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

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.

Email Classification using SVM

In [ ]:
```
import numpy as np # linear algebra
import pandas as pd
df = pd.read_csv('spambase_csv.csv')

df.head()
```

Out[ ]:

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	wo
0	0.00	0.64	0.64	0.0	0.32	
1	0.21	0.28	0.50	0.0	0.14	
2	0.06	0.00	0.71	0.0	1.23	
3	0.00	0.00	0.00	0.0	0.63	
4	0.00	0.00	0.00	0.0	0.63	

5 rows × 58 columns

Count of each class of the data,

Where 1 = Spam 0 = Not a spam

In [ ]:
```
df['class'].value_counts()
```

Out[ ]:
```
class
0    2788
1    1813
Name: count, dtype: int64
```

Hence we 1813 spam emails , 2788 non-spam

A heatmap is created using seaborn's heatmap function, visualizing missing values (NaNs) in the DataFrame df. The DataFrame df.isna() generates a DataFrame of the same shape as df, where each element is True if the corresponding element in df is NaN and False otherwise. The seaborn heatmap function then visualizes this DataFrame as a heatmap, where missing values are represented by a different color (usually a shade of blue) compared to non-missing values.

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize']=[10,10]
sns.heatmap(df.isna())
```

Out[ ]:
```
<Axes: >
```

This updated code computes the percentage of missing values in each column by first determining the proportion of NaN values in each column using df.isna().mean() * 100. This result is then plotted as a bar chart, where each bar represents the percentage of missing values for a specific column.

In [ ]:
```
X = df.iloc[:,:57]
y = df.iloc[:,-1]
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=
from sklearn.svm import SVC
```

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
# Train the Support Vector Machine model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

# Calculate accuracy
accuracy_svm = svm_model.score(X_test, y_test)
print("Accuracy:", accuracy_svm)
```

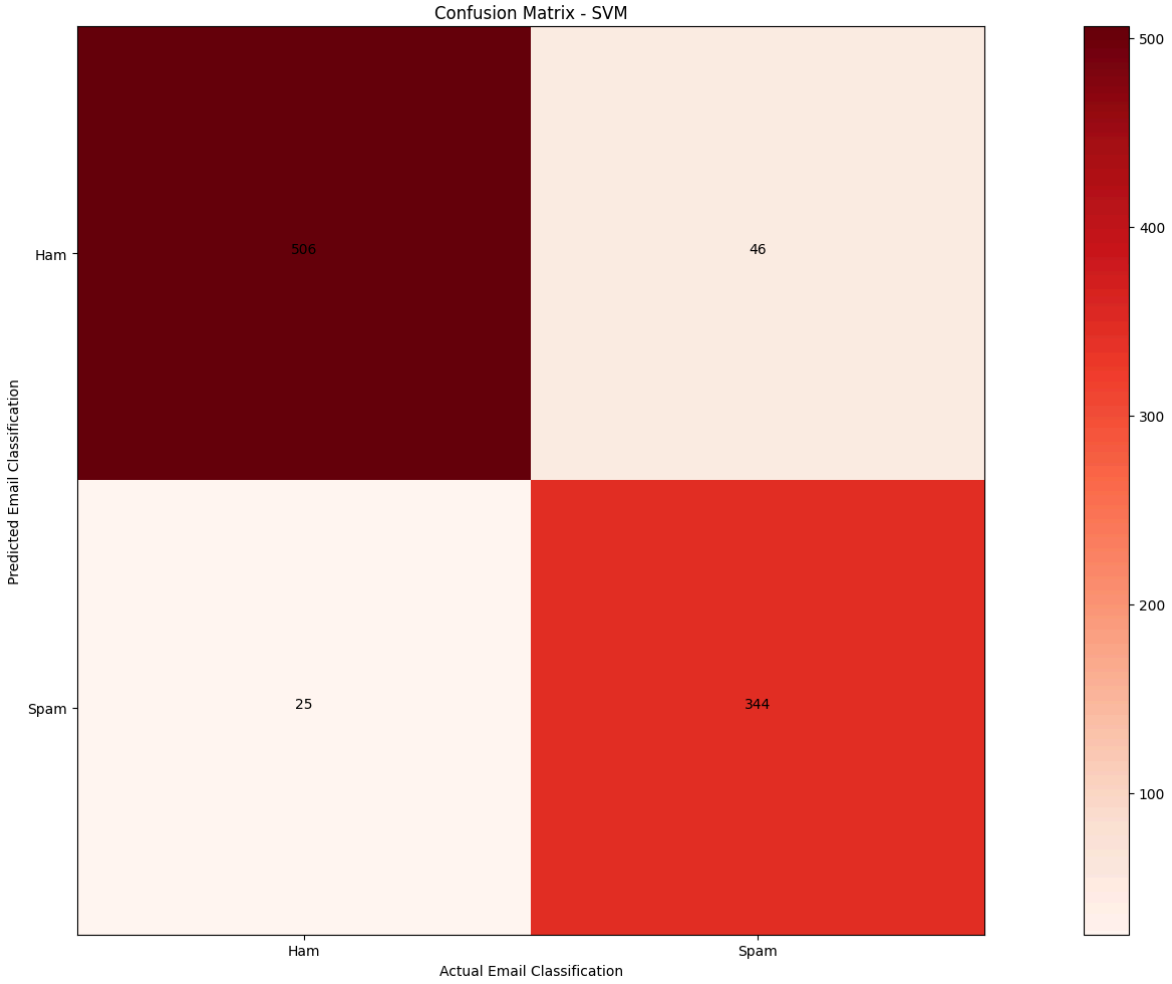
```
Accuracy: 0.9229098805646037
```

In [ ]:
```
# Predict on the test set
preds_svm = svm_model.predict(X_test)
```

In [ ]:
```
# Confusion matrix
cm_svm = confusion_matrix(preds_svm, y_test)
print("Confusion Matrix:")
print(cm_svm)
```

```
Confusion Matrix:
[[506  46]
 [ 25 344]]
```

In [ ]:
```
# Visualize confusion matrix
plt.figure(figsize=(20,10))
plt.imshow(cm_svm, interpolation='nearest', cmap=plt.cm.Reds)
plt.title('Confusion Matrix - SVM')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Ham', 'Spam'], rotation=0)
plt.yticks(tick_marks, ['Ham', 'Spam'])
plt.xlabel('Actual Email Classification')
plt.ylabel('Predicted Email Classification')
for i in range(2):
    for j in range(2):
        plt.text(j, i, format(cm_svm[i, j], 'd'),
                 horizontalalignment="center",
                 color="black" if cm_svm[i, j] > cm_svm.max() / 2. else "black")
plt.tight_layout()
plt.show()
```



Methodology:

Import necessary libraries: Import SVC from sklearn.svm, confusion_matrix from sklearn.metrics, and matplotlib.pyplot as plt.

Train the Support Vector Machine (SVM) model: Initialize an SVM model with a linear kernel (kernel='linear') and fit it to the training data (X_train and y_train).

Calculate accuracy: Use the score method of the SVM model to calculate the accuracy on the test set (X_test and y_test).

Make predictions: Use the trained SVM model to predict the labels for the test set (X_test).

Compute the confusion matrix: Use the confusion_matrix function to compute the confusion matrix based on the predicted labels (preds_svm) and the actual labels (y_test).

Print the confusion matrix: Print the confusion matrix to examine the performance of the SVM model.

Visualize the confusion matrix: Use matplotlib.pyplot to visualize the confusion matrix with a heatmap, where each cell's color intensity represents the number of predictions made by the model.

Inference:

- The code snippet trains a Support Vector Machine (SVM) model with a linear kernel for a classification task (presumably spam classification given the class labels 'Ham' and 'Spam').
- After training the model, it calculates the accuracy on the test set and prints it out.
- Then, it predicts the labels for the test set and computes the confusion matrix to evaluate the model's performance.
- Finally, it visualizes the confusion matrix using a heatmap, providing a visual representation of how well the model is performing in terms of correctly classifying instances as 'Ham' or 'Spam'.
- The diagonal elements of the confusion matrix represent the correct classifications, while off-diagonal elements represent misclassifications.
- The intensity of colors in the heatmap indicates the number of instances classified into each category.
- The visualization helps in quickly identifying any patterns of misclassification and assessing the overall performance of the SVM model.

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.

Digit Recognition using SVM

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Download the MNIST dataset
mnist = fetch_openml('mnist_784', version=1)
```

```
C:\Users\Ananya\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\datasets\_openml.py:1002: FutureWarning: The default value of `parser` will chan
ge from `'liac-arff'` to `'auto'` in 1.4. You can set `parser='auto'` to silence
this warning. Therefore, an `ImportError` will be raised from 1.4 if the dataset
is dense and pandas is not installed. Note that the pandas parser may return diff
erent data types. See the Notes Section in fetch_openml's API doc for details.
  warn(
```

In [ ]:
```
# Extract features (X) and labels (y)
X, y = mnist.data, mnist.target

# Step 2: Preprocess the dataset
# Normalize the features
X = X / 255.0

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Step 3: Train a Support Vector Machine (SVM) model
# Initialize the SVM classifier
svm_clf = SVC(kernel='rbf', gamma='scale', C=1.0, random_state=42)

# Train the SVM model
svm_clf.fit(X_train, y_train)

# Evaluate the trained model
y_pred = svm_clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9764285714285714
```

In [ ]:
```
# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

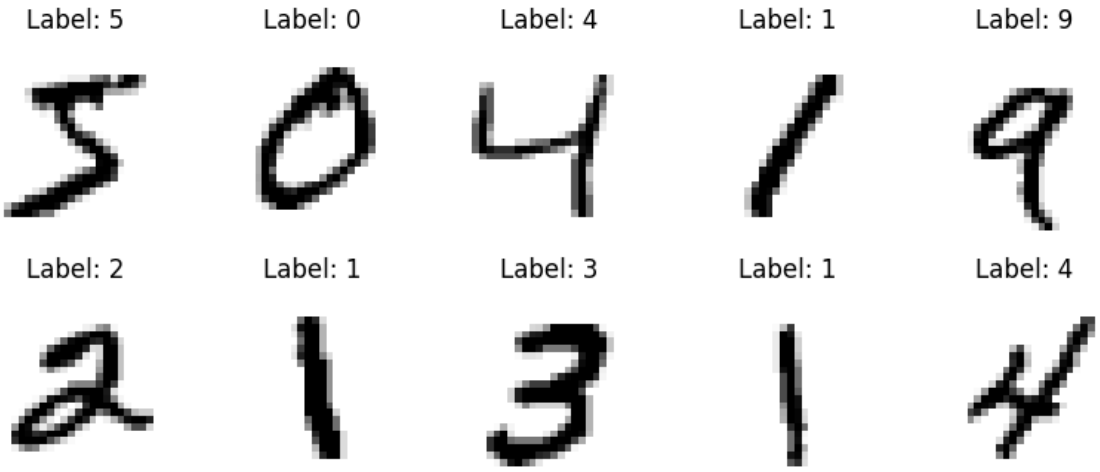
```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1343
           1       0.98      0.99      0.99      1600
           2       0.97      0.98      0.97      1380
           3       0.97      0.97      0.97      1433
           4       0.97      0.98      0.98      1295
           5       0.98      0.97      0.97      1273
           6       0.98      0.99      0.99      1396
           7       0.97      0.97      0.97      1503
           8       0.97      0.96      0.97      1357
           9       0.97      0.96      0.97      1420

    accuracy                           0.98     14000
   macro avg       0.98      0.98      0.98     14000
weighted avg       0.98      0.98      0.98     14000
```

In [ ]:
```python
# Convert X to a numpy array
X = np.array(X)

# Visualize some sample images from the dataset
plt.figure(figsize=(10, 4))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X[i].reshape(28, 28), cmap='binary', interpolation='nearest')  #
    plt.title('Label: ' + str(y[i]))
    plt.axis('off')
plt.show()
```



Methodology:

1. **Downloading Dataset:** The code fetches the MNIST dataset using the `fetch_openml` function from `sklearn.datasets`. MNIST is a widely used dataset consisting of handwritten digits.

2. **Preprocessing:**

   - The features (images) are extracted as `X` and labels (corresponding digits) as `y`.

- The pixel values of the images are normalized to a range of 0 to 1 by dividing by 255.0.

3. **Splitting Dataset:** The dataset is split into training and testing sets using `train_test_split` from `sklearn.model_selection`. The testing set size is 20% of the total dataset.

4. **Training SVM Model:**

   - An SVM classifier (`SVC`) with a radial basis function (RBF) kernel is initialized with certain hyperparameters (`kernel='rbf'`, `gamma='scale'`, `C=1.0`).
   - The model is trained on the training data using the `fit` method.

5. **Model Evaluation:**

   - The trained model is evaluated on the testing set by making predictions using `predict` method.
   - Accuracy of the model is computed using `accuracy_score` from `sklearn.metrics`.
   - Classification report, including precision, recall, and F1-score, is printed using `classification_report`.

6. **Visualization:**

   - The first 10 samples from the dataset are visualized using matplotlib. Each sample is reshaped to a 28x28 image and displayed with its corresponding label.

Inference:

1. **Accuracy:** The accuracy of the SVM model on the test set is printed (0.97). It gives an overall measure of how well the model performs.

2. **Classification Report:** The classification report provides a detailed breakdown of model performance for each class. It includes precision, recall, F1-score, and support (number of occurrences of each class in the test set).

3. **Visualization:** The visualization allows us to qualitatively inspect the dataset by displaying some sample images along with their labels. This gives an intuitive understanding of the data the model is trained on.