CSE 4020 - MACHINE LEARNING

Lab 29+30

Support Vector Machine(SVM)

Submitted by: Alokam Nikhitha(19BCE2555)

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

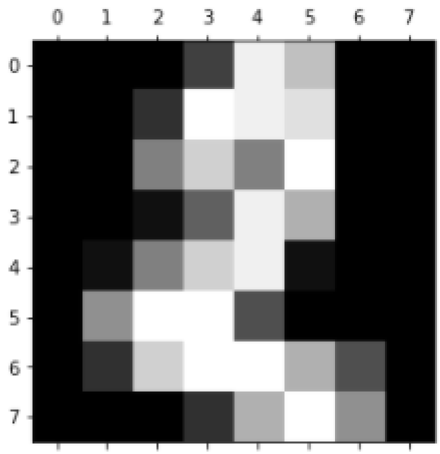
```
In [5]: #Printing shape of each image
        dataset.images.shape

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

Print the image shape

In [6]:
```python
#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]:
```python
# Set of Independent and Dependent Attributes
X = pd.DataFrame(dataset.data)
y = dataset.target
```

Taking the Independent and Depending Attributes

In [8]:
```python
#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]:
```python
# 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]:
```python
# 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