Question 2.1

One Vs All:

One Vs All is also known as One vs Rest is a heuristic method for Binary classification algorithms in order to perform Multi-Class classification. It splitts the Multi-Class Dataset into multiple binary classification problem. After splitting the dataset into multiple binary classes it trains the binary classifier on each of the classes and then the prediction is made for the multiclass.

One vs One

One vs One is another heuristic technique that is used for Multiclass Class classification using binary Classification. Similar to One Vs All it splits the dataset into one binary data for each class vs every other class.

For the Training of the 'one vs one' and 'one vs all' strategy the dataset is divided into Training set and Testing set in the ratio of 70:30. Then the train and test dataset is Standardised and passed to the Linear SVM for Training. The Test is used for prediction in order to check the Generality of the Classifier on the unseen dataset. This is the main procedure for the Training of SVM.

From [sklearn] the Training and Classification procedure is as follows:

SVC in Sklearn implements the "one-versus-one" method for multi-class classification and this is done by finding the total number of Classifiers using the equation n_classes * (n_classes - 1) / 2 and then the classifiers are constructed and each one trains data from the two classes. In order to be consistent with the interface of other classifiers, in addition, the decision_function_shape option allows us to monotonically transform the results of the "one-versus-one" classifiers to a "one-vs-rest" decision function of shape (n_samples, n_classes).

In addition, from the [Paper] the algorithm is as follows for the Training and Classification Procedure:

1: First of all we Split X into two parts, X1 for training and X2 for validation

2: In each of the base learner bi∈T do

3: The classification model is: ClassificationModeli←Train(X1,bi,method);

4: Oi←Validate(X2,bi,ClassificationModeli,method);

5: end for

6: OT←∪Oi;

7: Find the correlation matrix A for OT;

8: Sort out the highly correlated groups of base learners, C, from A

9: Find a CPT for each group c⊂C of highly correlated classifiers using O;

10: k←0;

11: (C(k),T(k))←(C,T);

12: if opt=1 then ▹ Further optimizations desired

13: Untill convergence or stopping criteria do

14: do Simplification. (Ctmp,Ttmp)← Simplification(C(k),T(k),X,method); ▹ Optimization 1

15: do Replacement. (C(k+1),T(k+1))← Replacement(Ctmp,Ttmp,X,method); ▹ Optimization 2

16: k←k+1;

17: end while

18: end if

19: (C(final),T(final))←(C(k),T(k));

20: Classify xt according to 5 using the set of base learners Tfinal and groups of highly correlated base learners Cfinal.

In [1]:
```
# Import the Necessary Packages
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
```

Question 2.2:

In [2]:
```
# Load the DatasetB
dataB = pd.read_csv('Dataset/DataB.csv')
```

In [3]:
```
# DataB Display
print("DataB: \n" ,dataB)
```

```
DataB:
      sepallength  sepalwidth  petallength  petalwidth           class
0            5.1         3.5          1.4         0.2     Iris-setosa
1            4.9         3.0          1.4         0.2     Iris-setosa
2            4.7         3.2          1.3         0.2     Iris-setosa
3            4.6         3.1          1.5         0.2     Iris-setosa
4            5.0         3.6          1.4         0.2     Iris-setosa
..           ...         ...          ...         ...             ...
145          6.7         3.0          5.2         2.3  Iris-virginica
146          6.3         2.5          5.0         1.9  Iris-virginica
147          6.5         3.0          5.2         2.0  Iris-virginica
148          6.2         3.4          5.4         2.3  Iris-virginica
149          5.9         3.0          5.1         1.8  Iris-virginica

[150 rows x 5 columns]
```

In [4]:
```
# Check for Null values
dataB.isna().sum()
```

Out[4]:
```
sepallength    0
sepalwidth     0
petallength    0
petalwidth     0
class          0
dtype: int64
```

In [5]:
```
# Collect all the features of the DataB
dataB_X = dataB.drop(['class'], axis=1)
```

In [6]:
```
dataB_X
```

Out[6]:

	sepallength	sepalwidth	petallength	petalwidth
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [7]:
```
# Collect the Target Variables
dataB_y = dataB['class']
print(" Target for the DataB: \n", dataB_y)
```

```
Target for the DataB:
 0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
            ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: class, Length: 150, dtype: object
```

In [8]:
```
# Check the Unique values in the Target
print("Unique Values in DataB: \n", dataB_y.unique())
```

```
Unique Values in DataB:
 ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

In [9]:
```
# Transform the Labels to Numbers
le = LabelEncoder()
dataB_y_num = le.fit_transform(dataB_y)
```

In [10]:
```
print(" Transformed Values of the Target: \n",dataB_y_num)
```

```
Transformed Values of the Target:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

In [11]:
```
# Split the dataset in the ratio of 70:30
X_train, X_test, y_train, y_test = train_test_split(dataB_X, dataB_y_num, test_size = 0.30, random_stat
e = 42)
```

In [12]:
```
# Standardise the dataset
ss = StandardScaler()
X_train_std = ss.fit_transform(X_train)
X_test_std = ss.fit_transform(X_test)
```

One vs all Classification

In [13]:
```
# Initalize the Class with linear and One vs all method
svm_ovr = SVC(kernel= 'linear', decision_function_shape= 'ovr')
```

In [14]:
```
# Train the dataset on the SVM classifier
svm_ovr.fit(X_train_std, y_train)
y_pred_ovr = svm_ovr.predict(X_test_std)
```

In [15]:
```
# Accuracy Score
acc_score = accuracy_score(y_test, y_pred_ovr)
print("Accuracy Score for One vs all Method: ", acc_score*100)
```

```
Accuracy Score for One vs all Method:  93.33333333333333
```

In [16]:
```
# Check the Categorical report for the Classifier
print("Classification Report for the One Vs All Method Using Kernel = 'Linear': \n", classification_rep
ort(le.inverse_transform(y_test), le.inverse_transform(y_pred_ovr)))
```

```
Classification Report for the One Vs All Method Using Kernel = 'Linear':
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        19
Iris-versicolor       1.00      0.77      0.87        13
 Iris-virginica       0.81      1.00      0.90        13

       accuracy                           0.93        45
      macro avg       0.94      0.92      0.92        45
   weighted avg       0.95      0.93      0.93        45
```

In [17]:
```
# Confusion Matrix for the predicted value
print("Confusion Matrix for One Vs All Method: \n", confusion_matrix(y_test, y_pred_ovr))
```

```
Confusion Matrix for One Vs All Method:
 [[19  0  0]
 [ 0 10  3]
 [ 0  0 13]]
```

One vs One Classification

In [18]:
```
# Initilaze the Classifier with one vs one method and linear kernel
svm_ovo = SVC(kernel= 'linear', decision_function_shape= 'ovo')
```

In [19]:
```
# Train and Test the Dataset
svm_ovo.fit(X_train_std, y_train)
y_pred_ovo = svm_ovo.predict(X_test_std)
acc_score_ovo = accuracy_score(y_test, y_pred_ovo)
```

In [20]:
```
# Check the Accuracy Score for the Dataset
print("Accuracy_Score for One Vs One: ", acc_score_ovo*100)
```

```
Accuracy_Score for One Vs One:  93.33333333333333
```

In [21]:
```
# Check the Classification Report for the Classifier
print("Classification Report for the One Vs All Method Using Kernel = 'Linear': \n", classification_rep
ort(le.inverse_transform(y_test), le.inverse_transform(y_pred_ovo)))
```

```
Classification Report for the One Vs All Method Using Kernel = 'Linear':
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        19
Iris-versicolor       1.00      0.77      0.87        13
 Iris-virginica       0.81      1.00      0.90        13

       accuracy                           0.93        45
      macro avg       0.94      0.92      0.92        45
   weighted avg       0.95      0.93      0.93        45
```

In [22]:
```
# Check the Confusion Matrix for the Classifier
print("Confusion Matrix for One Vs All Method: \n", confusion_matrix(y_test, y_pred_ovo))
```

```
Confusion Matrix for One Vs All Method:
 [[19  0  0]
 [ 0 10  3]
 [ 0  0 13]]
```

Question 2.3: Decision Tree

In the Decision Tree a series of questions are asked and every time an answer is obtained, a follow-up question is asked until a conclusion is reached about the label of the class. So each leaf node is given a class label and the non-terminal nodes and other internal nodes which contain the test conditions are used to separate records with different characteristics. A Decision Tree is constructed and then starting from the root node the test conditions are applied and appropriate branch is followed based on the new test condition. This guides to either internal node or to a leaf node. If it guides to the internal node then a new test condition is applied. The class label which is given to the leaf node is then given to that particular record. So for the multi-class classification the split in the decision tree depends on the number of definite values for that particular attribute. Example, if the gender of a person can be male, female or others then an attribute like gender will have a three split with three distinct value like male, female and others. One such approach could be Hunt's algorithm as in case of multiple classes an attribute that is selected partitions the dataset into smaller dataset and for each value of the attribute the child node is created. This algorithm grows the tree in recursive way until the full tree is obtained. Classification and Regression Trees (CART) does not work for multi classes as it only gives binary splits by taking into account all the 2^(k-1)-1 ways of forming binary partition of k attributes.

In [23]:
```
# Initialize the Classifier
dtclf = DecisionTreeClassifier()
dtclf.fit(X_train_std, y_train)
y_pred_DT = dtclf.predict(X_test_std)
DT_accuracy_store = accuracy_score(y_test, y_pred_DT)
print("Accuracy Score for Decision Tree: ", DT_accuracy_store*100)
```

```
Accuracy Score for Decision Tree:  95.55555555555556
```

In [24]:
```
# Check the Classification Report for the Classifier
print("Classification Report for the One Vs All Method Using Kernel = 'Linear': \n", classification_rep
ort(le.inverse_transform(y_test), le.inverse_transform(y_pred_DT)))
```

```
Classification Report for the One Vs All Method Using Kernel = 'Linear':
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        19
Iris-versicolor       1.00      0.85      0.92        13
 Iris-virginica       0.87      1.00      0.93        13

       accuracy                           0.96        45
      macro avg       0.96      0.95      0.95        45
   weighted avg       0.96      0.96      0.96        45
```

In [25]:
```
# Check the Confusion Matrix for the Classifier
print("Confusion Matrix for One Vs All Method: \n", confusion_matrix(y_test, y_pred_DT))
```

```
Confusion Matrix for One Vs All Method:
 [[19  0  0]
 [ 0 11  2]
 [ 0  0 13]]
```

Comment on the Result:

Question 2.2:

The accuracy score for Linear SVM classifier with 'one vs one' classification is 93.33%. The precision for Iris-setosa is 1.00, recall is 1.00 and f1-score is 1.00, however, for the value for recall and f1-score decreases for Iris-Versicolor to 0.77 and 0.87 respectively keeping the value of precision to 1.00. The value of precision for Iris-virginica is 0.81 which is lower than that of Iris-setosa and Iris-Versicolor. The values obtained from the Confusion Matrix shows that there are 3 samples in Iris-Versicolor which has been classified as Iris-virginica which is incorrect. Apart from that there is no misclassification for the samples. The results are identical for 'one vs all' case as well.

The results obtained by Decision Tree is different from that of Linear SVM. In Decision Tree, for Iris-setosa, the value of precision is 1.00, recall is 1.00 and f1-score is 1.00 which is identical to that of linear SVM. For Iris-Versicolor the value of precision is 1.00, recall is 0.85 and f1-score is 0.92 here the value of recall and f1-score is higher than that of Linear SVM. However the precision value for Iris-virginica is 0.87 which is higher than that of linear SVM, the recall value is 1.00 which is same as in the case of Linear SVM and the value of f1-score for Iris-virginica increases by 0.03 for Decision tree to 0.93. In addition, from the value of Confusion Matrix it is clear that only 2 samples of Iris-Versicolor was misclassified as Iris-virginica which is lower than that of Linear SVM.

Overall the accuracy score for Decision Tree is 95.55% which is higher than that of Linear SVM's 93.33%.