Artificial Intelligence
(CS13217)

Lab Report

Name:	Sameea Naeem
Registration #:	CSU-XS16-139
Lab Report #:	14
Submitted To:	Sir Usman Ahmed

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

Experiment # 14
Sentiment Analysis tool using Random Forest and Nave Bayes.

Objective
Understanding of Sentiment Analysis tool using Random Forest and Nave Bayes.

Software Tool
1. Python
2. Sublime text
3. Windows 10

1 Theory

Sentiment analysis is becoming a popular area of research and social media analysis, especially around user reviews and tweets. It is a special case of text mining generally focused on identifying opinion polarity, and while its often not very accurate, it can still be useful. For simplicity (and because the training data is easily accessible) Ill focus on 2 possible sentiment classifications: positive and negative. One common use of sentiment analysis is to figure out if a text expresses negative or positive feelings. Written reviews are great datasets for doing sentiment analysis, because they often come with a score that can be used to train an algorithm.

2 Task

2.1 Procedure: Task 2

```
import pandas as pd
from bs4 import BeautifulSoup
import re
```

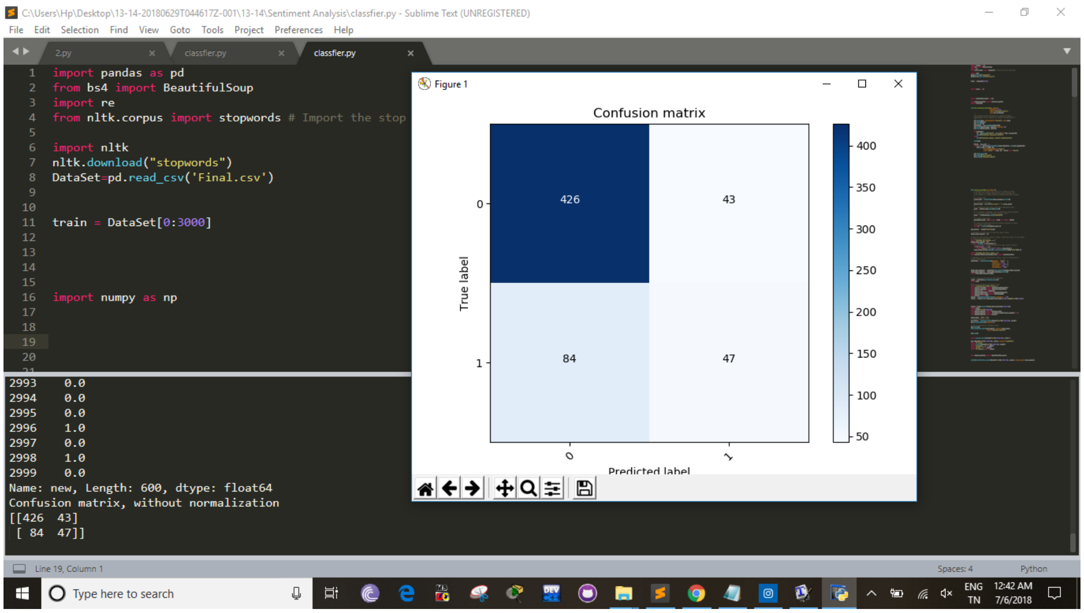

Figure 1: sentiment analysis graph

```
from nltk.corpus import stopwords # Import the stop word list

import nltk

nltk.download("stopwords")
DataSet=pd.read_csv('Final.csv')

train = DataSet[0:3000]

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
```

```
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting ‘normalize=True‘.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

def review_to_words( raw_review ):
    # Function to convert a raw review to a string of words
    # The input is a single string (a raw movie review), and
    # the output is a single string (a preprocessed movie review)
    #
    # 1. Remove HTML
    review_text = BeautifulSoup(raw_review).get_text()
    #
    # 2. Remove non-letters
    letters_only = re.sub("[^a-zA-Z]", " ", review_text)
```

```
    #
    # 3. Convert to lower case, split into individual words
    words = letters_only.lower().split()
    #
    # 4. In Python, searching a set is much faster than searching
    #   a list, so convert the stop words to a set
    stops = set(stopwords.words("english"))
    #
    # 5. Remove stop words
    meaningful_words = [w for w in words if not w in stops]
    #
    # 6. Join the words back into one string separated by space,
    # and return the result.
    return( " ".join( meaningful_words ))

num_reviews = train["Input"].size

# Initialize an empty list to hold the clean reviews
clean_train_reviews = []

# Loop over each review; create an index i that goes from 0 to the length
# of the movie review list
print "Cleaning and parsing ...\n"
clean_train_reviews = []
for i in xrange( 0, num_reviews ):
    # If the index is evenly divisible by 1000, print a message
    if( (i+1)%1000 == 0 ):
        print "Review %d of %d\n" % ( i+1, num_reviews )
    clean_train_reviews.append( review_to_words( train["Input"][i] ))

print "Creating the bag of words...\n"
from sklearn.feature_extraction.text import CountVectorizer

# Initialize the "CountVectorizer" object, which is scikit-learn's
# bag of words tool.
vectorizer = CountVectorizer(analyzer = "word",    \
                             tokenizer = None,     \
                             preprocessor = None, \
                             stop_words = None,    \
                             max_features = 5000)
```

```
train_data_features = vectorizer.fit_transform(clean_train_reviews)
train_data_features = train_data_features.toarray()
print train_data_features.shape

# Take a look at the words in the vocabulary
vocab = vectorizer.get_feature_names()
print vocab

print "Training the random forest..."
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn import tree
forest = RandomForestClassifier(n_estimators = 100)
forest = forest.fit( train_data_features[0:2400], train["new"][0:2400] )

result =forest.predict(train_data_features[2400:3000])
print result
print train["new"][2400:3000]
from sklearn.metrics import accuracy_score
from sklearn.metrics import  precision_recall_fscore_support as t
from sklearn.metrics import r2_score

class_names = ["0", "1"]
# Compute confusion matrix
cnf_matrix = confusion_matrix(train["new"][2400:3000], result)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix')

plt.show()
print accuracy_score(train["new"][2400:3000], result )

q21=t(train["new"][2400:3000], result, average='weighted')
```

```
print "R2 Score"
print r2_score(train["new"][2400:3000], result)
print 'Precision = %s '% q21[0]
print 'Recall = %s '% q21[1]
print 'F1 Measure = %s '% q21[2]

from sklearn.metrics import classification_report
print(classification_report(train["new"][2400:3000], result, target_names=
```

3 Conclusion

Successfully build graph of sentiment analysis.