In [ ]:

In [ ]:

In [1]:

In [2]:

Out[2]:



Practical No:2

Classify the email using the binary classification method. Email Spam detection has two states:

Normal State – Not Spam

Abnormal State – Spam. Use K**-**Nearest Neighbors **and** Support Vector Machine **for** classification. Analyze their performance.



**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.svm **import** SVC

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn **import** metrics

df **=** pd.read\_csv('emails.csv') df

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Email No.** | **the** | **to** | **ect** | **and** | **for** | **of** | **a** | **you** | **hou** | **...** | **connevey** | **jay** | **valued** | **lay** | **infr** |
| **0** | Email  1 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 0 | 0 | ... | 0 | 0 | 0 | 0 |  |
| **1** | Email  2 | 8 | 13 | 24 | 6 | 6 | 2 | 102 | 1 | 27 | ... | 0 | 0 | 0 | 0 |  |
| **2** | Email  3 | 0 | 0 | 1 | 0 | 0 | 0 | 8 | 0 | 0 | ... | 0 | 0 | 0 | 0 |  |
| **3** | Email  4 | 0 | 5 | 22 | 0 | 5 | 1 | 51 | 2 | 10 | ... | 0 | 0 | 0 | 0 |  |
| **4** | Email  5 | 7 | 6 | 17 | 1 | 5 | 2 | 57 | 0 | 9 | ... | 0 | 0 | 0 | 0 |  |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |  |
| **5167** | Email 5168 | 2 | 2 | 2 | 3 | 0 | 0 | 32 | 0 | 0 | ... | 0 | 0 | 0 | 0 |  |
| **5168** | Email 5169 | 35 | 27 | 11 | 2 | 6 | 5 | 151 | 4 | 3 | ... | 0 | 0 | 0 | 0 |  |
| **5169** | Email 5170 | 0 | 0 | 1 | 1 | 0 | 0 | 11 | 0 | 0 | ... | 0 | 0 | 0 | 0 |  |
| **5170** | Email 5171 | 2 | 7 | 1 | 0 | 2 | 1 | 28 | 2 | 0 | ... | 0 | 0 | 0 | 0 |  |
| **5171** | Email | 22 | 24 | 5 | 1 | 6 | 5 | 148 | 8 | 2 | ... | 0 | 0 | 0 | 0 |  |

5172

5172 rows × 3002 columns



In [3]:

df.shape

Out[3]: (5172, 3002)

In [4]:

df.isnull().any()

Out[4]: Email No. False

the False

to False

ect False

and False

...

military False

allowing False

ff False

dry False

Prediction False Length: 3002, dtype: bool

In [5]:

df.drop(columns**=**'Email No.', inplace**=True**) df

Out[5]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **the** | **to** | **ect** | **and** | **for** | **of** | **a** | **you** | **hou** | **in** | **...** | **connevey** | **jay** | **valued** | **lay** | **infrast** |
| **0** | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 |  |
| **1** | 8 | 13 | 24 | 6 | 6 | 2 | 102 | 1 | 27 | 18 | ... | 0 | 0 | 0 | 0 |  |
| **2** | 0 | 0 | 1 | 0 | 0 | 0 | 8 | 0 | 0 | 4 | ... | 0 | 0 | 0 | 0 |  |
| **3** | 0 | 5 | 22 | 0 | 5 | 1 | 51 | 2 | 10 | 1 | ... | 0 | 0 | 0 | 0 |  |
| **4** | 7 | 6 | 17 | 1 | 5 | 2 | 57 | 0 | 9 | 3 | ... | 0 | 0 | 0 | 0 |  |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |  |
| **5167** | 2 | 2 | 2 | 3 | 0 | 0 | 32 | 0 | 0 | 5 | ... | 0 | 0 | 0 | 0 |  |
| **5168** | 35 | 27 | 11 | 2 | 6 | 5 | 151 | 4 | 3 | 23 | ... | 0 | 0 | 0 | 0 |  |
| **5169** | 0 | 0 | 1 | 1 | 0 | 0 | 11 | 0 | 0 | 1 | ... | 0 | 0 | 0 | 0 |  |
| **5170** | 2 | 7 | 1 | 0 | 2 | 1 | 28 | 2 | 0 | 8 | ... | 0 | 0 | 0 | 0 |  |
| **5171** | 22 | 24 | 5 | 1 | 6 | 5 | 148 | 8 | 2 | 23 | ... | 0 | 0 | 0 | 0 |  |

5172 rows × 3001 columns



In [6]:

df.columns

Out[6]: Index(['the', 'to', 'ect', 'and', 'for', 'of', 'a', 'you', 'hou', 'in',

...

'connevey', 'jay', 'valued', 'lay', 'infrastructure', 'military', 'allowing', 'ff', 'dry', 'Prediction'], dtype='object', length=3001)

In [7]:

df.Prediction.unique()

Out[7]: array([0, 1], dtype=int64)

In [8]:

df['Prediction'] **=** df['Prediction'].replace({0:'Not spam',

1:'Spam'})

df

Out[8]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **the** | **to** | **ect** | **and** | **for** | **of** | **a** | **you** | **hou** | **in** | **...** | **connevey** | **jay** | **valued** | **lay** | **infrast** |
| **0** | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 |  |
| **1** | 8 | 13 | 24 | 6 | 6 | 2 | 102 | 1 | 27 | 18 | ... | 0 | 0 | 0 | 0 |  |
| **2** | 0 | 0 | 1 | 0 | 0 | 0 | 8 | 0 | 0 | 4 | ... | 0 | 0 | 0 | 0 |  |
| **3** | 0 | 5 | 22 | 0 | 5 | 1 | 51 | 2 | 10 | 1 | ... | 0 | 0 | 0 | 0 |  |
| **4** | 7 | 6 | 17 | 1 | 5 | 2 | 57 | 0 | 9 | 3 | ... | 0 | 0 | 0 | 0 |  |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |  |
| **5167** | 2 | 2 | 2 | 3 | 0 | 0 | 32 | 0 | 0 | 5 | ... | 0 | 0 | 0 | 0 |  |
| **5168** | 35 | 27 | 11 | 2 | 6 | 5 | 151 | 4 | 3 | 23 | ... | 0 | 0 | 0 | 0 |  |
| **5169** | 0 | 0 | 1 | 1 | 0 | 0 | 11 | 0 | 0 | 1 | ... | 0 | 0 | 0 | 0 |  |
| **5170** | 2 | 7 | 1 | 0 | 2 | 1 | 28 | 2 | 0 | 8 | ... | 0 | 0 | 0 | 0 |  |
| **5171** | 22 | 24 | 5 | 1 | 6 | 5 | 148 | 8 | 2 | 23 | ... | 0 | 0 | 0 | 0 |  |

5172 rows × 3001 columns



KNN

In [9]:

X **=** df.drop(columns**=**'Prediction',axis **=** 1) Y **=**

df['Prediction']

In [10]:

X.columns

Out[10]: Index(['the', 'to', 'ect', 'and', 'for', 'of', 'a', 'you', 'hou', 'in',

...

'enhancements', 'connevey', 'jay', 'valued', 'lay', 'infrastructur

e',

'military', 'allowing', 'ff', 'dry'], dtype='object', length=3000)

In [11]:

Y.head()

Out[11]: 0 Not spam

1. Not spam
2. Not spam
3. Not spam
4. Not spam

Name: Prediction, dtype: object

In [12]:

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(X, Y,

test\_size**=**0.2, random\_state**=**1)

In [13]:

KN **=** KNeighborsClassifier knn **=** KN(n\_neighbors**=**7) knn.fit(x\_train, y\_train)

y\_pred **=** knn.predict(x\_test)

In [14]:

print("Prediction: \n") print(y\_pred)

Prediction:

['Not spam' 'Spam' 'Not spam' ... 'Not spam' 'Not spam' 'Not spam']

In [15]:

M **=** metrics.accuracy\_score(y\_test,y\_pred) print("KNN

accuracy: ", M)

KNN accuracy: 0.8714975845410629

In [16]:

C **=** metrics.confusion\_matrix(y\_test,y\_pred) print("Confusion matrix: ", C)

Confusion matrix: [[635 84]

[ 49 267]]

SVM Classifier

In [17]:

model **=** SVC(C **=** 1)

*# cost C = 1*

model.fit(x\_train, y\_train)

y\_pred **=** model.predict(x\_test)

In [18]:

kc **=** metrics.confusion\_matrix(y\_test, y\_pred) print("SVM accuracy: ", kc)

SVM accuracy: [[700 19]

[189 127]]