######################################################################################## SCRIPT TO PREDICT SENTIMENT OF 25,000 MOVIE REVIEWS ###  
#######################################################

## READING THE FILES FROM CSV FORMAT

train = pd.read\_csv("train.csv", sep=",")

test = pd.read\_csv("test.csv", sep=",")

## WRITING FUNCTION TO PROCESS THE TEXT AND GET CLEAN WORDS AT THE END

Import re

from bs4 import BeautifulSoup

def text\_to\_words( raw\_review ):

# Function to convert a raw review to a string of words

# The input is a single string (a raw review), and

# the output is a single string (a preprocessed text)

# 1. Remove HTML using BeautifulSoup

review\_text = BeautifulSoup(raw\_review).get\_text()

#

# 2. Remove non-letters i.e. punctuations, numbers and symbols using regular expression

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 ))

### APPLYING FUNCTION ON THE RAW REVIEW TO GET PROCESSED TEXT

# Initialize an empty list to hold the clean reviews

clean\_train\_reviews = []

num\_review = train["review"].size

# Loop over each review; create an index i that goes from 0 to the length

# of the movie review list

for i in range( 0, num\_review ):

# Call our function for each one, and add the result to the list of

# clean reviews

if( (i+1)%10 == 0 ):

print ("Review %d of %d\n" % ( i+1, num\_review ))

clean\_train\_reviews.append( text\_to\_words( train["review"].iloc[i] ) )

## AFTER WE HAVE PROCESSED THE REVIEW’S WE WILL CONVERT THEM INTO NUMERICAL FORM SO ##THAT THEY CAN BE USED TO FIT A MODEL

### USING COUNT VECTORIZER TO FIT THE MODEL ON WORDS’ FREQUENCY

from sklearn.feature\_extraction.text import CountVectorizer

vectorizer = CountVectorizer(analyzer='word', Tokenizer = None, preprocessor = None, stop\_words = None, max\_features = 5000)

## fit\_transform() FIRST FITS THE MODEL AND LEARNS THE VOCABULARY AND SECOND TRANSFORM   
## OUR TRAIN DATA INTO FEATURE VECTORS

train\_data\_features = vectorizer.fit\_transform(clean\_train\_reviews)

train\_data\_features = train\_data\_features.toarray()

#### PERFORMING PREPROCESSING ON TEST DATA

# Create an empty list and append the clean reviews one by one

num\_reviews = test["review"].size

clean\_test\_reviews = []

print ("Cleaning and parsing the test set movie reviews...\n")

for i in range(0,num\_reviews):

if( (i+1) % 1000 == 0 ):

print ("Review %d of %d\n" % (i+1, num\_reviews))

clean\_review = review\_to\_words( test["review"].iloc[i] )

clean\_test\_reviews.append( clean\_review )

## USING COUNT VECTORIZER

# Get a bag of words for the test set, and convert to a numpy array

test\_data\_features = vectorizer.fit\_transform(clean\_test\_reviews)

test\_data\_features = test\_data\_features.toarray()

### APPLYING RANDOM FOREST FOR CLASSIFICATION

# Initialize a Random Forest classifier with 100 trees

forest = RandomForestClassifier(n\_estimators = 100, max\_features=2000, verbose=2)

# Fit the forest to the training set, using the bag of words as

# features and the sentiment labels as the response variable

forest = forest.fit( train\_data\_features, train["sentiment"] )

##### USING THE FOREST TO FIT TO TEST DATA TO CHECK ACCURACY

result = forest.predict(test\_data\_features)

# Copy the results to a pandas dataframe with an "id" column and

# a "sentiment" column

output = pd.DataFrame( data={"Id":test["Category1"], "Cat":result} )

output.to\_csv( "forest\_100.csv", index=False, quoting=3 )

##### ANALYZE THE PERFORMANCE OF YOUR CLASSIFIER BY CREATING A CONFUSION MATRIX

y\_actu = pd.Series(test["sentiment"], name='Actual')

y\_pred = pd.Series(result, name='Predicted')

df\_confusion = pd.crosstab(y\_actu, y\_pred)

df\_confusion

### APPLYING RANDOM FOREST FOR CLASSIFICATION

# Initialize a Random Forest classifier with 150 trees

forest = RandomForestClassifier(n\_estimators = 150, max\_features=2000, verbose=2)

# Fit the forest to the training set, using the bag of words as

# features and the sentiment labels as the response variable

forest = forest.fit( train\_data\_features, train["sentiment"] )

##### USING THE FOREST TO FIT TO TEST DATA TO CHECK ACCURACY

result = forest.predict(test\_data\_features)

# Copy the results to a pandas dataframe with an "id" column and

# a "sentiment" column

output = pd.DataFrame( data={"Id":test["Category1"], "Cat":result} )

output.to\_csv( "forest\_150.csv", index=False, quoting=3 )

##### ANALYZE THE PERFORMANCE OF YOUR CLASSIFIER BY CREATING A CONFUSION MATRIX

y\_actu = pd.Series(test["sentiment"], name='Actual')

y\_pred = pd.Series(result, name='Predicted')

df\_confusion = pd.crosstab(y\_actu, y\_pred)

df\_confusion

####################################################################################

## USING TF IDF OF DOCUMENTS TO EXTRACT FEATURES

## USING TF-IDF VECTORIZER

from sklearn.feature\_extraction.text import TfidfVectorizer

tf = TfidfVectorizer(analyzer='word', min\_df = 0, stop\_words = None, max\_features = 5000)

train\_data\_features = tf.fit\_transform(clean\_train\_reviews)

train\_data\_features = train\_data\_features.toarray()

# Get a bag of words for the test set, and convert to a numpy array

test\_data\_features = tf.fit\_transform(clean\_test\_reviews)

test\_data\_features = test\_data\_features.toarray()

### APPLYING RANDOM FOREST FOR CLASSIFICATION

# Initialize a Random Forest classifier with 100 trees

forest = RandomForestClassifier(n\_estimators = 100, max\_features=2000, verbose=2)

# Fit the forest to the training set, using the bag of words as

# features and the sentiment labels as the response variable

forest = forest.fit( train\_data\_features, train["sentiment"] )

##### USING THE FOREST TO FIT TO TEST DATA TO CHECK ACCURACY

result = forest.predict(test\_data\_features)

# Copy the results to a pandas dataframe with an "id" column and

# a "sentiment" column

output = pd.DataFrame( data={"Id":test["Category1"], "Cat":result} )

output.to\_csv( "forest\_100.csv", index=False, quoting=3 )

##### ANALYZE THE PERFORMANCE OF YOUR CLASSIFIER BY CREATING A CONFUSION MATRIX

y\_actu = pd.Series(test["sentiment"], name='Actual')

y\_pred = pd.Series(result, name='Predicted')

df\_confusion = pd.crosstab(y\_actu, y\_pred)

df\_confusion

### APPLYING RANDOM FOREST FOR CLASSIFICATION

# Initialize a Random Forest classifier with 150 trees

forest = RandomForestClassifier(n\_estimators = 150, max\_features=2000, verbose=2)

# Fit the forest to the training set, using the bag of words as

# features and the sentiment labels as the response variable

forest = forest.fit( train\_data\_features, train["sentiment"] )

##### USING THE FOREST TO FIT TO TEST DATA TO CHECK ACCURACY

result = forest.predict(test\_data\_features)

# Copy the results to a pandas dataframe with an "id" column and

# a "sentiment" column

output = pd.DataFrame( data={"Id":test["Category1"], "Cat":result} )

output.to\_csv( "forest\_150.csv", index=False, quoting=3 )

##### ANALYZE THE PERFORMANCE OF YOUR CLASSIFIER BY CREATING A CONFUSION MATRIX

y\_actu = pd.Series(test["sentiment"], name='Actual')

y\_pred = pd.Series(result, name='Predicted')

df\_confusion = pd.crosstab(y\_actu, y\_pred)

df\_confusion

############## BIGRAM VECTORIZER

bigram\_vectorizer = CountVectorizer(ngram\_range=(1, 2), token\_pattern=r'\b\w+\b', min\_df=1, max\_features = 4000)

analyze = bigram\_vectorizer.build\_analyzer()

## converting train and test data to bigram model

train\_data\_features = bigram\_vectorizer.fit\_transform (clean\_train\_reviews)

train\_data\_features = train\_data\_features.toarray()

# Get a bag of words for the test set, and convert to a numpy array

test\_data\_features = bigram\_vectorizer.fit\_transform (clean\_test\_reviews)

test\_data\_features = test\_data\_features.toarray()

### APPLYING RANDOM FOREST FOR CLASSIFICATION

# Initialize a Random Forest classifier with 150 trees

forest = RandomForestClassifier(n\_estimators = 150, max\_features=2000, verbose=2)

# Fit the forest to the training set, using the bag of words as

# features and the sentiment labels as the response variable

forest = forest.fit( train\_data\_features, train["sentiment"] )

##### USING THE FOREST TO FIT TO TEST DATA TO CHECK ACCURACY

result = forest.predict(test\_data\_features)

##### ANALYZE THE PERFORMANCE OF YOUR CLASSIFIER BY CREATING A CONFUSION MATRIX

y\_actu = pd.Series(test["sentiment"], name='Actual')

y\_pred = pd.Series(result, name='Predicted')

df\_confusion = pd.crosstab(y\_actu, y\_pred)

df\_confusion