****

Assignment No: 2

Title Name: Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance.

Name: Vasant Kumar

Class : BE **Div**: B **Batch**: C

Roll No: 405B091

****

In [4]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics
df=pd.read_csv('emails.csv')
df.head()
df.columns
df.isnull().sum()
df.dropna(inplace = True)
df.drop(['Email  No.'],axis=1,inplace=True)
X = df.drop(['Prediction'],axis = 1)
y = df['Prediction']
from sklearn.preprocessing import scale
X = scale(X)
# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_
```

In [5]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Prediction",y_pred)
print("KNN accuracy = ",metrics.accuracy_score(y_test,y_pred))
print("Confusion  matrix",metrics.confusion_matrix(y_test,y_pred))
```

```
Prediction [0 0 1 ... 1 1 1]
KNN accuracy =  0.8009020618556701
Confusion matrix [[804 293]
 [ 16 439]]
```

In [6]:

```
# cost C = 1
model = SVC(C = 1)
# fit
model.fit(X_train, y_train)
# predict
y_pred = model.predict(X_test)
metrics.confusion_matrix(y_true=y_test,  y_pred=y_pred)
print("SVM accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

```
SVM accuracy =  0.9381443298969072
```

****

Assignment No: 3

Title Name: Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months

Name: Abhishek Kumar

Class : BE **Div**: 1 **Batch**: A

Roll No: 405A002

****

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries
df = pd.read_csv("Churn_Modelling.csv")
```

In [2]:

```
df.head()
df.shape
df.describe()
df.isnull()
df.isnull().sum()
df.info()
df.dtypes
df.columns
df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) #Dropping the unnece
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   RowNumber       10000 non-null  int64  
 1   CustomerId      10000 non-null  int64  
 2   Surname         10000 non-null  object 
 3   CreditScore     10000 non-null  int64  
 4   Geography       10000 non-null  object 
 5   Gender          10000 non-null  object 
 6   Age             10000 non-null  int64  
 7   Tenure          10000 non-null  int64  
 8   Balance         10000 non-null  float64
 9   NumOfProducts   10000 non-null  int64  
 10  HasCrCard       10000 non-null  int64  
 11  IsActiveMember  10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited          10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

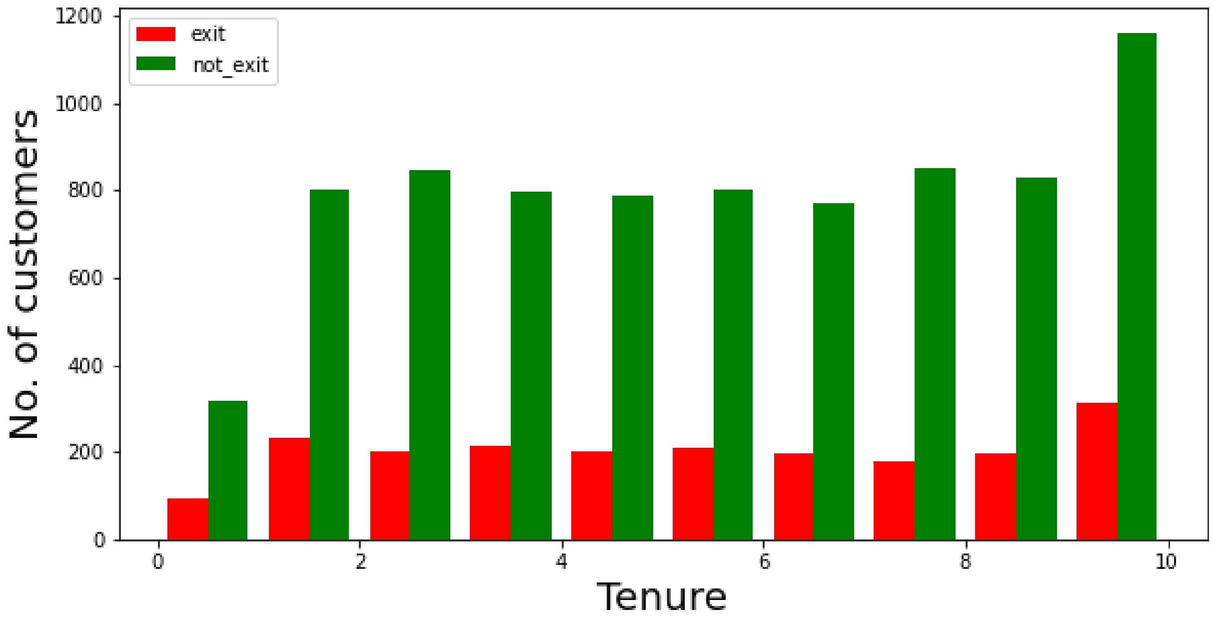
Out[2]:

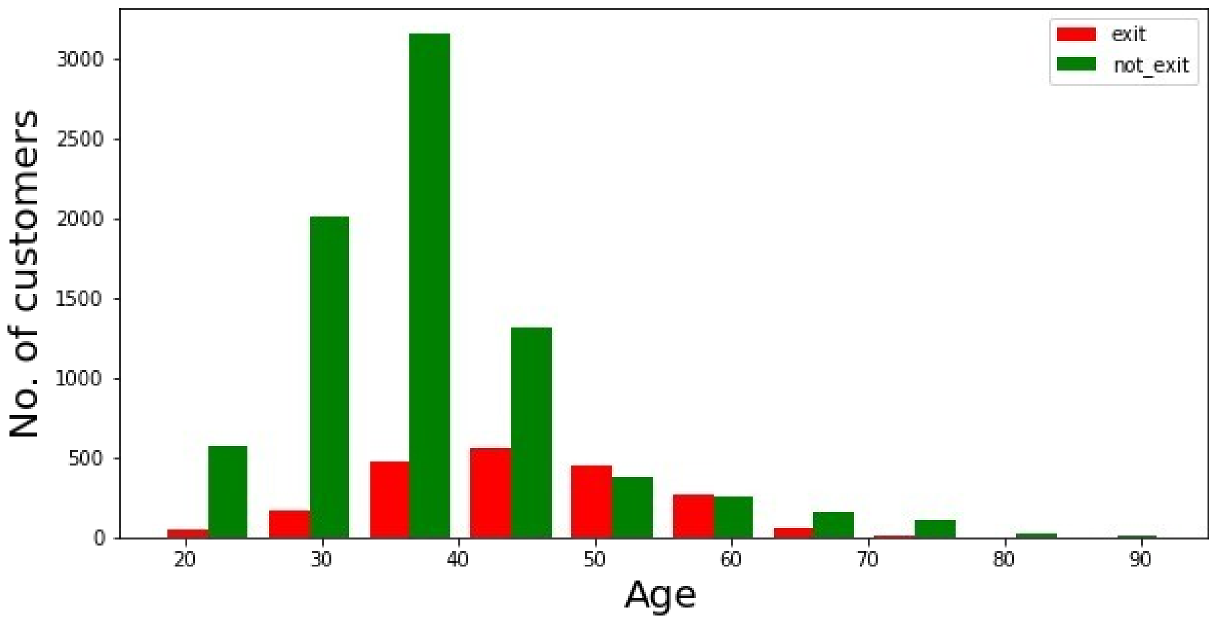
	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveM
0	619	France	Female	42	2	0.00	1	1	
1	608	Spain	Female	41	1	83807.86	1	0	
2	502	France	Female	42	8	159660.80	3	1	
3	699	France	Female	39	1	0.00	2	0	
4	850	Spain	Female	43	2	125510.82	1	1	

In [3]:
```
def visualization(x, y, xlabel):
    plt.figure(figsize=(10,5))
    plt.hist([x, y], color=['red', 'green'], label = ['exit', 'not_exit'])
    plt.xlabel(xlabel,fontsize=20)
    plt.ylabel("No. of customers", fontsize=20)
    plt.legend()
```

In [4]:
```
df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']
```

```
visualization(df_churn_exited,  df_churn_not_exited,  "Tenure")
df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']
visualization(df_churn_exited2,  df_churn_not_exited2,  "Age")
```





In [6]:

```
X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard',
states = pd.get_dummies(df['Geography'],drop_first = True)
gender = pd.get_dummies(df['Gender'],drop_first = True)
df = pd.concat([df,gender,states], axis = 1)
```

In [8]:

```
df.head()
X  =  df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiv
y = df['Exited']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.30)
```

In [9]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
X_train
X_test
```

Out[9]:
```
array([[ 0.08909172,  2.03556129, -1.04195601, ...,  0.90636285,
        -0.57581067,  1.7581737 ],
       [-0.6935785 , -0.3592006 ,  0.33616247, ...,  0.90636285,
        -0.57581067, -0.56877202],
       [ 1.7066102 ,  3.18504699, -1.04195601, ..., -1.10331088,
        -0.57581067,  1.7581737 ],
       ...,
       [ 0.07865612, -0.93394345, -0.35289677, ...,  0.90636285,
        -0.57581067, -0.56877202],
       [-0.46399524, -0.3592006 , -1.73101525, ..., -1.10331088,
        -0.57581067, -0.56877202],
       [ 1.59181856, -0.55078155,  0.33616247, ...,  0.90636285,
        -0.57581067, -0.56877202]])
```

In [10]:
```
import keras#Can use Tenserflow as well but won't be able to understand the errors
from keras.models import Sequential #To create sequential neural network
from keras.layers import Dense #To create hidden layers
classifier = Sequential()
#To add the layers
#Dense helps to contruct the neurons
#Input Dimension means we have 11 features
# Units is to create the hidden layers
```

In [11]:
```
classifier.add(Dense(activation = "relu",input_dim = 11,units = 6,kernel_initializ
classifier.add(Dense(activation = "relu",units = 6,kernel_initializer = "uniform")
classifier.add(Dense(activation = "sigmoid",units = 1,kernel_initializer = "unifor
classifier.compile(optimizer="adam",loss  =  'binary_crossentropy',metrics =  ['accur
classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 2nd layer and
classifier.fit(X_train,y_train,batch_size=10,epochs=50) #Fitting the ANN to training
y_pred =classifier.predict(X_test)
y_pred = (y_pred > 0.5) #Predicting the result
from  sklearn.metrics  import  confusion_matrix,accuracy_score,classification_report
cm = confusion_matrix(y_test,y_pred)
cm
accuracy = accuracy_score(y_test,y_pred)
accuracy
plt.figure(figsize = (10,7))
sns.heatmap(cm,annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
print(classification_report(y_test,y_pred))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 6)                 72

 dense_1 (Dense)             (None, 6)                 42

 dense_2 (Dense)             (None, 1)                 7

=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
700/700 [==============================] - 1s 675us/step - loss: 0.4841 - accuracy:
0.7970
Epoch 2/50
```

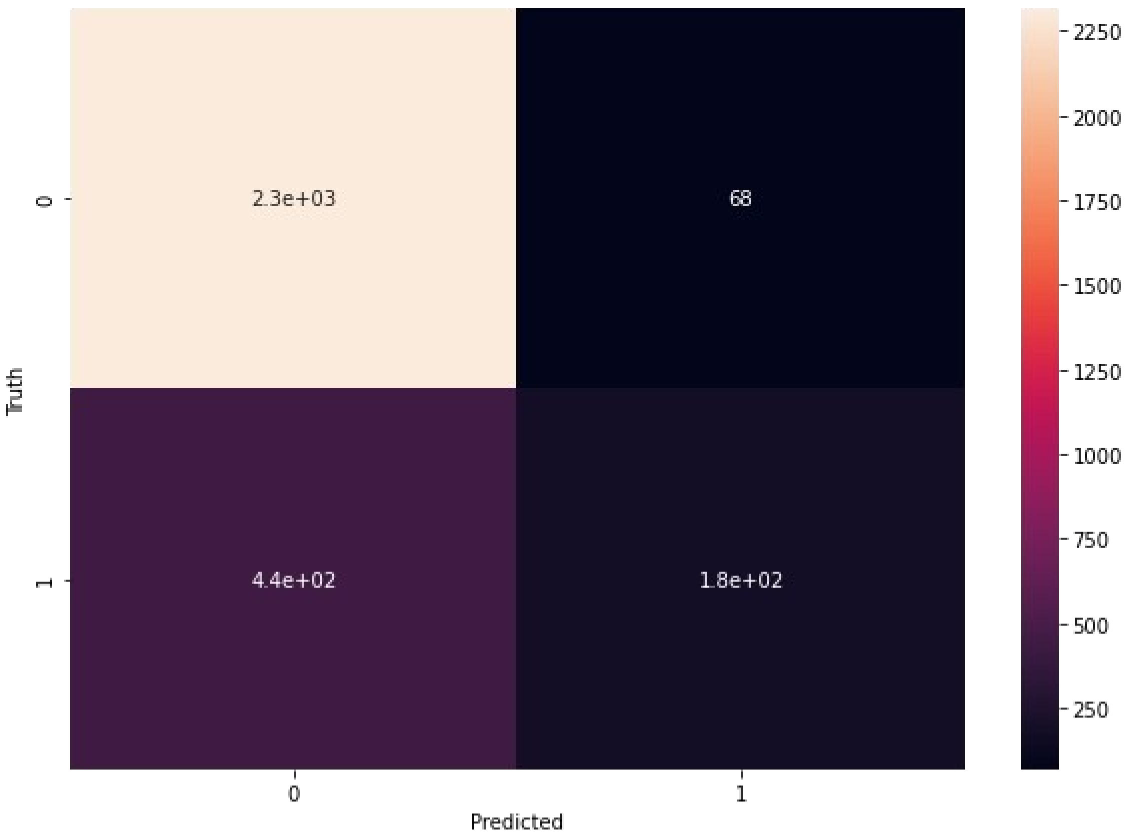
```
700/700 [==============================] - 5s 711us/step - loss: 0.4220 - accuracy:
0.7970
Epoch 3/50
700/700 [==============================] - 0s 715us/step - loss: 0.4165 - accuracy:
0.7990
Epoch 4/50
700/700 [==============================] - 0s 625us/step - loss: 0.4129 - accuracy:
0.8277
Epoch 5/50
700/700 [==============================] - 0s 634us/step - loss: 0.4109 - accuracy:
0.8280
Epoch 6/50
700/700 [==============================] - 1s 736us/step - loss: 0.4093 - accuracy:
0.8304
Epoch 7/50
700/700 [==============================] - 0s 650us/step - loss: 0.4078 - accuracy:
0.8321
Epoch 8/50
700/700 [==============================] - 0s 653us/step - loss: 0.4065 - accuracy:
0.8319
Epoch 9/50
700/700 [==============================] - 0s 640us/step - loss: 0.4053 - accuracy:
0.8350
Epoch 10/50
700/700 [==============================] - 0s 621us/step - loss: 0.4044 - accuracy:
0.8351
Epoch 11/50
700/700 [==============================] - 0s 643us/step - loss: 0.4035 - accuracy:
0.8360
Epoch 12/50
700/700 [==============================] - 0s 700us/step - loss: 0.4024 - accuracy:
0.8360
Epoch 13/50
700/700 [==============================] - 0s 648us/step - loss: 0.4016 - accuracy:
0.8349
Epoch 14/50
700/700 [==============================] - 0s 649us/step - loss: 0.4010 - accuracy:
0.8356
Epoch 15/50
700/700 [==============================] - 0s 712us/step - loss: 0.4005 - accuracy:
0.8361
Epoch 16/50
700/700 [==============================] - 1s 721us/step - loss: 0.3997 - accuracy:
0.8367
Epoch 17/50
700/700 [==============================] - 1s 716us/step - loss: 0.3996 - accuracy:
0.8364
Epoch 18/50
700/700 [==============================] - 0s 703us/step - loss: 0.3988 - accuracy:
0.8371
Epoch 19/50
700/700 [==============================] - 0s 695us/step - loss: 0.3985 - accuracy:
0.8360
Epoch 20/50
700/700 [==============================] - 0s 650us/step - loss: 0.3973 - accuracy:
0.8364
Epoch 21/50
700/700 [==============================] - 0s 641us/step - loss: 0.3978 - accuracy:
0.8369
Epoch 22/50
700/700 [==============================] - 0s 615us/step - loss: 0.3975 - accuracy:
0.8363
Epoch 23/50
700/700 [==============================] - 0s 639us/step - loss: 0.3962 - accuracy:
0.8377
Epoch 24/50
700/700 [==============================] - 0s 649us/step - loss: 0.3966 - accuracy:
0.8359
Epoch 25/50
```

```
700/700 [==============================] - 0s 666us/step - loss: 0.3961 - accuracy:
0.8371
Epoch 26/50
700/700 [==============================] - 0s 630us/step - loss: 0.3959 - accuracy:
0.8373
Epoch 27/50
700/700 [==============================] - 0s 666us/step - loss: 0.3957 - accuracy:
0.8369
Epoch 28/50
700/700 [==============================] - 0s 644us/step - loss: 0.3956 - accuracy:
0.8391
Epoch 29/50
700/700 [==============================] - 0s 594us/step - loss: 0.3956 - accuracy:
0.8379
Epoch 30/50
700/700 [==============================] - 0s 668us/step - loss: 0.3950 - accuracy:
0.8371
Epoch 31/50
700/700 [==============================] - 0s 715us/step - loss: 0.3949 - accuracy:
0.8393
Epoch 32/50
700/700 [==============================] - 0s 705us/step - loss: 0.3950 - accuracy:
0.8376
Epoch 33/50
700/700 [==============================] - 0s 652us/step - loss: 0.3955 - accuracy:
0.8386
Epoch 34/50
700/700 [==============================] - 0s 684us/step - loss: 0.3948 - accuracy:
0.8381
Epoch 35/50
700/700 [==============================] - 0s 661us/step - loss: 0.3944 - accuracy:
0.8387
Epoch 36/50
700/700 [==============================] - 0s 678us/step - loss: 0.3947 - accuracy:
0.8394
Epoch 37/50
700/700 [==============================] - 0s 663us/step - loss: 0.3942 - accuracy:
0.8394
Epoch 38/50
700/700 [==============================] - 0s 711us/step - loss: 0.3939 - accuracy:
0.8397
Epoch 39/50
700/700 [==============================] - 1s 996us/step - loss: 0.3941 - accuracy:
0.8383
Epoch 40/50
700/700 [==============================] - 1s 799us/step - loss: 0.3939 - accuracy:
0.8404
Epoch 41/50
700/700 [==============================] - 1s 863us/step - loss: 0.3938 - accuracy:
0.8396
Epoch 42/50
700/700 [==============================] - 1s 739us/step - loss: 0.3943 - accuracy:
0.8377
Epoch 43/50
700/700 [==============================] - 1s 761us/step - loss: 0.3933 - accuracy:
0.8374
Epoch 44/50
700/700 [==============================] - 1s 919us/step - loss: 0.3938 - accuracy:
0.8389
Epoch 45/50
700/700 [==============================] - 0s 630us/step - loss: 0.3938 - accuracy:
0.8387
Epoch 46/50
700/700 [==============================] - 1s 766us/step - loss: 0.3936 - accuracy:
0.8381
Epoch 47/50
700/700 [==============================] - 0s 623us/step - loss: 0.3935 - accuracy:
0.8386
Epoch 48/50
```

```
700/700 [==============================] - 0s 614us/step - loss: 0.3937 - accuracy:
0.8380
Epoch 49/50
700/700 [==============================] - 1s 763us/step - loss: 0.3934 - accuracy:
0.8399
Epoch 50/50
700/700 [==============================] - 1s 837us/step - loss: 0.3935 - accuracy:
0.8391
              precision    recall  f1-score   support

           0       0.84      0.97      0.90      2384
           1       0.72      0.29      0.41       616

    accuracy                           0.83      3000
   macro avg       0.78      0.63      0.66      3000
weighted avg       0.82      0.83      0.80      3000
```



In [ ]: