RAMAIAH
Institute of Technology

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Report submitted for the Subject

Introduction to Artificial Intelligence & Machine Learning – CSOE09

Submitted by

Name	USN
Harsha Aryan	1MS20EC033
Rehan Khan	1MS20EC080

Under the Supervision of

Dr. Meeradevi

Associate Professor

RAMAIAH INSTITUTE OF TECHNOLOGY

(Autonomous Institute, Affiliated to VTU)

BANGALORE-560054

www.msrit.edu, Sept-Dec 2022

CONTENTS

ABSTRACT

Globally, depression remains a pressing issue, especially in developed and emerging nations. The World Health Organization (WHO) estimates that 322 million people around the world suffer from depression. Suicidal ideation and attempted suicide can often result from depression if left untreated.

Access to and understanding of mental health interventions remains low. Also, with popular beliefs including a supernatural basis for mental illness, the doublethink and disparity in knowledge across demographics remain considerable despite the increasing awareness.

The diagnosis of depression and identifying when a depressed individual is at risk of attempting suicide are important problems at both the individual and population level. Users with struggling social skills can be identified quickly, efficiently, and unbiasedly using Artificial Intelligence and Machine Learning.

DATA SET DESCRIPTION

The raw data is collected through web scraping Subreddits and is cleaned using multiple NLP techniques. The data is only in English.It mainly targets mental health classification.

It consists of two columns one being the text and another being whether that person is depressed or not.

It consists of 7731 samples with 49.6% entries being not depressed and the rest 50.4% being depressed,this makes it an almost perfectly balanced dataset.

Here ‘1’ represents being depressed and ‘0’ not depressed. Some of the entries are given below

	clean_text	is_depression
0	we understand that most people who reply immed...	1
1	welcome to r depression s check in post a plac...	1
2	anyone else instead of sleeping more when depr...	1
3	i ve kind of stuffed around a lot in my life d...	1
4	sleep is my greatest and most comforting escap...	1

	clean_text	is_depression
7726	is that snow	0
7727	moulin rouge mad me cry once again	0
7728	trying to shout but can t find people on the list	0
7729	ughh can t find my red sox hat got ta wear thi...	0
7730	slept wonderfully finally tried swatching for ...	0

MODEL DESCRIPTION (CODE)

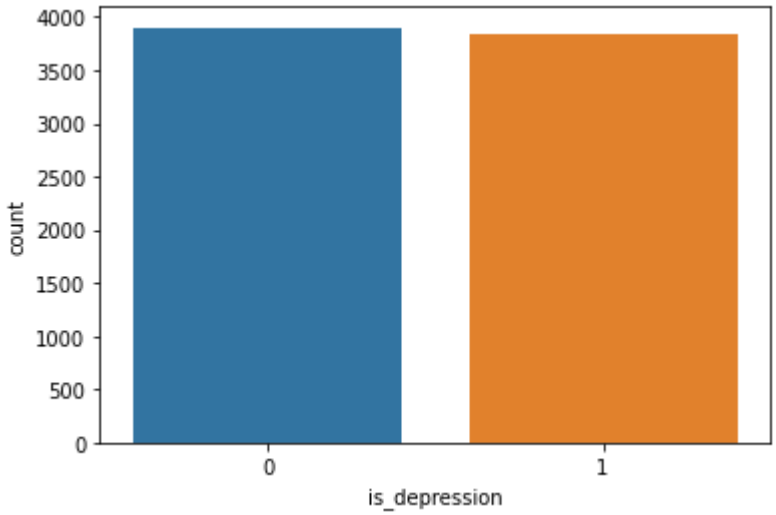
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

df=pd.read_csv('depression_dataset_reddit.csv')

df.head()
```

	clean_text	is_depression
0	we understand that most people who reply immed...	1
1	welcome to r depression s check in post a plac...	1
2	anyone else instead of sleeping more when depr...	1
3	i ve kind of stuffed around a lot in my life d...	1
4	sleep is my greatest and most comforting escap...	1

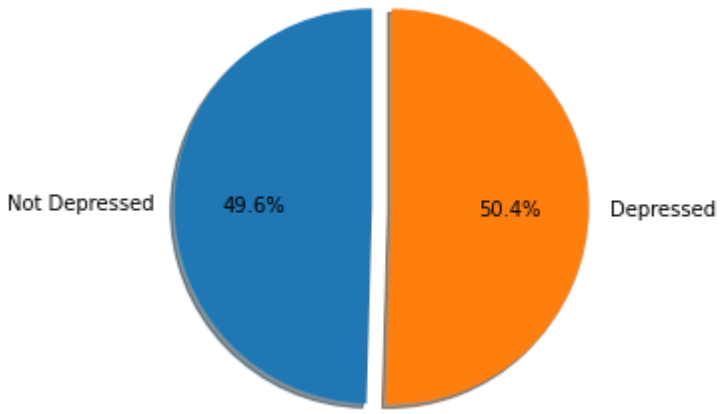
```
df.tail()
```

	clean_text	is_depression
7726	is that snow	0
7727	moulin rouge mad me cry once again	0
7728	trying to shout but can t find people on the list	0
7729	ughh can t find my red sox hat got ta wear thi...	0
7730	slept wonderfully finally tried swatching for ...	0

```
#Graphs and plots


sns.countplot(df['is_depression'])
```




```
plt.rcParams['figure.facecolor'] = 'white'

labels = 'Not Depressed', 'Depressed'

sizes =
[df['is_depression'].value_counts()[1],df['is_depression'].value_counts
()[0]]

explode = (0, 0.1)

fig1, ax1 = plt.subplots()

ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
shadow=True, startangle=90)

ax1.axis('equal')

plt.show()
```



Input other libraries and necessary files

```
import re

import nltk

nltk.download('omw-1.4')

nltk.download('wordnet')

nltk.download("stopwords")

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize
```

Lemmatization

```
lemmatize=nltk.WordNetLemmatizer()

corpus=[]

for i in range(0,len(df)):

    review=re.sub('[^a-zA-Z]',' ',df['clean_text'][i])

    review=review.lower()

    review=review.split()

    review=[lemmatize.lemmatize(word) for word in review if not word in
stopwords.words('english')]

    review=' '.join(review)

    corpus.append(review)

from sklearn.feature_extraction.text import CountVectorizer

count=CountVectorizer(stop_words="english",max_features=550)#it removes
unnecessary words and finds most using 550 words

from wordcloud import WordCloud

from wordcloud import STOPWORDS

#We need convert all reivews to single text

all_words=' '.join([text for text in corpus])

wordcloud=WordCloud(width=800,height=500,random_state=21,max_font_size=
110).generate(all_words)

plt.figure(figsize=[15,15])

plt.imshow(wordcloud, interpolation="bilinear")
```

```
plt.axis("off")

plt.title("Most used words in Posts")

plt.show()
```


```
matrix=count.fit_transform(corpus).toarray()

matrix

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

Defining the target and attributes

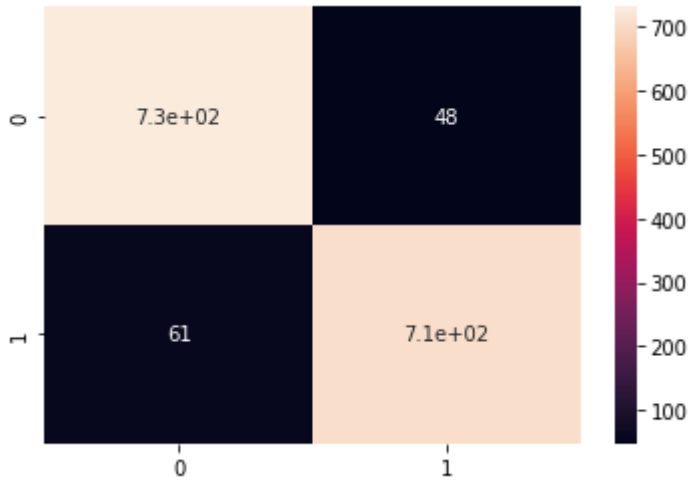
```
x=matrix

y=df["is_depression"]
```

Splitting x and y into training and testing sets

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)

from sklearn.naive_bayes import GaussianNB
classifier=GaussianNB()
classifier.fit(x_train,y_train)
classifier.score(x_test,y_test)
0.929541047188106
```

Plotting the confusion matrix

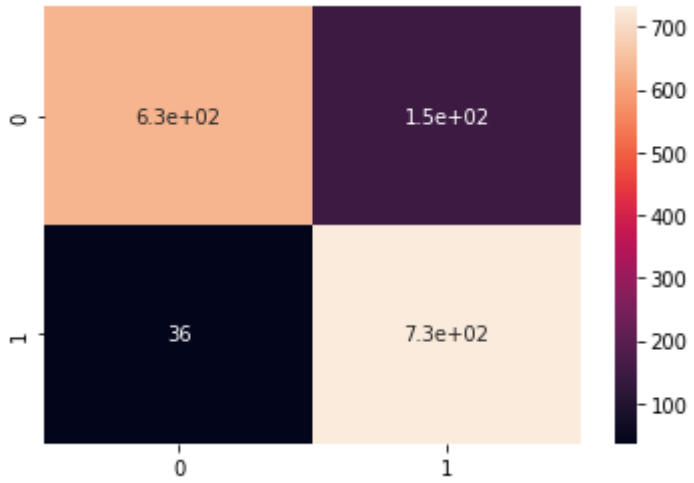
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,classifier.predict(x_test))
sns.heatmap(cm,annot=True)
```



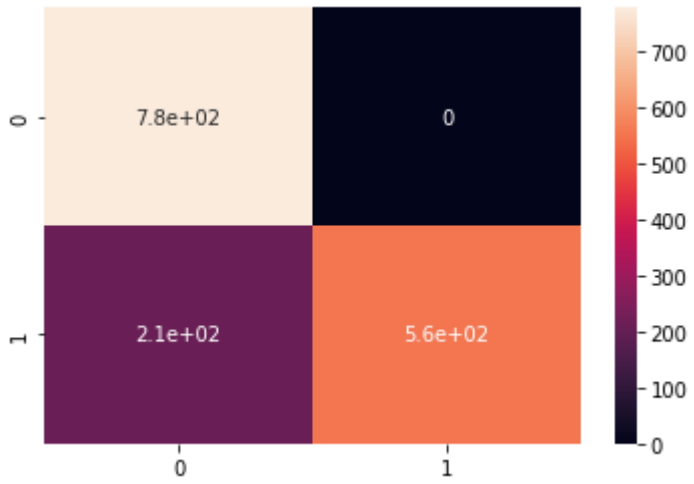
Trying other naive bayes models

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
nb=MultinomialNB()
nb.fit(x_train,y_train)
nb.score(x_test,y_test)
0.8823529411764706
cm=confusion_matrix(y_test,nb.predict(x_test))
```

```
sns.heatmap(cm,annot=True)
```




```
bn=BernoulliNB()

bn.fit(x_train,y_train)

bn.score(x_test,y_test)

0.8623141564318035

cm=confusion_matrix(y_test,bn.predict(x_test))

sns.heatmap(cm,annot=True)
```

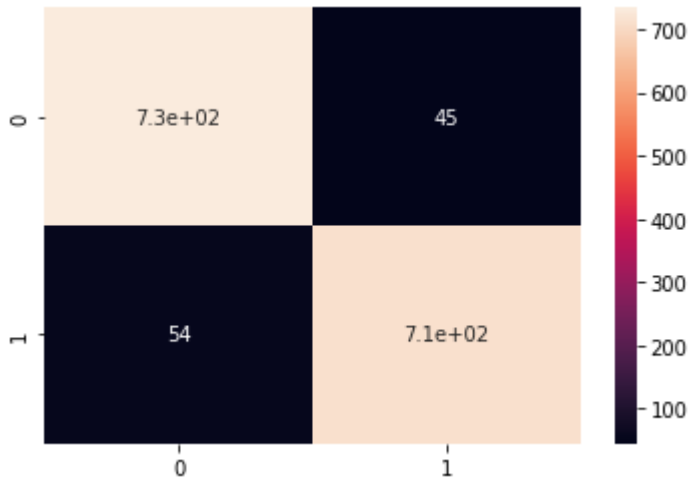


Finding the optimal value for var_smoothing in GaussianNB

```
from sklearn.model_selection import GridSearchCV

from sklearn.naive_bayes import GaussianNB
```

```
param_grid = {'var_smoothing': np.logspace(0,-9, num=100)}

grid_search = GridSearchCV(GaussianNB(), param_grid, cv=5)

grid_search.fit(x_train, y_train)

grid_search.best_params_


classifier=GaussianNB(var_smoothing=0.0001)

classifier.fit(x_train,y_train)

classifier.score(x_test,y_test)


cm=confusion_matrix(y_test,classifier.predict(x_test))

sns.heatmap(cm,annot=True)


from sklearn.metrics import classification_report

print(classification_report(y_test,classifier.predict(x_test)))
```

	precision	recall	f1-score	support
0	0.93	0.94	0.94	779
1	0.94	0.93	0.94	768
accuracy			0.94	1547
macro avg	0.94	0.94	0.94	1547
weighted avg	0.94	0.94	0.94	1547



```
sample="There are people out there enjoying life, are rich as hell and have made more money in a day than I will in my entire life. Yet here I am struggling, jobless actually avoiding working for my families sake while this pandemic goes on. Either way I just feel like a failure, I'd love to be rich doesn't everyone? Yet I just don't have the confidence or see myself ever making it. I have a degree in computer networking too, yet I've only had one IT related job since then and it was pretty basic and poor pay and I eventually lost the job. Being an extreme introvert really doesn't help in social life, job life and if you are an introvert I'd say there's probably less chance of being rich and successful though obviously it isn't impossible. My life is in no means actually bad though I have it fairly okay, even if I am eventually going to have to get a job if my money gets too low. It just sucks feeling the way I do and feeling like a failure."

#lemmatize the sample

sample=re.sub('[^a-zA-Z]',' ',sample)

sample=sample.lower()

sample=sample.split()

sample=[lemmatize.lemmatize(word) for word in sample if not word in stopwords.words('english')]

sample=' '.join(sample)

#convert sample to matrix

sample=count.transform([sample]).toarray()

sample
```

```
array([[0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1,
        0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
        0, 0, 0, 5, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
        1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

```
#predict the sample

classifier.predict(sample)

prediction=classifier.predict(sample)

if prediction==1:

    print("Depressed")

else:

    print("Not Depressed")
```

Output: Depressed

ALGORITHM DESCRIPTION

Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable y and dependent feature vector x1 through xn:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Using the naive conditional independence assumption that

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y),$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Since P(x1,x2…xn) is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters. (For theoretical reasons why naive Bayes works well, and on which types of data it does, see the references below.)

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.

Gaussian Naive Bayes

GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

The parameters σy and μy are estimated using maximum likelihood.

Multinomial Naive Bayes

`MultinomialNB` implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors θy=(θy1,...,θyn) for each class y, where n is the number of features (in text classification, the size of the vocabulary) and θyi is the probability P(xi∣y) of feature i appearing in a sample belonging to class y.

The parameters θy is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

θyi=(Nyi+α)/(Ny+αn)

where $N_{yi}=\sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_y=\sum N_{yi}$ is the total count of all features for class y.

The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha=1$ is called Laplace smoothing, while $\alpha<1$ is called Lidstone smoothing.

Bernoulli Naive Bayes

BernoulliNB implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. Therefore, this class requires samples to be represented as binary-valued feature vectors; if handed any other kind of data, a BernoulliNB instance may binarize its input (depending on the binarize parameter).

The decision rule for Bernoulli naive Bayes is based on

$$P(x_i \mid y) = P(x_i = 1 \mid y)x_i + (1 - P(x_i = 1 \mid y))(1 - x_i)$$

which differs from multinomial NB's rule in that it explicitly penalizes the non-occurrence of a feature that is an indicator for class , where the multinomial variant would simply ignore a non-occurring feature.

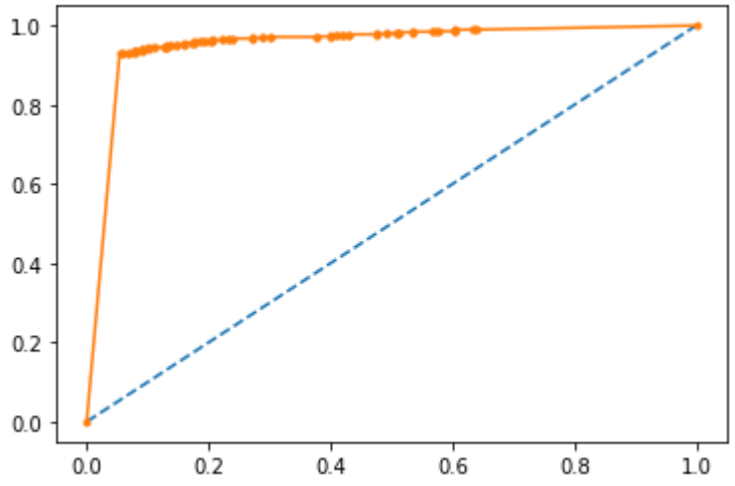
In the case of text classification, word occurrence vectors (rather than word count vectors) may be used to train and use this classifier. BernoulliNB might perform better on some datasets, especially those with shorter documents.

RESULTS AND INFERENCES

roc curve and auc for the model

```
from sklearn.metrics import roc_curve

from sklearn.metrics import roc_auc_score

from matplotlib import pyplot

probs = classifier.predict_proba(x_test)

probs = probs[:, 1]

auc = roc_auc_score(y_test, probs)

print('AUC: %.2f' % auc)

fpr, tpr, thresholds = roc_curve(y_test, probs)

pyplot.plot([0, 1], [0, 1], linestyle='--')

pyplot.plot(fpr, tpr, marker='.')

pyplot.show()
```

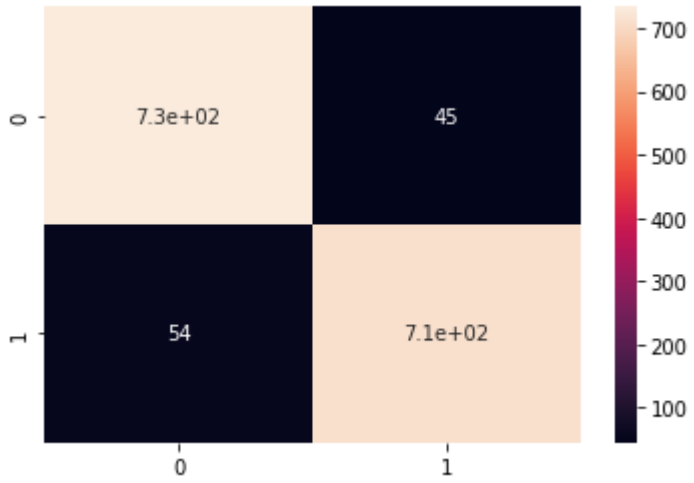
AUC Score:0.95



Accuracy: 0.94

	precision	recall	f1-score	support
0	0.93	0.94	0.94	779
1	0.94	0.93	0.94	768
accuracy			0.94	1547
macro avg	0.94	0.94	0.94	1547
weighted avg	0.94	0.94	0.94	1547

Confusion Matrix



On the test dataset, the model achieves 94% accuracy. False Positives and False Negatives are very low compared to True Positives and True Negatives.