UCS2612 Machine Learning Laboratory

A4 – Classification of Email spam and MNIST data using Support Vector Machines

Name: C B Ananya
Reg No: 3122215001010

GitHub Main Branch Link:

https://github.com/CB-Ananya/ML-Lab

EMAIL CLASSIFICATION:

4. a Question:

Download the Email spam dataset from the link given below:

https://www.kaggle.com/datasets/somesh24/spambase

Develop a python program to classify Emails as Spam or Ham using **Support Vector Machine (SVM) Model.**

Visualize the features from the dataset and interpret the results obtained by the model using Matplotlib library.

4. c Question:

Classification of Email Spam or Ham using **Naïve Bayes Algorithm**

Build the Naïve Bayes model for the classification of Email as Spam or Ham.

Google Colab Link:

https://drive.google.com/file/d/1n8XROGsLxglNAFkwY-_qnW3566RDpxFO/view?usp=sharing

CODE and OUTPUT:

Classfication of Email Spam Using Support Vector Machines

Import Necessary Libraries

In [ ]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import seaborn as sns
```

Load data, split, plot

In [ ]:
```
# Load data from CSV file using pandas
data = pd.read_csv("spambase_csv.csv")
print(data)
# Split data into features (X) and target variable (y)
X = data.drop(columns=['class'])
y = data['class']
# Plot each instance based on target label
spam_data = data[data['class'] == 1]
non_spam_data = data[data['class'] == 0]
```

```
      word_freq_make  word_freq_address  word_freq_all  word_freq_3d  \
0               0.00               0.64           0.64           0.0   
1               0.21               0.28           0.50           0.0   
2               0.06               0.00           0.71           0.0   
3               0.00               0.00           0.00           0.0   
4               0.00               0.00           0.00           0.0   
...              ...                ...            ...           ...   
4596            0.31               0.00           0.62           0.0   
4597            0.00               0.00           0.00           0.0   
4598            0.30               0.00           0.30           0.0   
4599            0.96               0.00           0.00           0.0   
4600            0.00               0.00           0.65           0.0   

      word_freq_our  word_freq_over  word_freq_remove  word_freq_internet  \
0              0.32            0.00              0.00                0.00   
1              0.14            0.28              0.21                0.07   
2              1.23            0.19              0.19                0.12   
3              0.63            0.00              0.31                0.63   
4              0.63            0.00              0.31                0.63   
...             ...             ...               ...                 ...   
4596           0.00            0.31              0.00                0.00   
4597           0.00            0.00              0.00                0.00   
4598           0.00            0.00              0.00                0.00   
4599           0.32            0.00              0.00                0.00   
4600           0.00            0.00              0.00                0.00   

      word_freq_order  word_freq_mail  ...  char_freq_%3B  char_freq_%28  \
0                0.00            0.00  ...          0.000          0.000   
1                0.00            0.94  ...          0.000          0.132   
2                0.64            0.25  ...          0.010          0.143   
3                0.31            0.63  ...          0.000          0.137   
4                0.31            0.63  ...          0.000          0.135   
...               ...             ...  ...            ...            ...   
4596             0.00            0.00  ...          0.000          0.232   
4597             0.00            0.00  ...          0.000          0.000   
4598             0.00            0.00  ...          0.102          0.718   
4599             0.00            0.00  ...          0.000          0.057   
4600             0.00            0.00  ...          0.000          0.000   

      char_freq_%5B  char_freq_%21  char_freq_%24  char_freq_%23  \
0               0.0          0.778          0.000          0.000   
1               0.0          0.372          0.180          0.048   
2               0.0          0.276          0.184          0.010   
3               0.0          0.137          0.000          0.000   
4               0.0          0.135          0.000          0.000   
...             ...            ...            ...            ...   
4596            0.0          0.000          0.000          0.000   
4597            0.0          0.353          0.000          0.000   
4598            0.0          0.000          0.000          0.000   
4599            0.0          0.000          0.000          0.000   
4600            0.0          0.125          0.000          0.000   

      capital_run_length_average  capital_run_length_longest  \
0                          3.756                          61   
1                          5.114                         101   
2                          9.821                         485   
3                          3.537                          40   
4                          3.537                          40   
...                          ...                         ...   
4596                       1.142                           3   
```

```
4597                         1.555                              4
4598                         1.404                              6
4599                         1.147                              5
4600                         1.250                              5

      capital_run_length_total  class
0                          278      1
1                         1028      1
2                         2259      1
3                          191      1
4                          191      1
...                        ...    ...
4596                        88      0
4597                        14      0
4598                       118      0
4599                        78      0
4600                        40      0

[4601 rows x 58 columns]
```

In [ ]:

```
print("\n\nThe Shape Of the dataset is : ",data.shape)
print("\n\nThe Attributes of the dataset is : ",data.columns)
print("The Number of Missing Values in the dataset\n")
print(data.isnull().sum())
```

```
The Shape Of the dataset is :  (4601, 58)


The Attributes of the dataset is :  Index(['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_receive', 'word_freq_will', 'word_freq_people',
       'word_freq_report', 'word_freq_addresses', 'word_freq_free',
       'word_freq_business', 'word_freq_email', 'word_freq_you',
       'word_freq_credit', 'word_freq_your', 'word_freq_font', 'word_freq_000',
       'word_freq_money', 'word_freq_hp', 'word_freq_hpl', 'word_freq_george',
       'word_freq_650', 'word_freq_lab', 'word_freq_labs', 'word_freq_telnet',
       'word_freq_857', 'word_freq_data', 'word_freq_415', 'word_freq_85',
       'word_freq_technology', 'word_freq_1999', 'word_freq_parts',
       'word_freq_pm', 'word_freq_direct', 'word_freq_cs', 'word_freq_meeting',
       'word_freq_original', 'word_freq_project', 'word_freq_re',
       'word_freq_edu', 'word_freq_table', 'word_freq_conference',
       'char_freq_%3B', 'char_freq_%28', 'char_freq_%5B', 'char_freq_%21',
       'char_freq_%24', 'char_freq_%23', 'capital_run_length_average',
       'capital_run_length_longest', 'capital_run_length_total', 'class'],
      dtype='object')
The Number of Missing Values in the dataset

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
word_freq_hp                  0
word_freq_hpl                 0
word_freq_george              0
word_freq_650                 0
word_freq_lab                 0
word_freq_labs                0
word_freq_telnet              0
word_freq_857                 0
word_freq_data                0
word_freq_415                 0
word_freq_85                  0
```

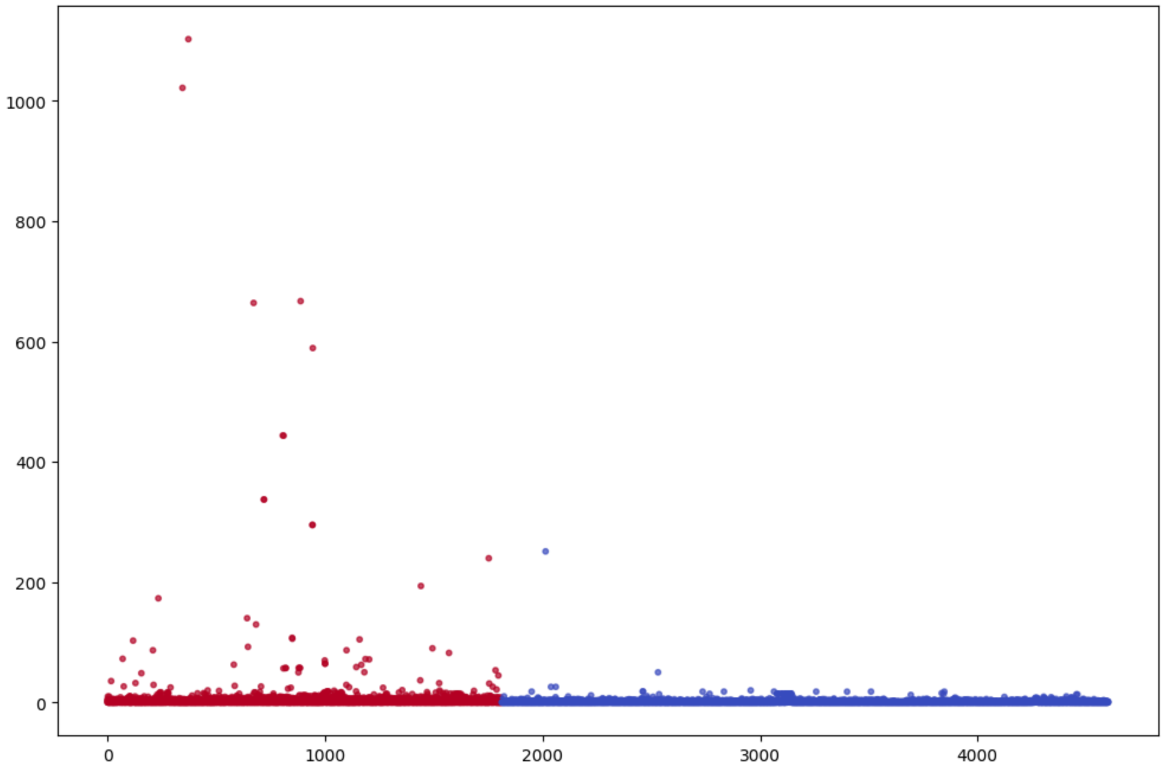
```
word_freq_technology          0
word_freq_1999                0
word_freq_parts               0
word_freq_pm                  0
word_freq_direct              0
word_freq_cs                  0
word_freq_meeting             0
word_freq_original            0
word_freq_project             0
word_freq_re                  0
word_freq_edu                 0
word_freq_table               0
word_freq_conference          0
char_freq_%3B                 0
char_freq_%28                 0
char_freq_%5B                 0
char_freq_%21                 0
char_freq_%24                 0
char_freq_%23                 0
capital_run_length_average    0
capital_run_length_longest    0
capital_run_length_total      0
class                         0
dtype: int64
```

In [ ]:

```
# Plot each instance with feature values, color-coded by class (spam or ham)
plt.figure(figsize=(12, 8))
plt.scatter(data.index, data['capital_run_length_average'], c=data['class'], cma
plt.scatter(data.index, data['word_freq_address'], c=data['class'], cmap='coolwa
```

Out[ ]:

```
<matplotlib.collections.PathCollection at 0x79d8eae284c0>
```



In [ ]:

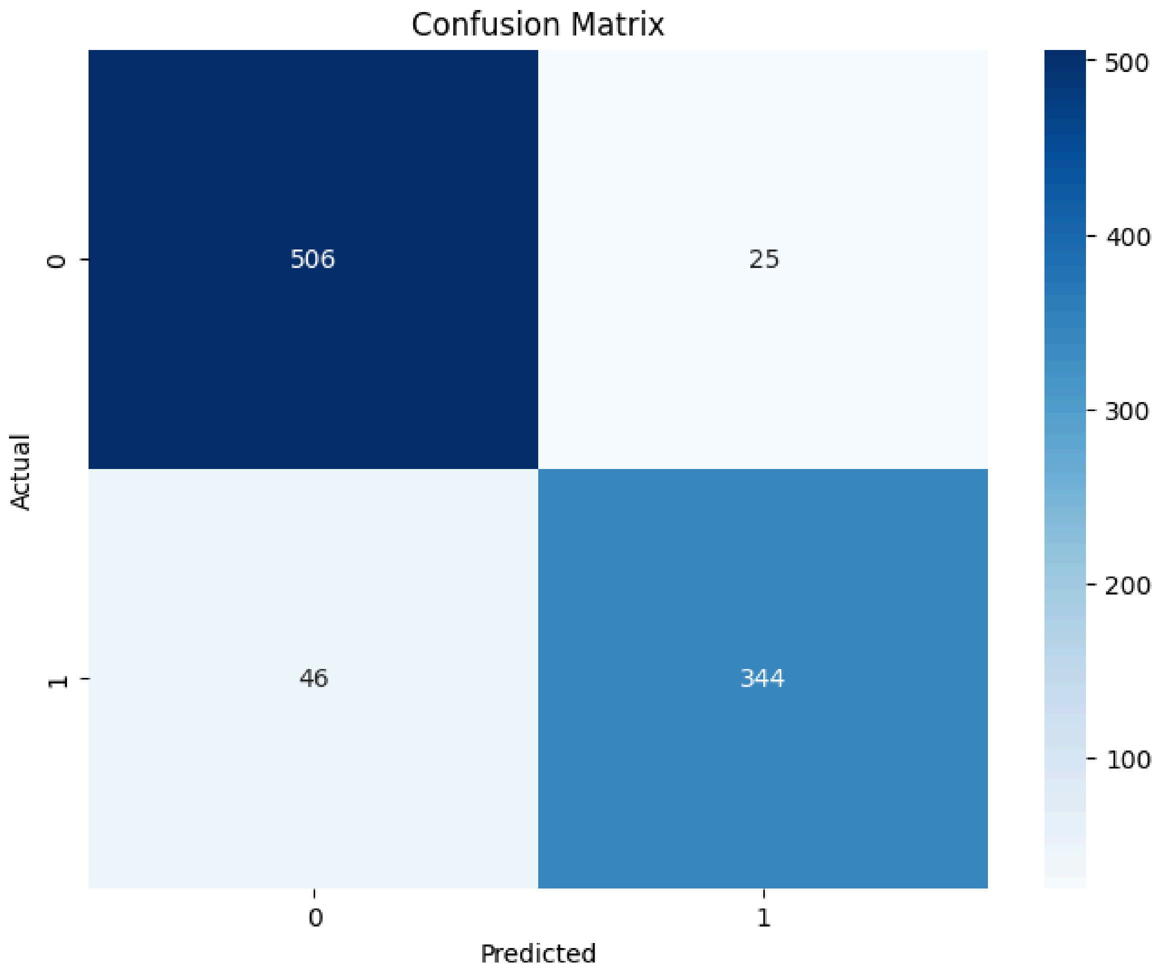
```
#Test Train Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

1. Linear Kernel

In [ ]:

```
#1.Linear Kernel
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)
y_pred = svm_linear.predict(X_test)
accuracy_linear = svm_linear.score(X_test, y_test)
print("Accuracy (Linear Kernel):", accuracy_linear)
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy (Linear Kernel): 0.9229098805646037
```



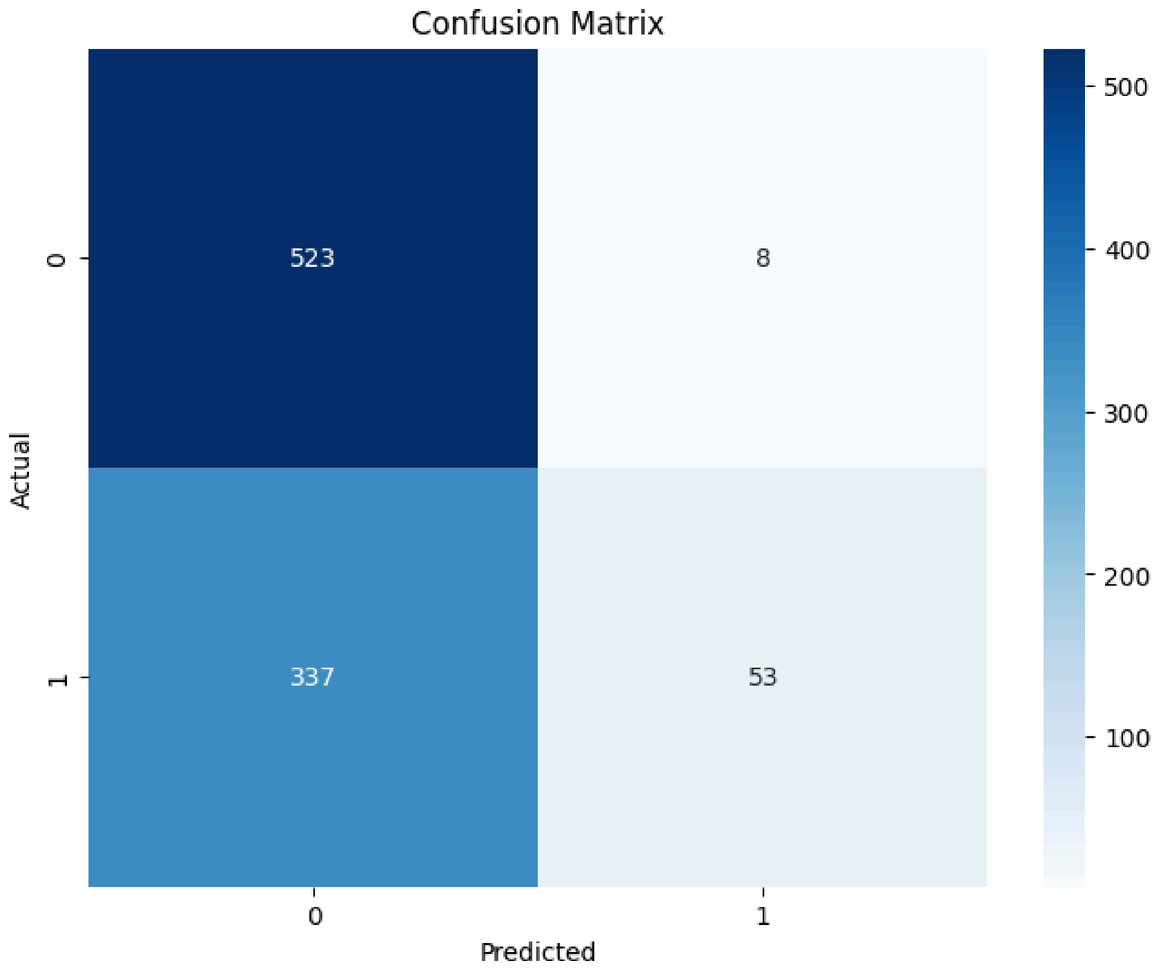
2. Polynomial Kernel

In [ ]:

```
#2.Polynomial Kernel
svm_poly = SVC(kernel='poly')
svm_poly.fit(X_train, y_train)
y_pred = svm_poly.predict(X_test)
accuracy_poly = svm_poly.score(X_test, y_test)
print("Accuracy (Polynomial Kernel):", accuracy_poly)
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```python
plt.title("Confusion Matrix")
plt.show()
```

Accuracy (Polynomial Kernel): 0.6254071661237784

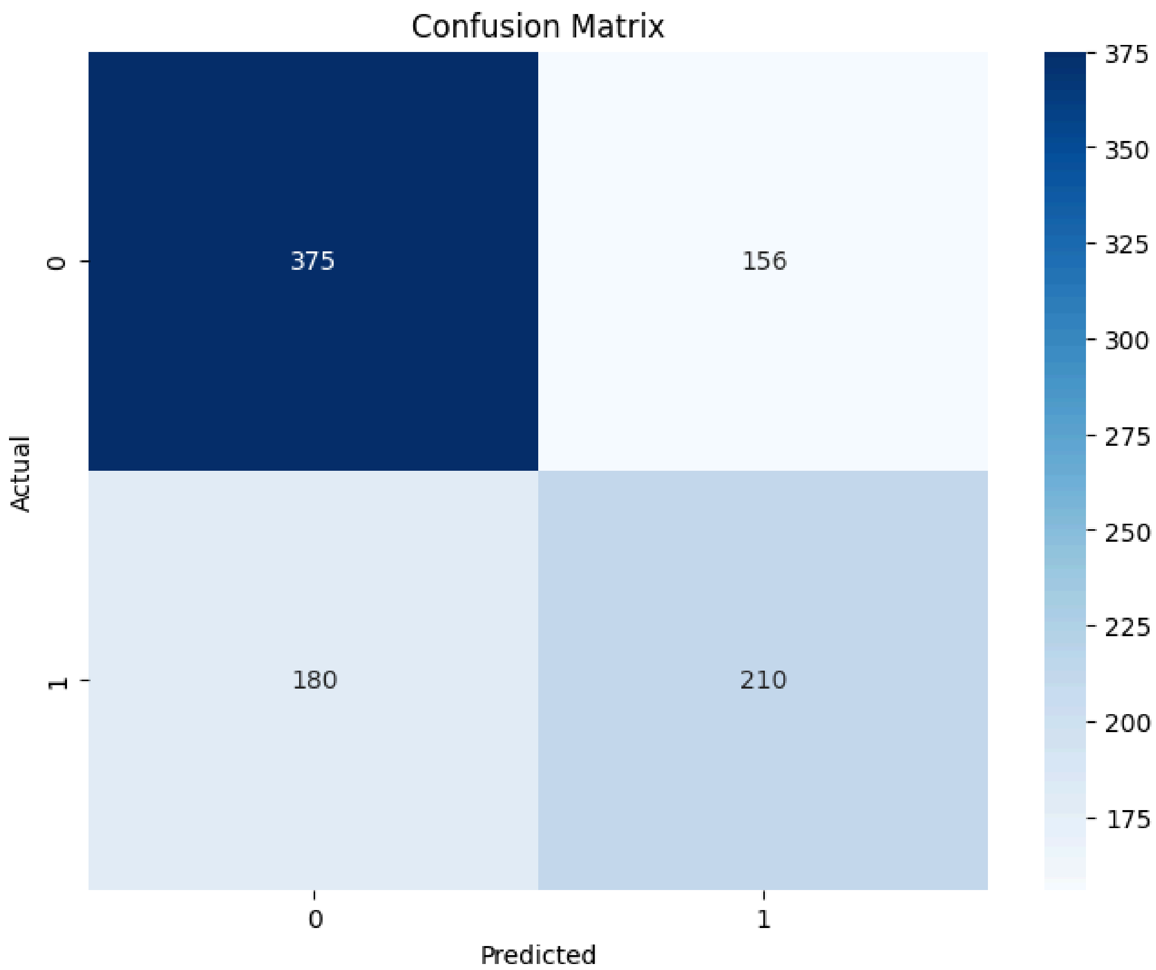


3. Sigmoid Kernel

In [ ]:

```python
#3.Sigmoid Kernel
svm_sigmoid = SVC(kernel='sigmoid')
svm_sigmoid.fit(X_train, y_train)
y_pred = svm_sigmoid.predict(X_test)
accuracy_sigmoid = svm_sigmoid.score(X_test, y_test)
print("Accuracy (Sigmoid Kernel):", accuracy_sigmoid)
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

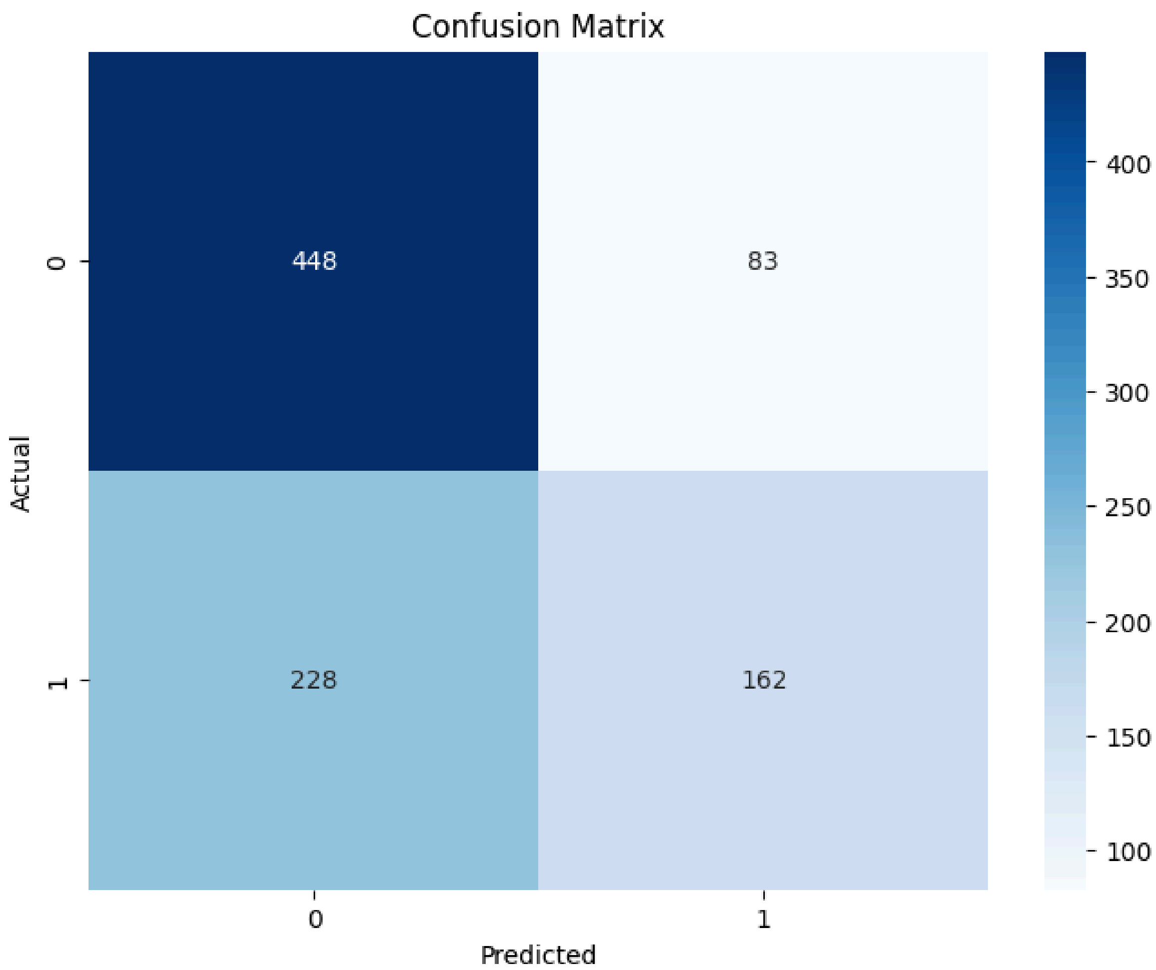
Accuracy (Sigmoid Kernel): 0.6351791530944625



4. RBF Kernel

In [ ]:
```
# 4.RBF Kernel
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)
y_pred = svm_rbf.predict(X_test)
accuracy_rbf = svm_rbf.score(X_test, y_test)
print("Accuracy (RBF Kernel):", accuracy_rbf)
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy (RBF Kernel): 0.6623235613463626
```



Accuracy:

```
In [ ]: print("Accuracy (Linear Kernel)     : ", accuracy_linear*100)
        print("Accuracy (Polynomial Kernel) : ", accuracy_poly*100)
        print("Accuracy (Sigmoid Kernel)    : ", accuracy_sigmoid*100)
        print("Accuracy (RBF Kernel)    : ", accuracy_rbf*100)
```

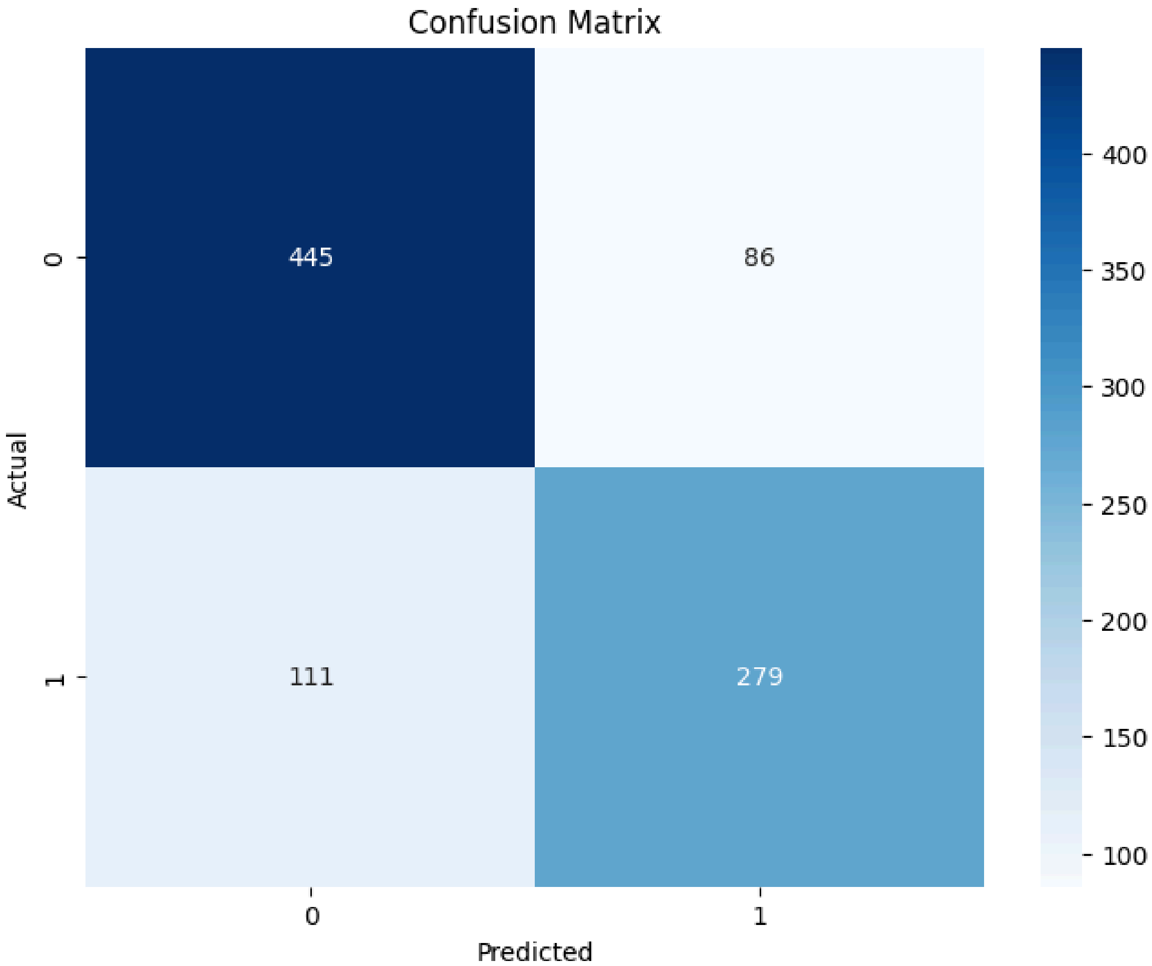
```
Accuracy (Linear Kernel)     :  92.29098805646036
Accuracy (Polynomial Kernel) :  62.54071661237784
Accuracy (Sigmoid Kernel)    :  63.51791530944625
Accuracy (RBF Kernel)    :  66.23235613463626
```

Classification of Email Spam or Ham using Naïve Bayes Algorithm

```
In [ ]: from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.naive_bayes import MultinomialNB

        # Naive Bayes model
        naive_bayes_model = MultinomialNB()
        naive_bayes_model.fit(X_train, y_train)

        # Predict the labels of test data
        y_pred = naive_bayes_model.predict(X_test)

        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)
        print("Accuracy:", accuracy)
```

```python
# Generate and plot confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.7861020629750272
```



MNIST Data – Digit Recognition using Support Vector Machines

4. b

Question:

Download the MNIST dataset from the link given below:
https://archive.ics.uci.edu/dataset/683/mnist+database+of+handwritten+digits

THE MNIST DATABASE:
http://yann.lecun.com/exdb/mnist/

Kaggle: https://www.kaggle.com/datasets/hojjatk/mnist-dataset/data

This is a database of 70,000 handwritten digits (10 class labels) with each example represented as an image of 28 x 28 gray-scale pixels.

Develop a python program to recognize the digits using Support Vector Machine (SVM) Model.

Visualize the features from the dataset and interpret the results obtained by the model using Matplotlib library. [CO1, K3]

Use the following steps to do implementation:
1. Loading the dataset.
2. Pre-Processing the data (Handling missing values, Encoding, Normalization, Standardization).
3. Exploratory Data Analysis.
4. Feature Engineering Techniques.
5. Split the data into training, testing and validation sets.
6. Train the model.
7. Test the model.
8. Measure the performance of the trained model.
9. Represent the results using graphs.

Google Colab link: https://drive.google.com/file/d/1vPfU-ArWwKy6rNbFy26j7cSFmM_41O12/view?usp=sharing

CODE and OUTPUT:

Classification of MNIST data using Support Vector Machines

Import Necessary Libraries

In [ ]:
```
# Import necessary libraries
import numpy as np
from skimage import io, color, exposure, feature
from skimage.filters import gaussian
from skimage.segmentation import slic
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,confusion_matrix
import numpy as np
from tensorflow.keras import datasets
import seaborn as sns
```

Load the dataset

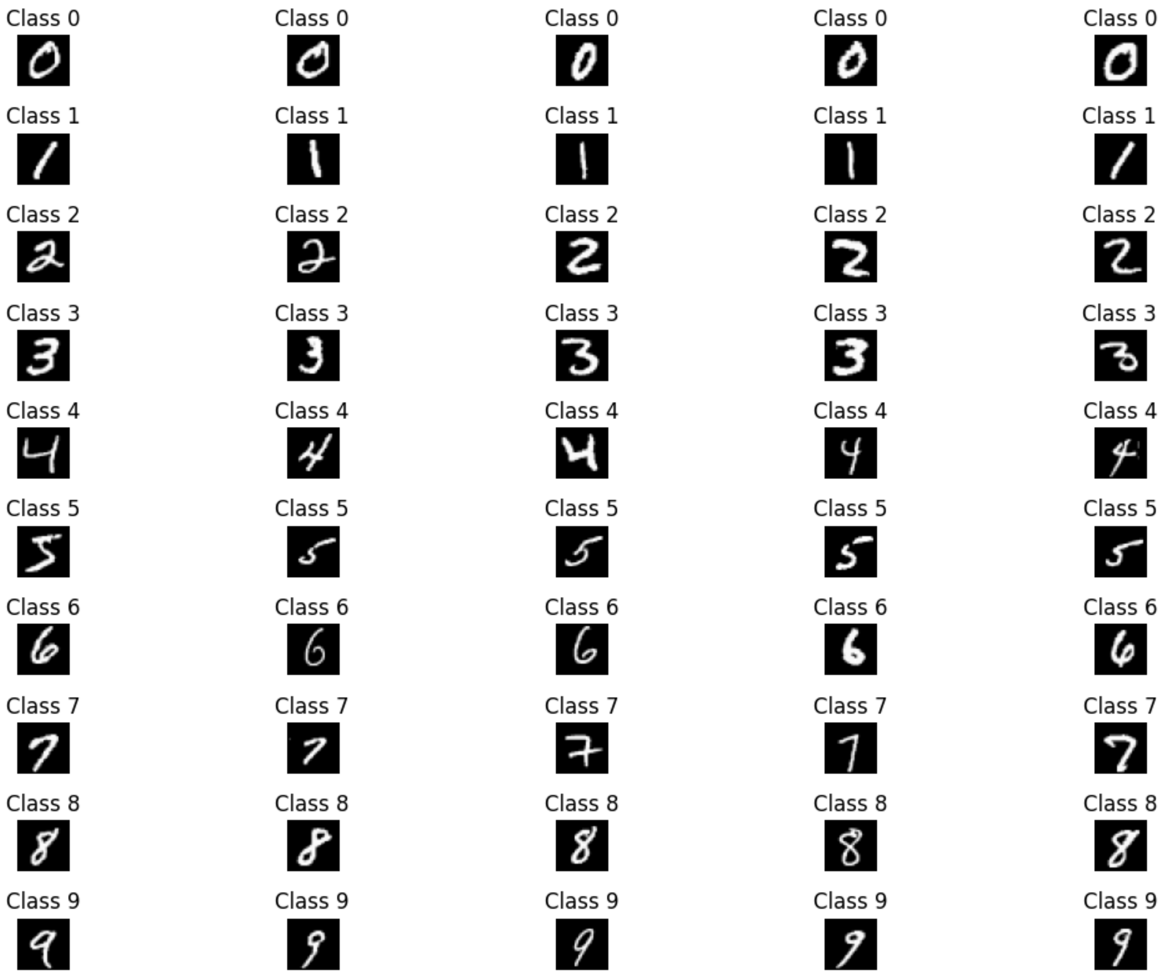
In [ ]:
```
#Loading dataset
(x_train, y_train), (x_test, y_test) = datasets.mnist.load_data()

print("\nThe Shape Of The Train dataset : ",x_train.shape)
print("\nThe Shape Of The Train dataset : ",x_test.shape)

x_train = x_train.reshape(x_train.shape[0], -1)
x_test = x_test.reshape(x_test.shape[0], -1)

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
s/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step

The Shape Of The Train dataset :  (60000, 28, 28)

The Shape Of The Train dataset :  (10000, 28, 28)
```

Display Samples for Each class

In [ ]:
```
import matplotlib.pyplot as plt

# Display samples for each class
num_classes = 10
num_samples_per_class = 5

plt.figure(figsize=(12, 8))
for i in range(num_classes):
    samples_for_class = x_train[y_train == i][:num_samples_per_class]
    for j, sample in enumerate(samples_for_class):
        plt.subplot(num_classes, num_samples_per_class, i * num_samples_per_clas
        plt.imshow(sample.reshape(28, 28), cmap='gray')
        plt.title(f"Class {i}")
        plt.axis('off')
```

```
plt.tight_layout()
plt.show()
```



1. Linear Kernel

In [ ]:
```
#1.Linear Kernel
svm_linear = SVC(kernel='linear')
svm_linear.fit(x_train, y_train)
y_pred = svm_linear.predict(x_test)
accuracy_linear = svm_linear.score(x_test, y_test)
print("Accuracy (Linear Kernel):", accuracy_linear)
```

2. Polynomial Kernel

In [ ]:
```
#2.Polynomial Kernel
svm_poly = SVC(kernel='poly')
svm_poly.fit(x_train, y_train)
y_pred = svm_poly.predict(x_test)
accuracy_poly = svm_poly.score(x_test, y_test)
print("Accuracy (Polynomial Kernel):", accuracy_poly)
```

3. Sigmoid Kernel

In [ ]:
```
#3.Sigmoid Kernel
svm_sigmoid = SVC(kernel='sigmoid')
svm_sigmoid.fit(x_train, y_train)
y_pred = svm_sigmoid.predict(x_test)
```

```
accuracy_sigmoid = svm_sigmoid.score(x_test, y_test)
print("Accuracy (Sigmoid Kernel):", accuracy_sigmoid)
```

4. RBF Kernel

In [ ]:
```
#4.RBF Kernel
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(x_train, y_train)
y_pred = svm_rbf.predict(x_test)
accuracy_rbf = svm_rbf.score(x_test, y_test)
print("Accuracy (RBF Kernel):", accuracy_rbf)
```

Accuracy:

In [ ]:
```
print("Accuracy (Linear Kernel)     : ", accuracy_linear*100)
print("Accuracy (Polynomial Kernel) : ", accuracy_poly*100)
print("Accuracy (Sigmoid Kernel)    : ", accuracy_sigmoid*100)
print("Accuracy (RBF Kernel)    : ", accuracy_rbf*100)
```

```
Accuracy (Linear Kernel)     :  94.04
Accuracy (Polynomial Kernel) :  97.71
Accuracy (Sigmoid Kernel)    :  77.59
Accuracy (RBF Kernel)    :  97.92
```

EMAIL CLASSIFICATION:

Inference:

- Using SVM, Linear kernel function gives higher accuracy (92.29%)
- SVM with Linear Kernel Function gives higher accuracy than Naïve Bayes Algorithm

Learning Outcome:

- Implemented SVM algorithm using scikit-learn library in Python, preprocessing email data with feature extraction and scaling techniques for effective spam or ham classification.
- Tuned SVM hyperparameters, including kernel selection and regularization parameter, to optimize classification performance on email datasets.
- Utilized Naive Bayes algorithm to classify emails into spam or ham categories, assessing its performance through confusion matrix analysis for accuracy evaluation.

DIGIT RECOGNITION:

Inference:

- Linear, Polynomial, Sigmoid and RBF Kernel Functions were tried out
- RBF Kernel gives higher accuracy (97.92%)

Learning Outcome:

Successfully trained a Support Vector Machine (SVM) model to recognize handwritten digits from the MNIST dataset, demonstrating proficiency in implementing machine learning algorithms for classification tasks.