MOHAMED FARHAN

ASSIGNMENT 2: SUPPORT VECTOR MACHINE

DATASET:

I programmed the support vector machine (SVM) classifier on the Glass dataset from the UCI Machine learning repository which consists of 214 training instances of 7 different types of glasses.

The dataset has a total of 11 attributes (features) using which the glass dataset is classified.

Attribute Information:
1. Id (Index) number: 1 to 214
2. RI: refractive index
3. Na: Sodium
4. Mg: Magnesium
5. Al: Aluminum
6. Si: Silicon
7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron
11. Type of glass (classified using numbers from 1 to 7)

Train set: 80% of the original dataset; 171 instances of dataset with 9 features (attributes) excluding class label ('Type') and 'Id' of the original dataset.

Train labels: 80% of the original dataset; 171 instances of dataset containing only class label ('Type').

Test data: 20% of the original dataset; 43 instances of dataset with 9 features (attributes) excluding class label ('Type') and 'Id' of the original dataset.

Test labels: 20% of the original dataset; 43 instances of dataset containing only class label ('Type').

HYPERPARAMETERS:

C

'C' is the penalty parameter used for SVM optimization. The 'C' parameter controls the misclassification of each training example. For large values of 'C', a narrow margin hyperplane would be chosen such that the SVM performs a better job of classifying all training points correctly. Conversely, a small value of 'C' will degrade the performance of SVM classifier because of a large margin separating hyperplane which causes it to misclassify training points.

Gamma

'gamma' parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. It is the kernel coefficient for 'rbf', 'poly' and 'sigmoid'. When 'gamma' is very small, the model is too constrained and cannot capture the complexity or "shape" of the data. Therefore, care should be taken to see that gamma is neither set to a very high or a very low value.

Degree of polynomial

Degree parameter controls the flexibility of the decision boundary. Higher degree kernels yield a more flexible decision boundary. The polynomial kernel of degree 1 leads to a linear separation. This is used while implementing SVM classifier using the polynomial kernel.

KERNELS USED:

LINEAR KERNEL: The Linear kernel is the simplest kernel function. It is given by the inner product (x,y) plus an optional constant **c**. Kernel algorithms using a linear kernel are often equivalent to their non-kernel counterparts.

$$k(x, y) = x^T y + c$$

POLYNOMIAL KERNEL: It represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing learning of non-linear models. The Polynomial kernel is a non-stationary kernel. It is well suited for problems where all the training samples are normalized. The parameters which must be settled are the slope gamma. The constant term r and the polynomial degree d (hence d=3, r=0).

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

Where d is degree of polynomial

RADIAL BASIS FUNCTION (RBF) KERNEL: RBF kernel, is a popular kernel function used in various kernelized learning algorithms. RBF (Gaussian) kernels are a family of kernels where a distance measure is smoothed by a radial function (exponential function) [10]. This kernel nonlinearly maps samples into a higher dimensional space so it, unlike the linear kernel, can handle the case when the relation between class labels and attributes is nonlinear.

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

SIGMOID KERNEL: The Sigmoid kernel, which, despite its wide use, it is not positive semi-definite for certain values of its parameters. Thus, the parameters gamma must be properly chosen otherwise, the results may be drastically wrong, so much so that the SVM performs worse than random.

$$k(x, y) = \tanh(\alpha x^T y + c)$$

CROSS VALIDATION PROCEDURE:

- Read data from **'uci_dataset.csv'** file.
- Store all attributes in variable **x** except for **'Type'** and **'Id'.**
- Store the **'Type'** attribute as label in **y.**
- Use random.seed(5) and random.shuffle(s) to shuffle **x** and **y.**
- Split the original data(x features and y labels) into training set and test set using the function **train_test_split(x, y, test_size = 0.20)** such that training set (**x_train, y_trainlabel**) has **80%** of the original data and the test set (**x_test, y_testlabel**) has the remaining **20%** data.
- Initialize two lists **Cs** and **gamma_range** which contain the values of the parameters 'C' and 'gamma' to be tested using cross validation technique.
- Run two loops (Outer loop iterates over values stored in **Cs** and inner loop iterates over values stored in **gamma_range**).
- **from sklearn.svm import SVC**, call the function **SVC(kernel='rbf', gamma=g,C=C)** which sets kernel to **rbf** and runs for values stored in **Cs** and **gamma_range.**
- **from sklearn.cross_validation import cross_val_score,** call the function **cross_val_score(svc, x_train, y_trainlabel, cv=5, scoring='accuracy'),** which performs the cross validation for 5 folds using **x_train** and **y_trainlabel** as inputs.
- Repeat the above procedure for other kernels (linear, polynomial and sigmoid).
- Optimal hyperparameters are found using the cross validation technique. Run the SVM classifier for the optimal hyperparameters.

SVC(kernel='rbf', C='optimal C', gamma='optimal gamma')

- Use **svclassifier.fit(x_train, y_trainlabel)** to fit the model with the appropriate training data and training labels.
- Determine the predicted values by giving **x_test** as input.

y_pred = svclassifier.predict(x_test)

HYPERPARAMETER EVALUATION:

- Read data from **'uci_dataset.csv'** file.
- Store all attributes in variable **x** except for **'Type'** and **'Id'.**
- Store the **'Type'** attribute as label in **y.**
- Use random.seed(5) and random.shuffle(s) to shuffle **x** and **y.**
- Split the original data(x features and y labels) into training set and test set using the function **train_test_split(x, y, test_size = 0.20)** such that training set (**x_train, y_trainlabel**) has 80% of the original data and the test set (**x_test, y_testlabel**) has the remaining 20% data.
- Initialize two lists **Cs** and **gamma_range** which contain the values of the parameters 'C' and 'gamma' to be tested using cross validation technique.
- Run two loops (Outer loop iterates over values stored in **Cs** and inner loop iterates over values stored in **gamma_range**).
- **from sklearn.svm import SVC**, call the function **SVC(kernel='rbf', gamma=g,C=C)** which sets kernel to **rbf** and runs for values stored in **Cs** and **gamma_range.**
- **from sklearn.cross_validation import cross_val_score,** call the function **cross_val_score(svc, x_train, y_trainlabel, cv=5, scoring='accuracy'),** which performs the cross validation for 5 folds using **x_train** and **y_trainlabel** as inputs.
- Determine the maximum accuracy for each fold and append it along with its corresponding **C** and **gamma** values to a new list **b.**

  b.append((C,g,max_accuracy))

- Sort the list **b** in descending order with respect to accuracy.

  b.sort(key=operator.itemgetter(2),reverse=True)

- The first element of the list **b** contains the **optimal parameters (maximum accuracy along with its corresponding C and gamma values)**.
- Repeat the above procedure to find the optimal hyperparameters for other kernels (linear, polynomial and sigmoid).

RESULTS FOR OPTIMAL HYPERPARAMETERS EVALUATION:

Set of values considered for each hyperparameter:

C = [2-5,2**-4,2**-3,2**-2,2**-1,1,2**1,2**2,2**3,2**4,2**5,2**6]**

gamma = [2-15,2**-13,2**-11,2**-9,2**-7,2**-5,2**-3,2**-1,2**1,2**3,2**5]**

degree=[1,2,3,4,5,6,7,8]

coef0=[1,0.9,0.8,0.7,-1]

```
One vs One
Optimal hyperparamters (rbf):  c= 32 gamma= 0.5 accuracy= 0.8108108108108109
Optimal hyperparameters (poly) : gamma= 0.25 degree= 2 c= 16 accuracy= 0.8648648648648649
Optimal hyperparameters (linear):  c= 4 acc= 0.7419354838709677
Optimal hyperparameters (sigmoid): gamma= 0.0001220703125 coef0= -1 c= 64 accuracy= 0.4666666666666667
```

ONE VS ONE

```
One Vs All
Optimal hyperparameters (rbf):  c= 2 gamma= 8 accuracy= 0.8064516129032258
Optimal hyperparameters (poly): gamma= 0.015625 degree= 2 c= 16 accuracy= 0.8064516129032258
Optimal hyperparameters (linear):  c= 32 acc= 0.7419354838709677
Optimal hyperparameters (sigmoid): gamma= 0.0001220703125 coef0= -1 c= 1 accuracy= 0.7
```

ONE VS ALL

```
Weighted
Optimal hyperparameters (rbf):  c= 64 gamma= 0.0078125 acc= 0.7837837837837838
Optimal hyperparameters (poly) : gamma= 0.015625 degree= 4 c= 0.03125 accuracy= 0.8108108108108109
Optimal hyperparameters (linear):  c= 8 acc= 0.7297297297297297
Optimal hyperparameters (sigmoid): gamma= 0.0001220703125 coef0= -1 c= 64 accuracy= 0.5227272727272727
```

WEIGHTED

ONE VS. ALL:

One-vs-the-rest (OvR) multiclass/multilabel strategy also known as one-vs-all, this strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. In addition to its computational efficiency (only n_classes classifiers are needed), one advantage of this approach is its interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier. This is the most commonly used strategy for multiclass classification.

- To implement the One vs All classifier, we make use of the following module:

  from sklearn.multiclass import OneVsRestClassifier

  svc =OneVsRestClassifier(SVC(kernel='rbf', gamma=g,C=C))

- Perform cross validation using the function **cross_val_score(svc, x_train, y_trainlabel, cv=5, scoring='accuracy')**

- Optimal hyperparameters are found using the cross validation technique. Run the SVM classifier for the optimal hyperparameters.

SVC(kernel='rbf', C='optimal C', gamma='optimal gamma')

- Use **svclassifier.fit(x_train, y_trainlabel)** to fit the model with the appropriate training data and training labels.
- Determine the predicted values by giving **x_test** as input.

y_pred = svclassifier.predict(x_test)

- Repeat the procedure for different kernels (linear, polynomial and sigmoid).

RESULTS FOR ONE VS ALL:

Set of values considered for each hyperparameter:

C = [2-5,2**-4,2**-3,2**-2,2**-1,1,2**1,2**2,2**3,2**4,2**5,2**6]**

gamma = [2-15,2**-13,2**-11,2**-9,2**-7,2**-5,2**-3,2**-1,2**1,2**3,2**5]**

degree=[1,2,3,4,5,6,7,8]

coef0=[1,0.9,0.8,0.7,-1]

```
ONE VS ALL

Optimal hyperparameters (rbf):  c= 2 gamma= 8 acc= 0.8064516129032258
Overall Accuracy (rbf) :  62.7906976744186
Training Time (rbf) : 0.016000032424926758


Optimal hyperparameters (poly) : gamma= 0.015625 degree= 2 c= 16 accuracy= 0.8064516129032258
Overall Accuracy (poly) :  67.44186046511628
Training Time (poly) : 0.033115386962890625


Optimal hyperparameters (linear):  c= 32 acc= 0.7419354838709677
Overall Accuracy (linear) :  62.7906976744186
Training Time (linear) : 0.025223255157470703


Optimal hyperparameters (sigmoid): gamma= 0.0001220703125 coef0= -1 c= 1 accuracy= 0.7
Overall Accuracy (sigmoid) :  44.18604651162791
Training Time (sigmoid) : 0.004021406173706055
```

NOTE: Overall Accuracy is calculated as %

REWEIGHTING PROCEDURE:

Using weights for the classes will stretch the decision boundary away from the center of the under-represented class more towards the over-represented class. This will make the dataset more balanced as the under-represented classes are given more importance by assigning greater weights to them.

- Define SVM classifier with any of the four kernels and assign tuned hyperparameters as arguments.
- Define a class_weight() function which computes weights for each class found in input dataset:
    - Determine number of classes in dataset.
    - Compute samples in each class.
    - Enumerate number of samples.

- Weight for i_{th} class = total number of data samples /(number of samples in i_{th} class * Number of classes)

- class_weight() returns the weights for all classes in the dataset.

- Perform cross validation and call class_weight() function for each of the five folds.

- Fit the model with reweighted training and test data.
- Determine optimal hyperparameters.
- Find overall accuracy as done in one vs one classifier with the optimal hyperparameters.
- Repeat above steps for all kernel and compute the results.

RESULTS:

Set of values considered for each hyperparameter:

C = [2-5,2**-4,2**-3,2**-2,2**-1,1,2**1,2**2,2**3,2**4,2**5,2**6]**

gamma = [2-15,2**-13,2**-11,2**-9,2**-7,2**-5,2**-3,2**-1,2**1,2**3,2**5]**

degree=[1,2,3,4,5,6,7,8]

coef0=[1,0.9,0.8,0.7,-1]

RBF

```
Optimal hyperparameters (rbf):  c= 64 gamma= 0.0078125 acc= 0.7837837837837838
Overall Accuracy (rbf) :  58.139534883720934
Training Time (rbf) : 0.002026081085205078
```

POLYNOMIAL

```
Optimal hyperparameters (poly) : gamma= 0.015625 degree= 4 c= 0.03125 accuracy= 0.81081081
Overall Accuracy (poly) :  72.09302325581395
Training Time (poly) : 0.06248760223388672
```

LINEAR

```
Optimal hyperparameters (linear):  c= 8 acc= 0.7297297297297297
Overall Accuracy (linear) :  67.44186046511628
Training Time (linear) : 0.0
```

SIGMOID

```
Optimal hyperparameters (sigmoid): gamma= 0.0001220703125 coef0= -1 c= 64 accuracy= 0.5151
Overall Accuracy (sigmoid) :  41.86046511627907
Training Time (sigmoid) : 0.0
```

<u>NOTE:</u> Overall Accuracy is calculated as %

ACCURACY AND TIME COMPARISONS FOR ALL PROCEDURES IMPLEMEDTED (ONE VS ONE, ONE VS ALL, WEIGHTED)

ONE VS ONE

```
Optimal hyperparameters (rbf):  c= 32  gamma= 0.5  acc= 0.8108108108108109
Overall Accuracy (rbf) :  79.06976744186046
Training Time (rbf) : 0.002992868423461914
```

```
Optimal hyperparameters (poly) : gamma= 0.25 degree= 2 c= 16 accuracy= 0.8648648648648649
Overall Accuracy (poly) :  72.09302325581395
Training Time (poly) : 0.22450470924377441
```

```
Optimal hyperparameters (linear):   c= 4  acc= 0.7419354838709677
Accuracy (linear) :  72.09302325581395
Training Time (linear) : 0.0
```

```
Optimal hyperparameters (sigmoid): gamma= 0.0001220703125 coef0= -1 c= 64 accuracy= 0.4838709677419355
Accuracy (sigmoid) :  48.837209302325576
Training Time (sigmoid) : 0.0019948482513427734
```

ONE VS ALL

```
Optimal hyperparameters (rbf):  c= 2 gamma= 8 acc= 0.8064516129032258
Overall Accuracy (rbf) :  62.7906976744186
Training Time (rbf) : 0.0
```

```
Optimal hyperparameters (poly) : gamma= 0.015625 degree= 2 c= 16 accuracy= 0.8064516129032258
Overall Accuracy (poly) :  67.44186046511628
Training Time (poly) : 0.04689955711364746
```

```
Optimal hyperparameters (linear):  c= 32 acc= 0.7419354838709677
Overall Accuracy (linear) :  62.7906976744186
Training Time (linear) : 0.031242847442626953
```

```
Optimal hyperparameters (sigmoid): gamma= 0.0001220703125 coef0= -1 c= 1 accuracy= 0.7096774193548387
Accuracy (sigmoid) :  44.18604651162791
Training Time (sigmoid) : 0.0011212825775146484
```

WEIGHTED

```
Optimal hyperparameters (rbf):  c= 64 gamma= 0.0078125 acc= 0.7837837837837838
Overall Accuracy (rbf) :  58.139534883720934
Training Time (rbf) : 0.002026081085205078
```

```
Optimal hyperparameters (poly) : gamma= 0.015625 degree= 4 c= 0.03125 accuracy= 0.8108108108108109
Overall Accuracy (poly) :  72.09302325581395
Training Time (poly) : 0.06248760223388672
```

```
Optimal hyperparameters (linear):  c= 8 acc= 0.7297297297297297
Overall Accuracy (linear) :  67.44186046511628
Training Time (linear) : 0.0
```

```
Optimal hyperparameters (sigmoid): gamma= 0.0001220703125 coef0= -1 c= 64 accuracy= 0.5151515151515151
Overall Accuracy (sigmoid) :  41.86046511627907
Training Time (sigmoid) : 0.0
```

NOTE 1: **Overall Accuracy is calculated as %**

NOTE 2: **Please refer files ovo_all_acc.txt, ovr_all_acc.txt and weighted_all_acc.txt for accuracies computed using the tested hyperparameters' values in the code for one vs one, one vs all and weighted procedures.**

CONCLUSION:

- We find that **'one vs one'** procedure yields better accuracies for the tested hyperparameters compared to **'one vs all'** procedure.
- We also find that **'one vs all'** procedure has longer training time periods compared to **'one vs one'** procedure.
- **'Weighted'** procedure yields slightly better accuracies for polynomial and linear kernels compared to **'one vs all'** procedure.