CSE 4020 - MACHINE LEARNING

Lab 29+30

Lab Assignment-3

Submitted by: Alokam Nikhitha(19BCE2555)

SUPPORT VECTOR MACHINE(SVM)

Ques: Train SVM classifier using sklearn digits dataset (i.e. from sklearn.datasets import load_digits) and then

1. Measure accuracy of your model using different kernels such as rbf, poly and linear.

2. Tune your model further using regularization and gamma parameters and try to come up with highest accuracy score.

3. Use 80% of samples as training data size.

Dataset Used: load_digits dataset from sklearn

Procedure:

- Using pandas, we first import the dataset into our workspace.
- The next step is to choose the independent and dependent variables that will be used in our regression model.
- After that, we divided our data into two sets: training and test.
- Then, using the 'rbf' kernel, we must initialise our Support Vector Machine classifier and fit it to the X_train and y_train attributes.
- Use the ‘linear' and ‘polynomial' kernels to repeat the previous process.
- Then, using the results predicted by X_test on the 'rbf', 'linear', and 'polynomial' kernels, we establish three variables to store the X_test result
- Finally, we compute evaluation metrics for each of the three kernels.

CODE SNIPPETS AND EXPLAINATION

```
In [1]: #Importing the Libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.datasets import load_digits
```

Importing the required Libraries

```
In [2]: #Importing the digits dataset
        dataset = load_digits()
```

Importing the digits Dataset

```
In [3]: # Attributes in our dataset
        dataset.keys()

Out[3]: dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])
```

Listing the Attributes in our Dataset

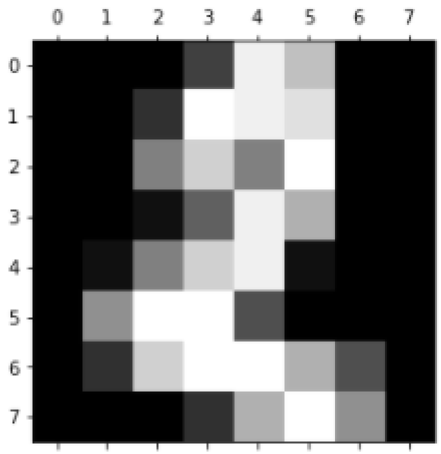
```
In [4]: # Printing the different class lables
        dataset.target_names

Out[4]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Printing the Different Class Labels

```
In [5]: #Printing shape of each image
        dataset.images.shape

Out[5]: (1797, 8, 8)
```

Print the image shape

```
In [6]: #Visualizing the third image
        import pylab as pl
        pl.gray()
        pl.matshow(dataset.images[2])

Out[6]: <matplotlib.image.AxesImage at 0x1f393a7d370>

        <Figure size 432x288 with 0 Axes>
```



Visualizing the Third Image in the Dataset

```
In [7]: # Set of Independent and Dependent Attributes
        X = pd.DataFrame(dataset.data)
        y = dataset.target
```

Taking the Independent and Depending Attributes

```
In [8]: #Splitting the dataset into training and test set
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
```

Splitting the dataset into Training set and Test set

```
In [9]: # Training the Support Vector Machine Model using rbf kernel
        from sklearn.svm import SVC
        rbfClassifier = SVC(kernel='rbf', random_state=0, probability=True)
        rbfClassifier.fit(X_train, y_train)

Out[9]: SVC(probability=True, random_state=0)
```

Training theSVM Model using rbf kernel

```
In [10]: # Training the Support Vector Machine Model using linear kernel
         from sklearn.svm import SVC
         linClassifier = SVC(kernel='linear', random_state=0, probability=True)
         linClassifier.fit(X_train, y_train)

Out[10]: SVC(kernel='linear', probability=True, random_state=0)
```

Training SVM model using Linear kernel

```
In [11]: # Training the Support Vector Machine Model using poly kernel
         from sklearn.svm import SVC
         polClassifier = SVC(kernel='poly', random_state=0, probability=True)
         polClassifier.fit(X_train, y_train)

Out[11]: SVC(kernel='poly', probability=True, random_state=0)
```

Training SVM model using poly kernel Model

```
In [12]: # Predicting results
         y_pred_rbf = rbfClassifier.predict(X_test)
         y_pred_lin = linClassifier.predict(X_test)
         y_pred_pol = polClassifier.predict(X_test)
```

Predicting Results for various Kernels

```
In [13]: from sklearn.metrics import confusion_matrix
         cm_rbf = confusion_matrix(y_test, y_pred_rbf)
         cm_lin = confusion_matrix(y_test, y_pred_lin)
         cm_pol = confusion_matrix(y_test, y_pred_pol)
```

Confusion Matrix for various Kernels

```
In [14]: cm_rbf

Out[14]: array([[27,  0,  0,  0,  0,  0,  0,  0,  0,  0],
                [ 0, 35,  0,  0,  0,  0,  0,  0,  0,  0],
                [ 0,  0, 36,  0,  0,  0,  0,  0,  0,  0],
                [ 0,  0,  0, 29,  0,  0,  0,  0,  0,  0],
                [ 0,  0,  0,  0, 30,  0,  0,  0,  0,  0],
                [ 0,  0,  0,  0,  0, 39,  0,  0,  0,  1],
                [ 0,  0,  0,  0,  0,  0, 44,  0,  0,  0],
                [ 0,  0,  0,  0,  0,  0,  0, 39,  0,  0],
                [ 0,  1,  0,  0,  0,  0,  0,  0, 38,  0],
                [ 0,  0,  0,  0,  0,  1,  0,  0,  0, 40]], dtype=int64)
```

Confusion Matrix for rbf Kernel

```
In [15]: cm_pol

Out[15]: array([[27,  0,  0,  0,  0,  0,  0,  0,  0,  0],
                [ 0, 35,  0,  0,  0,  0,  0,  0,  0,  0],
                [ 0,  0, 36,  0,  0,  0,  0,  0,  0,  0],
                [ 0,  0,  0, 29,  0,  0,  0,  0,  0,  0],
                [ 0,  0,  0,  0, 30,  0,  0,  0,  0,  0],
                [ 0,  0,  0,  0,  0, 39,  0,  0,  0,  1],
                [ 0,  1,  0,  0,  0,  0, 43,  0,  0,  0],
                [ 0,  0,  0,  0,  0,  0,  0, 39,  0,  0],
                [ 0,  1,  0,  0,  0,  0,  0,  0, 38,  0],
                [ 0,  0,  0,  0,  0,  1,  0,  0,  0, 40]], dtype=int64)
```

Confusion Matrix for poly Kernel

```
cm_lin
```

```
array([[27,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 34,  0,  0,  0,  0,  0,  0,  1,  0],
       [ 0,  0, 36,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 29,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 30,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0, 39,  0,  0,  0,  1],
       [ 0,  1,  0,  0,  0,  0, 43,  0,  0,  0],
       [ 0,  0,  0,  0,  1,  0,  0, 38,  0,  0],
       [ 0,  1,  1,  0,  0,  0,  0,  0, 37,  0],
       [ 0,  0,  0,  1,  0,  1,  0,  0,  0, 39]], dtype=int64)
```

Confusion Matrix for linear Kernel

```
In [16]: from sklearn.metrics import accuracy_score
         accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
         accuracy_lin = accuracy_score(y_test, y_pred_lin)
         accuracy_pol = accuracy_score(y_test, y_pred_pol)
```

Calculating Accuracy for various Kernels

```
In [17]: print("Accuracy with RBF kernel: \t", accuracy_rbf)
         print("Accuracy with Linear kernel: \t", accuracy_lin)
         print("Accuracy with Polynomial kernel:", accuracy_pol)
```

```
Accuracy with RBF kernel:        0.9916666666666667
Accuracy with Linear kernel:     0.9777777777777777
Accuracy with Polynomial kernel: 0.9888888888888889
```

Printing Accuracy for Different Kernels

We can see here that the accuracy with rbf kernel is max and thus it is most suitable.

Accuracy(Linear) < Accuracy(Poly) < Accuracy(rbf)

```
: from sklearn.metrics import mean_squared_error
  print("Mean Squared Error with RBF kernel: \t", mean_squared_error(y_test, y_pred_rbf))
  print("Mean Squared Error with Linear kernel: \t", mean_squared_error(y_test, y_pred_lin))
  print("Mean Squared Error with Polynomial kernel:", mean_squared_error(y_test, y_pred_pol))
```

```
Mean Squared Error with RBF kernel:       0.225
Mean Squared Error with Linear kernel:    0.6555555555555556
Mean Squared Error with Polynomial kernel: 0.29444444444444445
```

Mean Squared Errors with various Kernels

From here we can again infer that MSE in least for rbf kernel and hence it is the most suitable kernel for our dataset in Support Vector Classifier.

MSE(rbf) < MSE(Poly) < MSE(Linear)

In [19]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_rbf))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        27
           1       0.97      1.00      0.99        35
           2       1.00      1.00      1.00        36
           3       1.00      1.00      1.00        29
           4       1.00      1.00      1.00        30
           5       0.97      0.97      0.97        40
           6       1.00      1.00      1.00        44
           7       1.00      1.00      1.00        39
           8       1.00      0.97      0.99        39
           9       0.98      0.98      0.98        41

    accuracy                           0.99       360
   macro avg       0.99      0.99      0.99       360
weighted avg       0.99      0.99      0.99       360
```

In [20]:
```
print(classification_report(y_test, y_pred_pol))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        27
           1       0.95      1.00      0.97        35
           2       1.00      1.00      1.00        36
           3       1.00      1.00      1.00        29
           4       1.00      1.00      1.00        30
           5       0.97      0.97      0.97        40
           6       1.00      0.98      0.99        44
           7       1.00      1.00      1.00        39
           8       1.00      0.97      0.99        39
           9       0.98      0.98      0.98        41

    accuracy                           0.99       360
   macro avg       0.99      0.99      0.99       360
weighted avg       0.99      0.99      0.99       360
```

In [27]:
```
print(classification_report(y_test, y_pred_lin))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        27
           1       0.94      0.97      0.96        35
           2       0.97      1.00      0.99        36
           3       0.97      1.00      0.98        29
           4       0.97      1.00      0.98        30
           5       0.97      0.97      0.97        40
           6       1.00      0.98      0.99        44
           7       1.00      0.97      0.99        39
           8       0.97      0.95      0.96        39
           9       0.97      0.95      0.96        41

    accuracy                           0.98       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.98      0.98       360
```

Here we have printed the classification report of Support Vector Classifier with all three kernels.

In [33]:
```
rbfClassifierC=SVC(kernel='rbf' , C=0.3)
rbfClassifierC.fit(X_train,y_train)
rbfClassifierC.score(X_test,y_test)
```

Out[33]: 0.9805555555555555

In [34]:
```
rbfClassifierC=SVC(kernel='rbf' , C=0.4)
rbfClassifierC.fit(X_train,y_train)
rbfClassifierC.score(X_test,y_test)
```

Out[34]: 0.9888888888888889

In [35]:
```
rbfClassifierC=SVC(kernel='rbf' , C=0.5)
rbfClassifierC.fit(X_train,y_train)
rbfClassifierC.score(X_test,y_test)
```

Out[35]: 0.9888888888888889

In [36]:
```
rbfClassifierC=SVC(kernel='rbf' , C=0.6)
rbfClassifierC.fit(X_train,y_train)
rbfClassifierC.score(X_test,y_test)
```

Out[36]: 0.9888888888888889

```
In [37]: rbfClassifierC=SVC(kernel='rbf' , C=0.7)
         rbfClassifierC.fit(X_train,y_train)
         rbfClassifierC.score(X_test,y_test)

Out[37]: 0.9916666666666667
```

```
In [38]: rbfClassifierC=SVC(kernel='rbf' , C=0.8)
         rbfClassifierC.fit(X_train,y_train)
         rbfClassifierC.score(X_test,y_test)

Out[38]: 0.9916666666666667
```

```
In [39]: rbfClassifierC=SVC(kernel='rbf' , C=0.9)
         rbfClassifierC.fit(X_train,y_train)
         rbfClassifierC.score(X_test,y_test)

Out[39]: 0.9916666666666667
```

Here we have tried to tune in the C value for rbf kernel. Initially we have used 0.3 as we increase the C value and we can see that the accuracy increases till C=0.7, after that C remains constant and there is no significant increase in models accuracy and hence the C value can be taken as C=0.7.

Result and Conclusion:

Accuracy

```
In [17]: print("Accuracy with RBF kernel: \t", accuracy_rbf)
         print("Accuracy with Linear kernel: \t", accuracy_lin)
         print("Accuracy with Polynomial kernel:", accuracy_pol)

         Accuracy with RBF kernel:        0.9916666666666667
         Accuracy with Linear kernel:     0.9777777777777777
         Accuracy with Polynomial kernel: 0.9888888888888889
```

RBF Kernel-

- **Model Accuracy = 99.1667%**

```
In [14]: cm_rbf

Out[14]: array([[27,  0,  0,  0,  0,  0,  0,  0,  0,  0],
                [ 0, 35,  0,  0,  0,  0,  0,  0,  0,  0],
                [ 0,  0, 36,  0,  0,  0,  0,  0,  0,  0],
                [ 0,  0,  0, 29,  0,  0,  0,  0,  0,  0],
                [ 0,  0,  0,  0, 30,  0,  0,  0,  0,  0],
                [ 0,  0,  0,  0,  0, 39,  0,  0,  0,  1],
                [ 0,  0,  0,  0,  0,  0, 44,  0,  0,  0],
                [ 0,  0,  0,  0,  0,  0,  0, 39,  0,  0],
                [ 0,  1,  0,  0,  0,  0,  0,  0, 38,  0],
                [ 0,  0,  0,  0,  0,  1,  0,  0,  0, 40]], dtype=int64)
```

```
In [19]: from sklearn.metrics import classification_report
         print(classification_report(y_test, y_pred_rbf))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        27
           1       0.97      1.00      0.99        35
           2       1.00      1.00      1.00        36
           3       1.00      1.00      1.00        29
           4       1.00      1.00      1.00        30
           5       0.97      0.97      0.97        40
           6       1.00      1.00      1.00        44
           7       1.00      1.00      1.00        39
           8       1.00      0.97      0.99        39
           9       0.98      0.98      0.98        41

    accuracy                           0.99       360
   macro avg       0.99      0.99      0.99       360
weighted avg       0.99      0.99      0.99       360
```

```
In [24]: print(classification_report(y_test,y_pred_rbf))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        27
           1       0.97      1.00      0.99        35
           2       1.00      1.00      1.00        36
           3       1.00      1.00      1.00        29
           4       1.00      1.00      1.00        30
           5       0.97      0.97      0.97        40
           6       1.00      1.00      1.00        44
           7       1.00      1.00      1.00        39
           8       1.00      0.97      0.99        39
           9       0.98      0.98      0.98        41

    accuracy                           0.99       360
   macro avg       0.99      0.99      0.99       360
weighted avg       0.99      0.99      0.99       360
```

- **Identified 0 = 27**
- **True 0 = 27**
- **Identified 1 = 34**
- **True 1 = 35**
- **Identified 2 = 36**
- **True 2 = 36**

- Identified 3 = 29
- True 3 = 29
- Identified 4 = 30
- True 4 = 30
- Identified 5 = 39
- True 5 = 40
- Identified 6 = 44
- True 6 = 44
- Identified 7 = 39
- True 7 = 39
- Identified 8 = 39
- True 8 = 39
- Identified 9 = 40
- True 9 = 41

- Precision of 0 = 1.00
- Precision of 1 = 0.97
- Precision of 2 = 1.00
- Precision of 3 = 1.00
- Precision of 4 = 1.00
- Precision of 5 = 0.97
- Precision of 6 = 1.00
- Precision of 7 = 1.00
- Precision of 8 = 1.00
- Precision of 9 = 0.98

- Recall of 0 = 1.00
- Recall of 1 = 1.00
- Recall of 2 = 1.00
- Recall of 3 = 1.00
- Recall of 4 = 1.00
- Recall of 5 = 0.97
- Recall of 6 = 1.00
- Recall of 7 = 1.00
- Recall of 8 = 0.97
- Recall of 9 = 0.98

Linear Kernel:

- **Model Accuracy = 97.77%**

```
cm_lin
```

```
array([[27,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 34,  0,  0,  0,  0,  0,  0,  1,  0],
       [ 0,  0, 36,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 29,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 30,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0, 39,  0,  0,  0,  1],
       [ 0,  1,  0,  0,  0,  0, 43,  0,  0,  0],
       [ 0,  0,  0,  0,  1,  0,  0, 38,  0,  0],
       [ 0,  1,  1,  0,  0,  0,  0,  0, 37,  0],
       [ 0,  0,  0,  1,  0,  1,  0,  0,  0, 39]], dtype=int64)
```

```
In [27]: print(classification_report(y_test, y_pred_lin))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        27
           1       0.94      0.97      0.96        35
           2       0.97      1.00      0.99        36
           3       0.97      1.00      0.98        29
           4       0.97      1.00      0.98        30
           5       0.97      0.97      0.97        40
           6       1.00      0.98      0.99        44
           7       1.00      0.97      0.99        39
           8       0.97      0.95      0.96        39
           9       0.97      0.95      0.96        41

    accuracy                           0.98       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.98      0.98       360
```

- **Identified 0 = 27**
- **True 0 = 27**
- **Identified 1 = 33**
- **True 1 = 35**
- **Identified 2 = 35**
- **True 2 = 36**
- **Identified 3 = 28**
- **True 3 = 29**
- **Identified 4 = 29**
- **True 4 = 30**
- **Identified 5 = 39**
- **True 5 = 40**

- Identified 6 = 44
- True 6 = 44
- Identified 7 = 39
- True 7 = 39
- Identified 8 = 38
- True 8 = 39
- Identified 9 = 40
- True 9 = 41

- Precision of 0 = 1.00
- Precision of 1 = 0.97
- Precision of 2 = 1.00
- Precision of 3 = 1.00
- Precision of 4 = 1.00
- Precision of 5 = 0.97
- Precision of 6 = 0.98
- Precision of 7 = 0.97
- Precision of 8 = 0.95
- Precision of 9 = 0.95

- Recall of 0 = 1.00
- Recall of 1 = 0.96
- Recall of 2 = 0.99
- Recall of 3 = 0.98
- Recall of 4 = 0.98
- Recall of 5 = 0.97
- Recall of 6 = 0.99
- Recall of 7 = 0.99
- Recall of 8 = 0.96
- Recall of 9 = 0.96

Poly Kernel:

• **Model Accuracy = 98.88%**

```
In [15]: cm_pol

Out[15]: array([[27,  0,  0,  0,  0,  0,  0,  0,  0,  0],
                [ 0, 35,  0,  0,  0,  0,  0,  0,  0,  0],
                [ 0,  0, 36,  0,  0,  0,  0,  0,  0,  0],
                [ 0,  0,  0, 29,  0,  0,  0,  0,  0,  0],
                [ 0,  0,  0,  0, 30,  0,  0,  0,  0,  0],
                [ 0,  0,  0,  0,  0, 39,  0,  0,  0,  1],
                [ 0,  1,  0,  0,  0,  0, 43,  0,  0,  0],
                [ 0,  0,  0,  0,  0,  0,  0, 39,  0,  0],
                [ 0,  1,  0,  0,  0,  0,  0,  0, 38,  0],
                [ 0,  0,  0,  0,  0,  1,  0,  0,  0, 40]], dtype=int64)
```

```
In [20]: print(classification_report(y_test, y_pred_pol))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        27
           1       0.95      1.00      0.97        35
           2       1.00      1.00      1.00        36
           3       1.00      1.00      1.00        29
           4       1.00      1.00      1.00        30
           5       0.97      0.97      0.97        40
           6       1.00      0.98      0.99        44
           7       1.00      1.00      1.00        39
           8       1.00      0.97      0.99        39
           9       0.98      0.98      0.98        41

    accuracy                           0.99       360
   macro avg       0.99      0.99      0.99       360
weighted avg       0.99      0.99      0.99       360
```

- **Identified 0 = 27**
- **True 0 = 27**
- **Identified 1 = 35**
- **True 1 = 35**
- **Identified 2 = 36**
- **True 2 = 36**
- **Identified 3 = 29**
- **True 3 = 29**
- **Identified 4 = 30**
- **True 4 = 30**
- **Identified 5 = 39**
- **True 5 = 40**
- **Identified 6 = 44**

- True 6 = 44
- Identified 7 = 39
- True 7 = 39
- Identified 8 = 39
- True 8 = 39
- Identified 9 = 40
- True 9 = 41

- Precision of 0 = 1.00
- Precision of 1 = 0.95
- Precision of 2 = 1.00
- Precision of 3 = 1.00
- Precision of 4 = 1.00
- Precision of 5 = 0.97
- Precision of 6 = 1.00
- Precision of 7 = 1.00
- Precision of 8 = 1.00
- Precision of 9 = 0.98

- Recall of 0 = 1.00
- Recall of 1 = 1.00
- Recall of 2 = 1.00
- Recall of 3 = 1.00
- Recall of 4 = 1.00
- Recall of 5 = 0.97
- Recall of 6 = 0.98
- Recall of 7 = 1.00
- Recall of 8 = 0.97
- Recall of 9 = 0.98

KNN

Question:

1. Load the data

2. Initialize K to your chosen number of neighbors

3. For each example in the data

3.1 Calculate the distance between the query example and the current example from the data.

3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances

5. Pick the first K entries from the sorted collection

6. Get the labels of the selected K entries

7. If regression, return the mean of the K labels

8. If classification, return the mode of the K labels

Dataset Used:

diabetes dataset from https://www.kaggle.com/uciml/pima-indians-diabetesdatabase/version/1

Procedure:

-Using pandas, we first import the dataset into our workspace.

-The independent and dependent attributes to be employed in our classification model must then be decided.

- After that, we divided our data into two sets: training and test.

- After that, we must Feature Scale our dataset.

-Scaling concerns should be accounted for in many attributes.

- Next, we determine the k value for which the classifier has the lowest error.

- After that, we use the best k value to fit our classifier model.

- After that, we construct a variable to record our expected result.

-the X test set's classifier

- Last, we compute our assessment metrics.

Code Snippets and Explanation:

```
In [1]: #Importing Libraries
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

```
In [2]: #Importing the dataset
        dataset = pd.read_csv("diabetes.csv")
        X = dataset.iloc[:, :-1].values
        y = dataset.iloc[:, -1].values
```

Importing the Libraries and Dataset

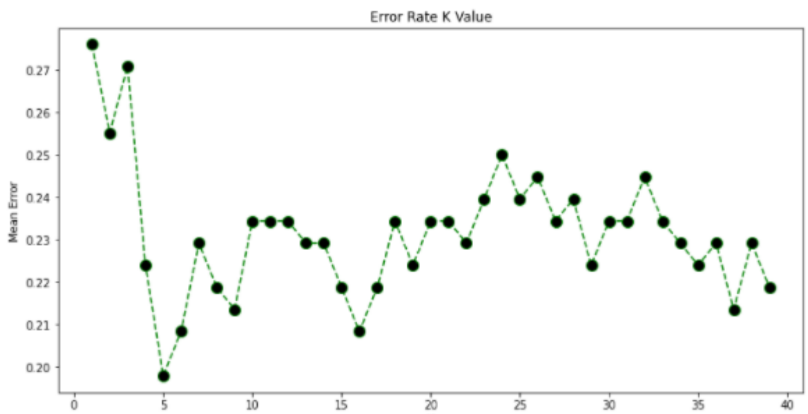
```
In [3]: #Splitting the datasets into training and test set
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=0)
```

```
In [4]: #Feature Scaling
        from sklearn.preprocessing import StandardScaler
        sc_X = StandardScaler()
        X_train = sc_X.fit_transform(X_train)
        X_test = sc_X.transform(X_test)
```

Splitting the dataset into Training and testing sets and Feature scaling

Here we are splitting our dataset into training set and test set with 25% of our dataset values in test set and remaining 75% in training set.

In [5]:
```
from sklearn.neighbors import KNeighborsClassifier
error = []
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='green', linestyle='dashed', marker='o', markerfacecolor='black', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K')
plt.ylabel('Mean Error')
plt.show()
```



We're trying to figure out what the best value for K is. When we choose K as 5, we can observe that the error value is the lowest.

In [6]:
```
#Fitting Classification model to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric='minkowski')
classifier.fit(X_train, y_train)
```

Out[6]:
```
KNeighborsClassifier()
```

We're using training sets to fit our KNN classifier. Because of the previous outcome, we chose K value of 5.

In [7]:
```
#Predicting the X_test result
y_pred = classifier.predict(X_test)
```

On the test set, we're creating a list of predictions based on the classifier's predictions. Our Confusion Matrix has also been created.

```
In [8]: #Printing the confusion matrix
        from sklearn.metrics import confusion_matrix
        confusion_matrix(y_test, y_pred)

Out[8]: array([[114,  16],
               [ 22,  40]], dtype=int64)
```

```
In [9]: #Printing the Accuracy and Mean Squared Error Value
        from sklearn.metrics import accuracy_score, mean_squared_error
        print("Accuracy value: \t", accuracy_score(y_test, y_pred))
        print("MSE value: \t", mean_squared_error(y_test, y_pred))

        Accuracy value:          0.8020833333333334
        MSE value:       0.19791666666666666
```

```
In [10]: #Printing the classification report
         from sklearn.metrics import classification_report
         print(classification_report(y_test, y_pred))

                       precision    recall  f1-score   support

                   0       0.84      0.88      0.86       130
                   1       0.71      0.65      0.68        62

            accuracy                           0.80       192
           macro avg       0.78      0.76      0.77       192
        weighted avg       0.80      0.80      0.80       192
```

We've created our numerous evaluation matrixes here. Precision, recall, and f1 score are all included, as well as accuracy and mean squared error value. Our model has an accuracy of 80 percent and an MSE score of 0.1979.

def knn(data, query, k, distance_fn, choice_fn):

neighbor_distances_and_indices = []

 # 3. For each example in the data

for index, example in enumerate(data):

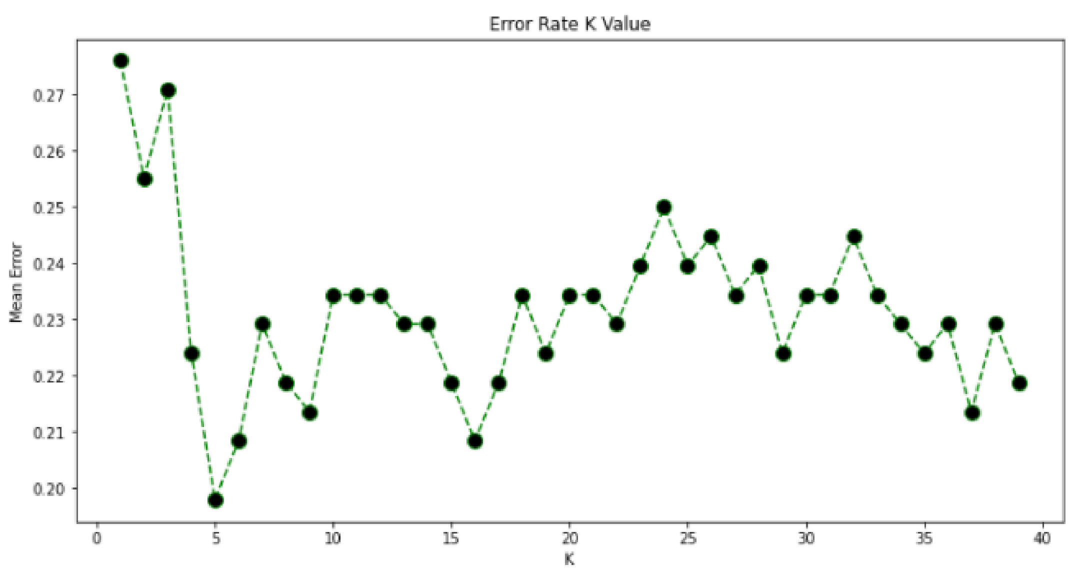
 # 3.1 Calculate the distance between the query example and the current

 # example from the data.

```
        distance = distance_fn(example[:-1], query)

        # 3.2 Add the distance and the index of the example to an ordere
d collection
        neighbor_distances_and_indices.append((distance, index))

    # 4. Sort the ordered collection of distances and indices from
    # smallest to largest (in ascending order) by the distances
    sorted_neighbor_distances_and_indices = sorted(neighbor_distan
ces_and_indices)

    # 5. Pick the first K entries from the sorted collection
    k_nearest_distances_and_indices = sorted_neighbor_distances_an
d_indices[:k]

    # 6. Get the labels of the selected K entries
    k_nearest_labels = [data[i][1] for distance, i in k_nearest_distances
_and_indices]

    # 7. If regression (choice_fn = mean), return the average of the K la
bels
    # 8. If classification (choice_fn = mode), return the mode of the K la
bels
    return k_nearest_distances_and_indices , choice_fn(k_nearest_lab
els)

#function to calculate the mean used in case of regression
def mean(labels):
```

```
    return sum(labels) / len(labels)

#function to calculate the mode used in case of classification
def mode(labels):
    return Counter(labels).most_common(1)[0][0]

#function to calculate the distance between two data points
def euclidean_distance(point1, point2):
    sum_squared_distance = 0
    for i in range(len(point1)):
        sum_squared_distance += math.pow(point1[i] - point2[i], 2)
    return math.sqrt(sum_squared_distance)
```

Result and Conclusion:



	precision	recall	f1-score	support
0	0.84	0.88	0.86	130
1	0.71	0.65	0.68	62
accuracy			0.80	192
macro avg	0.78	0.76	0.77	192
weighted avg	0.80	0.80	0.80	192

Modal Accuracy:80%

MLP

Question:

To train a Multi-Layer Perceptron (MLP) model to classify the network traffic record whether it is a normal or attack...

1. Read and parse the dataset.

2. Create Multi-Layer Perceptron Model (MLP)

3. Train and evaluate a Multi-Layer Perceptron (MLP) model

Dataset Used:

NSL KDD – Intrusion Detection Dataset
https://www.unb.ca/cic/datasets/nsl.html

Procedure:

-Using pandas, we first import the dataset into our workspace.

-Assign the column names to our dataset as it doesn't have one.

- **Pick out** and encode our **specific** variable.

- After encoding **the specific** variable, we **want** to dummy encode them **on the way to** keep away from ordinality **among** nominal **information**.

- We then **want** to re-assign our label **information**. All labels **different** than **ordinary** are assigned as attacks.

- We then **want** to divide the **schooling** set and **check** set **information** into set of **structured** attributes and **impartial** attributes.

- Next, we lay down the Multi-Layer Perceptron and **byskip** our **enter** records into **enter** layer of our neural network.

- Finally, we generate our **check** set **consequences** and evaluation metrices.

Code Snippets and Explanation:

```
In [1]: #Importing the Libraries
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

Here we are importing the necessary libraries in our workspace.

```
In [2]: col_names = ["duration","protocol_type","service","flag","src_bytes",
                     "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
                     "logged_in","num_compromised","root_shell","su_attempted","num_root",
                     "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
                     "is_host_login","is_guest_login","count","srv_count","serror_rate",
                     "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
                     "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
                     "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
                     "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
                     "dst_host_rerror_rate","dst_host_srv_rerror_rate","label", "difficulty"]

        dataset_train = pd.read_csv("KDDTrain+.txt", header=None, names=col_names)
        dataset_test = pd.read_csv("KDDTest+.txt", header=None, names=col_names)
```

Here we're uploading the dataset into our workspace and are assigning them with the column names because it isn't always pre-blanketed in the given dataset.

```
In [3]: for col_name in dataset_train.columns:
            if dataset_train[col_name].dtypes == 'object':
                unique_cat = len(dataset_train[col_name].unique())
                print("Feature '{col_name}' has {unique_cat} categories". format(col_name=col_name, unique_cat=unique_cat))

        Feature 'protocol_type' has 3 categories
        Feature 'service' has 70 categories
        Feature 'flag' has 11 categories
        Feature 'label' has 23 categories
```

```
In [4]: #Identifying Categorical Variables in test set
        for col_name in dataset_test.columns:
            if dataset_test[col_name].dtypes == 'object':
                unique_cat = len(dataset_test[col_name].unique())
                print("Feature '{col_name}' has {unique_cat} categories". format(col_name=col_name, unique_cat=unique_cat))

        Feature 'protocol_type' has 3 categories
        Feature 'service' has 64 categories
        Feature 'flag' has 11 categories
        Feature 'label' has 38 categories
```

Here we've identified all of the express attributes in our training set and take a look at set. We have additionally identified the range of

classes inculcating **inside** every attribute

```
In [5]: #Encoding Categorical Variables
        from sklearn.preprocessing import LabelEncoder, OneHotEncoder
        categorical_columns = ['protocol_type', 'service', 'flag']
        cat_train = dataset_train[categorical_columns]
        cat_test = dataset_test[categorical_columns]
```

Here we have created 2 dummy data frames to include the categorical attributes in them

```
In [6]: #Making column names for dummies
        #Protocol Type
        unique_protocol = sorted(dataset_train.protocol_type.unique())
        string1 = 'Protocol_type_'
        unique_protocol2 = [string1 + x for x in unique_protocol]

        #Service
        unique_service = sorted(dataset_train.service.unique())
        string2 = 'service_'
        unique_service2 = [string2 + x for x in unique_service]

        #Flag
        unique_flag = sorted(dataset_train.flag.unique())
        string3 = 'flag_'
        unique_flag2 = [string3 + x for x in unique_flag]

        dumcols = unique_protocol2 + unique_service2 + unique_flag2

        #For test set
        unique_service_test = sorted(dataset_test.service.unique())
        unique_service2_test = [string2 + x for x in unique_service_test]

        test_dumcols = unique_protocol2 + unique_service2_test + unique_flag2
```

Here **we've** created the dummy attributes to **keep away from** the ordinal **introduction** among those nominal **specific** attributes.

```
In [7]: #Dummy encoding the Categorical Variable
        train_categorical = cat_train.apply(LabelEncoder().fit_transform)
        test_categorical = cat_test.apply(LabelEncoder().fit_transform)
        enc = OneHotEncoder()
        train_categorical = enc.fit_transform(train_categorical)
        train_categorical_data = pd.DataFrame(train_categorical.toarray(), columns=dumcols)
        test_categorical = enc.fit_transform(test_categorical)
        test_categorical_data = pd.DataFrame(test_categorical.toarray(), columns=test_dumcols)
```

Here **we've** used the label encoder to fill **withinside the** dummy attributes in **every** of **the specific** attributes.

```
In [8]: #Adding 6 missing classes from service variable in test set
        train_service = dataset_train['service'].tolist()
        test_service = dataset_test['service'].tolist()

        difference = list(set(train_service) - set(test_service))
        string = 'service_'

        difference = [string + x for x in difference]
        print("Unknown classes in test set are: ")
        difference

        Unknown classes in test set are:

Out[8]: ['service_http_2784',
         'service_red_i',
         'service_aol',
         'service_urh_i',
         'service_harvest',
         'service_http_8001']
```

While checking **the specific** variables we **noticed** that service **characteristic** in **check** set has 70 **training** whilst schooling set has **sixty four** training. Hence, we **want** to **encompass** the ones 6 dummy attributes with **zero** fee in **every** our **schooling** set. This is what **we've got** diagnosed and **achieved** here.

```
In [9]: for col in difference:
            test_categorical_data[col] = 0
        print(test_categorical_data.shape)
        print(train_categorical_data.shape)

        (22544, 84)
        (125973, 84)
```

Here we have finalised our data frames with the dummy values of categorical attributes.

```
In [10]: #Joining the encoded dataframe with non-encoded one
         train = dataset_train.join(train_categorical_data)
         train.drop('flag', axis=1, inplace=True)
         train.drop('protocol_type', axis=1, inplace=True)
         train.drop('service', axis=1, inplace=True)


         test = dataset_test.join(test_categorical_data)
         test.drop('flag', axis=1, inplace=True)
         test.drop('protocol_type', axis=1, inplace=True)
         test.drop('service', axis=1, inplace=True)
```

Next, **we've got** combined our **original** dataset with dummy attributes that we **acquired** in our **specific** assignment. Also **right here** we've got dropped the **original** specific attributes **for you to** inculcate only the non-**specific** attributes.

```
In [11]: print("Training Set Shape: \t", train.shape)
         print("Test Set Shape: \t", test.shape)

         Training Set Shape:      (125973, 124)
         Test Set Shape:          (22544, 124)
```

Here we have checked the number of attributes in both the training set and test set to see if they are equal… we can see that they have 124 attributes each and hence are compatible.

```
In [12]: #Categorising Attacks into common type
         train_label = train['label']
         test_label = test['label']

         train_label = train_label.replace({'normal':0, 'neptune':1, 'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'mailbomb':
                                 'ipsweep' : 1,'nmap' : 1,'portsweep' : 1,'satan' : 1,'mscan' : 1,'saint' : 1
                                 ,'ftp_write': 1,'guess_passwd': 1,'imap': 1,'multihop': 1,'phf': 1,'spy': 1,'warezclient': 1,'warezmas
                                 'buffer_overflow': 1,'loadmodule': 1,'perl': 1,'rootkit': 1,'ps': 4,'sqlattack': 1,'xterm': 1})
         test_label = test_label.replace({ 'normal' : 0, 'neptune' : 1 ,'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'mailbom
                                 'ipsweep' : 1,'nmap' : 1,'portsweep' : 1,'satan' : 1,'mscan' : 1,'saint' : 1
                                 ,'ftp_write': 1,'guess_passwd': 1,'imap': 1,'multihop': 1,'phf': 1,'spy': 1,'warezclient': 1,'warezmas
                                 'buffer_overflow': 1,'loadmodule': 1,'perl': 1,'rootkit': 1,'ps': 1,'sqlattack': 1,'xterm': 1})
         train['label'] = train_label
         test['label'] = test_label
```

Here we have categorised each of the label attribute either as "normal" or an "attack". All the labels which are normal are given a label of 0 and all those that indicate an attack are labelled as 1.

In [13]:
```python
print(train['label'].unique())
print(test['label'].unique())
```

```
[0 1]
[1 0]
```

In [14]:
```python
X_train = train.iloc[:, :-1]
X_test = test.iloc[:, :-1]
y_train = train['label']
y_test = test['label']
```

In [15]:
```python
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Since all the attributes in our dataset don't follow a common scale, we need to feature scale the dataset in order to avoid any preassumed weight amongst them. We have used standard scalar to do this and it scales down each attribute to a range in -1 to 1.

In [16]:
```python
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(125973, 123)
(22544, 123)
(125973,)
(22544,)
```

Here we have assigned the set of dependent and independent attributes. Also, we have printed the shape of each category that we have in order to check if they are compatible with each other.

In [17]:
```python
#Training the Model
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(5, 5), max_iter=100)
mlp.fit(X_train, y_train)
```

Out[17]:
```
MLPClassifier(hidden_layer_sizes=(5, 5), max_iter=100)
```

Here we have laid our neural network and then passed our input and output set to it in-order for it to adjust the weight biases.

```
In [18]: y_pred = mlp.predict(X_test)
```

```
In [19]: from sklearn.metrics import confusion_matrix, accuracy_score
         confusion_matrix(y_test, y_pred)

Out[19]: array([[ 8087,  1624],
                [   63, 12770]], dtype=int64)
```

Here we have generated a vector y_pred that stores the result as predicted by our mlp classifier on test set. We have also generated the confusion matrix to check the performance of our classifier.

```
In [20]: accuracy_score(y_pred, y_test)

Out[20]: 0.9251685592618879
```

we have printed the accuracy of our model and printed the classification reported to finally check the performance of our model. We can see that the accuracy of the model is 92.51%.

```
In [21]: from sklearn.metrics import classification_report
         print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.99      0.83      0.91      9711
           1       0.89      1.00      0.94     12833

    accuracy                           0.93     22544
   macro avg       0.94      0.91      0.92     22544
weighted avg       0.93      0.93      0.92     22544
```

Result and Conclusion:

Confusion Matrix:

```
In [19]: from sklearn.metrics import confusion_matrix, accuracy_score
         confusion_matrix(y_test, y_pred)

Out[19]: array([[ 8087,  1624],
                [   63, 12770]], dtype=int64)
```

Accuracy:

```
In [20]: accuracy_score(y_pred, y_test)

Out[20]: 0.9251685592618879
```

Accuracy:92.51%.

- ✓ Identified Normal = 9711
- ✓ Actual Normal = 9362
- ✓ Identified Attack = 12833
- ✓ Actual Attack = 13182
- ✓ True Normal = 8087
- ✓ True Attack = 12770
- ✓ False Normal = 1624
- ✓ False Attack = 63
- ✓ Precision Normal = 0.99
- ✓ Precision Attack = 0.89
- ✓ Recall Normal = 0.83
- ✓ Recall Attack = 1.00