Project Report - Sentiment Analysis

IMDb MOVIE REVIEWS

Meghana Joshi

July 2020

OBJECTIVE

With the availability of a huge volume of online review data (Amazon, Imdb, etc.), sentiment analysis plays a major role. Sentiment analysis refers to the use of natural language processing and text analysis to systematically identify and classify subjective information.

The main purpose of this project is to build a classifier to determine whether a review is associated with a positive or a negative sentiment. Additionally, the model can be used to determine words that are most strongly associated with the positive or negative reviews.

PROJECT COMPONENTS

- Load Data in Python
- Data Cleansing and Pre-processing
    - Convert reviews to lowercase
    - Remove Punctuations
    - Remove numeric values
    - Remove all English Stop Words
    - Stemming
    - Lemmatization
- Data Representation
    - Count Vectorization
    - Word Count Vectorization
    - TF - IDF
- Building the Classifier
    - Logistic Regressions
    - Support Vector Machines
- Determining positive and negative coefficients
- Visualizing Results - accuracy, precision etc

ENVIRONMENT

Language: Python 3

Libraries: Scikit learn, Pandas, NumPy, NLTK, Seaborn, matplotlib

DATASET OVERVIEW

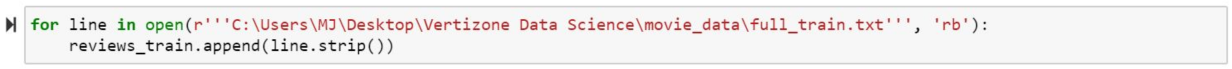
Dataset: Large Movie Review Dataset (http://ai.stanford.edu/~amaas/data/sentiment/)

- This dataset consists of 50,000 movie reviews from IMDb.
- It is evenly split with 25,000 reviews intended for training, and the remaining for testing.
- The numbers of positive and negative reviews are equal. (Within each set, 12,500 are positive and 12,500 are negative reviews).
- Negative reviews have scores less or equal than 4 out of 10 while a positive review have scores greater or equal than 7 out of 10.
- Neutral reviews are not included.

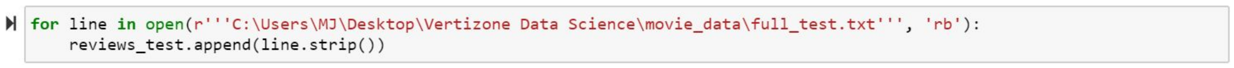
PROCESS

LOADING DATA INTO PYTHON

In order to perform analysis on the reviews, the testing and training data are read into two separate lists.



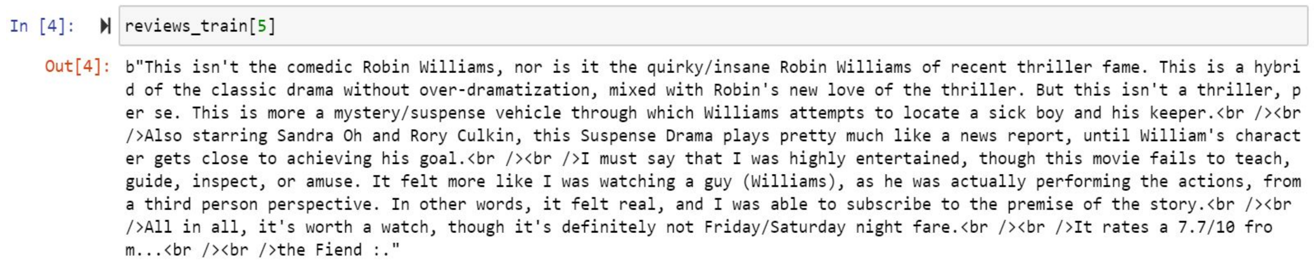
```
for line in open(r'''C:\Users\MJ\Desktop\Vertizone Data Science\movie_data\full_train.txt''', 'rb'):
    reviews_train.append(line.strip())
```



```
for line in open(r'''C:\Users\MJ\Desktop\Vertizone Data Science\movie_data\full_test.txt''', 'rb'):
    reviews_test.append(line.strip())
```

DATA CLEANSING AND PRE-PROCESSING

Consider the below review:



```
In [4]:  reviews_train[5]

Out[4]: b"This isn't the comedic Robin Williams, nor is it the quirky/insane Robin Williams of recent thriller fame. This is a hybri
        d of the classic drama without over-dramatization, mixed with Robin's new love of the thriller. But this isn't a thriller, p
        er se. This is more a mystery/suspense vehicle through which Williams attempts to locate a sick boy and his keeper.<br /><br
        />Also starring Sandra Oh and Rory Culkin, this Suspense Drama plays pretty much like a news report, until William's charact
        er gets close to achieving his goal.<br /><br />I must say that I was highly entertained, though this movie fails to teach,
        guide, inspect, or amuse. It felt more like I was watching a guy (Williams), as he was actually performing the actions, from
        a third person perspective. In other words, it felt real, and I was able to subscribe to the premise of the story.<br /><br
        />All in all, it's worth a watch, though it's definitely not Friday/Saturday night fare.<br /><br />It rates a 7.7/10 fro
        m...<br /><br />the Fiend :."
```

1. **Convert all reviews to lowercase:**

   Converting to lowercase simplifies the further analysis process by introducing uniformity and will help in the steps like removing stop words.

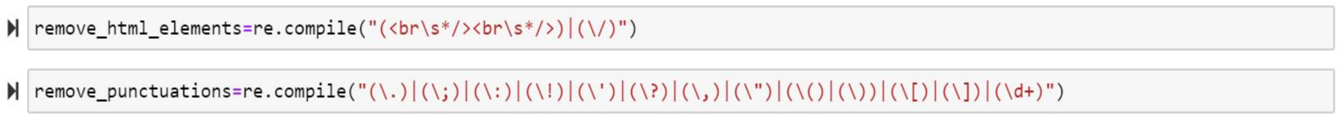
2. **Remove punctuations and digits**

   Since the raw data is pretty messy for these reviews, it is required to remove punctuations and digits. They also do not offer any significance for classification, and hence removing them will not affect the data.

3. **Remove HTML elements**

   Some HTML elements like <br /> also have been included in the data. These can be removed, as they serve no specific purpose.

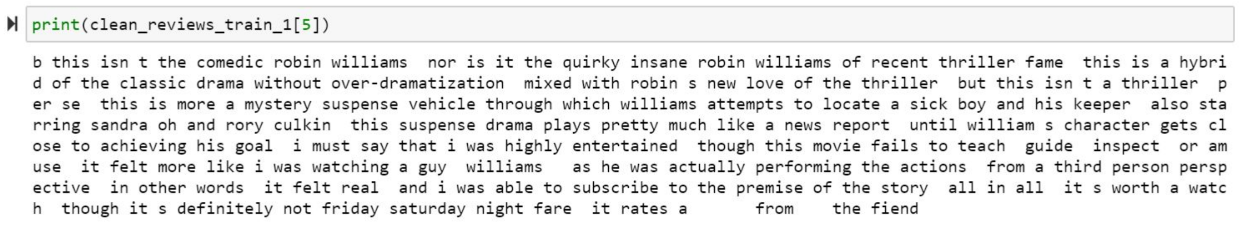
Steps 1,2,3 are carried out using regular expressions (regex) in Python as shown:



```
remove_html_elements=re.compile("(<br\s*/><br\s*/>)|(\/)")
```

```
remove_punctuations=re.compile("(\.)|(\;)|(\:)|(\!)|(\')|(\?)|(\,)|(\")|(\()|(\))|(\[)|(\])|(\d+)")
```

The previous review now looks like this:



```
print(clean_reviews_train_1[5])
```

```
b this isn t the comedic robin williams  nor is it the quirky insane robin williams of recent thriller fame  this is a hybri
d of the classic drama without over-dramatization  mixed with robin s new love of the thriller  but this isn t a thriller  p
er se  this is more a mystery suspense vehicle through which williams attempts to locate a sick boy and his keeper  also sta
rring sandra oh and rory culkin  this suspense drama plays pretty much like a news report  until william s character gets cl
ose to achieving his goal  i must say that i was highly entertained  though this movie fails to teach  guide  inspect  or am
use  it felt more like i was watching a guy  williams   as he was actually performing the actions  from a third person persp
ective  in other words  it felt real  and i was able to subscribe to the premise of the story  all in all  it s worth a watc
h  though it s definitely not friday saturday night fare  it rates a       from    the fiend
```

4. Remove English Stop Words

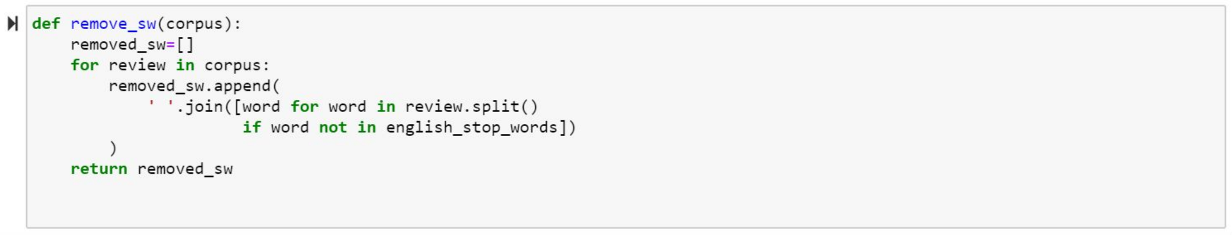
Stop words are the ordinary words like ‘he’, ‘she’, ‘we’, etc., that occur frequently in English sentences. These words can be removed without altering the basic sentiment associated with the review. Furthermore, these words take up space and add up to processing time, which can be saved.

Library used: NLTK (Natural Language Toolkit)

The NLTK library is one of the most common libraries in Python used for stop word removal. A list of the stop words can be found in the corpus module. In order to delete the stop words, text can be tokenized and a word can be removed if it belongs to the list of stopwords belonging to NLTK.

First, a collection of stopwords has to be imported from the nltk.corpus module.

The code snippet below is used to split the reviews into words and then remove all stop words.



```python
def remove_sw(corpus):
    removed_sw=[]
    for review in corpus:
        removed_sw.append(
            ' '.join([word for word in review.split()
                      if word not in english_stop_words])
        )
    return removed_sw
```

After execution, the previous review looks like this:

```
print(clean_reviews_train_2[5])
```

```
b comedic robin williams quirky insane robin williams recent thriller fame hybrid classic drama without over-dramatization m
ixed robin new love thriller thriller per se mystery suspense vehicle williams attempts locate sick boy keeper also starring
sandra oh rory culkin suspense drama plays pretty much like news report william character gets close achieving goal must say
highly entertained though movie fails teach guide inspect amuse felt like watching guy williams actually performing actions
third person perspective words felt real able subscribe premise story worth watch though definitely friday saturday night fa
re rates fiend
```

5. Stemming

Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form. Stemming is used extensively for information retrieval systems in search engines and also in domain analysis.

Library used: NLTK (Natural Language Toolkit)

Porter Stemmer:

This is a relatively older stemming algorithm and it is mainly concerned with removing the common endings to words so that they can be resolved to a common form. It is generally regarded as a basic and simple stemming algorithm and is the least aggressive in comparison with Lancaster Stemmer, and Snowball Stemmer.

```
from nltk.stem.porter import PorterStemmer
```

```
def stem(corpus):
    stemmer = PorterStemmer()
    return [' '.join([stemmer.stem(word) for word in review.split()]) for review in corpus]
```

Once stemming is executed, the previous review looks like this:

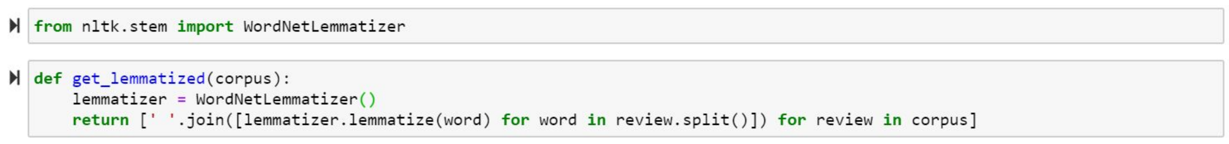
```
print(stemmed_reviews_train[5])
```

```
b comed robin william quirki insan robin william recent thriller fame hybrid classic drama without over-dramat mix robin new
love thriller thriller per se mysteri suspens vehicl william attempt locat sick boy keeper also star sandra oh rori culkin s
uspens drama play pretti much like news report william charact get close achiev goal must say highli entertain though movi f
ail teach guid inspect amus felt like watch guy william actual perform action third person perspect word felt real abl subsc
rib premis stori worth watch though definit friday saturday night fare rate fiend
```

6. Lemmatization

Lemmatization is concerned with resolving words to their dictionary form. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meaning to one word.

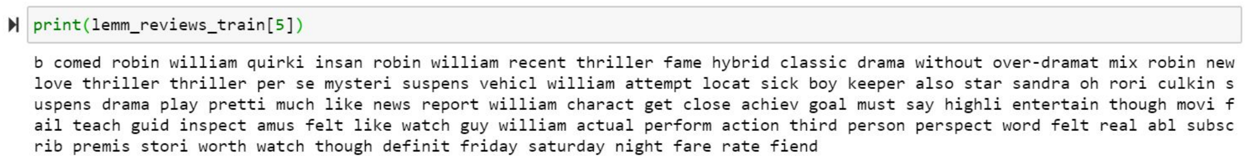
Library used: NLTK (Natural Language Toolkit)



```python
from nltk.stem import WordNetLemmatizer
```

```python
def get_lemmatized(corpus):
    lemmatizer = WordNetLemmatizer()
    return [' '.join([lemmatizer.lemmatize(word) for word in review.split()]) for review in corpus]
```

Once lemmatization is executed, the review looks like:



```python
print(lemm_reviews_train[5])
```

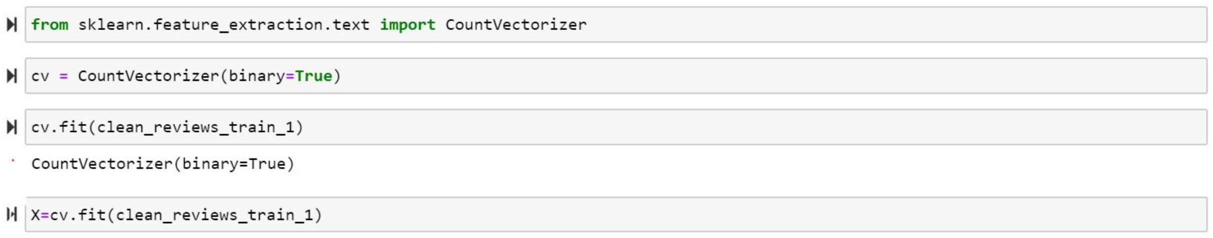
```
b comed robin william quirki insan robin william recent thriller fame hybrid classic drama without over-dramat mix robin new
love thriller thriller per se mysteri suspens vehicl william attempt locat sick boy keeper also star sandra oh rori culkin s
uspens drama play pretti much like news report william charact get close achiev goal must say highli entertain though movi f
ail teach guid inspect amus felt like watch guy william actual perform action third person perspect word felt real abl subsc
rib premis stori worth watch though definit friday saturday night fare rate fiend
```

DATA REPRESENTATION

In order to ensure that the code is computationally efficient, it is important to use vectorization. To process natural language text and to extract useful information from the given text, it needs to be converted into a set of real numbers (vectors) - word embeddings. Word Embeddings or Word vectorization is a methodology in NLP to map words or phrases from vocabulary to a corresponding vector of real numbers which used to find word predictions, word similarities/semantics.

1. Count Vectorization (One-hot encoding)

This technique converts a collection of text documents to a matrix of token counts. It creates a vector that has as many dimensions as the corpora has unique words. Each unique word has a unique dimension and will be represented by a 1 in that dimension with 0s everywhere else. This process is also known as one hot encoding.



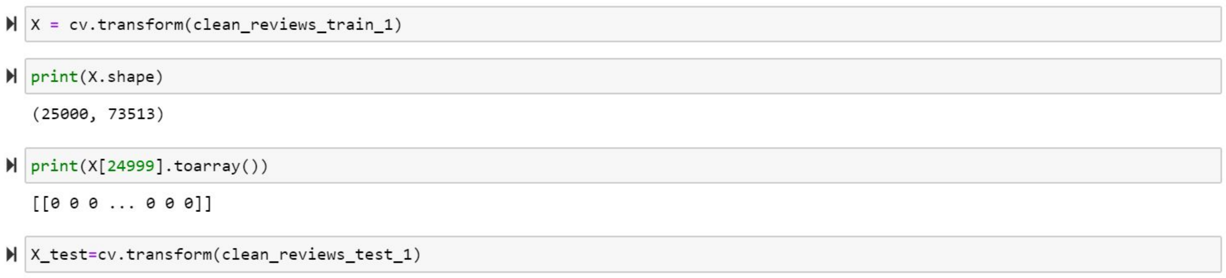
```
from sklearn.feature_extraction.text import CountVectorizer
```

```
cv = CountVectorizer(binary=True)
```

```
cv.fit(clean_reviews_train_1)
```

```
CountVectorizer(binary=True)
```

```
X=cv.fit(clean_reviews_train_1)
```



```
X = cv.transform(clean_reviews_train_1)
```

```
print(X.shape)
```

```
(25000, 73513)
```

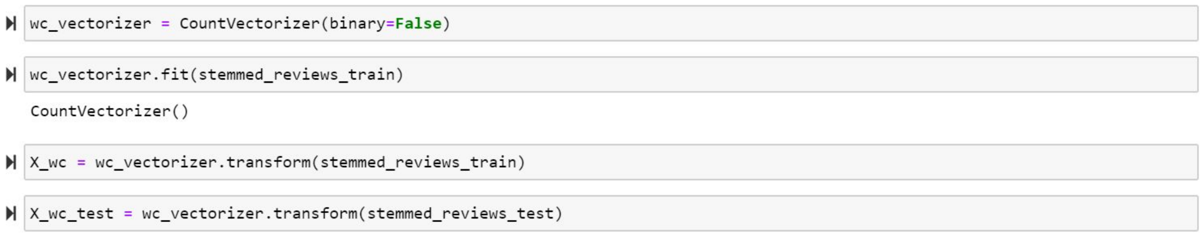
```
print(X[24999].toarray())
```

```
[[0 0 0 ... 0 0 0]]
```

```
X_test=cv.transform(clean_reviews_test_1)
```

2. Word Counts

Instead of simply noting whether or not a word appears, this approach takes the number of occurrences into account. This method can help to give the sentiment classifier more predictive power. For example, if a certain positive word occurs frequently in a review, then it is considerably more probable that the review is associated with a positive sentiment overall.



```
wc_vectorizer = CountVectorizer(binary=False)
```

```
wc_vectorizer.fit(stemmed_reviews_train)
```

```
CountVectorizer()
```

```
X_wc = wc_vectorizer.transform(stemmed_reviews_train)
```

```
X_wc_test = wc_vectorizer.transform(stemmed_reviews_test)
```

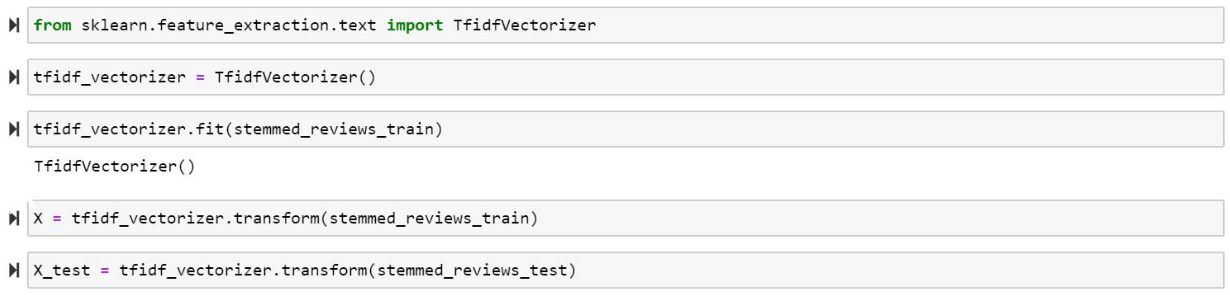
3. TF-IDF (term frequency - inverse document frequency)

tf-idf aims to represent the number of times a given word appears in a document (a movie review in our case) relative to the number of documents in the corpus that the word appears in — where words that appear in many documents have a value closer to zero and words that appear in less documents have values closer to 1.

TF-IDF for a word in a document is calculated by multiplying two different metrics:

- The term frequency of a word in a document. There are several ways of calculating this frequency, with the simplest being a raw count of instances a word appears in a document. Then, there are ways to adjust the frequency, by length of a document, or by the raw frequency of the most frequent word in a document.
- The inverse document frequency of the word across a set of documents. This means, how common or rare a word is in the entire document set. The closer it is to 0, the more common a word is. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm.
- If the word is very common and appears in many documents, this number will approach 0. Otherwise, it will approach 1.

Multiplying these two numbers results in the TF-IDF score of a word in a document. The higher the score, the more relevant that word is in that particular document.



```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tfidf_vectorizer = TfidfVectorizer()
```

```
tfidf_vectorizer.fit(stemmed_reviews_train)
```

```
TfidfVectorizer()
```

```
X = tfidf_vectorizer.transform(stemmed_reviews_train)
```

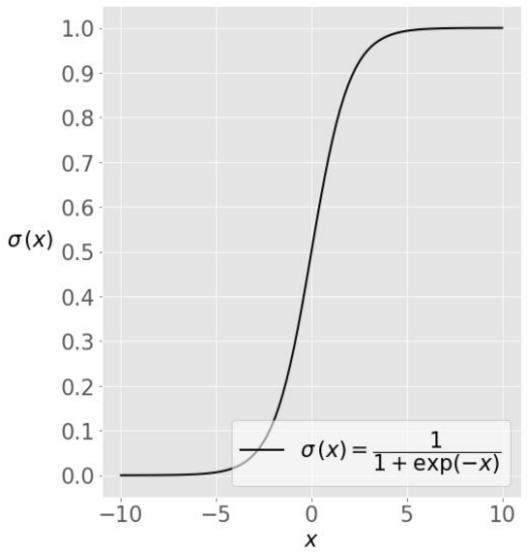
```
X_test = tfidf_vectorizer.transform(stemmed_reviews_test)
```

SENTIMENT CLASSIFIER

Once the dataset is transformed into a format suitable for modeling, the next step is to build the classifier. It is suggested to try out a few different models and test for accuracy in order to gain more predictive power.

1. Logistic Regressions

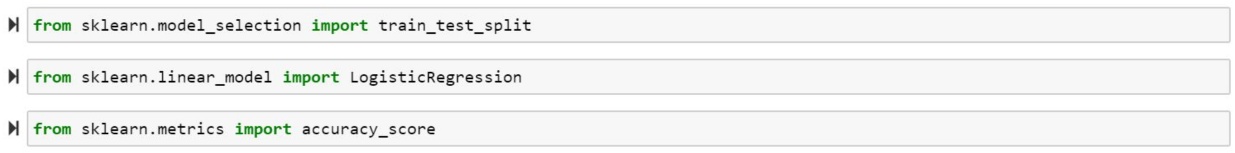
Logistic regression is a fundamental classification technique. It belongs to the group of linear classifiers and is somewhat similar to polynomial and linear regression. Logistic regression is fast and relatively uncomplicated.



The sigmoid function has values very close to either 0 or 1 across most of its domain. This fact makes it suitable for application in classification methods.

Library used: Sklearn

- The following need to be imported from sklearn in order to carry out and determine accuracy of the logistic regression approach - train_test_split, LogisticRegression and accuracy_score.



```
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import accuracy_score
```

- In order to comprehend model performance, the strategy is to divide the dataset into training sets and a testing set. The dataset is split using train_test_split.

```
X2_train, X2_val, Y2_train, Y2_val = train_test_split(
    X2, target, train_size = 0.7
)
```

- Then, fit the model on the train set using fit() and perform prediction on the test set using predict().

  - Logistic Regressions with CountVectorizer

```
for n in [0.01, 0.05,0.1, 0.25, 0.5,0.75, 1]:
    lr = LogisticRegression(C=n)
    lr.fit(X2_train, Y2_train)
    print ("Accuracy for C=%s: %s"
           % (n, accuracy_score(Y2_val, lr.predict(X2_val))))
```

```
Accuracy for C=0.01: 0.87808
Accuracy for C=0.05: 0.88736
Accuracy for C=0.1: 0.88608
Accuracy for C=0.25: 0.88256
```

Here, we are able to achieve an accuracy of 88.7% with hyperparameter c=0.05.

  - Logistic Regressions with TF IDF

```
for c in [0.01, 0.05, 0.25, 0.5, 1]:

    lr = LogisticRegression(C=c)
    lr.fit(X_train, y_train)
    print ("Accuracy for C=%s: %s"
           % (c, accuracy_score(y_val, lr.predict(X_val))))
```

```
Accuracy for C=0.01: 0.8282
Accuracy for C=0.05: 0.8566
Accuracy for C=0.25: 0.8828
Accuracy for C=0.5: 0.891
Accuracy for C=1: 0.896
```

Here, we are able to achieve accuracy of 89.6% with hyperparameter as c=1.

  - Logistic Regressions with Word Count

```
for c in [0.01, 0.05, 0.1, 0.25, 0.5, 1]:
    lr = LogisticRegression(C=c)
    lr.fit(X_train, y_train)
    print ("Accuracy for C=%s: %s"
           % (c, accuracy_score(y_val, lr.predict(X_val))))
```

```
Accuracy for C=0.01: 0.8802666666666666

Accuracy for C=0.1: 0.8890666666666667
```

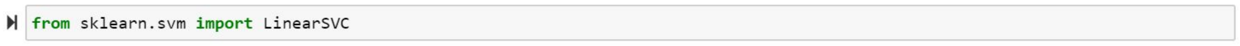
Here, we are able to achieve accuracy of 88.9% with hyperparameter as c=0.1.

2. Support Vector Machines

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points. The goal is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Library used: Sklearn

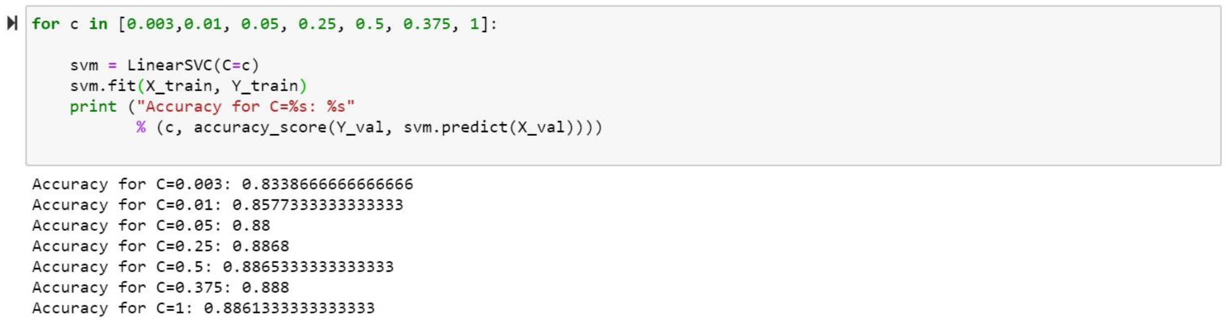
- In addition to train_test_split and accuracy_score, we also need to import Linear SVC in order to use the support vector machine algorithm.



```
from sklearn.svm import LinearSVC
```

- Similar to Logistic Regressions, the dataset is split into a training set and a testing set using train_test_split.
- Next, fit the model on the train set using fit() and perform prediction on the test set using predict().

    - SVM with CountVectorizer



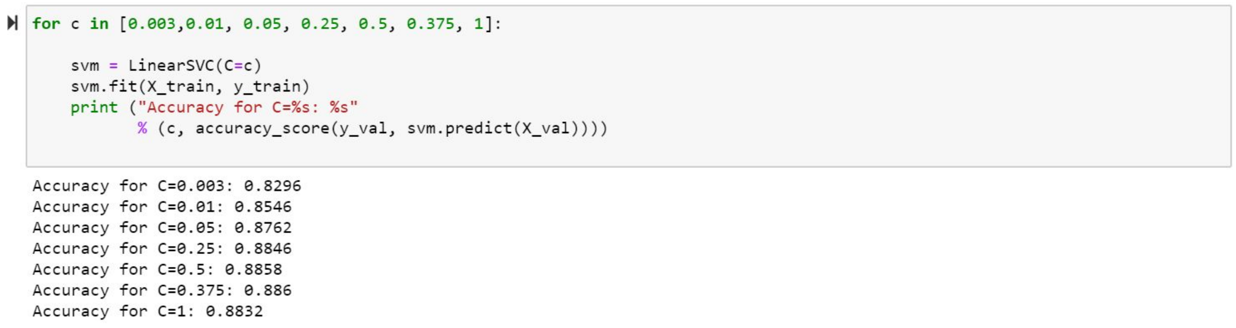
```
for c in [0.003,0.01, 0.05, 0.25, 0.5, 0.375, 1]:

    svm = LinearSVC(C=c)
    svm.fit(X_train, Y_train)
    print ("Accuracy for C=%s: %s"
           % (c, accuracy_score(Y_val, svm.predict(X_val))))
```

```
Accuracy for C=0.003: 0.8338666666666666
Accuracy for C=0.01: 0.8577333333333333
Accuracy for C=0.05: 0.88
Accuracy for C=0.25: 0.8868
Accuracy for C=0.5: 0.8865333333333333
Accuracy for C=0.375: 0.888
Accuracy for C=1: 0.8861333333333333
```

Here, we are able to achieve an accuracy of 88.8% with hyperparameter c= 0.375.

- SVM with TF IDF



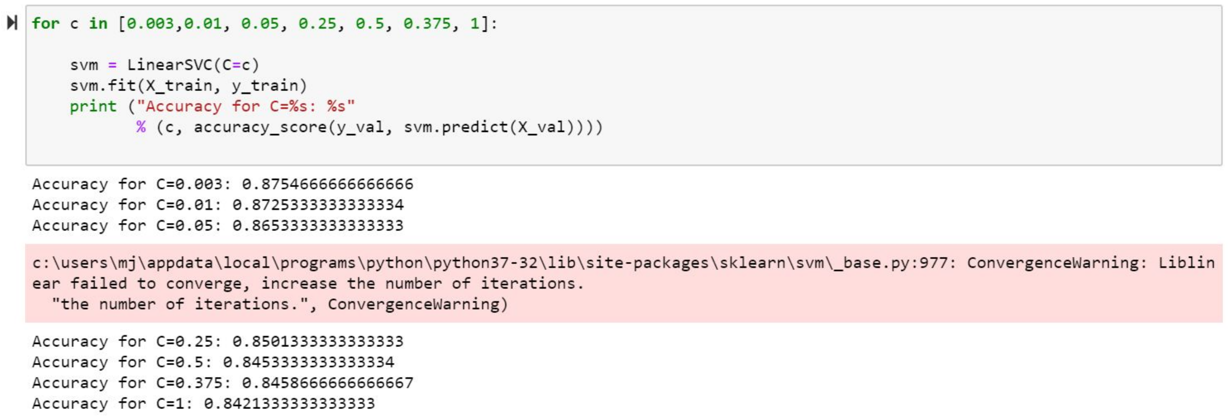
```
for c in [0.003,0.01, 0.05, 0.25, 0.5, 0.375, 1]:

    svm = LinearSVC(C=c)
    svm.fit(X_train, y_train)
    print ("Accuracy for C=%s: %s"
           % (c, accuracy_score(y_val, svm.predict(X_val))))
```

```
Accuracy for C=0.003: 0.8296
Accuracy for C=0.01: 0.8546
Accuracy for C=0.05: 0.8762
Accuracy for C=0.25: 0.8846
Accuracy for C=0.5: 0.8858
Accuracy for C=0.375: 0.886
Accuracy for C=1: 0.8832
```

Here, we are able to achieve an accuracy of 88.6% with hyperparameter c= 0.375.

- SVM with Word Count



```
for c in [0.003,0.01, 0.05, 0.25, 0.5, 0.375, 1]:

    svm = LinearSVC(C=c)
    svm.fit(X_train, y_train)
    print ("Accuracy for C=%s: %s"
           % (c, accuracy_score(y_val, svm.predict(X_val))))
```

```
Accuracy for C=0.003: 0.8754666666666666
Accuracy for C=0.01: 0.8725333333333334
Accuracy for C=0.05: 0.8653333333333333
```

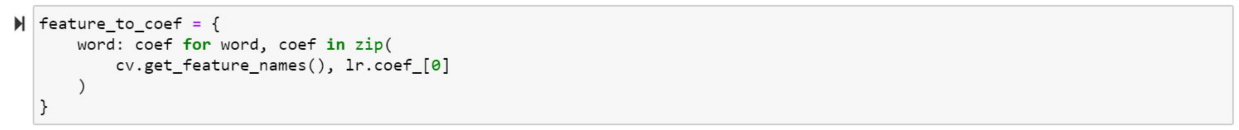
```
c:\users\mj\appdata\local\programs\python\python37-32\lib\site-packages\sklearn\svm\_base.py:977: ConvergenceWarning: Liblin
ear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
Accuracy for C=0.25: 0.8501333333333333
Accuracy for C=0.5: 0.8453333333333334
Accuracy for C=0.375: 0.8458666666666667
Accuracy for C=1: 0.8421333333333333
```

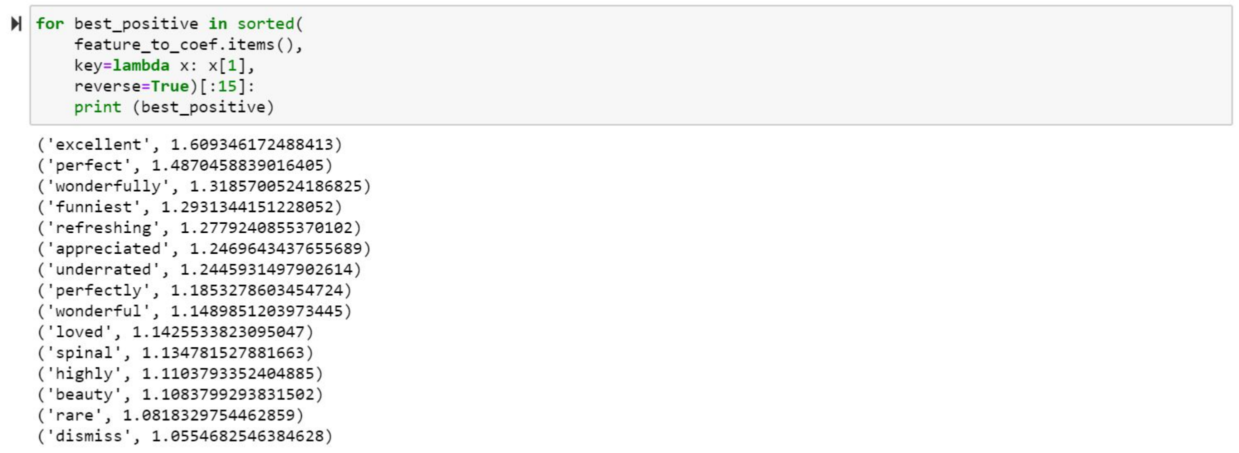
Here, we are able to achieve an accuracy of around 87.5%.

DETERMINING POSITIVE AND NEGATIVE COEFFICIENTS

It is important to determine the most discriminating words that are associated with both positive and negative reviews as well in order to get an idea of how negative/positive a word in the review is.

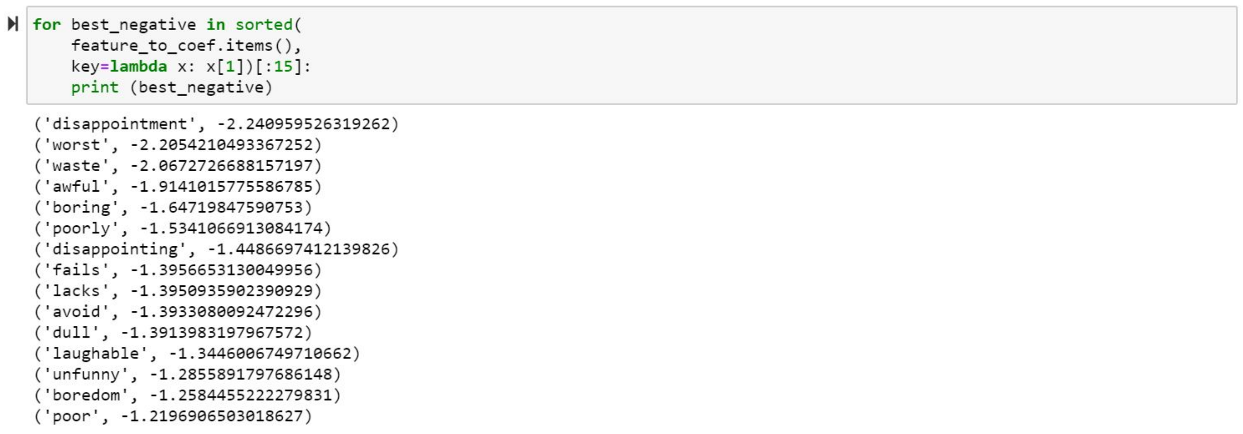


```
feature_to_coef = {
    word: coef for word, coef in zip(
        cv.get_feature_names(), lr.coef_[0]
    )
}
```



```
for best_positive in sorted(
    feature_to_coef.items(),
    key=lambda x: x[1],
    reverse=True)[:15]:
    print (best_positive)
```

```
('excellent', 1.609346172488413)
('perfect', 1.4870458839016405)
('wonderfully', 1.3185700524186825)
('funniest', 1.2931344151228052)
('refreshing', 1.2779240855370102)
('appreciated', 1.2469643437655689)
('underrated', 1.2445931497902614)
('perfectly', 1.1853278603454724)
('wonderful', 1.1489851203973445)
('loved', 1.1425533823095047)
('spinal', 1.134781527881663)
('highly', 1.1103793352404885)
('beauty', 1.1083799293831502)
('rare', 1.0818329754462859)
('dismiss', 1.0554682546384628)
```



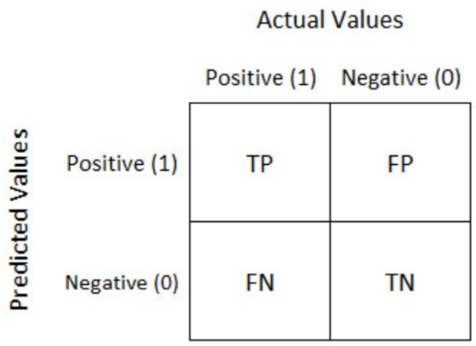
```
for best_negative in sorted(
    feature_to_coef.items(),
    key=lambda x: x[1])[:15]:
    print (best_negative)
```

```
('disappointment', -2.240959526319262)
('worst', -2.2054210493367252)
('waste', -2.0672726688157197)
('awful', -1.9141015775586785)
('boring', -1.64719847590753)
('poorly', -1.5341066913084174)
('disappointing', -1.4486697412139826)
('fails', -1.3956653130049956)
('lacks', -1.3950935902390929)
('avoid', -1.3933080092472296)
('dull', -1.3913983197967572)
('laughable', -1.3446006749710662)
('unfunny', -1.2855891797686148)
('boredom', -1.2584455222279831)
('poor', -1.2196906503018627)
```

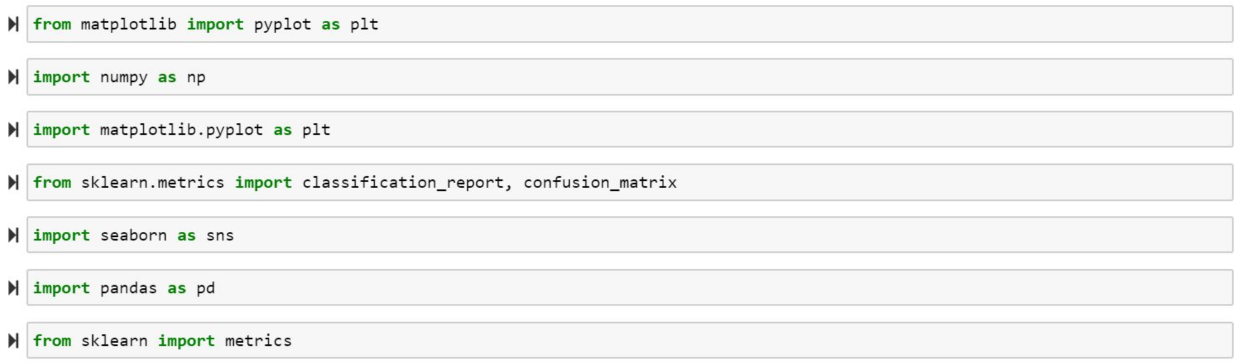
VISUALIZING RESULTS - PERFORMANCE

1. Confusion Matrix

A confusion matrix is a performance measurement for machine learning classification problems where the output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.

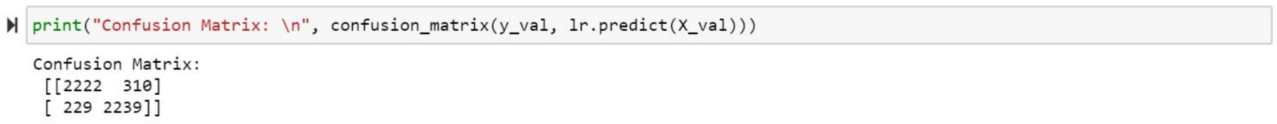


- First, we need to import the following libraries:



```
from matplotlib import pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import pandas as pd
from sklearn import metrics
```

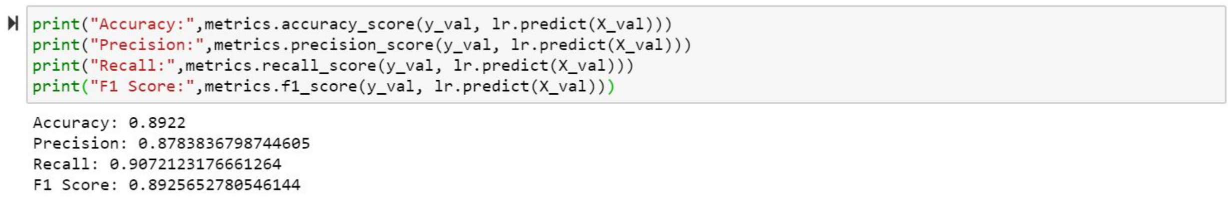
- Print the confusion matrix



```
print("Confusion Matrix: \n", confusion_matrix(y_val, lr.predict(X_val)))

Confusion Matrix:
 [[2222  310]
 [ 229 2239]]
```

2. Performance Metrics



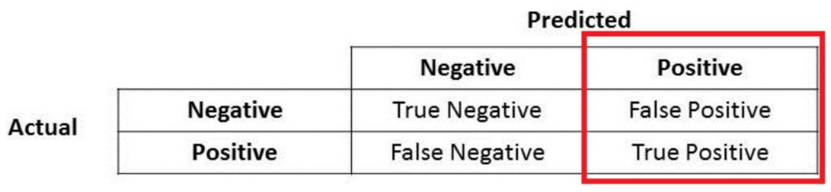
```
print("Accuracy:",metrics.accuracy_score(y_val, lr.predict(X_val)))
print("Precision:",metrics.precision_score(y_val, lr.predict(X_val)))
print("Recall:",metrics.recall_score(y_val, lr.predict(X_val)))
print("F1 Score:",metrics.f1_score(y_val, lr.predict(X_val)))
```

```
Accuracy: 0.8922
Precision: 0.8783836798744605
Recall: 0.9072123176661264
F1 Score: 0.8925652780546144
```

- Accuracy: Out of all the classes, how much we predicted correctly, which will be, in this case 89%. It should be as high as possible.

- Precision: Out of all the positive classes we have predicted correctly, how many are actually positive. In this case, precision is 87.8%. Precision is a good measure to determine when the cost of False Positive is high.

$$\text{Precision} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Positive}}$$



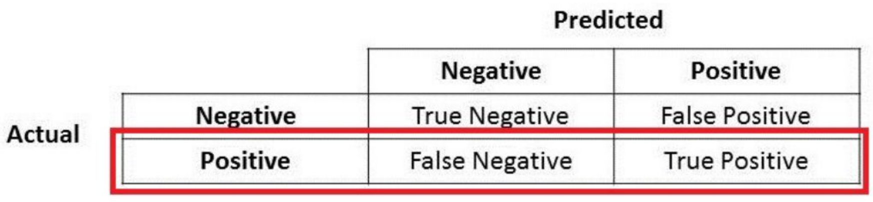
		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

True Positive + False Positive = Total Predicted Positive

- Recall: Out of all the positive classes, how much we predicted correctly. It should be as high as possible. In this case, recall is 90.7%. So Recall actually calculates how many of the Actual Positives our model captures through labeling it as Positive (True Positive). Applying the same understanding, we know that Recall shall be the model metric we use to select our best model when there is a high cost associated with False Negative.

$$\text{Recall} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Negative}}$$

$$= \frac{\textit{True Positive}}{\textit{Total Actual Positive}}$$



		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

True Positive + False Negative = Actual Positive

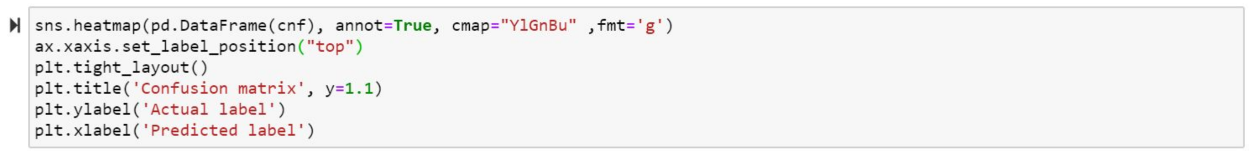
- F1 score: It is difficult to compare two models with low precision and high recall or vice versa. So to make them comparable, we use F-Score. F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

$$\text{F1} = 2 \times \frac{\textit{Precision} * \textit{Recall}}{\textit{Precision} + \textit{Recall}}$$

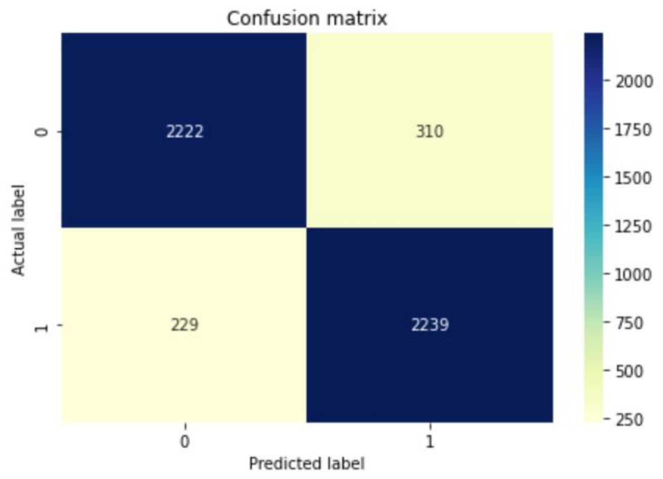
In this case, we have an F1 score of 89.2%.

3. Heat Map

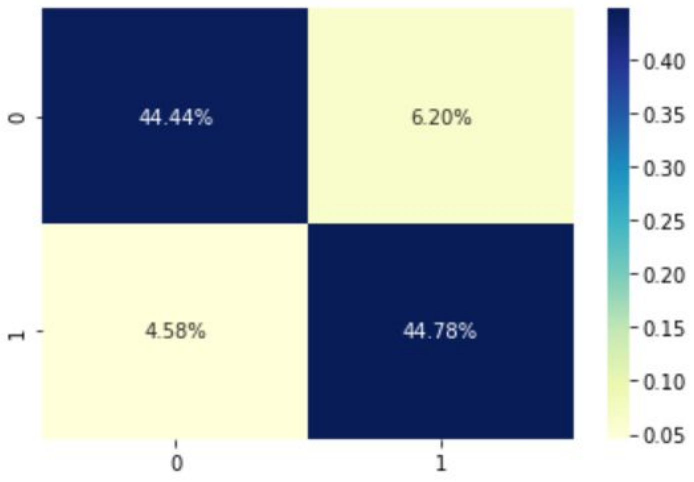
A better way to visualize a confusion matrix is by utilizing seaborn's heatmap.



```
sns.heatmap(pd.DataFrame(cnf), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```



```
sns.heatmap(cnf/np.sum(cnf), annot=True,
            fmt='.2%', cmap='YlGnBu')
```

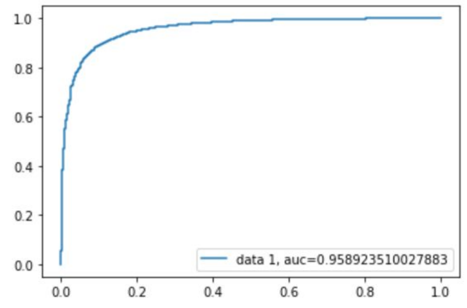


4. AUC-ROC Curve

AUC - ROC curve is a performance measurement for classification problems at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much a model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s.

The ROC curve is plotted with TPR(or recall) against the FPR(or precision) where TPR is on the y-axis and FPR is on the x-axis.

```
y_pred_proba = lr.predict_proba(X_val)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_val,  y_pred_proba)
auc = metrics.roc_auc_score(y_val, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



An AUC of 0.958 means that there is a 95.8% chance that the model will be able to distinguish between positive class and negative class.