Name: Imam Mahamad Shaikh

Div: C

Batch: C2

Roll No: B23025

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. Dataset link: The emails.csv dataset on the Kaggle https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score
from sklearn.neighbors import KNeighborsClassifier

df=pd.read_csv('emails.csv')
df
```

```
       Email No.  the  to  ect  and  for  of    a  you  hou  ...
connevey  \
0         Email 1    0   0    1    0    0   0    2    0    0  ...
0
1         Email 2    8  13   24    6    6   2  102    1   27  ...
0
2         Email 3    0   0    1    0    0   0    8    0    0  ...
0
3         Email 4    0   5   22    0    5   1   51    2   10  ...
0
```

```
4         Email 5    7   6   17    1    5   2   57    0    9  ...
0
...           ...  ...  ..  ...  ...  ...  ..  ...  ...  ...  ...
...
5167  Email 5168    2   2    2    3    0   0   32    0    0  ...
0
5168  Email 5169   35  27   11    2    6   5  151    4    3  ...
0
5169  Email 5170    0   0    1    1    0   0   11    0    0  ...
0
5170  Email 5171    2   7    1    0    2   1   28    2    0  ...
0
5171  Email 5172   22  24    5    1    6   5  148    8    2  ...
0
```

```
      jay  valued  lay  infrastructure  military  allowing  ff  dry  \
0       0       0    0                0         0         0   0    0
1       0       0    0                0         0         0   1    0
2       0       0    0                0         0         0   0    0
3       0       0    0                0         0         0   0    0
4       0       0    0                0         0         0   1    0
...   ...     ...  ...              ...       ...       ...  ..  ...
5167    0       0    0                0         0         0   0    0
5168    0       0    0                0         0         0   1    0
5169    0       0    0                0         0         0   0    0
5170    0       0    0                0         0         0   1    0
5171    0       0    0                0         0         0   0    0
```

```
      Prediction
0              0
1              0
2              0
3              0
4              0
...          ...
5167           0
5168           0
5169           1
5170           1
5171           0
```

```
[5172 rows x 3002 columns]
```

```
df.head()
```

```
  Email No.   the  to  ect  and  for  of    a  you  hou  ...  connevey
jay  \
0   Email 1     0   0    1    0    0   0    2    0    0  ...          0
0
1   Email 2     8  13   24    6    6   2  102    1   27  ...          0
```

```
0
2   Email 3     0    0     1     0     0    0     8     0     0  ...           0
0
3   Email 4     0    5    22     0     5    1    51     2    10  ...           0
0
4   Email 5     7    6    17     1     5    2    57     0     9  ...           0
0

   valued  lay  infrastructure  military  allowing  ff  dry
Prediction
0       0    0               0         0         0   0    0
0
1       0    0               0         0         0   1    0
0
2       0    0               0         0         0   0    0
0
3       0    0               0         0         0   0    0
0
4       0    0               0         0         0   1    0
0

[5 rows x 3002 columns]

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5172 entries, 0 to 5171
Columns: 3002 entries, Email No. to Prediction
dtypes: int64(3001), object(1)
memory usage: 118.5+ MB

df.isnull().sum()

Email No.      0
the            0
to             0
ect            0
and            0
              ..
military       0
allowing       0
ff             0
dry            0
Prediction     0
Length: 3002, dtype: int64

X = df.iloc[:, 1:-1].values
y = df.iloc[:, -1].values

scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(X)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_scaled, y,
test_size=0.30, random_state=101)

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

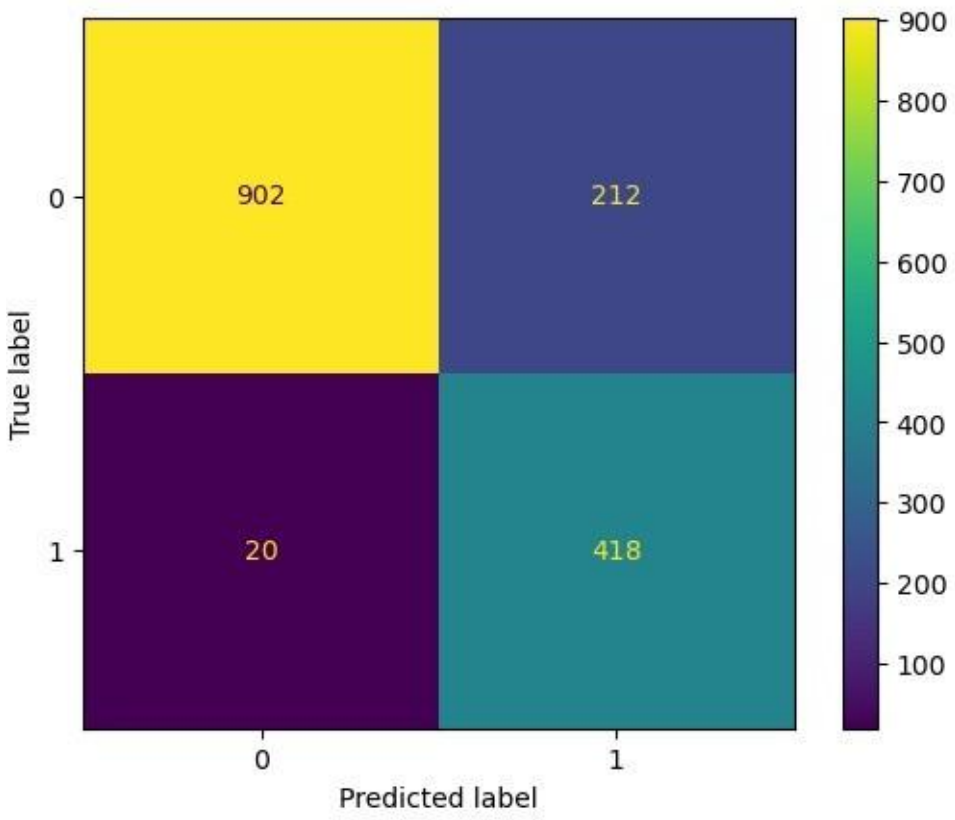
```
KNeighborsClassifier()
```

```
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)

cm
```

```
array([[902, 212],
       [ 20, 418]], dtype=int64)
```

```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x252bc29e910>
```



```
from sklearn.metrics import classification_report
cl_report=classification_report(y_test,y_pred)
print(cl_report)
```

```
              precision    recall  f1-score   support

           0       0.98      0.81      0.89      1114
           1       0.66      0.95      0.78       438

    accuracy                           0.85      1552
   macro avg       0.82      0.88      0.83      1552
weighted avg       0.89      0.85      0.86      1552
```

```
print("Accuracy Score for KNN : ", accuracy_score(y_pred,y_test))
```

```
Accuracy Score for KNN :  0.8505154639175257
```

```
cm = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print("Confusion Matrix:\n", cm)
print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
```

```
Confusion Matrix:
 [[902 212]
 [ 20 418]]
Accuracy: 0.8505154639175257
Error Rate: 0.14948453608247425
Precision: 0.6634920634920635
Recall: 0.954337899543379
```

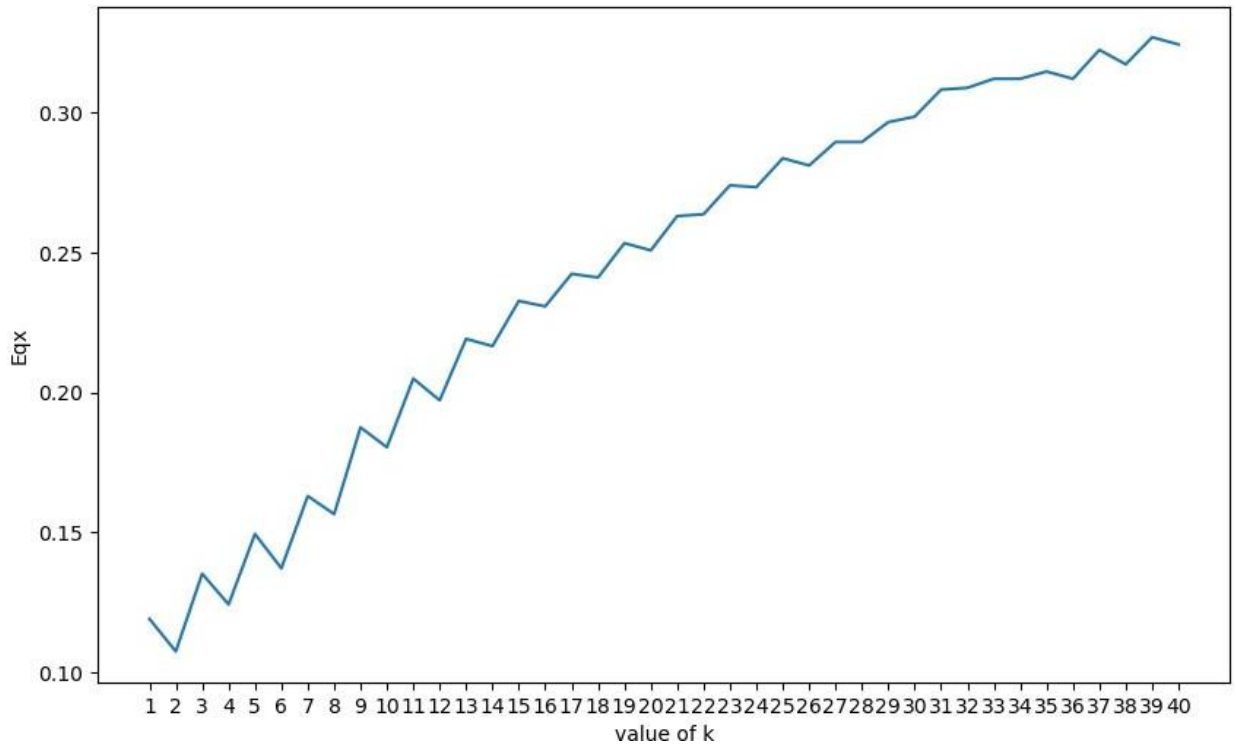
```
error = []

for k in range(1, 41):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    pred_k = knn.predict(X_test)
    error_rate = np.mean(pred_k != y_test)
    error.append(error_rate)

for i, e in enumerate(error, start=1):
    print(f"k={i}: Error Rate={e:.4f}")
```

```
k=1: Error Rate=0.1192
k=2: Error Rate=0.1076
```

```
k=3: Error Rate=0.1353
k=4: Error Rate=0.1244
k=5: Error Rate=0.1495
k=6: Error Rate=0.1372
k=7: Error Rate=0.1630
k=8: Error Rate=0.1566
k=9: Error Rate=0.1875
k=10: Error Rate=0.1804
k=11: Error Rate=0.2049
k=12: Error Rate=0.1972
k=13: Error Rate=0.2191
k=14: Error Rate=0.2165
k=15: Error Rate=0.2326
k=16: Error Rate=0.2307
k=17: Error Rate=0.2423
k=18: Error Rate=0.2410
k=19: Error Rate=0.2532
k=20: Error Rate=0.2506
k=21: Error Rate=0.2629
k=22: Error Rate=0.2635
k=23: Error Rate=0.2738
k=24: Error Rate=0.2732
k=25: Error Rate=0.2835
k=26: Error Rate=0.2809
k=27: Error Rate=0.2893
k=28: Error Rate=0.2893
k=29: Error Rate=0.2964
k=30: Error Rate=0.2983
k=31: Error Rate=0.3080
k=32: Error Rate=0.3086
k=33: Error Rate=0.3119
k=34: Error Rate=0.3119
k=35: Error Rate=0.3144
k=36: Error Rate=0.3119
k=37: Error Rate=0.3222
k=38: Error Rate=0.3170
k=39: Error Rate=0.3267
k=40: Error Rate=0.3241
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.xlabel('value of k')
plt.ylabel('Eqx')
plt.xticks(range(1,41))
plt.plot(range(1,41), error)
```

```
[<matplotlib.lines.Line2D at 0x25298923110>]
```




```
knn = KNeighborsClassifier(2)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.91      0.92      1114
           1       0.79      0.84      0.81       438

    accuracy                           0.89      1552
   macro avg       0.86      0.88      0.87      1552
weighted avg       0.89      0.89      0.89      1552
```

```
cm = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print("Confusion Matrix:\n", cm)
print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
```

```
Confusion Matrix:
 [[1019   95]
 [  72  366]]
Accuracy: 0.8923969072164949
Error Rate: 0.1076030927835051
Precision: 0.7939262472885033
Recall: 0.8356164383561644
```

```
from sklearn.svm import SVC

# Create an SVM classifier
svm_model = SVC(kernel='linear')

svm_model.fit(X_train, y_train)
```

```
SVC(kernel='linear')
```

```
y_pred = svm_model.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print("Confusion Matrix:\n", cm)
print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
```

```
Confusion Matrix:
 [[1078   36]
 [  12  426]]
Accuracy: 0.9690721649484536
Error Rate: 0.030927835051546393
Precision: 0.922077922077922
Recall: 0.9726027397260274
```

`