-null object
dtypes: int64(2), object(1)
memory usage: 749.2+ KB
```

```
Out[4]: ((31962, 3), None)
```

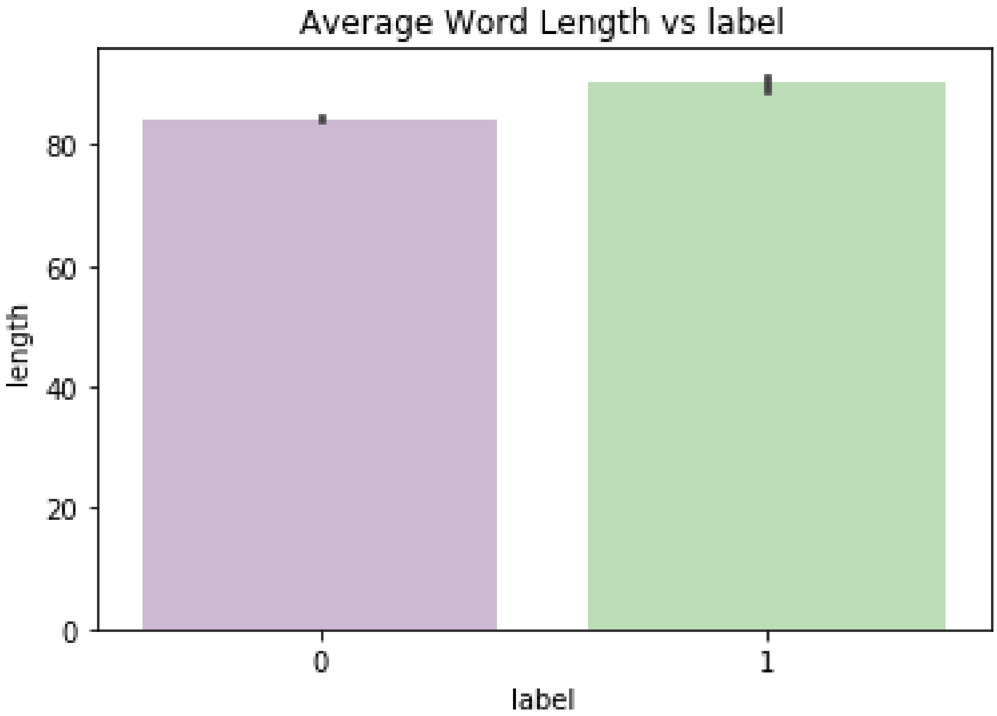
```
In [5]: df['label'].value_counts()
```

```
Out[5]: 0    29720
        1     2242
        Name: label, dtype: int64
```

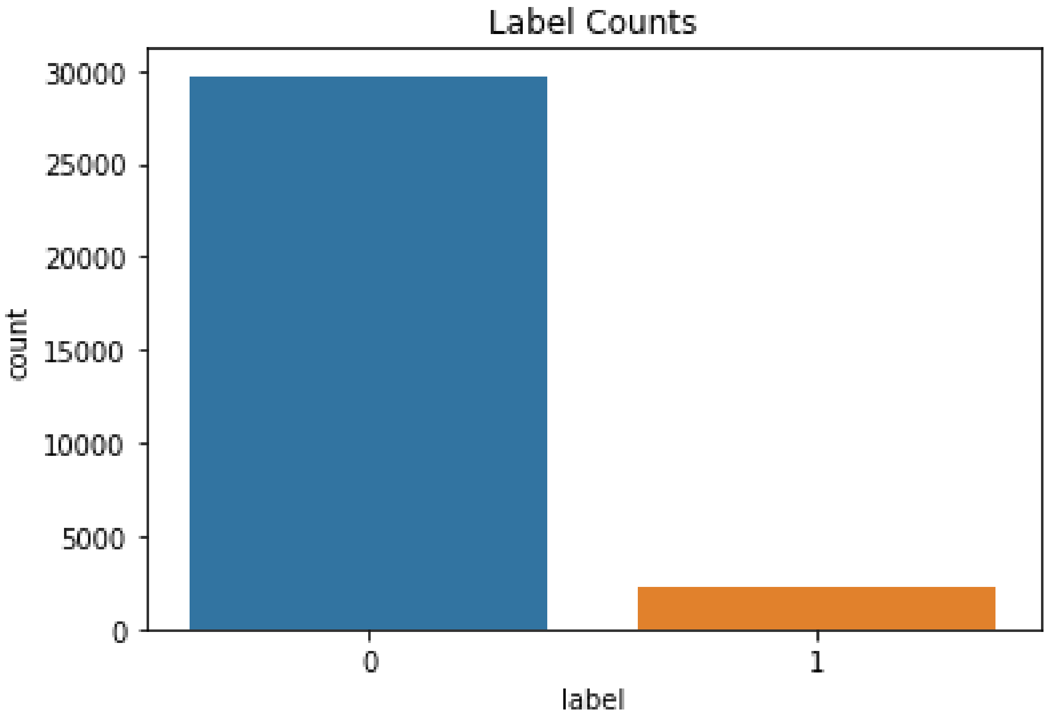
```
In [7]: import seaborn as sns
        import re
        import matplotlib.pyplot as plt
        % matplotlib inline

        import seaborn as sns
        from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
```

0.0.4 Exploratory Data Analysis about the tweets

```
In [9]: #Exploratory Data Analysis
        df['length'] = df['tweet'].apply(len)
        fig1 = sns.barplot('label','length',data = df,palette='PRGn')
        plt.title('Average Word Length vs label')
        plot = fig1.get_figure()
        plot.savefig('Barplot.png')
```



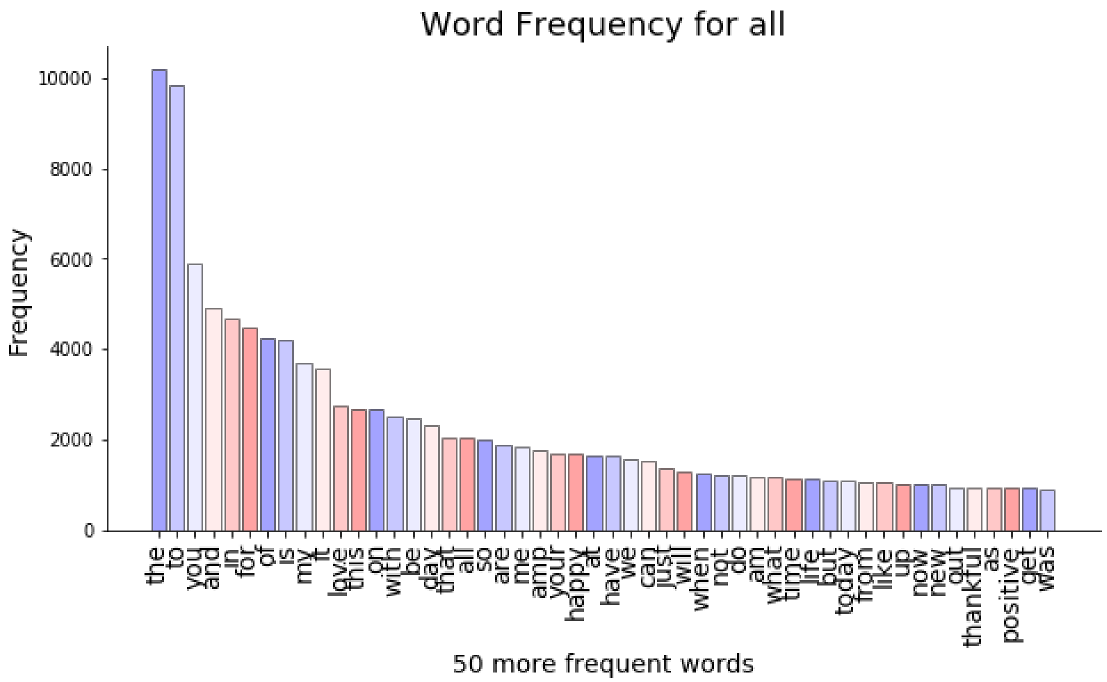
```
In [10]: #bar graph to count positive negative label
         fig2 = sns.countplot(x= 'label',data = df)
         plt.title('Label Counts')
         plot = fig2.get_figure()
         plot.savefig('Count Plot.png')
```



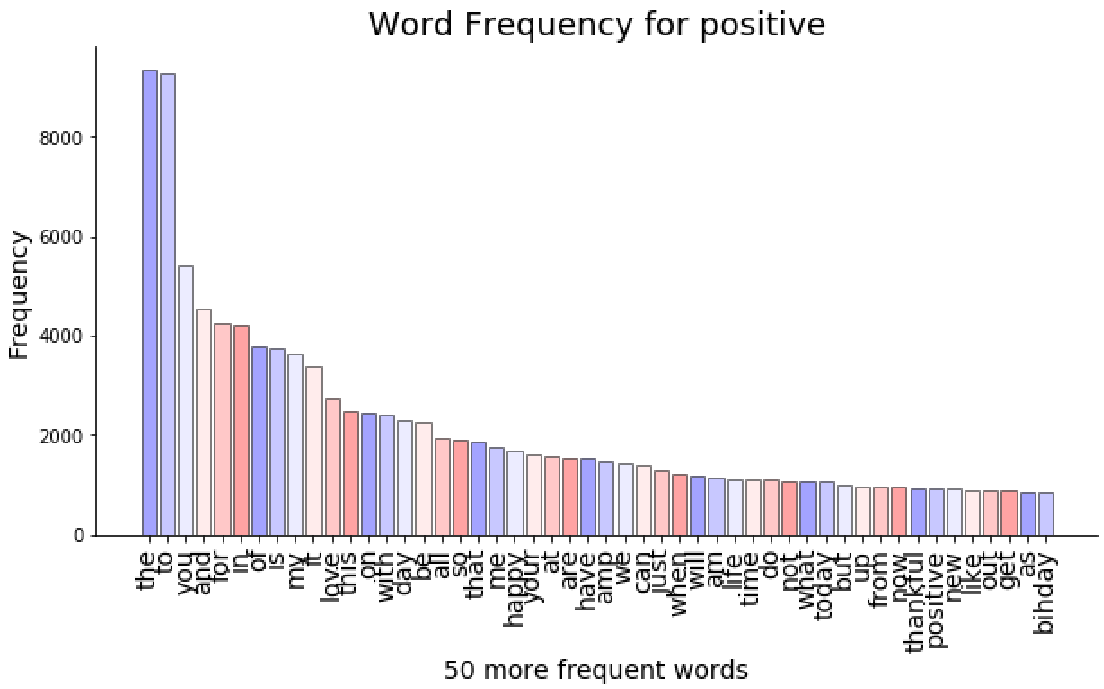
```
In [16]: df['length'].head()

Out[16]: 0    102
         1    122
         2     21
         3     86
         4     39
         Name: length, dtype: int64

In [17]: def vectorization(table):
             #CountVectorizer will convert a collection of text documents to a matrix of token
             #Produces a sparse representation of the counts
             #Initialize
             vector = CountVectorizer()
             #We fit and transform the vector created
             frequency_matrix = vector.fit_transform(table.tweet)
             #Sum all the frequencies for each word
             sum_frequencies = np.sum(frequency_matrix, axis=0)
             #Now we use squeeze to remove single-dimensional entries from the shape of an arr
             #the sum of frequencies.
             frequency = np.squeeze(np.asarray(sum_frequencies))
             #Now we get into a dataframe all the frequencies and the words that they correspo
             frequency_df = pd.DataFrame([frequency], columns=vector.get_feature_names()).trans
             return frequency_df
```

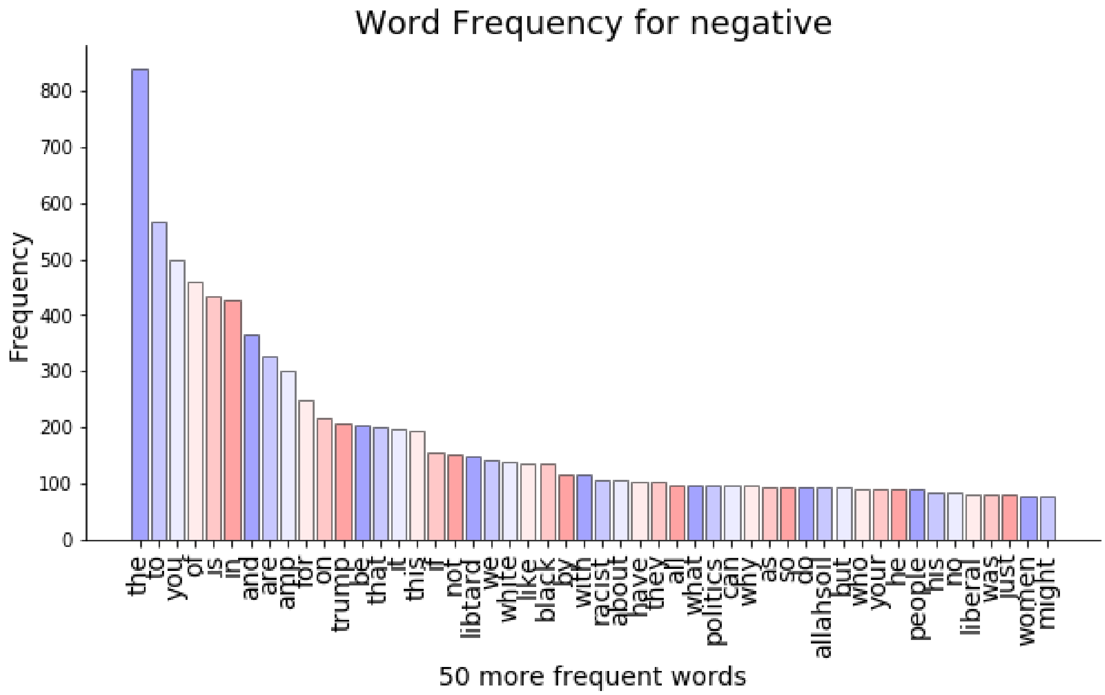
```
In [18]: def graph(word_frequency, sent):
             labels = word_frequency[0][1:51].index
             title = "Word Frequency for %s" %sent
             #Plot the figures
             plt.figure(figsize=(10,5))
             plt.bar(np.arange(50), word_frequency[0][1:51], width = 0.8, color = sns.color_pal
                     edgecolor = "black", capsize=8, linewidth=1);
             plt.xticks(np.arange(50), labels, rotation=90, size=14);
             plt.xlabel("50 more frequent words", size=14);
             plt.ylabel("Frequency", size=14);
             #plt.title('Word Frequency for %s', size=18) %sent;
             plt.title(title, size=18)
             plt.grid(False);
             plt.gca().spines["top"].set_visible(False);
             plt.gca().spines["right"].set_visible(False);
             plt.show()
```

```
In [21]: word_frequency = vectorization(df).sort_values(0, ascending = False)
         graph(word_frequency, 'all')
```



```
In [22]: word_frequency_pos = vectorization(df[df['label'] == 0]).sort_values(0, ascending = Fa
         word_frequency_neg = vectorization(df[df['label'] == 1]).sort_values(0, ascending = Fa

         graph(word_frequency_pos, 'positive')
         graph(word_frequency_neg, 'negative')
```

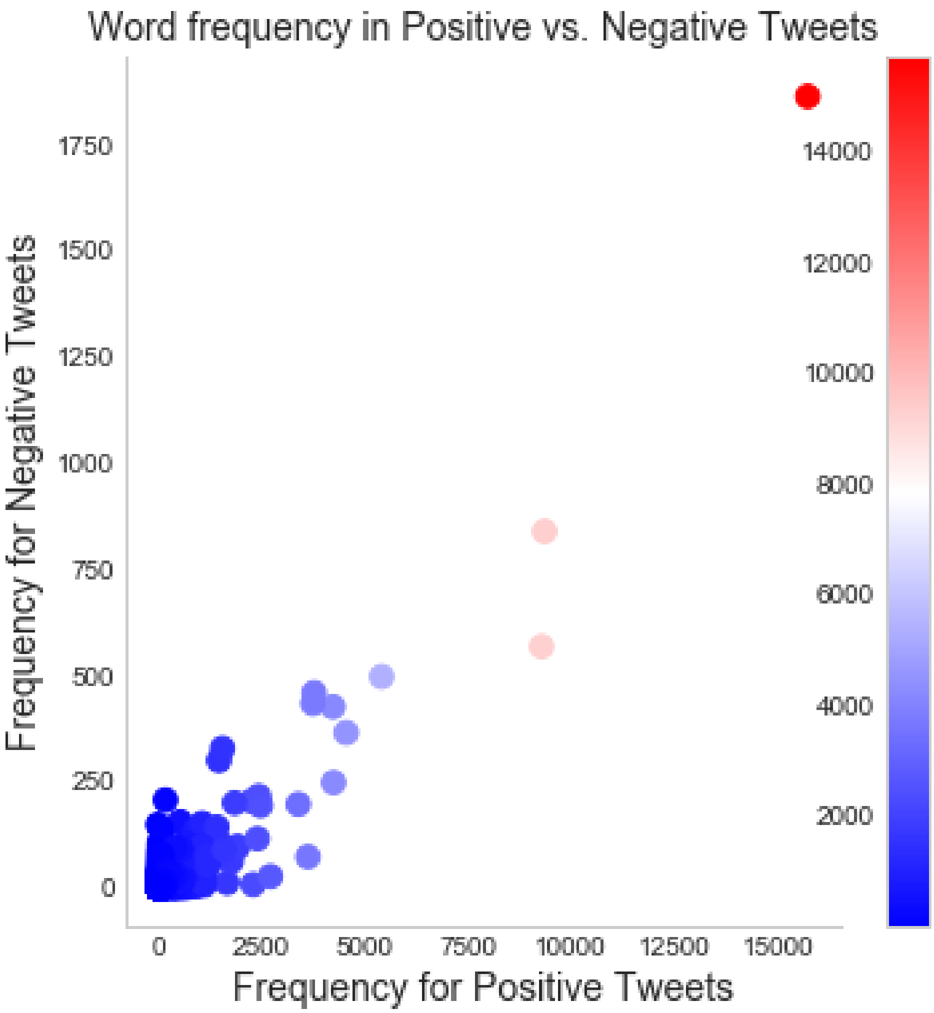




```
In [23]: def regression_graph(table):
             table = table[1:]
```

```
        #We set the style of seaborn
        sns.set_style("whitegrid")
        #Initialize the figure
        plt.figure(figsize=(6,6))

        #we obtain the points from matplotlib scatter
        points = plt.scatter(table["Positive"], table["Negative"], c=table["Positive"], s=
        #graph the colorbar
        plt.colorbar(points)
        #we graph the regplot from seaborn
        sns.regplot(x="Positive", y="Negative",fit_reg=False, scatter=False, color=".1", c
        plt.xlabel("Frequency for Positive Tweets", size=14)
        plt.ylabel("Frequency for Negative Tweets", size=14)
        plt.title("Word frequency in Positive vs. Negative Tweets", size=14)
        plt.grid(False)
        sns.despine()
```

```
In [25]: table_regression = pd.concat([word_frequency_pos, word_frequency_neg], axis=1)
         table_regression.columns = ["Positive", "Negative"]
         regression_graph(table_regression)
```



0.0.5 Appying Preprocessing and Cleaning on the tweets

```
In [26]: def drop_features(features,data):
             data.drop(features,inplace=True,axis=1)
```

```
In [27]: import re
         ## example ##
         re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])", " ","ouch...junior is angryð§ŸŘ#got7 #juni
```

```
Out[27]: 'ouch   junior is angry     got7  junior  yugyo      '
```

```
In [28]: def process_tweet(tweet):
             return " ".join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])", " ",tweet.lower()).spl
```

```
In [29]: df['processed_tweets'] = df['tweet'].apply(process_tweet)
```

```
In [30]: df.head()
```

```
Out[30]:    id  label                                              tweet  length  \
         0   1      0   @user when a father is dysfunctional and is s...     102
         1   2      0  @user @user thanks for #lyft credit i can't us...     122
         2   3      0                                bihday your majesty      21
         3   4      0  #model   i love u take with u all the time in ...      86
         4   5      0             factsguide: society now    #motivation      39

                                             processed_tweets
         0  when a father is dysfunctional and is so selfi...
         1  thanks for lyft credit i can t use cause they ...
         2                                bihday your majesty
         3      model i love u take with u all the time in ur
         4                  factsguide society now motivation
```

```
In [31]: drop_features(['id','tweet'],df)
```

```
In [32]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31962 entries, 0 to 31961
Data columns (total 3 columns):
label               31962 non-null int64
length              31962 non-null int64
processed_tweets    31962 non-null object
dtypes: int64(2), object(1)
memory usage: 749.2+ KB
```

Train Test Split and Determining TF-IDF vectorizer values

```
In [33]: from sklearn.model_selection import train_test_split
         x_train, x_test, y_train, y_test = train_test_split(df["processed_tweets"], df["label
```

```
In [34]: from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
         count_vect = CountVectorizer(stop_words='english')
         transformer = TfidfTransformer(norm='l2',sublinear_tf=True)
```

```
In [35]: x_train_counts = count_vect.fit_transform(x_train)
         x_train_tfidf = transformer.fit_transform(x_train_counts)
```

```
In [36]: print(x_train_counts.shape)
         print(x_train_tfidf.shape)
```

```
(25569, 33735)
(25569, 33735)
```

```
In [37]: x_test_counts = count_vect.transform(x_test)
         x_test_tfidf = transformer.transform(x_test_counts)
```

```
In [38]: print(x_test_counts.shape)
         print(x_test_tfidf.shape)

(6393, 33735)
(6393, 33735)
```

0.0.6 Applying Model Implementation :

```
In [39]: from sklearn.linear_model import LogisticRegression
         from sklearn.naive_bayes import MultinomialNB
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.linear_model import SGDClassifier
         from sklearn.metrics import f1_score

In [49]: #### Model building: Ordinary Logistic Regression

In [40]: modelLR = LogisticRegression(C=100).fit(x_train_tfidf,y_train)

In [50]: #### Model evaluation: Accuracy, recall, and f_1 score.

In [41]: predictionsLR = modelLR.predict(x_test_tfidf)
         sum(predictionsLR==1),len(y_test),f1_score(y_test,predictionsLR)

Out[41]: (334, 6393, 0.7063291139240505)

In [51]: #### Train again with the adjustment and evaluate :Regularization and Hyperparameter

In [42]: from sklearn.pipeline import Pipeline
         from sklearn.naive_bayes import MultinomialNB
         from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
         from sklearn.model_selection import train_test_split, GridSearchCV

In [43]: text_clf = Pipeline([('vect', CountVectorizer()),
                              ('tfidf', TfidfTransformer()),
                              ('clf', MultinomialNB())])
         tuned_parameters = {
             'vect__ngram_range': [(1, 1), (1, 2), (2, 2)],
             'tfidf__use_idf': (True, False),
             'tfidf__norm': ('l1', 'l2'),
             'clf__alpha': [1, 1e-1, 1e-2]
         }
         from sklearn.metrics import classification_report
         clf = GridSearchCV(text_clf, tuned_parameters, cv=10)
         clf.fit(x_train, y_train)

Out[43]: GridSearchCV(cv=10, error_score='raise',
                estimator=Pipeline(memory=None,
              steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='stri
                 dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                 lowercase=True, max_df=1.0, max_features=None, min_df=1,
                 ngram_range=(1, 1), preprocessor=None, stop_words=None,
                 strip...inear_tf=False, use_idf=True)), ('clf', MultinomialNB(alpha=1.0, clas
                fit_params=None, iid=True, n_jobs=1,
                param_grid={'vect__ngram_range': [(1, 1), (1, 2), (2, 2)], 'tfidf__use_idf': (
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=None, verbose=0)
```

```
In [45]: print(classification_report(y_test, clf.predict(x_test), digits=4))
```

```
             precision    recall  f1-score   support

          0     0.9666    0.9939    0.9801      5937
          1     0.8750    0.5526    0.6774       456

avg / total     0.9601    0.9625    0.9585      6393
```

```
In [52]: ### we have received the best parameters and stats .....
```