WARNING

Please make sure to "COPY AND EDIT NOTEBOOK" to use compatible library dependencies! DO NOT CREATE A NEW NOTEBOOK AND COPY+PASTE THE CODE - this will use latest Kaggle dependencies at the time you do that, and the code will need to be modified to make it work. Also make sure internet connectivity is enabled on your notebook

Preliminaries

Write requirements to file, anytime you run it, in case you have to go back and recover Kaggle dependencies. **MOST OF THESE REQUIREMENTS WOULD NOT BE NECESSARY FOR LOCAL INSTALLATION**

Latest known such requirements are hosted for each notebook in the companion github repo, and can be pulled down and installed here if needed. Companion github repo is located at https://github.com/azunre/transfer-learning-for-nlp

```
!pip freeze > kaggle_image_requirements.txt
```

Download IMDB Movie Review Dataset

Download IMDB dataset

```
import random
import pandas as pd

## Read-in the reviews and print some basic descriptions of them

!wget -q "http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"
!tar xzf aclImdb_v1.tar.gz
```

Define Tokenization, Stop-word and Punctuation Removal Functions

Before proceeding, we must decide how many samples to draw from each class. We must also decide the maximum number of tokens per email, and the maximum length of each token. This is done by setting the following overarching hyperparameters

```
Nsamp = 1000 # number of samples to generate in each class - 'spam', 'not spam'
maxtokens = 200 # the maximum number of tokens per document
maxtokenlen = 100 # the maximum length of each token
```

Tokenization

```
def tokenize(row):
    if row is None or row == '':
        tokens = ""
    else:
        tokens = row.split(" ")[:maxtokens]
    return tokens
```

Use regular expressions to remove unnecessary characters

Next, we define a function to remove punctuation marks and other nonword characters (using regular expressions) from the emails with the help of the ubiquitous python regex library. In the same step, we truncate all tokens to hyperparameter maxtokenlen defined above.

```
import re

def reg_expressions(row):
    tokens = []
    try:
        for token in row:
            token = token.lower() # make all characters lower case
            token = re.sub(r'[\W\d]', "", token)
```

```
            token = token[:maxtokenlen] # truncate token
            tokens.append(token)
    except:
        token = ""
        tokens.append(token)
    return tokens
```

Stop-word removal

Stop-words are also removed. Stop-words are words that are very common in text but offer no useful information that can be used to classify the text. Words such as is, and, the, are are examples of stop-words. The NLTK library contains a list of 127 English stop-words and can be used to filter our tokenized strings.

```
import nltk

nltk.download('stopwords')
from nltk.corpus import stopwords
stopwords = stopwords.words('english')

# print(stopwords) # see default stopwords
# it may be beneficial to drop negation words from the removal list, as they can change the positive/negative meaning
# of a sentence
# stopwords.remove("no")
# stopwords.remove("nor")
# stopwords.remove("not")

```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
def stop_word_removal(row):
    token = [token for token in row if token not in stopwords]
    token = filter(None, token)
    return token
```

Bag-of-words model

For the computer to make inferences of the e-mails, it has to be able to interpret the text by making a numerical representation of it. One way to do this is by using something called a "bag-of-words" model. This model simply counts the frequency of word tokens for each email and thereby represents it as a vector of these counts.

** Assemble matrices function**

The `assemble_bag()` function assembles a new dataframe containing all the unique words found in the text documents. It counts the word frequency and then returns the new dataframe.

```
def assemble_bag(data):
    used_tokens = []
    all_tokens = []

    for item in data:
        for token in item:
            if token in all_tokens:
                if token not in used_tokens:
                    used_tokens.append(token)
            else:
                all_tokens.append(token)

    df = pd.DataFrame(0, index = np.arange(len(data)), columns = used_tokens)

    for i, item in enumerate(data):
        for token in item:
            if token in used_tokens:
                df.iloc[i][token] += 1
    return df
```

Putting It All Together To Assemble Dataset

Now, putting all the preprocessing steps together we assemble our dataset...

```
import os
import numpy as np

# shuffle raw data first
def unison_shuffle_data(data, header):
    p = np.random.permutation(len(header))
    data = [data[i] for i in p]  # Shuffle data as a list
    header = np.asarray(header)[p]
    return data, header

# load data in appropriate form
def load_data(path):
    data, sentiments = [], []
    for folder, sentiment in (('neg', 0), ('pos', 1)):
        folder = os.path.join(path, folder)
        for name in os.listdir(folder):
            with open(os.path.join(folder, name), 'r') as reader:
                  text = reader.read()
            text = tokenize(text)
            text = stop_word_removal(text)
            text = reg_expressions(text)
            data.append(text)
            sentiments.append(sentiment)

    data, sentiments = unison_shuffle_data(data, sentiments)
    return data, sentiments

train_path = os.path.join('aclImdb', 'train')
test_path = os.path.join('aclImdb', 'test')
raw_data, raw_header = load_data(train_path)

# Aquí ya no intentamos usar np.array con data
print(len(raw_data))  # Número total de documentos
print(len(raw_header))

```

```
25000
25000
```

```
# Subsample required number of samples
random_indices = np.random.choice(range(len(raw_header)), size=(Nsamp*2,), replace=False)
data_train = [raw_data[i] for i in random_indices]
header = raw_header[random_indices]

print("DEBUG::data_train::")
print(data_train[:5])  # Muestra las primeras 5 filas
```

```
DEBUG::data_train::
[['first', 'lets', 'agree', 'lorenzo', 'lamas', 'could', 'never', 'considered', 'skilled', 'actor', 'barely', 'even', 'decent', 'sometim
```

Display sentiments and their frequencies in the dataset, to ensure it is roughly balanced between classes

```
unique_elements, counts_elements = np.unique(header, return_counts=True)
print("Sentiments and their frequencies:")
print(unique_elements)
print(counts_elements)
```

```
Sentiments and their frequencies:
[0 1]
[1002  998]
```

Featurize and Create Labels

```
MixedBagOfReviews = assemble_bag(data_train)
# this is the list of words in our bag-of-words model
predictors = [column for column in MixedBagOfReviews.columns]

# expand default pandas display options to make emails more clearly visible when printed
pd.set_option('display.max_colwidth', 300)
```

```

MixedBagOfReviews # you could do print(MixedBagOfReviews), but Jupyter displays this nicer for pandas DataFrames
```

	first	lets		sister	br	actor	acting	lamas	never	the	...	sox	fanbr	uncanny	pupils	hopelessly	brainy	raving	flashy
0	2	2	2	2	3	2	2	2	2	2	...	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	1	1	...	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
4	0	0	0	0	2	0	0	0	1	2	...	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	2	0	4	0	0	0	0	0	0	4	...	0	0	0	0	0	0	0	0
1996	0	0	6	0	6	0	1	0	0	1	...	0	0	0	0	0	0	0	0
1997	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0
1998	0	0	0	0	1	0	0	0	0	2	...	2	1	1	1	1	1	0	0
1999	1	0	0	0	0	0	1	0	0	2	...	0	0	0	0	0	0	1	1

2000 rows × 11588 columns

```
# split into independent 70% training and 30% testing sets
data = MixedBagOfReviews.values

idx = int(0.7*data.shape[0])

# 70% of data for training
train_x = data[:idx,:]
train_y = header[:idx]
# remaining 30% for testing
test_x = data[idx:,:]
test_y = header[idx:]

print("train_x/train_y list details, to make sure it is of the right form:")
print(len(train_x))
print(train_x)
print(train_y[:5])
print(len(train_y))
```

```
train_x/train_y list details, to make sure it is of the right form:
1400
[[2 2 2 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 ...
 [0 0 1 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
[0 0 0 0 1]
1400
```

How about other vectorization strategies?

We present other vectorization strategies below, for readers who are interested in exploring them...

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, HashingVectorizer

# create the transform - uncomment the one you want to focus on
# vectorizer = CountVectorizer() # this is equivalent to the bag of words
vectorizer = TfidfVectorizer() # tf-idf vectorizer
# vectorizer = HashingVectorizer(n_features=3000) # hashing vectorizer
```

```
# build vocabulary
vectorizer.fit([' '.join(sublst) for sublst in data_train])
# summarize
print(len(vectorizer.vocabulary_))
#print(vectorizer.idf_)
# encode one document
vector = vectorizer.transform([' '.join(data_train[0])])
# summarize encoded vector
```

```
# summarize encoded vector
print(vector.shape)
print(vector.toarray())

USE = False # set this to 'True' if you want to use the vectorizer featurizers instead of the bag-of-words done before
if(USE):
    data = vectorizer.transform([' '.join(sublst) for sublst in data_train]).toarray()
    # 70% of data for training
    train_x = data[:idx,:]
    # remaining 30% for testing
    test_x = data[idx:,:]

    print("train_x/train_y list details, to make sure it is of the right form:")
    print(train_x.shape[0])
    print(train_x)
    print(train_y[:5])
    print(len(train_y))
    predictors = [column for column in vectorizer.vocabulary_]
```

```
24762
(1, 24762)
[[0. 0. 0. ... 0. 0. 0.]]
```

Logistic Regression Classifier

```
from sklearn.linear_model import LogisticRegression

def fit(train_x,train_y):
    model = LogisticRegression()

    try:
        model.fit(train_x, train_y)
    except:
        pass
    return model

model = fit(train_x,train_y)
```

```
predicted_labels = model.predict(test_x)

# print all labels for full trasparency
print("DEBUG::The logistic regression predicted labels are::")
print(predicted_labels)
```

```
DEBUG::The logistic regression predicted labels are::
[0 1 1 1 0 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1 0 0 1 0 1 0 0 1 1 1 1 1 1 0 0 1 1
 0 1 0 1 1 0 0 1 1 0 0 1 0 1 1 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 0 1 1 0 1 0 0
 0 0 1 1 0 1 0 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 0 0 1 0 1 0 1 1 1 1 0 0 1
 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0 0 0 0 1 0 1 0
 0 1 0 0 0 0 1 1 1 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0 0 0 1
 1 0 1 1 1 1 0 0 0 0 1 0 1 1 0 1 1 0 0 0 1 1 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0
 1 1 0 0 0 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 0 0 0 0 1 1 1 1
 1 1 1 1 1 1 1 0 1 0 1 0 1 1 0 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0 0 0 0
 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 0 1 0 1 0
 1 0 0 0 1 1 0 0 1 1 0 0 1 1 1 0 0 0 1 0 1 1 0 1 0 0 0 1 1 1 0 1 0 1 0 0 0
 1 0 1 0 1 1 1 1 0 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 1 1 0 0 0 0 1 0 0 1 0
 0 0 1 1 0 0 1 1 0 1 0 0 1 0 1 1 0 1 1 1 1 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 0
 1 0 1 0 1 0 0 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1
 0 0 0 1 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 1 1 1 0 0
 0 0 1 1 1 1 1 0 0 1 0 0 0 0 1 0 0 1 0 1 1 1 1 0 0 1 0 0 1 1 0 0 1 1 1 1 1
 0 1 1 1 1 1 0 0 1 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
 1 0 1 1 1 1 1 0]
```

```
from sklearn.metrics import accuracy_score

acc_score = accuracy_score(test_y, predicted_labels)

print("The logistic regression accuracy score is::")
print(acc_score)
```

```
The logistic regression accuracy score is::
0.7866666666666666
```

Support Vector Machine Classifier

```
import time
from sklearn.svm import SVC # Support Vector Classification model
```

```
# Create a support vector classifier
clf = SVC(C=1, gamma="auto", kernel='linear',probability=False)

# Fit the classifier using the training data
start_time = time.time()
clf.fit(train_x, train_y)
end_time = time.time()
print("Training the SVC Classifier took %3d seconds"%(end_time-start_time))

# test and evaluate
predicted_labels = clf.predict(test_x)
print("DEBUG::The SVC Classifier predicted labels are::")
print(predicted_labels)

acc_score = accuracy_score(test_y, predicted_labels)
print("The SVC Classifier testing accuracy score is::")
print(acc_score)
```

```
Training the SVC Classifier took  22 seconds
DEBUG::The SVC Classifier predicted labels are::
[1 1 1 1 0 0 0 1 1 0 1 0 1 1 1 1 0 1 1 1 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 1 1
 0 0 0 1 1 0 0 1 1 0 0 1 0 0 1 1 1 0 1 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 1 0 0
 0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 0 0 1 0 1 0 1 0 1 1 0 1 1
 0 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0
 0 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0 0 0 1
 1 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 0 0 0 1 1 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0
 1 1 0 0 0 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 0 0 0 0 0 1 1 1 1
 1 1 1 1 1 1 1 0 1 0 1 0 0 1 0 1 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 0 0 1 0 0
 1 1 0 1 0 0 0 0 1 0 1 0 0 0 1 1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0 0 1 0 1 0
 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 1 0 1 0 0 0 1 1 1 0 1 1 1 0 0 0
 1 0 1 0 1 1 1 1 0 0 1 0 0 0 1 1 1 0 0 1 1 1 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0
 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 1 0 1 1 1 1 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 0
 1 0 1 0 1 0 0 1 1 0 0 1 0 0 0 1 1 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1
 0 0 0 1 0 1 1 1 1 0 1 0 1 0 0 0 1 1 0 0 0 0 1 1 0 1 0 1 1 0 1 0 1 1 1 0 0
 0 0 1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 0 1 1 0 0 1 1 1 1 1
 0 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 1 1 0 1 0 0 0 0 0
 1 0 0 1 1 1 0 0]
The SVC Classifier testing accuracy score is::
0.7533333333333333
```

Random Forests

```
# Load scikit's random forest classifier library
from sklearn.ensemble import RandomForestClassifier

# Create a random forest Classifier. By convention, clf means 'Classifier'
clf = RandomForestClassifier(n_jobs=1, random_state=0)

# Train the Classifier to take the training features and learn how they relate
# to the training y (spam, not spam?)
start_time = time.time()
clf.fit(train_x, train_y)
end_time = time.time()
print("Training the Random Forest Classifier took %3d seconds"%(end_time-start_time))

predicted_labels = clf.predict(test_x)
print("DEBUG::The RF predicted labels are::")
print(predicted_labels)

acc_score = accuracy_score(test_y, predicted_labels)

print("DEBUG::The RF testing accuracy score is::")
print(acc_score)
```

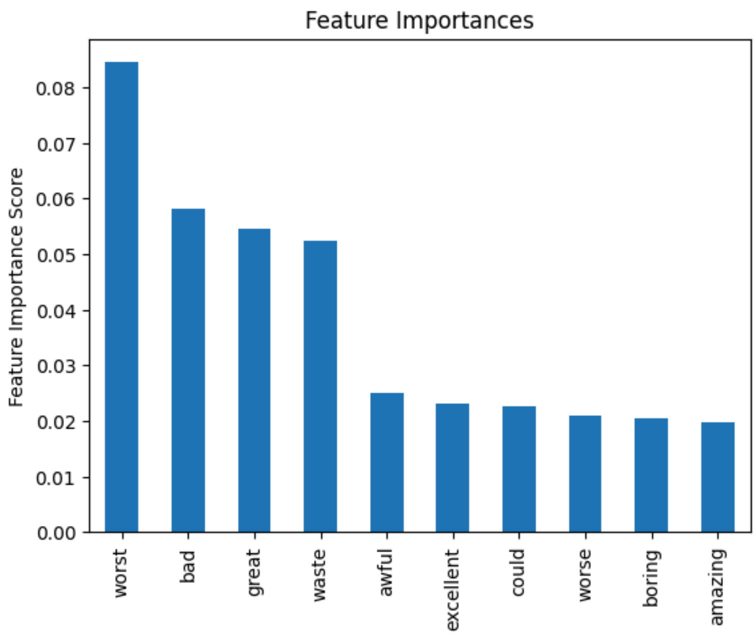
```
Training the Random Forest Classifier took   4 seconds
DEBUG::The RF predicted labels are::
[0 1 1 1 1 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1 0 0 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1
 0 0 0 0 1 0 0 1 1 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0
```

```
 1 0 1 1 1 0 0 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1 0 1 1 1 0 1 1 0 1 0 1 1
 0 0 1 1 1 1 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 1 1
 1 1 0 0 1 1 1 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0 0 1 1
 1 0 0 1 0 1 0 0 0 0 1 0 1 1 0 1 1 0 0 0 1 1 0 1 1 1 1 1 0 0 0 0 1 1 0 0 0
 1 0 0 0 0 1 1 1 1 1 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 0 1 1 0 0 0 0 1 1 1 0
 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 1 0 0 1 1 0 0 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0
 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 1 1 0 1 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 1
 1 0 0 1 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 1 0 1 1 1 0 1 1 1 0 1 0 1 0 0 1
 1 0 1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 0 0 1 1 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0
 0 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0 1 0 1 1 1 0 1 0 0 0 0 1 0 0
 1 0 1 0 0 0 0 1 1 1 0 1 0 0 1 1 0 1 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1
 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 1 1 1 1 0
 0 0 1 1 1 1 1 0 0 1 0 0 0 1 1 0 0 1 0 1 1 1 1 0 0 1 1 0 1 1 0 0 1 1 1 1 1
 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0
 1 1 1 0 1 1 1 0]
DEBUG::The RF testing accuracy score is::
0.7816666666666666
```

Gradient Boosting Machines

```
from sklearn.ensemble import GradientBoostingClassifier  # GBM algorithm
from sklearn import metrics   #Additional scklearn functions
from sklearn.model_selection import cross_val_score, GridSearchCV

def modelfit(alg, train_x, train_y, predictors, test_x, performCV=True, printFeatureImportance=True, cv_folds=5):
    #Fit the algorithm on the data
    alg.fit(train_x, train_y)

    #Predict training set:
    predictions = alg.predict(train_x)
    predprob = alg.predict_proba(train_x)[:,1]

    #Perform cross-validation:
    if performCV:
        cv_score = cross_val_score(alg, train_x, train_y, cv=cv_folds, scoring='roc_auc')

    #Print model report:
    print("\nModel Report")
    print("Accuracy : %.4g" % metrics.accuracy_score(train_y,predictions))
    print("AUC Score (Train): %f" % metrics.roc_auc_score(train_y, predprob))

    if performCV:
        print("CV Score : Mean - %.7g | Std - %.7g | Min - %.7g | Max - %.7g" % (np.mean(cv_score),np.std(cv_score),np.min(cv_score),np.m

    #Print Feature Importance:
    import matplotlib.pyplot as plt

    if printFeatureImportance:
        fig,ax = plt.subplots()
        feat_imp = pd.Series(alg.feature_importances_, predictors).sort_values(ascending=False)
        feat_imp[:10].plot(kind='bar', title='Feature Importances',ax=ax)
        plt.ylabel('Feature Importance Score')

        fig.savefig('GBMimportances.eps', format='eps',bbox_inches='tight')
        fig.savefig('GBMimportances.pdf', format='pdf',bbox_inches='tight')
        fig.savefig('GBMimportances.png', format='png',bbox_inches='tight')
        fig.savefig('GBMimportances.svg', format='svg',bbox_inches='tight')

    return alg.predict(test_x)

gbm = GradientBoostingClassifier(random_state=10)

start_time = time.time()
test_predictions = modelfit(gbm, train_x, train_y, predictors, test_x)
end_time = time.time()
print("Training the Gradient Boosting Classifier took %3d seconds"%(end_time-start_time))

predicted_labels = test_predictions
print("DEBUG::The Gradient Boosting predicted labels are::")
print(predicted_labels)

acc_score = accuracy_score(test_y, predicted_labels)

print("DEBUG::The Gradient Boosting testing accuracy score is::")
print(acc_score)
```

```
Model Report
Accuracy : 0.9264
AUC Score (Train): 0.983428
CV Score : Mean - 0.8419613 | Std - 0.01045559 | Min - 0.8242002 | Max - 0.8566508
Training the Gradient Boosting Classifier took 292 seconds
DEBUG::The Gradient Boosting predicted labels are::
[0 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 1
 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 1 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 1 0 1
 1 0 1 1 1 1 0 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1
 0 0 1 1 1 1 0 0 1 0 1 1 0 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 1 0 0 0 1 0 1 1
 1 1 0 0 0 1 0 1 1 1 1 0 0 0 1 1 1 0 1 0 1 0 0 1 0 0 0 0 1 0 1 1 1 0 0 1 1
 1 1 1 1 0 1 0 1 0 0 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 0 0
 1 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 1 0 1 0
 1 1 1 1 0 1 0 0 1 1 1 1 1 1 0 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
 0 1 0 1 1 0 0 0 1 1 1 0 0 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 1 0 0 0 0 0 1 1
 1 0 0 1 1 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1 1 1 1 0 1 0 1 1 0 0
 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0
 0 1 0 1 0 0 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0 1 0 1 1 1 0 1 0 0 0 0 1 1 0
 1 0 1 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 0 1 0 1 0 0 1
 1 0 0 1 0 0 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0 1 1 1 1 0
 0 0 1 0 1 1 1 0 0 1 0 0 0 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1
 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0 1 0 1
 1 1 1 1 0 1 1 0]
DEBUG::The Gradient Boosting testing accuracy score is::
0.76
```



Make figures downloadable to local system in interactive mode

```
from IPython.display import HTML
import base64

def create_download_link(file_path, title="Download file"):
    # Open the file in binary mode
    with open(file_path, "rb") as f:
        data = f.read()

    # Convert the binary data to a base64 encoded string
    b64 = base64.b64encode(data).decode()

    # Create a download link
    html = f'<a download="{file_path}" href="data:application/octet-stream;base64,{b64}">{title}</a>'
    return HTML(html)

# Uso con el archivo que deseas descargar
create_download_link('GBMimportances.svg')

```

```
# you must remove all downloaded files - having too many of them on completion will make Kaggle reject your notebook
!rm -rf aclImdb
!rm aclImdb_v1.tar.gz
```