Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Ex4 - Classification of Email spam and MNIST data

GitHub Link:

GitHub Link

Colab Links:

4.1 and 4.3

4.2

Aim:

To develop a python program

(i) To classify Emails as Spam or Ham

(ii) To recognize the digits of the MNIST dataset

Using Support Vector Machine (SVM) Model

4.1, 4.3 Classification of Email Spam or Ham using Support Vector Machine (SVM) and Naïve Bayes Algorithm

Clone GitHub Repo For Data

!git clone https://github.com/Ojus999/Machine-Learning-Sem-6.git

Import Dependencies

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn import svm
```

Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Read Data

df = pd.read_csv("/content/Machine-Learning-Sem-6/Ex 4/spambase_csv.csv")

Read First Few Rows

df.head()

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove	word_freq_internet	word_freq_order	word_freq_mail	...	char_freq_%3B	char_fr
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.00	
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.00	
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.01	
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	

5 rows × 58 columns

DataFrame Info

df.info()

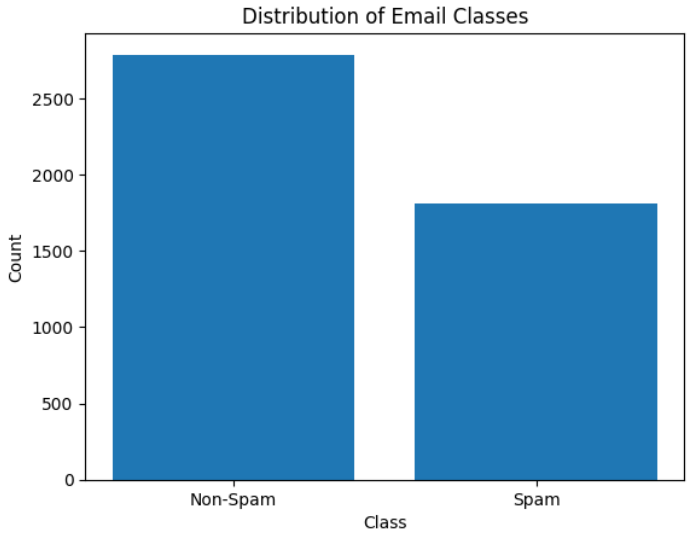
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4601 entries, 0 to 4600
Data columns (total 58 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   word_freq_make         4601 non-null   float64
 1   word_freq_address      4601 non-null   float64
 2   word_freq_all          4601 non-null   float64
 3   word_freq_3d           4601 non-null   float64
 4   word_freq_our          4601 non-null   float64
 5   word_freq_over         4601 non-null   float64
 6   word_freq_remove       4601 non-null   float64
 7   word_freq_internet     4601 non-null   float64
 8   word_freq_order        4601 non-null   float64
 9   word_freq_mail         4601 non-null   float64
 10  word_freq_receive      4601 non-null   float64
 11  word_freq_will         4601 non-null   float64
 12  word_freq_people       4601 non-null   float64
 13  word_freq_report       4601 non-null   float64
 14  word_freq_addresses    4601 non-null   float64
 15  word_freq_free         4601 non-null   float64
 16  word_freq_business     4601 non-null   float64
 17  word_freq_email        4601 non-null   float64
 18  word_freq_you          4601 non-null   float64
 19  word_freq_credit       4601 non-null   float64
 20  word_freq_your         4601 non-null   float64
 21  word_freq_font         4601 non-null   float64
 22  word_freq_000          4601 non-null   float64
 23  word_freq_money        4601 non-null   float64
 24  word_freq_hp           4601 non-null   float64
```

Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Data Visualization

Data Distribution

```
class_counts = df['class'].value_counts()
plt.bar(class_counts.index, class_counts.values)
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Distribution of Email Classes')
plt.xticks(class_counts.index, ['Non-Spam', 'Spam'])
plt.show()
```



Correlation Heatmap

```
correlation_matrix = df.corr()

# Create a heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```

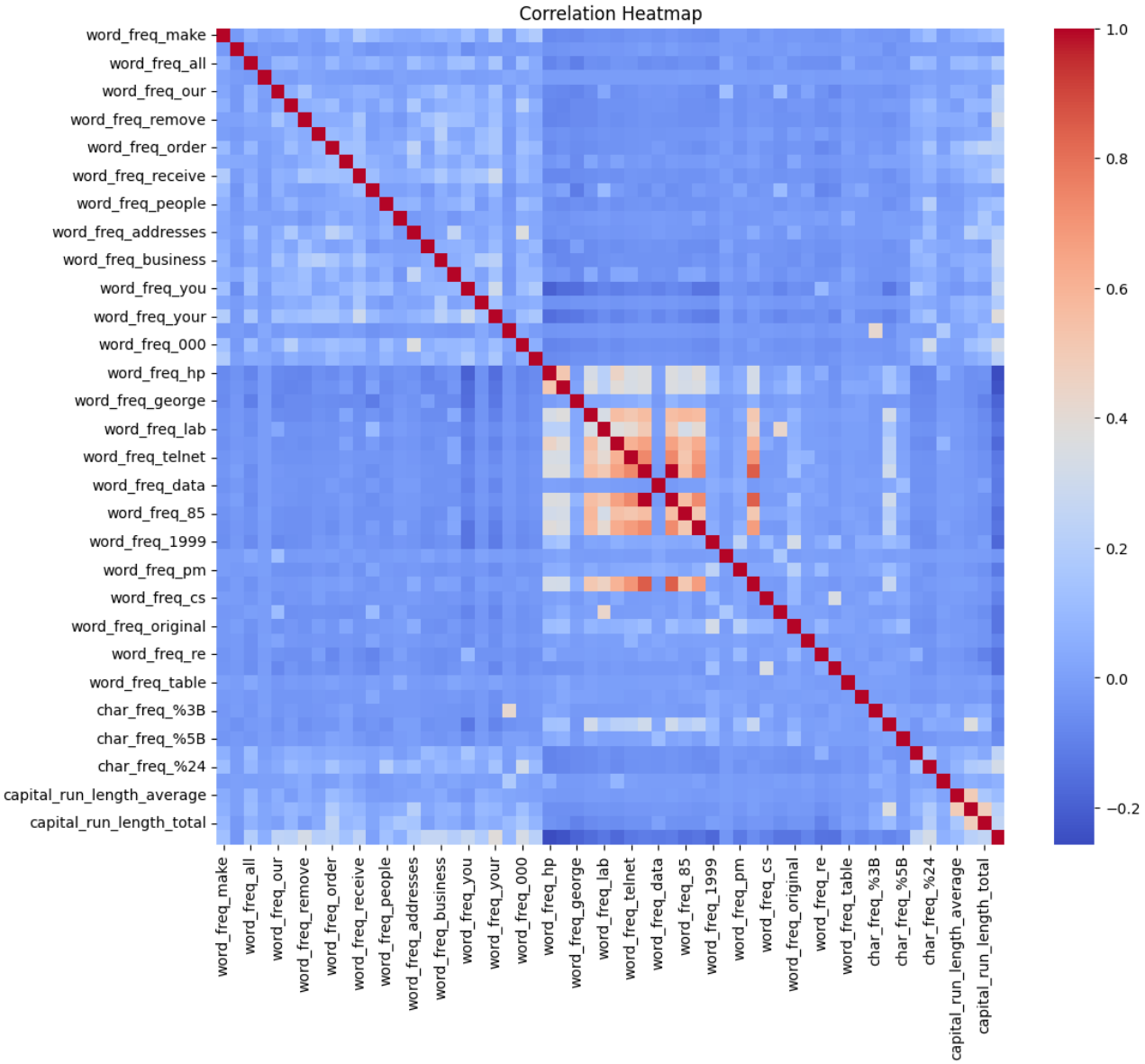
Machine Learning Lab

UCS2612

Classification of Email spam and MNIST data

Ex. No: 4 Y.V.Ojus

4-3-24 3122 21 5001 125

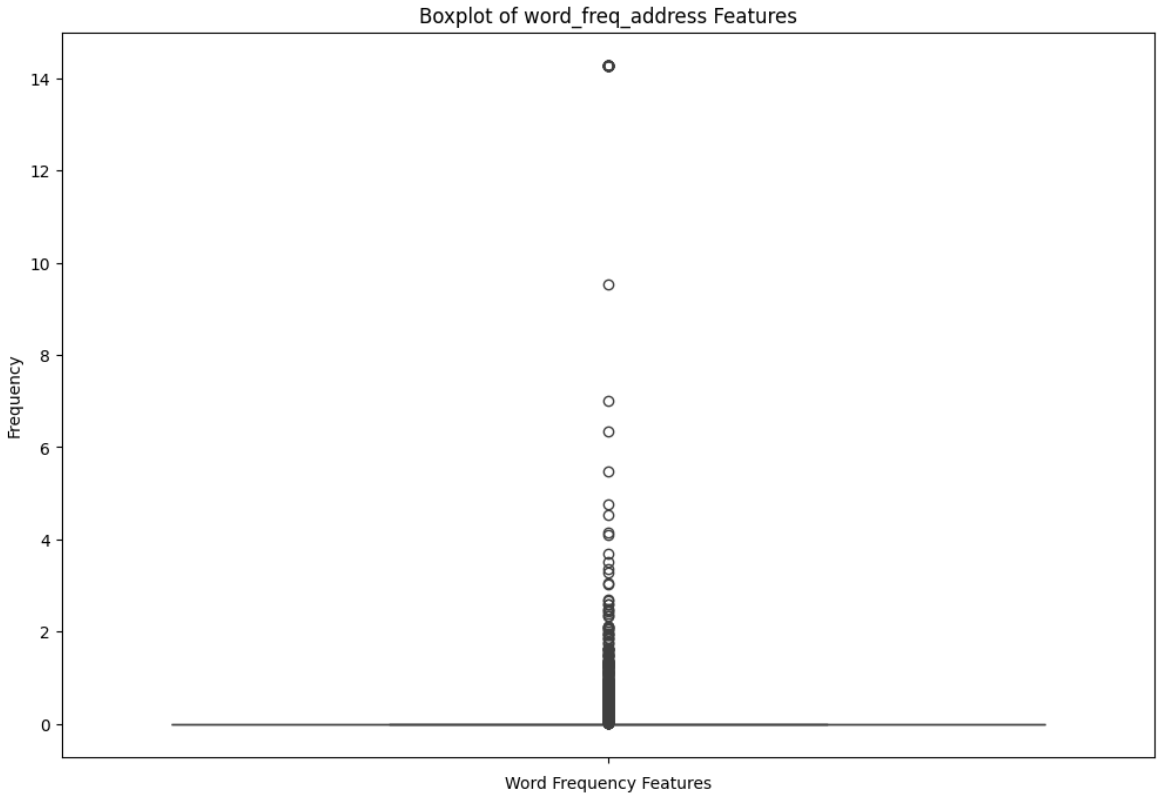


Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Histograms & Boxplot

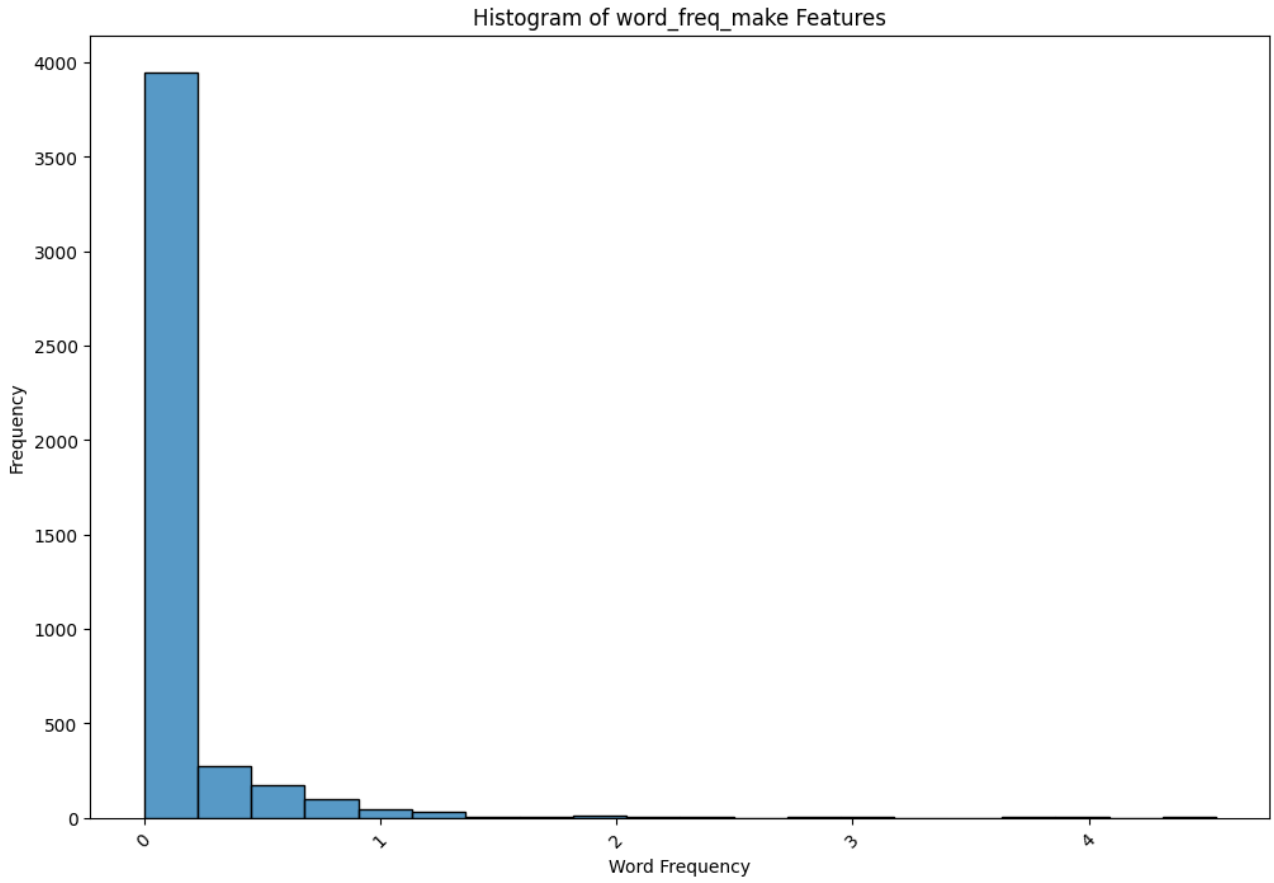
```
word_freq_columns = df.loc[:, 'word_freq_make':'word_freq_conference'].columns
index = 1

# Plot boxplots for word frequency features
plt.figure(figsize=(12, 8))
sns.boxplot(data=df[word_freq_columns[index]])
plt.xlabel('Word Frequency Features')
plt.ylabel('Frequency')
plt.title(f'Boxplot of {word_freq_columns[index]} Features')
plt.xticks(rotation=45)
plt.show()
```



```
word_freq_columns = df.loc[:, 'word_freq_make':'word_freq_conference'].columns
index = 0

# Plot histogram for the selected word frequency feature
plt.figure(figsize=(12, 8))
sns.histplot(data=df[word_freq_columns[index]], bins=20)  # Adjust bins and kde as needed
plt.xlabel('Word Frequency')
plt.ylabel('Frequency')
plt.title(f'Histogram of {word_freq_columns[index]} Features')
plt.xticks(rotation=45)
plt.show()
```



Null Values

df.isnull().sum()

word_freq_make	0
word_freq_address	0
word_freq_all	0
word_freq_3d	0
word_freq_our	0
word_freq_over	0
word_freq_remove	0
word_freq_internet	0
word_freq_order	0
word_freq_mail	0
word_freq_receive	0
word_freq_will	0
word_freq_people	0
word_freq_report	0
word_freq_addresses	0
word_freq_free	0
word_freq_business	0
word_freq_email	0
word_freq_you	0
word_freq_credit	0
word_freq_your	0
word_freq_font	0
word_freq_000	0
word_freq_money	0

Statistics of Data

df.describe().transpose()

	count	mean	std	min	25%	50%	75%	max
word_freq_make	4601.0	0.104553	0.305358	0.0	0.000	0.000	0.000	4.540
word_freq_address	4601.0	0.213015	1.290575	0.0	0.000	0.000	0.000	14.280
word_freq_all	4601.0	0.280656	0.504143	0.0	0.000	0.000	0.420	5.100
word_freq_3d	4601.0	0.065425	1.395151	0.0	0.000	0.000	0.000	42.810
word_freq_our	4601.0	0.312223	0.672513	0.0	0.000	0.000	0.380	10.000
word_freq_over	4601.0	0.095901	0.273824	0.0	0.000	0.000	0.000	5.880
word_freq_remove	4601.0	0.114208	0.391441	0.0	0.000	0.000	0.000	7.270
word_freq_internet	4601.0	0.105295	0.401071	0.0	0.000	0.000	0.000	11.110
word_freq_order	4601.0	0.090067	0.278616	0.0	0.000	0.000	0.000	5.260
word_freq_mail	4601.0	0.239413	0.644755	0.0	0.000	0.000	0.160	18.180
word_freq_receive	4601.0	0.059824	0.201545	0.0	0.000	0.000	0.000	2.610
word_freq_will	4601.0	0.541702	0.861698	0.0	0.000	0.100	0.800	9.670
word_freq_people	4601.0	0.093930	0.301036	0.0	0.000	0.000	0.000	5.550
word_freq_report	4601.0	0.058626	0.335184	0.0	0.000	0.000	0.000	10.000
word_freq_addresses	4601.0	0.049205	0.258843	0.0	0.000	0.000	0.000	4.410
word_freq_free	4601.0	0.248848	0.825792	0.0	0.000	0.000	0.100	20.000
word_freq_business	4601.0	0.142586	0.444055	0.0	0.000	0.000	0.000	7.140
word_freq_email	4601.0	0.184745	0.531122	0.0	0.000	0.000	0.000	9.090
word_freq_you	4601.0	1.662100	1.775481	0.0	0.000	1.310	2.640	18.750

Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Building Model – SVM

Define Train And Target Columns

```
X = df.loc[:,'word_freq_make':'capital_run_length_total']
X
```

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove	word_freq_internet	word_freq_order	word_freq_mail	...	word_freq_conference	char_freq_%3B	ch
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.0	0.000	
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.0	0.010	
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.000	
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.000	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4596	0.31	0.00	0.62	0.0	0.00	0.31	0.00	0.00	0.00	0.00	...	0.0	0.000	
4597	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	
4598	0.30	0.00	0.30	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.102	
4599	0.96	0.00	0.00	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	
4600	0.00	0.00	0.65	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	

4601 rows × 57 columns

```
y = df['class']
```

Train Test Split

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,random_state=109)
```

Perform Feature Scaling – Standardization

```
# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Fit And Predict

```
kernels = ['linear','poly','rbf','sigmoid']

for ker in kernels:
  #Create a svm Classifier
  clf = svm.SVC(kernel=ker) # Linear Kernel

  #Train the model using the training sets
  clf.fit(X_train, y_train)

  #Predict the response for test dataset
  y_pred = clf.predict(X_test)

  # Model Accuracy: how often is the classifier correct?
  accuracy = metrics.accuracy_score(y_test, y_pred)
  print(f"Kernel: {ker}")
  print("Accuracy:", accuracy)

  # Model Precision: what percentage of positive tuples are labeled as such?
  precision = metrics.precision_score(y_test, y_pred)
  print("Precision:", precision)

  # Model Recall: what percentage of positive tuples are labelled as such?
  recall = metrics.recall_score(y_test, y_pred)
  print("Recall:", recall)

  print()
```

Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

```
Kernel: linear
Accuracy: 0.9203475742215785
Precision: 0.9023508137432188
Recall: 0.8990990990990991

Kernel: poly
Accuracy: 0.7863866763215062
Precision: 0.9513888888888888
Recall: 0.4936936936936937

Kernel: rbf
Accuracy: 0.9217958001448225
Precision: 0.9146567717996289
Recall: 0.8882882882882883

Kernel: sigmoid
Accuracy: 0.8776249094858798
Precision: 0.8547794117647058
Recall: 0.8378378378378378
```

```
#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

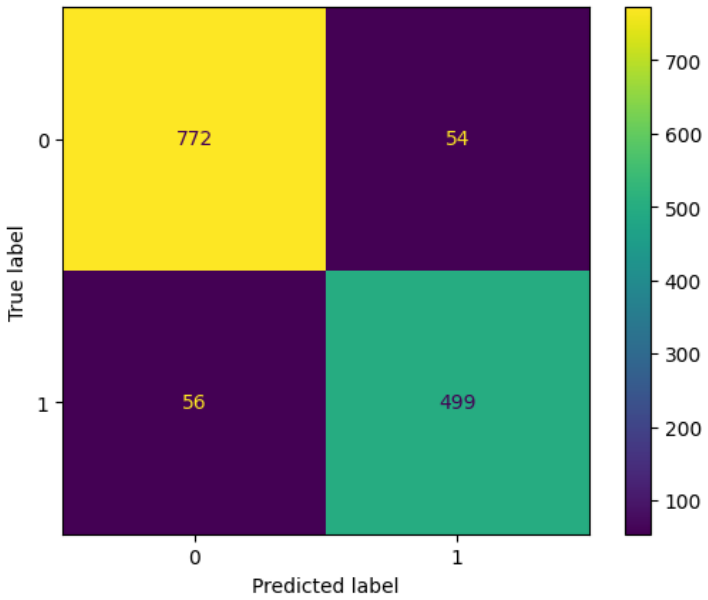
Visualize Output

Classification Report

	precision	recall	f1-score	support
0	0.93	0.93	0.93	826
1	0.90	0.90	0.90	555
accuracy			0.92	1381
macro avg	0.92	0.92	0.92	1381
weighted avg	0.92	0.92	0.92	1381

Confusion Matrix

```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
cm_display = metrics.ConfusionMatrixDisplay(cm)
cm_display.plot()
```



Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Building Model - Naive Bayes

Train Test Split

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,random_state=109)
```

Feature Scaling

```
# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Initialize Model

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

Fit And Predict

```
y_pred = gnb.fit(X_train, y_train).predict(X_test)
```

Accuracy

```
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8037653874004345
```

Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Precision & Recall

```
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Precision: 0.6815856777493606
Recall: 0.9603603603603603
```

Visualizing Output

Classification Report

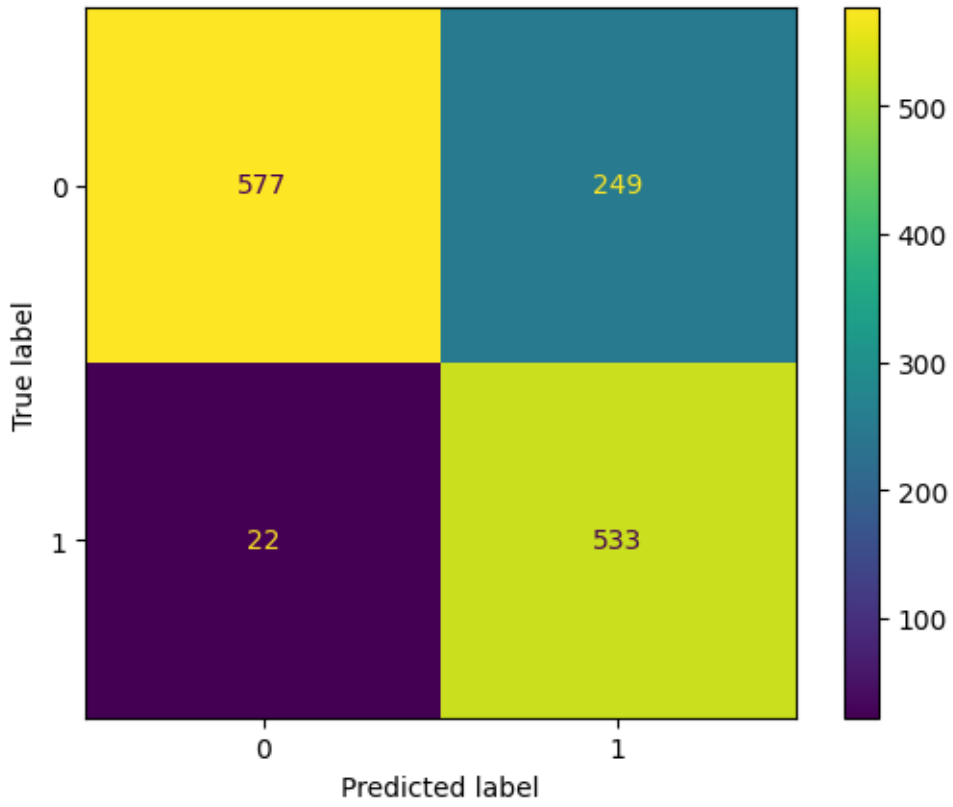
```
# Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.96	0.70	0.81	826
1	0.68	0.96	0.80	555
accuracy			0.80	1381
macro avg	0.82	0.83	0.80	1381
weighted avg	0.85	0.80	0.80	1381

Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Confusion Matrix

```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
cm_display = metrics.ConfusionMatrixDisplay(cm)
cm_display.plot()
```



Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

4.2 Classification of MNIST dataset using Support Vector Machine (SVM)

Clone Repo

```
!git clone https://github.com/Ojus999/Machine-Learning-Sem-6.git
```

Import Dependencies

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, svm, metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Loading the Dataset

```
def load_mnist_images(path):
    with open(path, 'rb') as f:
        data = np.frombuffer(f.read(), dtype=np.uint8, offset=16)
    return data.reshape(-1, 28*28)

def load_mnist_labels(path):
    with open(path, 'rb') as f:
        data = np.frombuffer(f.read(), dtype=np.uint8, offset=8)
    return data

X_train = load_mnist_images('/content/Machine-Learning-Sem-6/Ex 4/mnist/train-images-idx3-ubyte/train-images.idx3-ubyte')
y_train = load_mnist_labels('/content/Machine-Learning-Sem-6/Ex 4/mnist/train-labels-idx1-ubyte/train-labels.idx1-ubyte')
X_test = load_mnist_images('/content/Machine-Learning-Sem-6/Ex 4/mnist/t10k-images-idx3-ubyte/t10k-images.idx3-ubyte')
y_test = load_mnist_labels('/content/Machine-Learning-Sem-6/Ex 4/mnist/t10k-labels-idx1-ubyte/t10k-labels.idx1-ubyte')
```

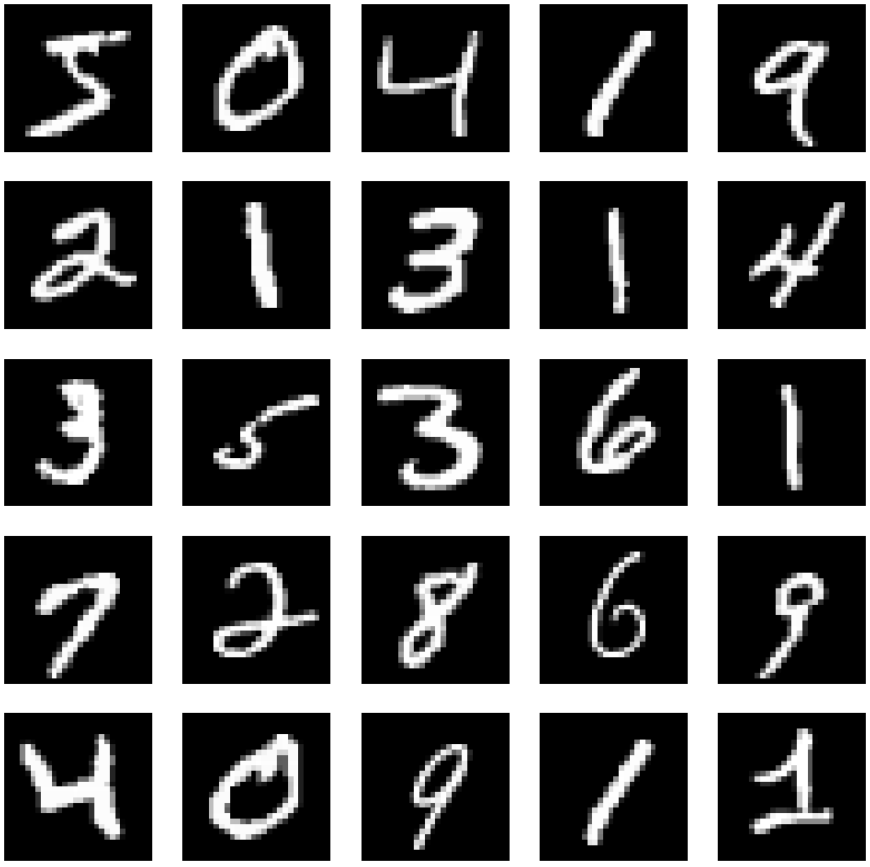
Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Pre-Processing the Data

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Exploratory Data Analysis:

```
# Visualization of some samples from the dataset
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.imshow(X_train[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
plt.show()
```



Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Train Test Split

```
#Split the data into training, testing, and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1,
random_state=42)
```

Train the Model

```
# Train the model
svm_model = svm.SVC(kernel='rbf', C=10, gamma='scale')
svm_model.fit(X_train, y_train)
```

Test the Model

```
#Test the model
y_pred = svm_model.predict(X_test)
```

Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Measure Performance

#Measure the performance of the trained model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)

```
Accuracy: 0.9841
Confusion Matrix:
 [[ 974    0    1    0    0    2    0    1    2    0]
 [   0 1129    3    0    0    1    0    1    1    0]
 [   5    1 1013    1    1    0    2    5    4    0]
 [   0    0    3  995    0    2    0    4    3    3]
 [   1    0    3    0  965    0    2    0    0   11]
 [   2    0    0    9    1  873    3    0    2    2]
 [   3    2    0    0    2    3  946    0    2    0]
 [   0    3    8    2    1    0    0 1007    0    7]
 [   1    0    2    3    1    2    1    2  959    3]
 [   1    3    0    7    7    3    1    6    1  980]]
Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.98      0.99      0.98      1010
           4       0.99      0.98      0.98       982
           5       0.99      0.98      0.98       892
           6       0.99      0.99      0.99       958
           7       0.98      0.98      0.98      1028
           8       0.98      0.98      0.98       974
           9       0.97      0.97      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

Ex. No: 4 Y.V.Ojus
4-3-24 3122 21 5001 125

Visualize Confusion Matrix

```
# Visualize confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, cmap='Blues')
plt.colorbar()
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```