Task 1

(a)

result of Training Confusion Matrix represents

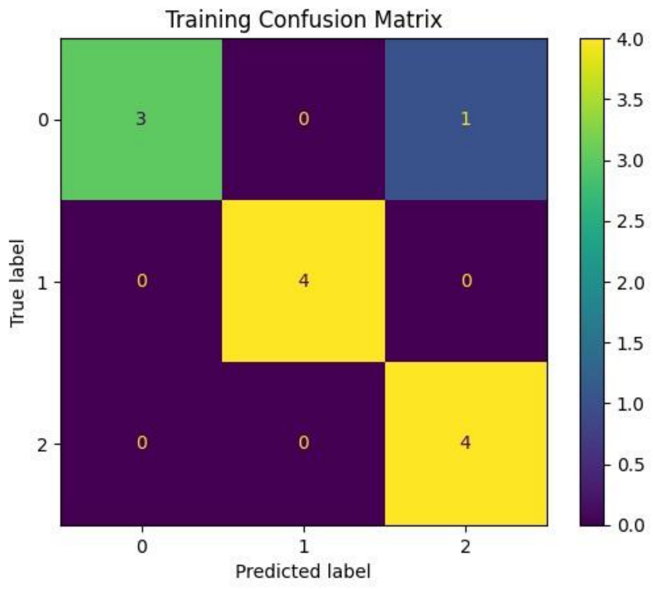
True label 0: There are 3 instances that belong to true label 0 and were correctly predicted as label 0 (true positive). There is 1 instance that belongs to true label 0 but was incorrectly predicted as label 2 (false negative).

True label 1: There are 4 instances that belong to true label 1 and were correctly predicted as label 1 (true positive). There are no instances that belong to true label 1 and were incorrectly predicted as any other label (false negative).

True label 2: There are 4 instances that belong to true label 2 and were correctly predicted as label 2

(true positive). There are no instances that belong to true label 2 and were incorrectly predicted as any other label (false negative).

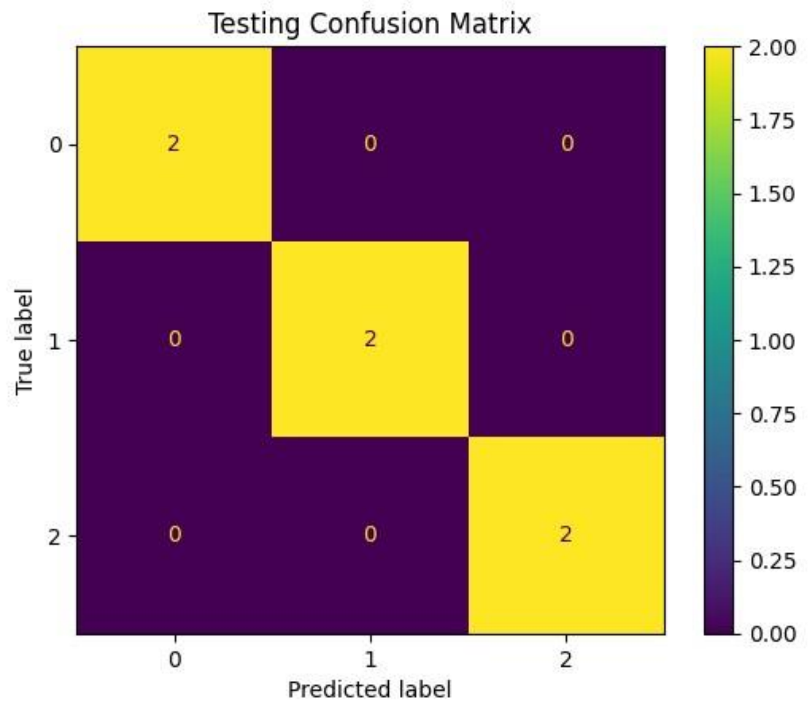
```
Training Confusion Matrix
[[3 0 1]
 [0 4 0]
 [0 0 4]]
```



```
Testing Confusion Matrix
[[2 0 0]
 [0 2 0]
 [0 0 2]]
```

This Testing confusion matrix suggests that the model achieved perfect accuracy on the testing dataset, as it correctly classified all instances for each class without any errors.

```
Testing Confusion Matrix
[[2 0 0]
 [0 2 0]
 [0 0 2]]
```

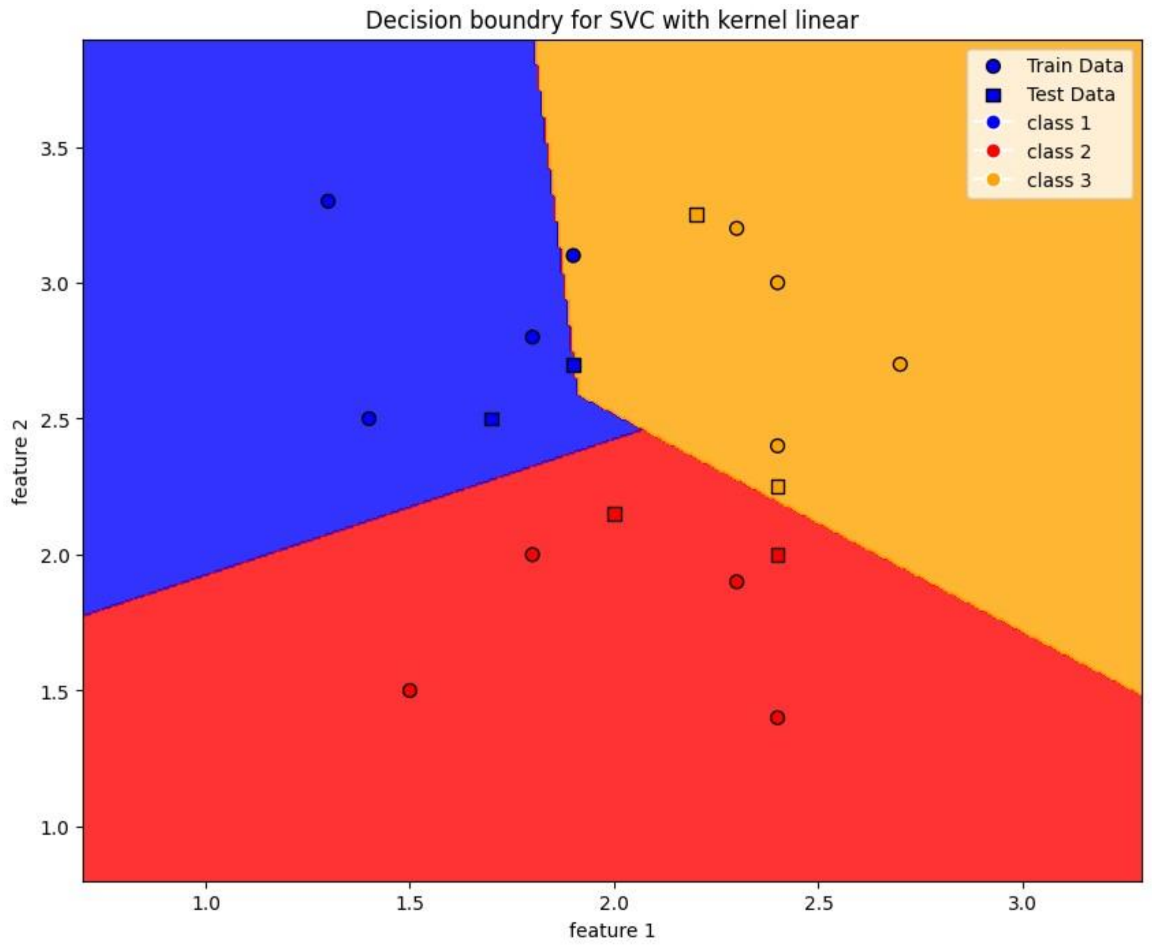


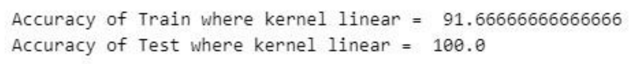
Accuracy of Train where kernel linear = 91.66666666666666:

This means that when evaluating the model on the training dataset, it achieved an accuracy of approximately 91.67%. this is seen from graph "plot_decision_boundary"where class one miss to know one point.

Accuracy of Test where kernel linear = 100.0:

This indicates that when evaluating the model on the testing dataset, it achieved a perfect accuracy of 100%. The model correctly classified all instances in the testing data, this is seen from graph where no class miss to know any point.

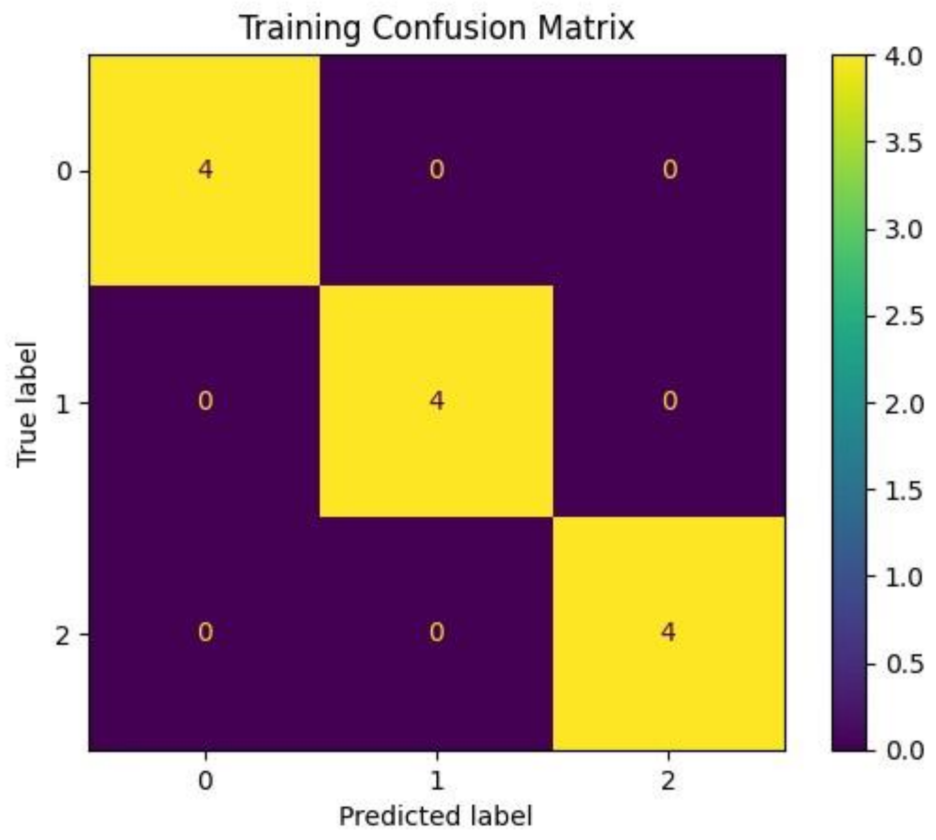




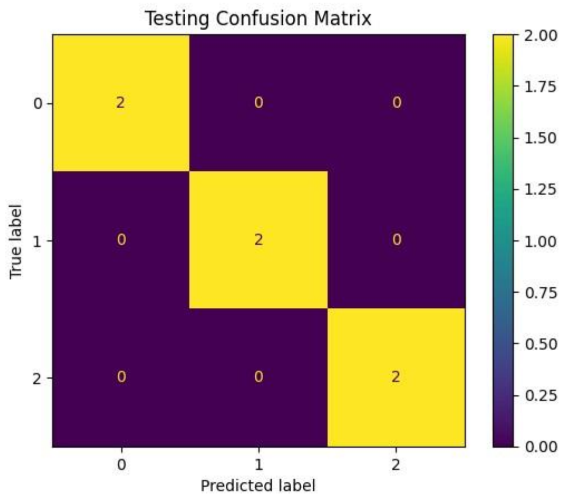
```
Accuracy of Train where kernel linear =  91.66666666666666
Accuracy of Test where kernel linear =  100.0
```

`Accuracy of Train for default SVC =  100.0.`
when evaluating the model on the training dataset, it achieved an accuracy of approximately 100%. this is seen from graph where no class miss to know any point.

```
Training Confusion Matrix
[[4 0 0]
 [0 4 0]
 [0 0 4]]
```

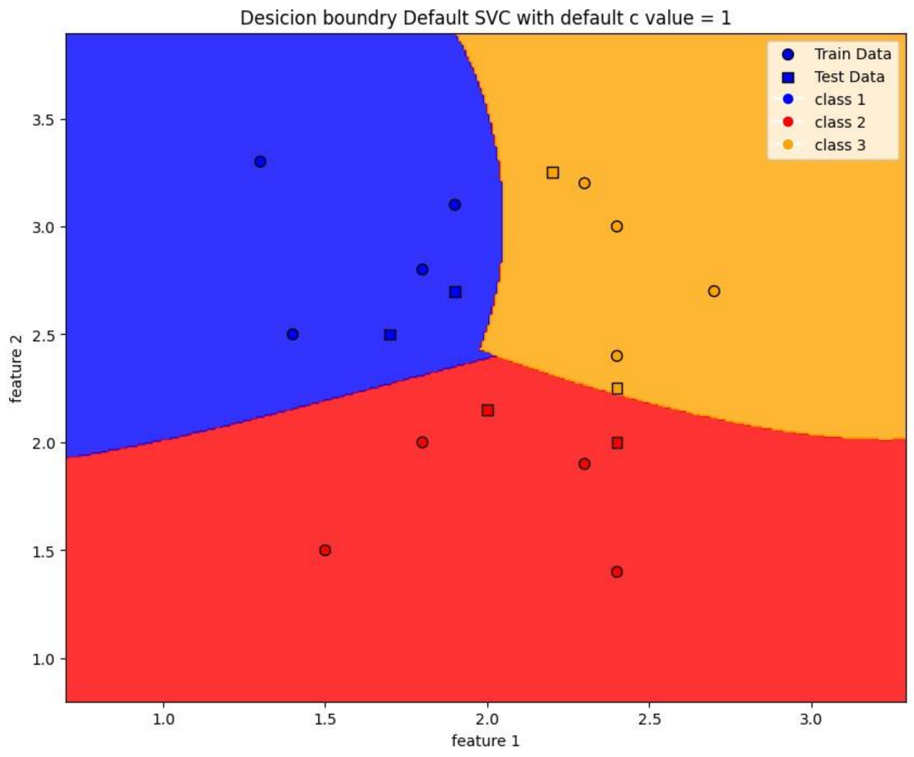


```
Testing Confusion Matrix
[[2 0 0]
 [0 2 0]
 [0 0 2]]
```



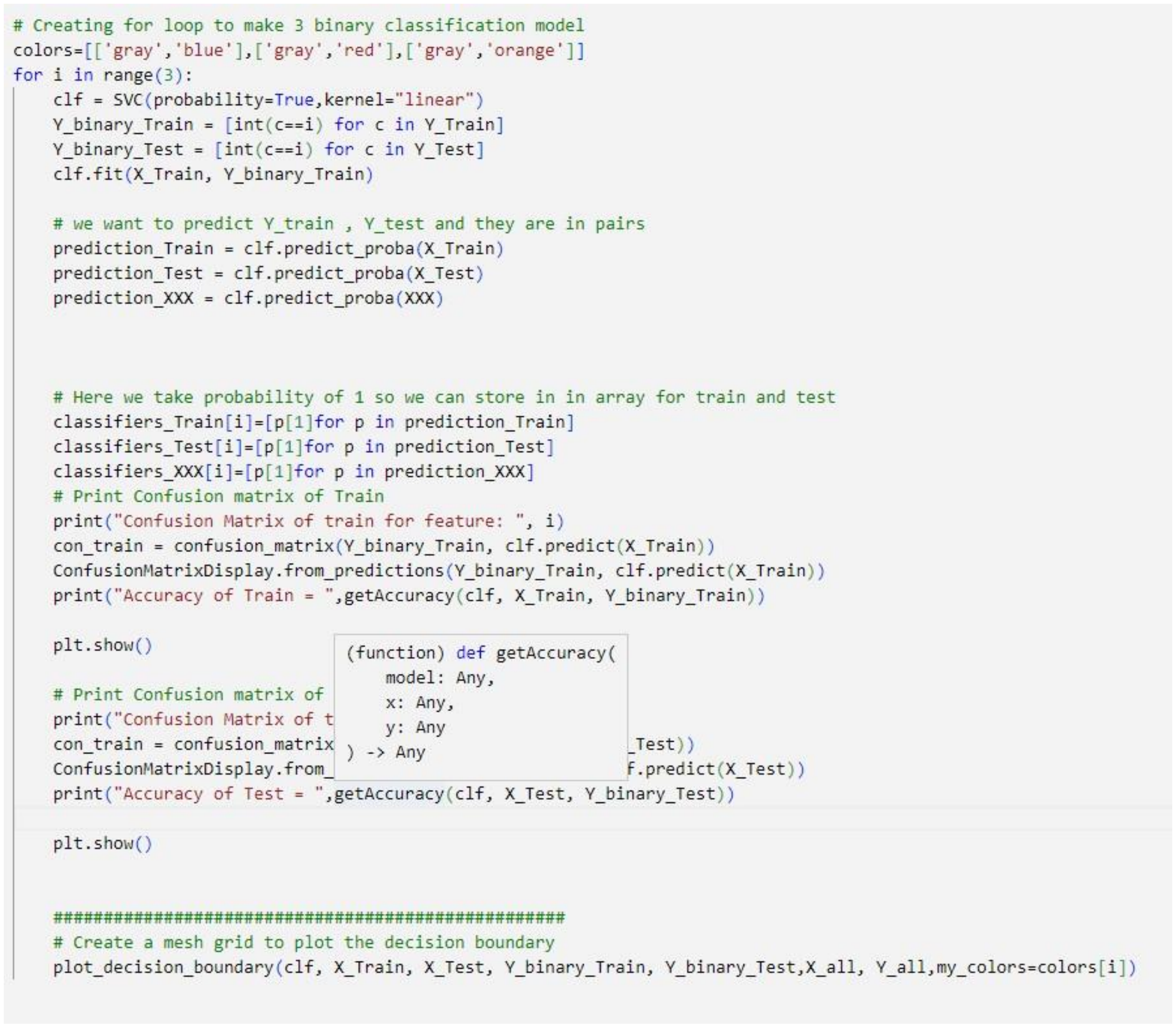
Accuracy of Test for `for default` SVC = 100.0:

when evaluating the model on the testing dataset, it achieved a perfect accuracy of 100%. The model correctly classified all instances in the testing data, this is seen from graph where no class miss to know any point.



This is graph that ensure the above accuracy for Test and train `for default SVC.`

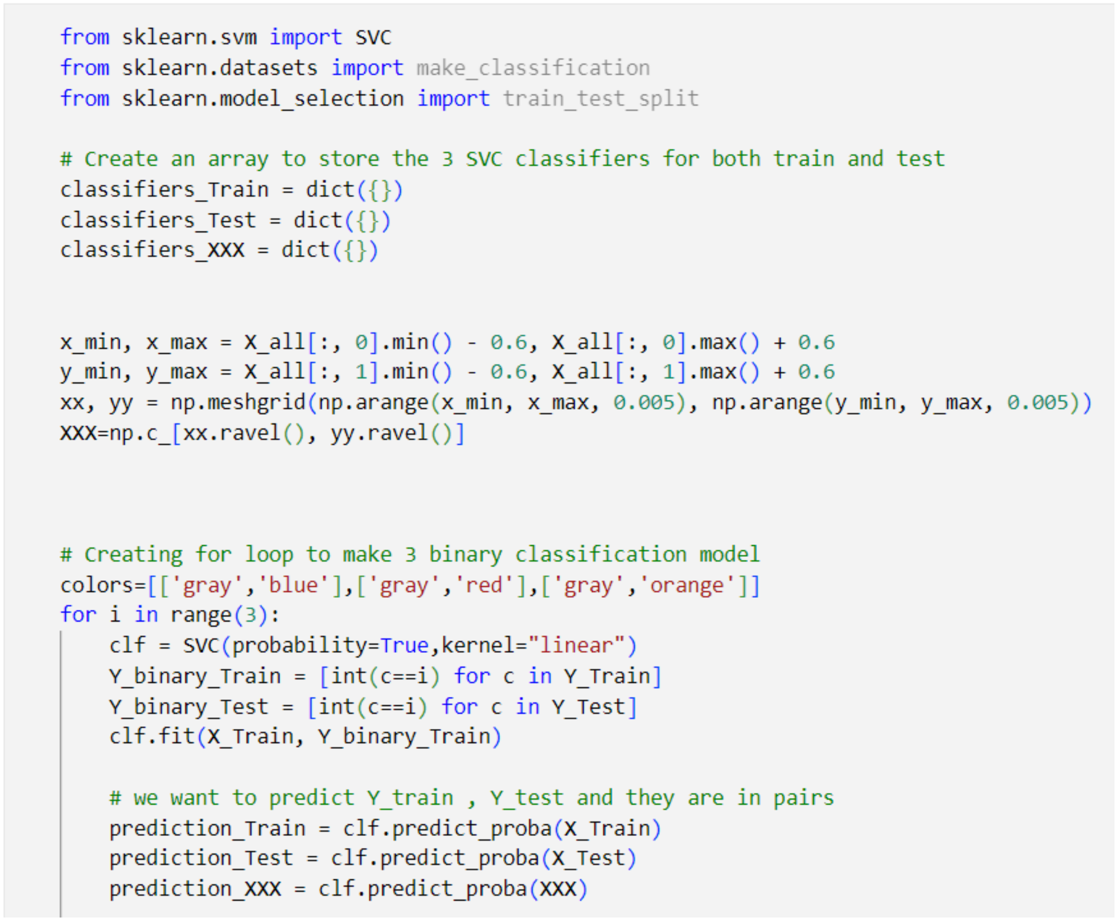
We train three binary classifiers using the Support Vector Classifier (SVC) with linear kernel. It then makes predictions, stores the probabilities, and visualizes the results including the confusion matrices, accuracies, and decision boundaries for each class.



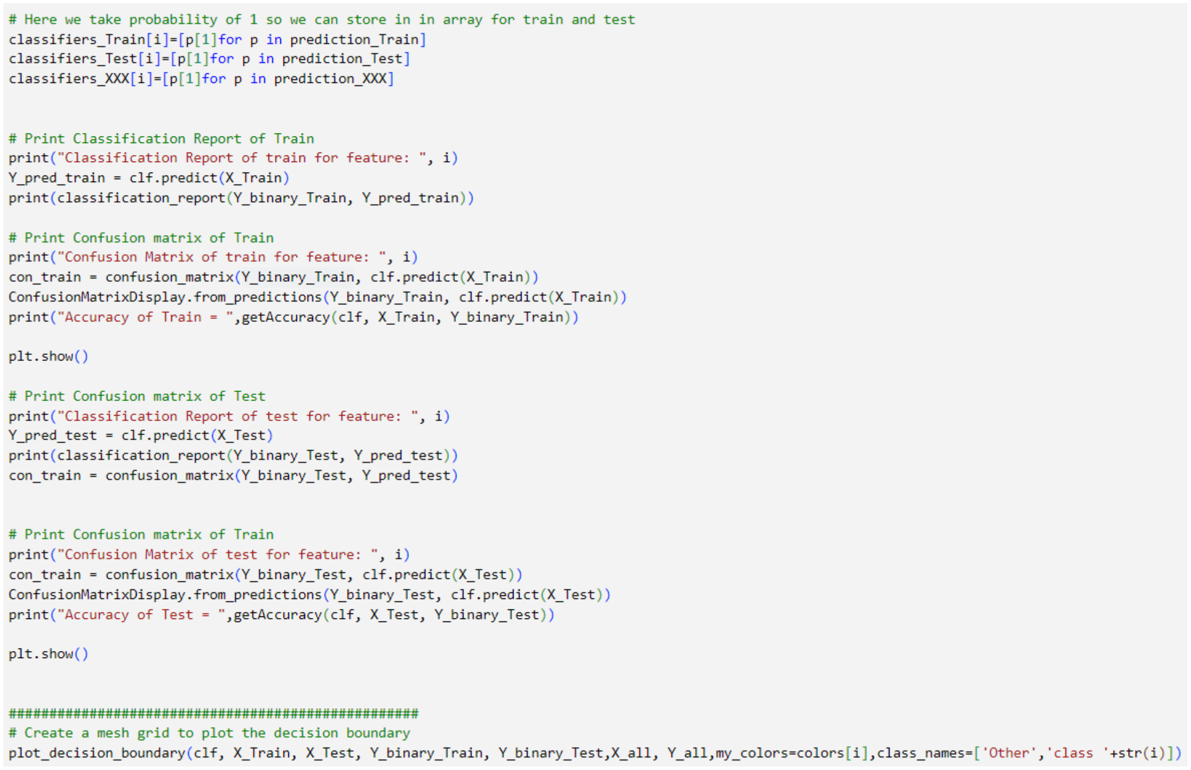
```
# Creating for loop to make 3 binary classification model
colors=[['gray','blue'],['gray','red'],['gray','orange']]
for i in range(3):
    clf = SVC(probability=True,kernel="linear")
    Y_binary_Train = [int(c==i) for c in Y_Train]
    Y_binary_Test = [int(c==i) for c in Y_Test]
    clf.fit(X_Train, Y_binary_Train)

    # we want to predict Y_train , Y_test and they are in pairs
    prediction_Train = clf.predict_proba(X_Train)
    prediction_Test = clf.predict_proba(X_Test)
    prediction_XXX = clf.predict_proba(XXX)



    # Here we take probability of 1 so we can store in in array for train and test
    classifiers_Train[i]=[p[1]for p in prediction_Train]
    classifiers_Test[i]=[p[1]for p in prediction_Test]
    classifiers_XXX[i]=[p[1]for p in prediction_XXX]
    # Print Confusion matrix of Train
    print("Confusion Matrix of train for feature: ", i)
    con_train = confusion_matrix(Y_binary_Train, clf.predict(X_Train))
    ConfusionMatrixDisplay.from_predictions(Y_binary_Train, clf.predict(X_Train))
    print("Accuracy of Train = ",getAccuracy(clf, X_Train, Y_binary_Train))

    plt.show()                  (function) def getAccuracy(
                                    model: Any,
    # Print Confusion matrix of         x: Any,
    print("Confusion Matrix of t        y: Any
    con_train = confusion_matrix ) -> Any                        _Test))
    ConfusionMatrixDisplay.from_                                 f.predict(X_Test))
    print("Accuracy of Test = ",getAccuracy(clf, X_Test, Y_binary_Test))

    plt.show()


    ####################################################
    # Create a mesh grid to plot the decision boundary
    plot_decision_boundary(clf, X_Train, X_Test, Y_binary_Train, Y_binary_Test,X_all, Y_all,my_colors=colors[i])
```

One-vs-Rest SVM

(b)



```python
from sklearn.svm import SVC
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Create an array to store the 3 SVC classifiers for both train and test
classifiers_Train = dict({})
classifiers_Test = dict({})
classifiers_XXX = dict({})


x_min, x_max = X_all[:, 0].min() - 0.6, X_all[:, 0].max() + 0.6
y_min, y_max = X_all[:, 1].min() - 0.6, X_all[:, 1].max() + 0.6
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.005), np.arange(y_min, y_max, 0.005))
XXX=np.c_[xx.ravel(), yy.ravel()]



# Creating for loop to make 3 binary classification model
colors=[['gray','blue'],['gray','red'],['gray','orange']]
for i in range(3):
    clf = SVC(probability=True,kernel="linear")
    Y_binary_Train = [int(c==i) for c in Y_Train]
    Y_binary_Test = [int(c==i) for c in Y_Test]
    clf.fit(X_Train, Y_binary_Train)

    # we want to predict Y_train , Y_test and they are in pairs
    prediction_Train = clf.predict_proba(X_Train)
    prediction_Test = clf.predict_proba(X_Test)
    prediction_XXX = clf.predict_proba(XXX)
```



```python
# Here we take probability of 1 so we can store in in array for train and test
classifiers_Train[i]=[p[1]for p in prediction_Train]
classifiers_Test[i]=[p[1]for p in prediction_Test]
classifiers_XXX[i]=[p[1]for p in prediction_XXX]


# Print Classification Report of Train
print("Classification Report of train for feature: ", i)
Y_pred_train = clf.predict(X_Train)
print(classification_report(Y_binary_Train, Y_pred_train))

# Print Confusion matrix of Train
print("Confusion Matrix of train for feature: ", i)
con_train = confusion_matrix(Y_binary_Train, clf.predict(X_Train))
ConfusionMatrixDisplay.from_predictions(Y_binary_Train, clf.predict(X_Train))
print("Accuracy of Train = ",getAccuracy(clf, X_Train, Y_binary_Train))

plt.show()

# Print Confusion matrix of Test
print("Classification Report of test for feature: ", i)
Y_pred_test = clf.predict(X_Test)
print(classification_report(Y_binary_Test, Y_pred_test))
con_train = confusion_matrix(Y_binary_Test, Y_pred_test)


# Print Confusion matrix of Train
print("Confusion Matrix of test for feature: ", i)
con_train = confusion_matrix(Y_binary_Test, clf.predict(X_Test))
ConfusionMatrixDisplay.from_predictions(Y_binary_Test, clf.predict(X_Test))
print("Accuracy of Test = ",getAccuracy(clf, X_Test, Y_binary_Test))

plt.show()


###################################################
# Create a mesh grid to plot the decision boundary
plot_decision_boundary(clf, X_Train, X_Test, Y_binary_Train, Y_binary_Test,X_all, Y_all,my_colors=colors[i],class_names=['Other','class '+str(i)])
```

- In previous code we take Create an array to store the 3 SVC classifiers for both train and test

We store them in dictionaries classifiers_Train, classifiers_Test, classifiers_XXX

- Define list of colors used in each classification to visualize it

- Creating for loop to make 3 binary classification model each iteration treated as binary classifier for example in first iteration class 0 vs rest means class 0 will be labeled as 1 and the rest (classes 1 and 2) will be labeled as 0

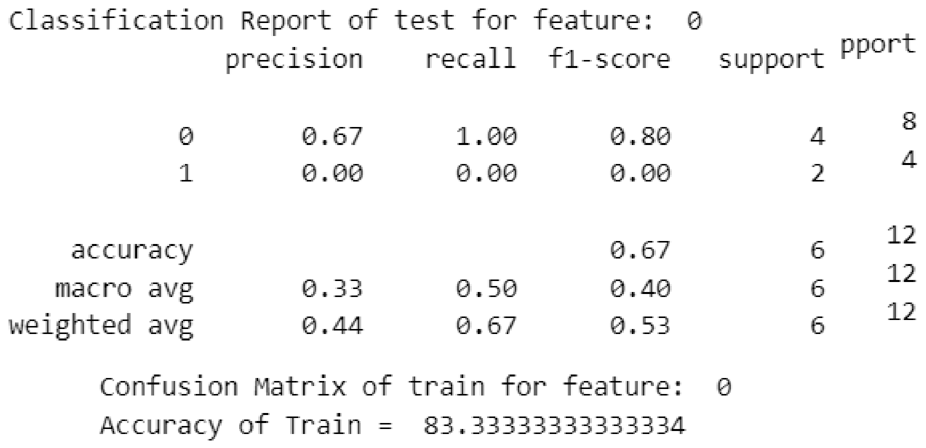
- we used predict probability to get a list with pairs to predict each element and get Y_binary classification for both train and test data that for each pair there is probability to be 0 or 1 so we want to have only the 1 probability so we created Classifiers to store only the one probability

- Print Classification Report for both train and test

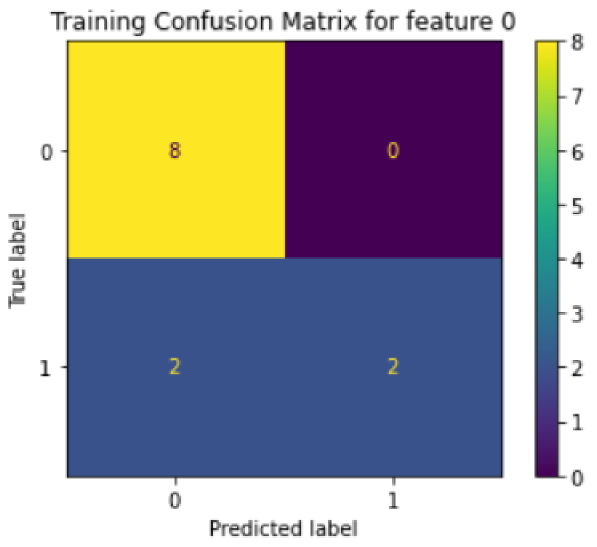
- Print Confusion matrix for both train and test

Finally Create a mesh grid to plot the decision boundary this function called: plot_decision_boundary this function takes the model name, train data, test data, Aggregated data of X and Y and a list for all colors used and name of class

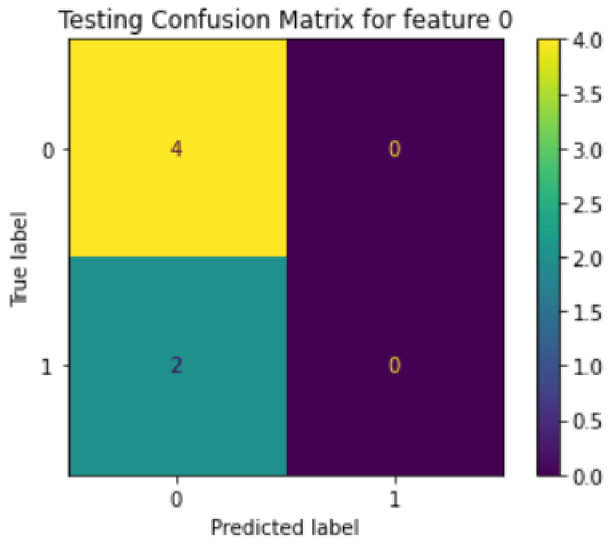
Now we will run code to visualize classification report and all confusion metrices and decision boundaries

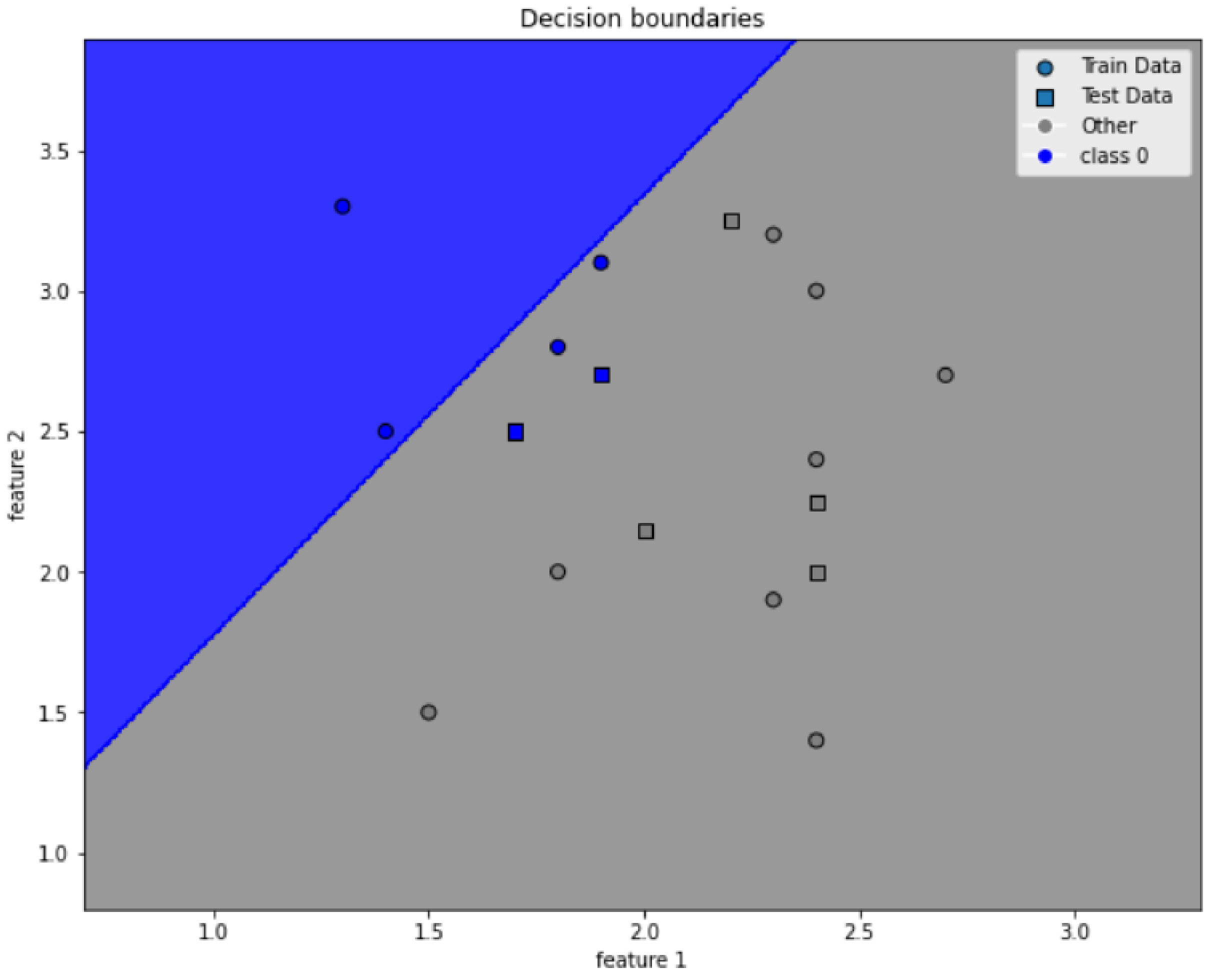


```
Classification Report of test for feature:  0
              precision    recall  f1-score   support pport

           0       0.67      1.00      0.80         4     8
           1       0.00      0.00      0.00         2     4

    accuracy                           0.67         6    12
   macro avg       0.33      0.50      0.40         6    12
weighted avg       0.44      0.67      0.53         6    12

      Confusion Matrix of train for feature:  0
      Accuracy of Train =  83.33333333333334
```

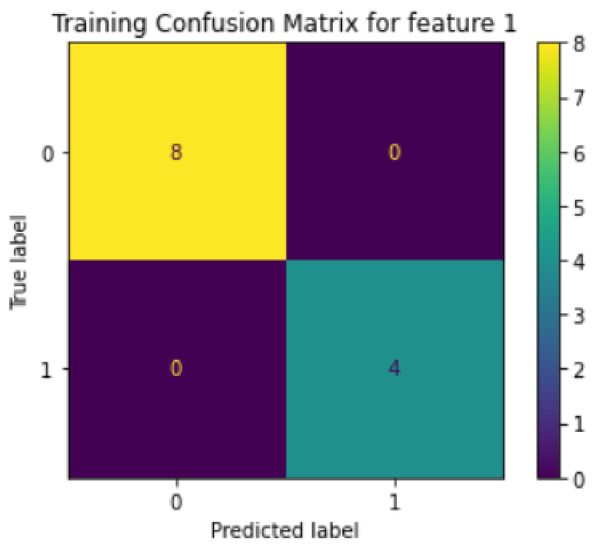


```
Confusion Matrix of test for feature:  0
Accuracy of Test =  66.66666666666666
```



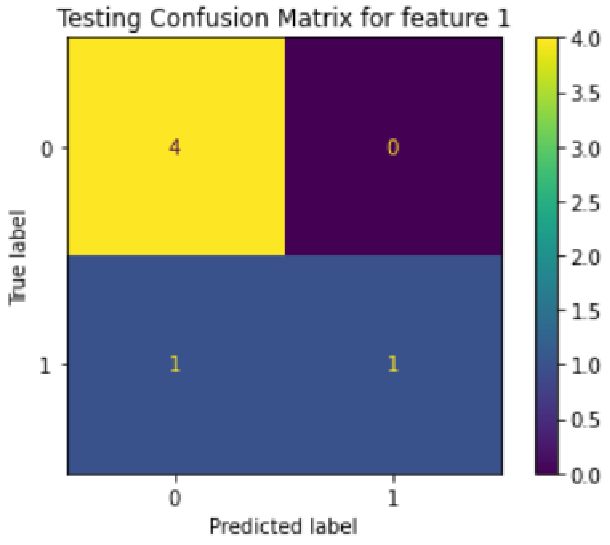


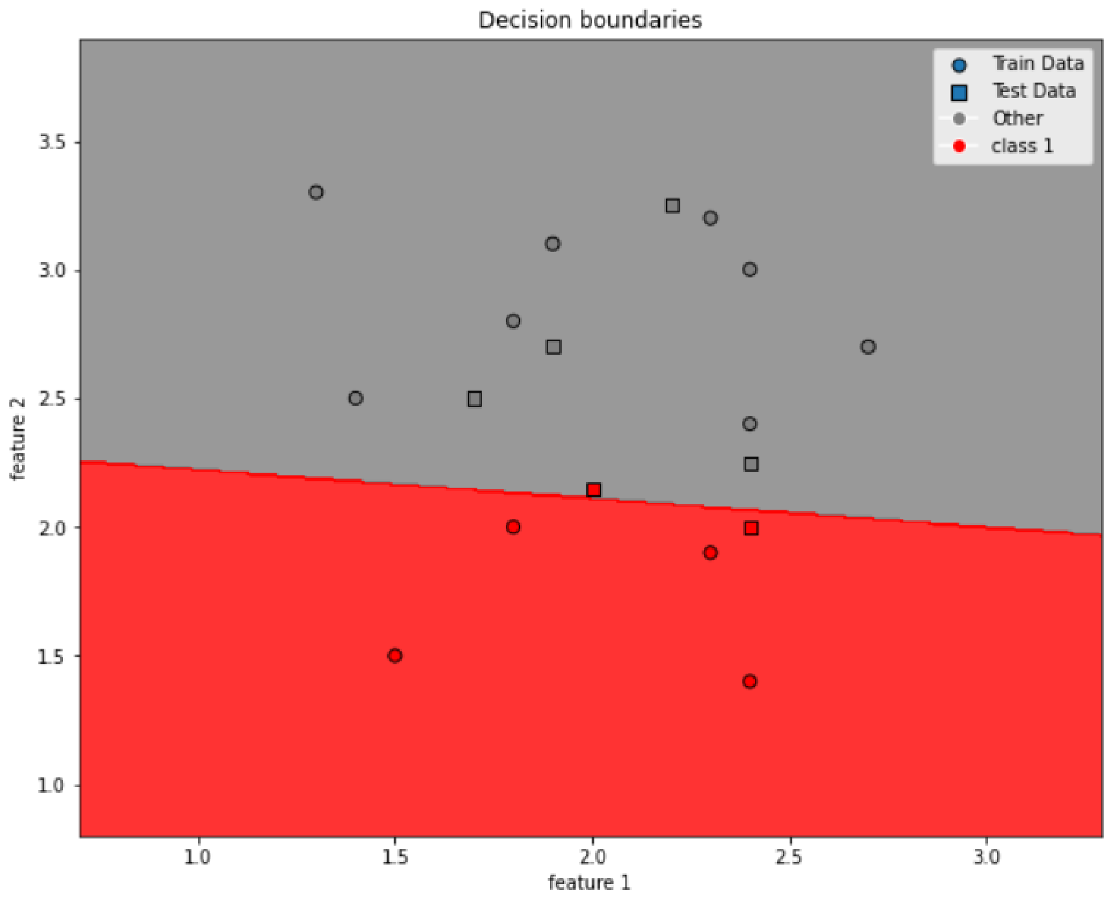
```
Classification Report of train for feature:  1
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       1.00      1.00      1.00         4

    accuracy                           1.00        12
   macro avg       1.00      1.00      1.00        12
weighted avg       1.00      1.00      1.00        12

Confusion Matrix of train for feature:  1
Accuracy of Train =  100.0
```

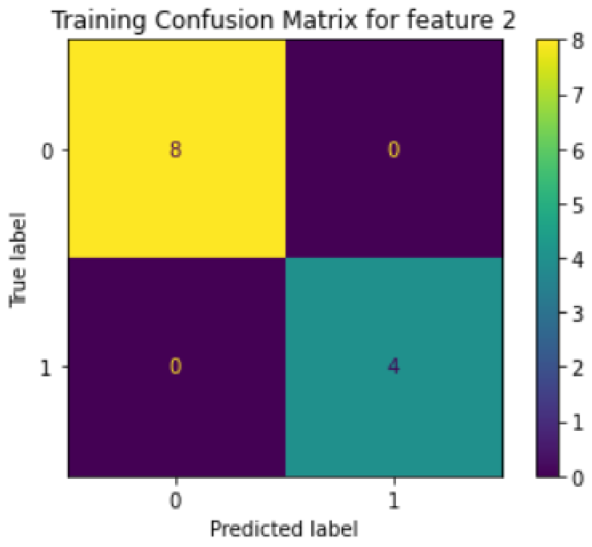


```
Classification Report of test for feature:  1
              precision    recall  f1-score   support

           0       0.80      1.00      0.89         4
           1       1.00      0.50      0.67         2

    accuracy                           0.83         6
   macro avg       0.90      0.75      0.78         6
weighted avg       0.87      0.83      0.81         6

Confusion Matrix of test for feature:  1
Accuracy of Test =  83.33333333333334
```

```
Classification Report of train for feature:  2
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       1.00      1.00      1.00         4

    accuracy                           1.00        12
   macro avg       1.00      1.00      1.00        12
weighted avg       1.00      1.00      1.00        12

Confusion Matrix of train for feature:  2
Accuracy of Train =  100.0
```

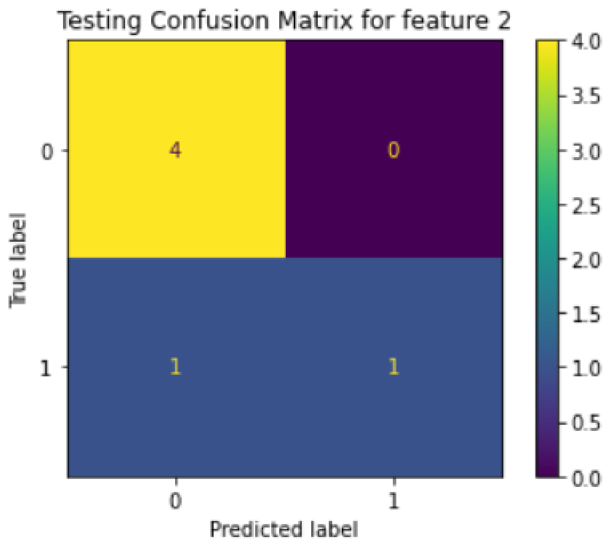


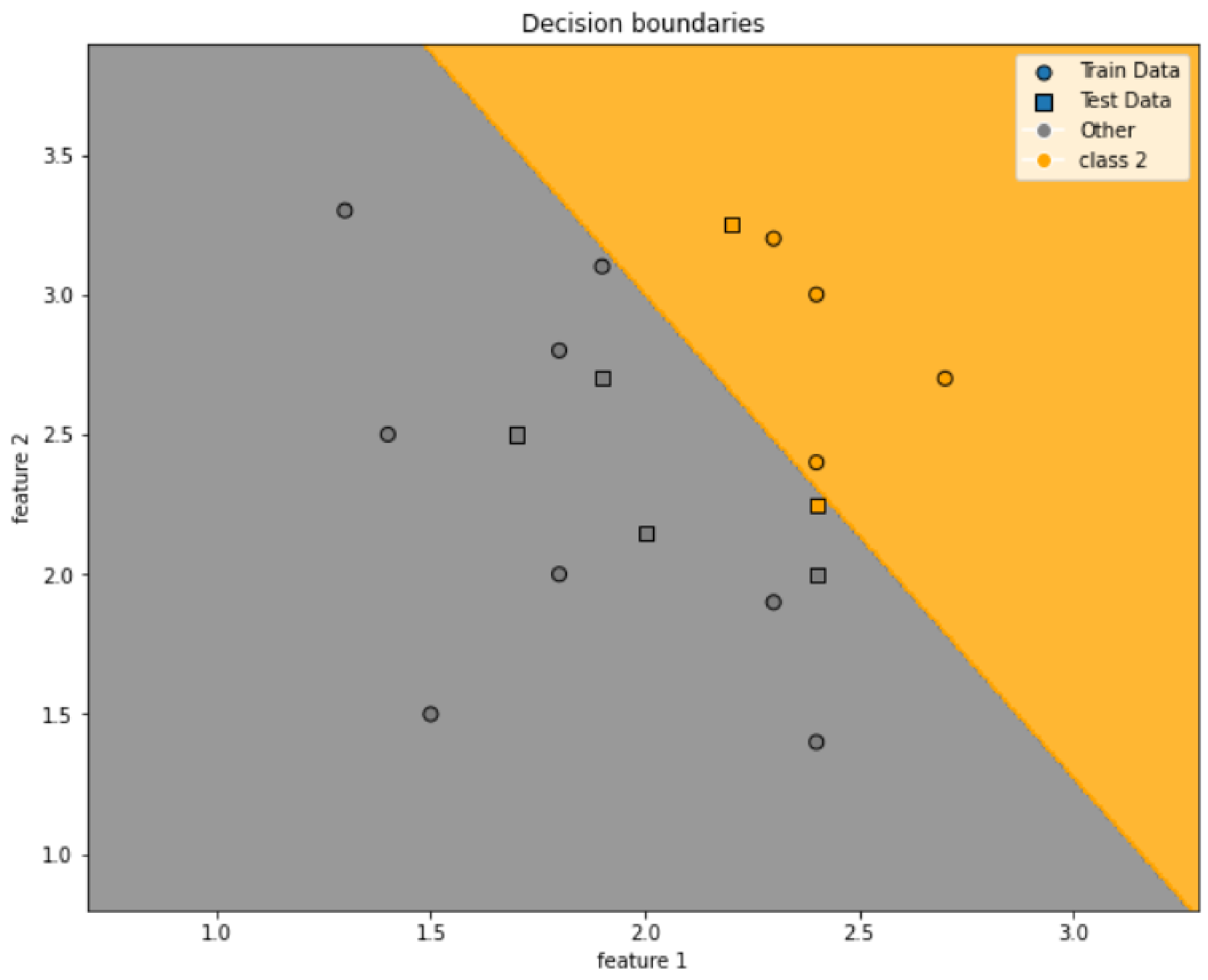




```
Classification Report of test for feature:  2
              precision    recall  f1-score   support

           0       0.80      1.00      0.89         4
           1       1.00      0.50      0.67         2

    accuracy                           0.83         6
   macro avg       0.90      0.75      0.78         6
weighted avg       0.87      0.83      0.81         6
```

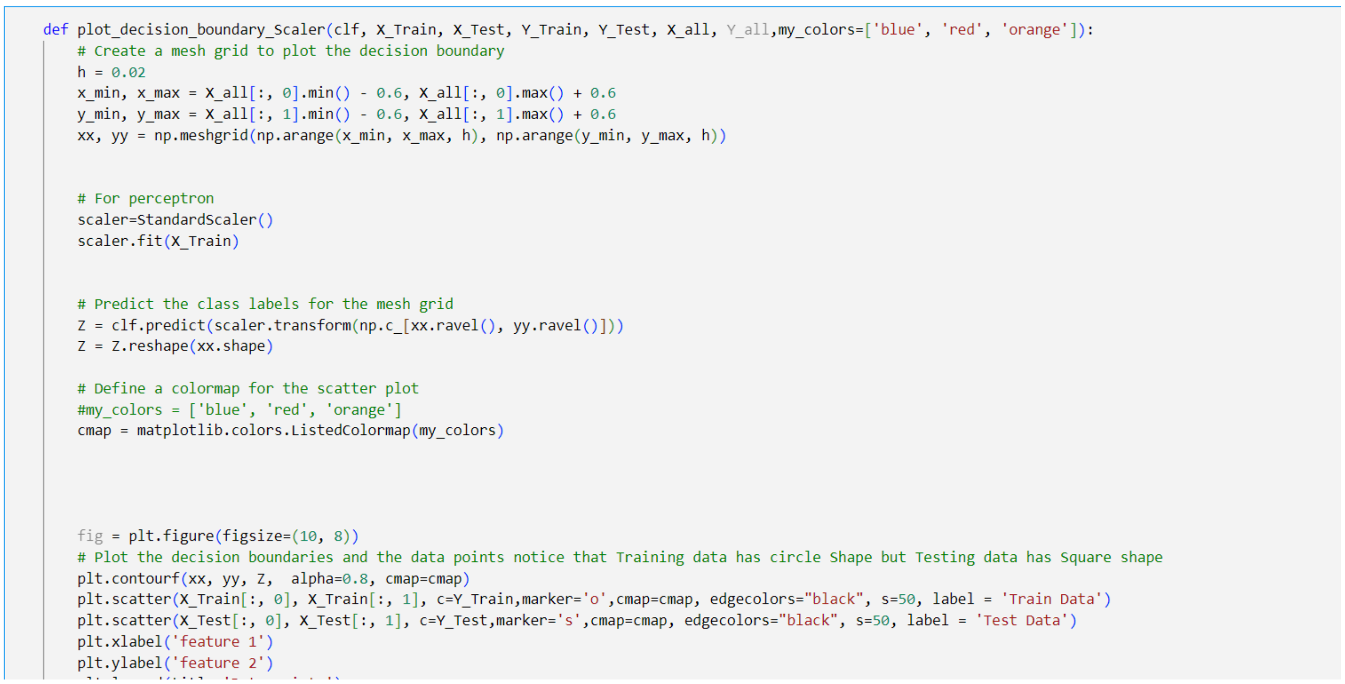
```
Confusion Matrix of test for feature:  2
Accuracy of Test =  83.33333333333334
```





Then we make Function to Plot Descision Boundry for Scaler One VS Rest Perceptron

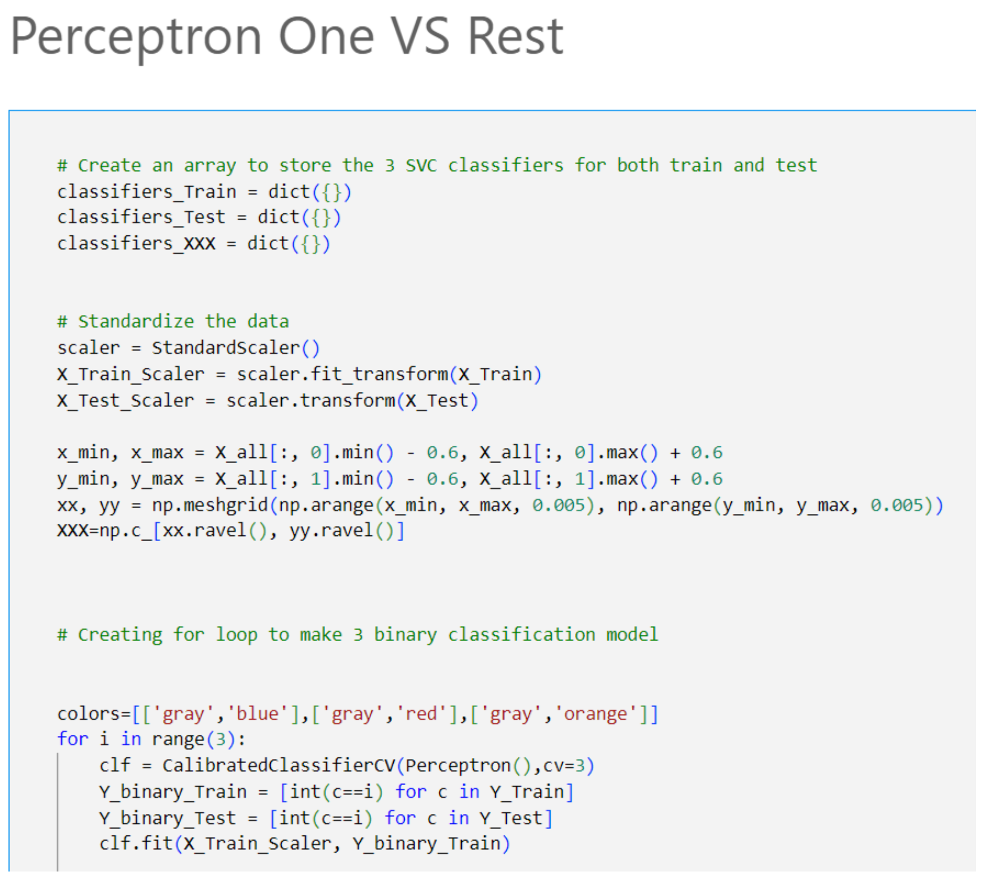
Function to Plot Descision Boundry for Scaler One VS Rest Perceptron



```
def plot_decision_boundary_Scaler(clf, X_Train, X_Test, Y_Train, Y_Test, X_all, Y_all,my_colors=['blue', 'red', 'orange']):
    # Create a mesh grid to plot the decision boundary
    h = 0.02
    x_min, x_max = X_all[:, 0].min() - 0.6, X_all[:, 0].max() + 0.6
    y_min, y_max = X_all[:, 1].min() - 0.6, X_all[:, 1].max() + 0.6
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))


    # For perceptron
    scaler=StandardScaler()
    scaler.fit(X_Train)


    # Predict the class labels for the mesh grid
    Z = clf.predict(scaler.transform(np.c_[xx.ravel(), yy.ravel()]))
    Z = Z.reshape(xx.shape)

    # Define a colormap for the scatter plot
    #my_colors = ['blue', 'red', 'orange']
    cmap = matplotlib.colors.ListedColormap(my_colors)




    fig = plt.figure(figsize=(10, 8))
    # Plot the decision boundaries and the data points notice that Training data has circle Shape but Testing data has Square shape
    plt.contourf(xx, yy, Z,  alpha=0.8, cmap=cmap)
    plt.scatter(X_Train[:, 0], X_Train[:, 1], c=Y_Train,marker='o',cmap=cmap, edgecolors="black", s=50, label = 'Train Data')
    plt.scatter(X_Test[:, 0], X_Test[:, 1], c=Y_Test,marker='s',cmap=cmap, edgecolors="black", s=50, label = 'Test Data')
    plt.xlabel('feature 1')
    plt.ylabel('feature 2')
```

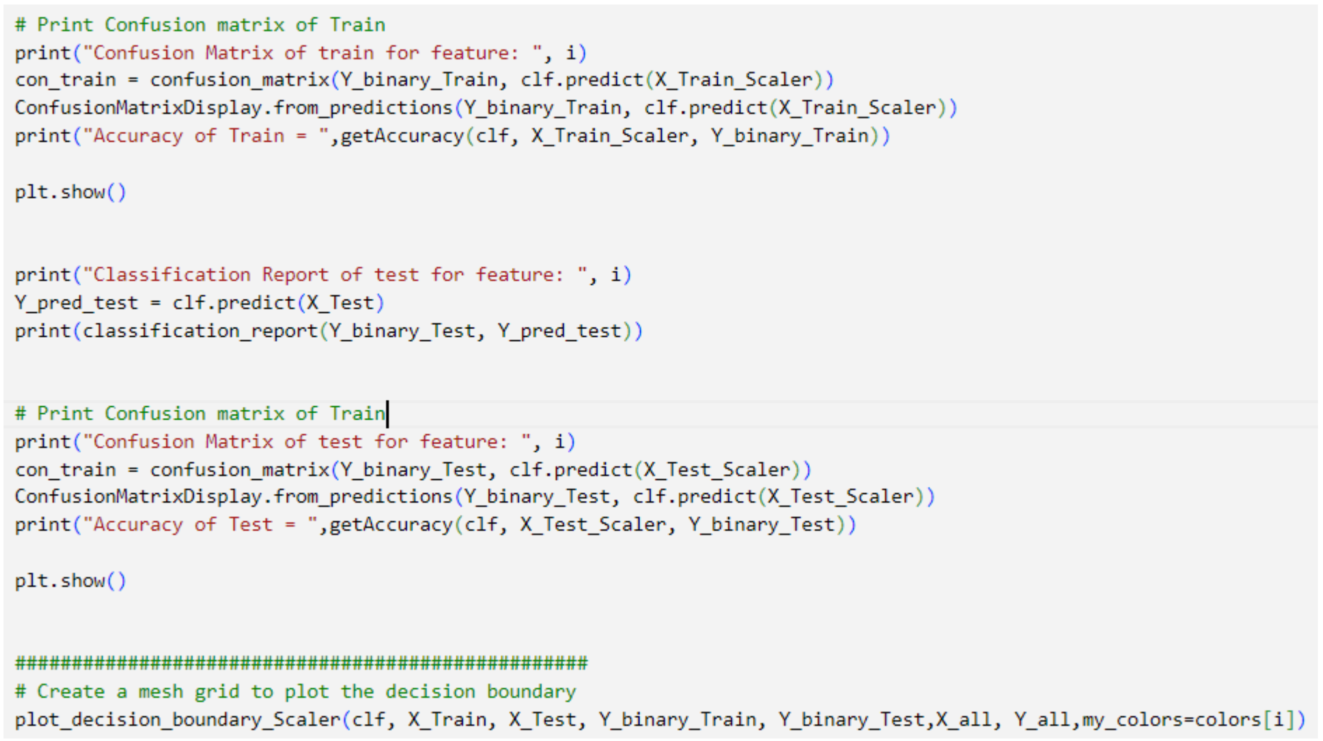
As we see in figure above It Looks like the function we used to plot OVR for SVC

Then we make Perceptron One VS Rest as follows:



Perceptron One VS Rest

```
# Create an array to store the 3 SVC classifiers for both train and test
classifiers_Train = dict({})
classifiers_Test = dict({})
classifiers_XXX = dict({})


# Standardize the data
scaler = StandardScaler()
X_Train_Scaler = scaler.fit_transform(X_Train)
X_Test_Scaler = scaler.transform(X_Test)

x_min, x_max = X_all[:, 0].min() - 0.6, X_all[:, 0].max() + 0.6
y_min, y_max = X_all[:, 1].min() - 0.6, X_all[:, 1].max() + 0.6
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.005), np.arange(y_min, y_max, 0.005))
XXX=np.c_[xx.ravel(), yy.ravel()]



# Creating for loop to make 3 binary classification model


colors=[['gray','blue'],['gray','red'],['gray','orange']]
for i in range(3):
    clf = CalibratedClassifierCV(Perceptron(),cv=3)
    Y_binary_Train = [int(c==i) for c in Y_Train]
    Y_binary_Test = [int(c==i) for c in Y_Test]
    clf.fit(X_Train_Scaler, Y_binary_Train)
```



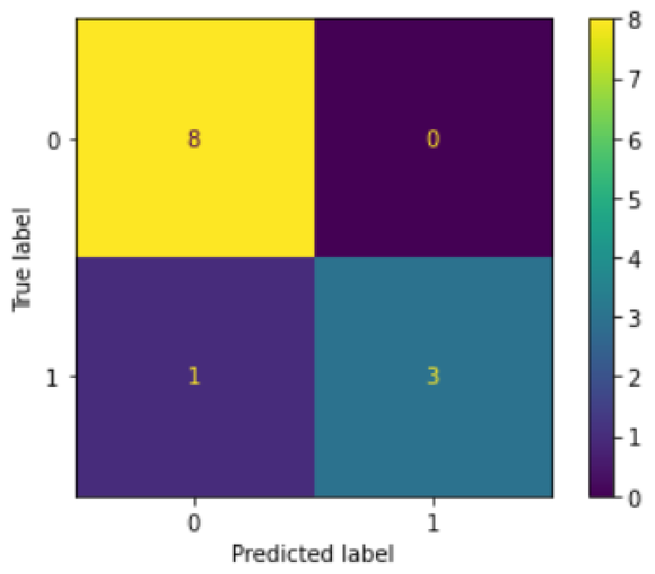
```
# Print Confusion matrix of Train
print("Confusion Matrix of train for feature: ", i)
con_train = confusion_matrix(Y_binary_Train, clf.predict(X_Train_Scaler))
ConfusionMatrixDisplay.from_predictions(Y_binary_Train, clf.predict(X_Train_Scaler))
print("Accuracy of Train = ",getAccuracy(clf, X_Train_Scaler, Y_binary_Train))

plt.show()


print("Classification Report of test for feature: ", i)
Y_pred_test = clf.predict(X_Test)
print(classification_report(Y_binary_Test, Y_pred_test))


# Print Confusion matrix of Train
print("Confusion Matrix of test for feature: ", i)
con_train = confusion_matrix(Y_binary_Test, clf.predict(X_Test_Scaler))
ConfusionMatrixDisplay.from_predictions(Y_binary_Test, clf.predict(X_Test_Scaler))
print("Accuracy of Test = ",getAccuracy(clf, X_Test_Scaler, Y_binary_Test))

plt.show()


#####################################################
# Create a mesh grid to plot the decision boundary
plot_decision_boundary_Scaler(clf, X_Train, X_Test, Y_binary_Train, Y_binary_Test,X_all, Y_all,my_colors=colors[i])
```

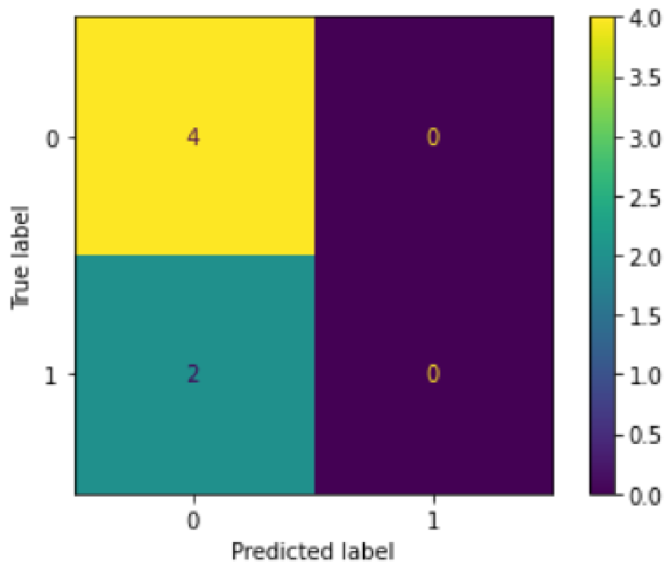
As we can see the code is very similar to OVR SVC with same concept except PERCEPTRON makes only output of 0 or 1 so we used standard scaler to utilize that

----

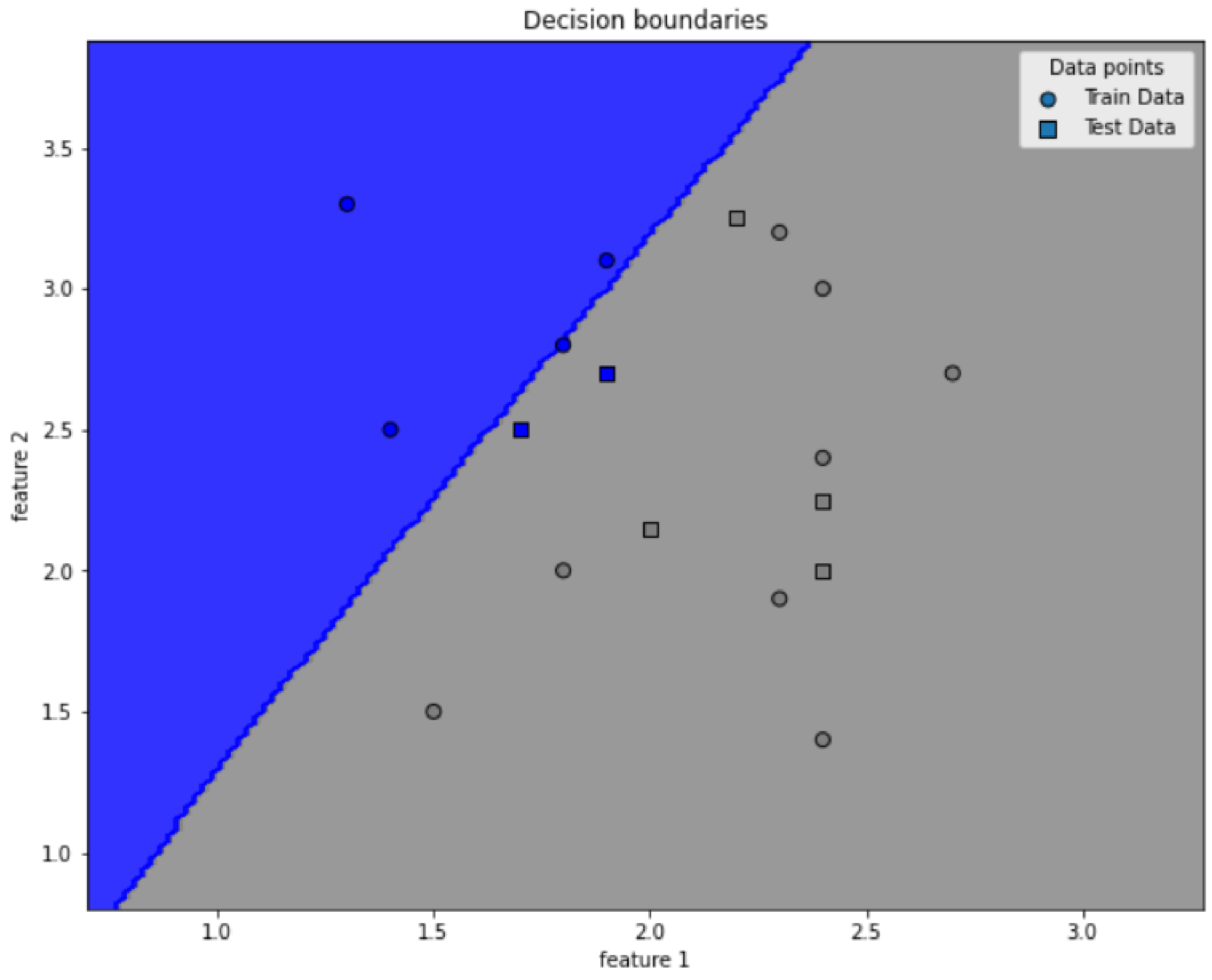
```
Classification Report of train for feature:  0
              precision    recall  f1-score   support

           0       0.73      1.00      0.84         8
           1       1.00      0.25      0.40         4

    accuracy                           0.75        12
   macro avg       0.86      0.62      0.62        12
weighted avg       0.82      0.75      0.69        12
```

```
Classification Report of test for feature:  0
              precision    recall  f1-score   support

           0       0.67      1.00      0.80         4
           1       0.00      0.00      0.00         2

    accuracy                           0.67         6
   macro avg       0.33      0.50      0.40         6
weighted avg       0.44      0.67      0.53         6

Confusion Matrix of test for feature:  0
Accuracy of Test =  66.66666666666666
```



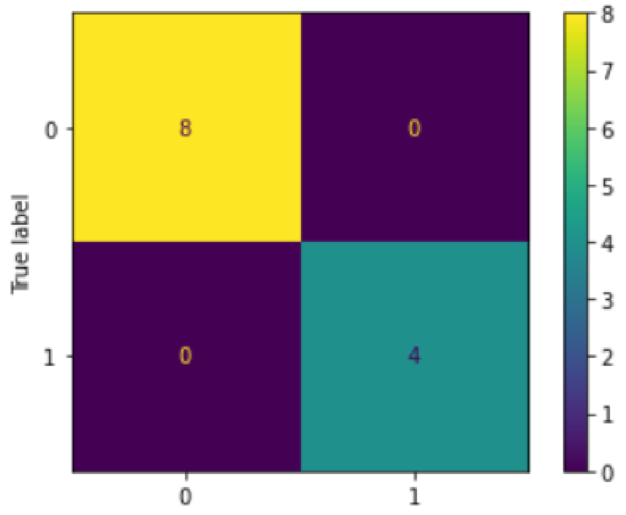


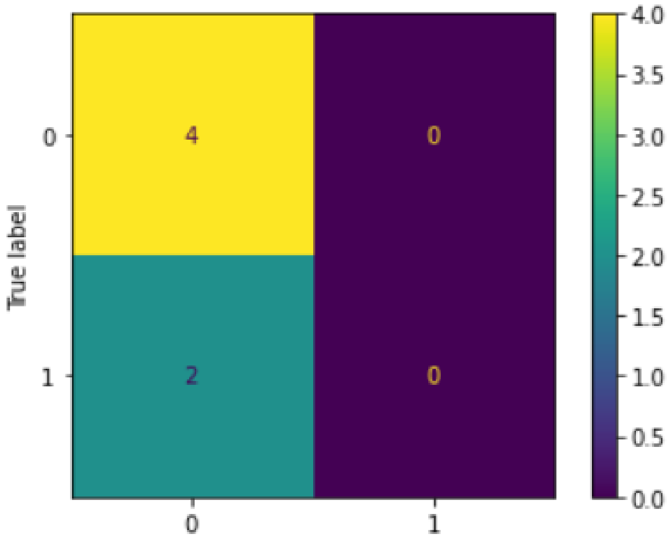
Desciision Boundary for feature 0:

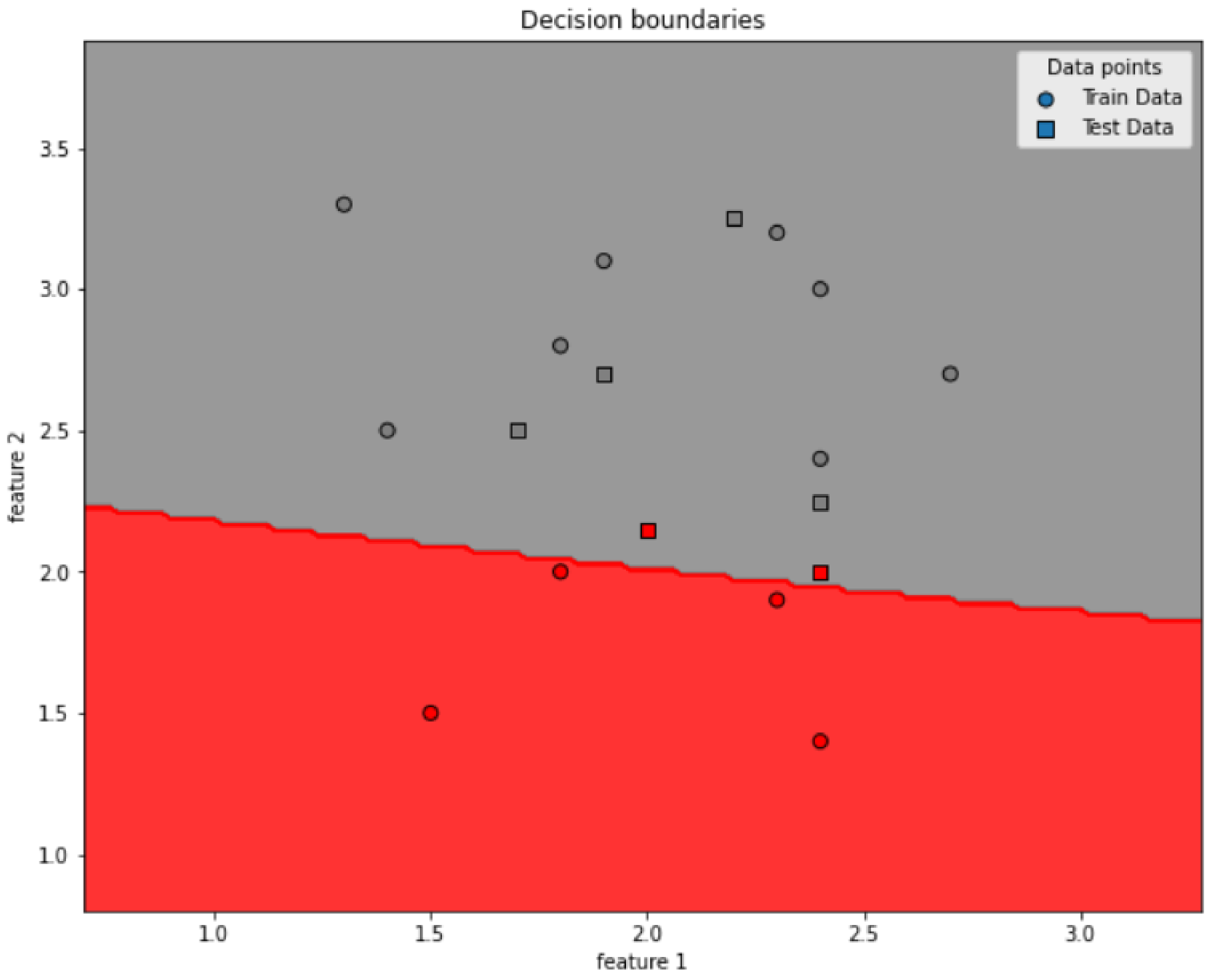


```
Classification Report of train for feature:  1
              precision    recall  f1-score   support

           0       0.67      1.00      0.80         8
           1       0.00      0.00      0.00         4

    accuracy                           0.67        12
   macro avg       0.33      0.50      0.40        12
weighted avg       0.44      0.67      0.53        12

Confusion Matrix of train for feature:  1
Accuracy of Train =  100.0
```

```
Classification Report of test for feature:  1
              precision    recall  f1-score   support

           0       0.67      1.00      0.80         4
           1       0.00      0.00      0.00         2

    accuracy                           0.67         6
   macro avg       0.33      0.50      0.40         6
weighted avg       0.44      0.67      0.53         6

Confusion Matrix of test for feature:  1
Accuracy of Test =  66.66666666666666
```

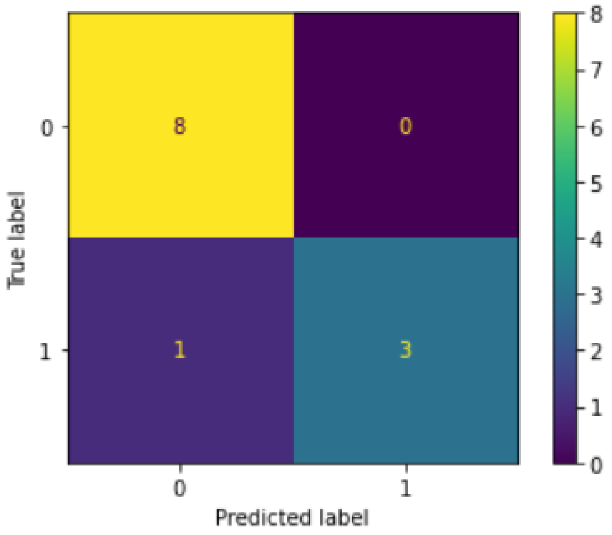


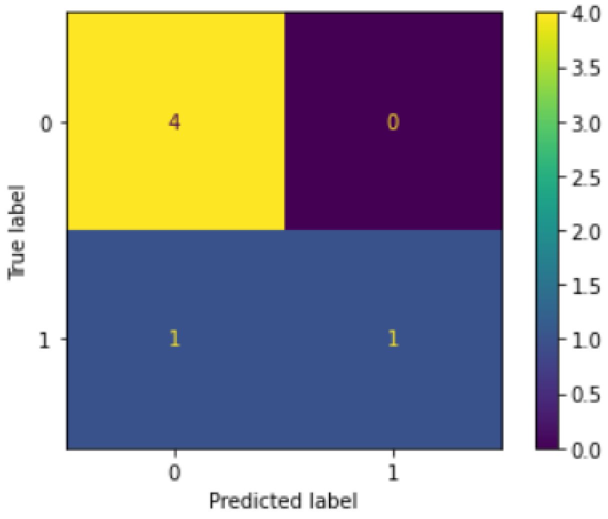


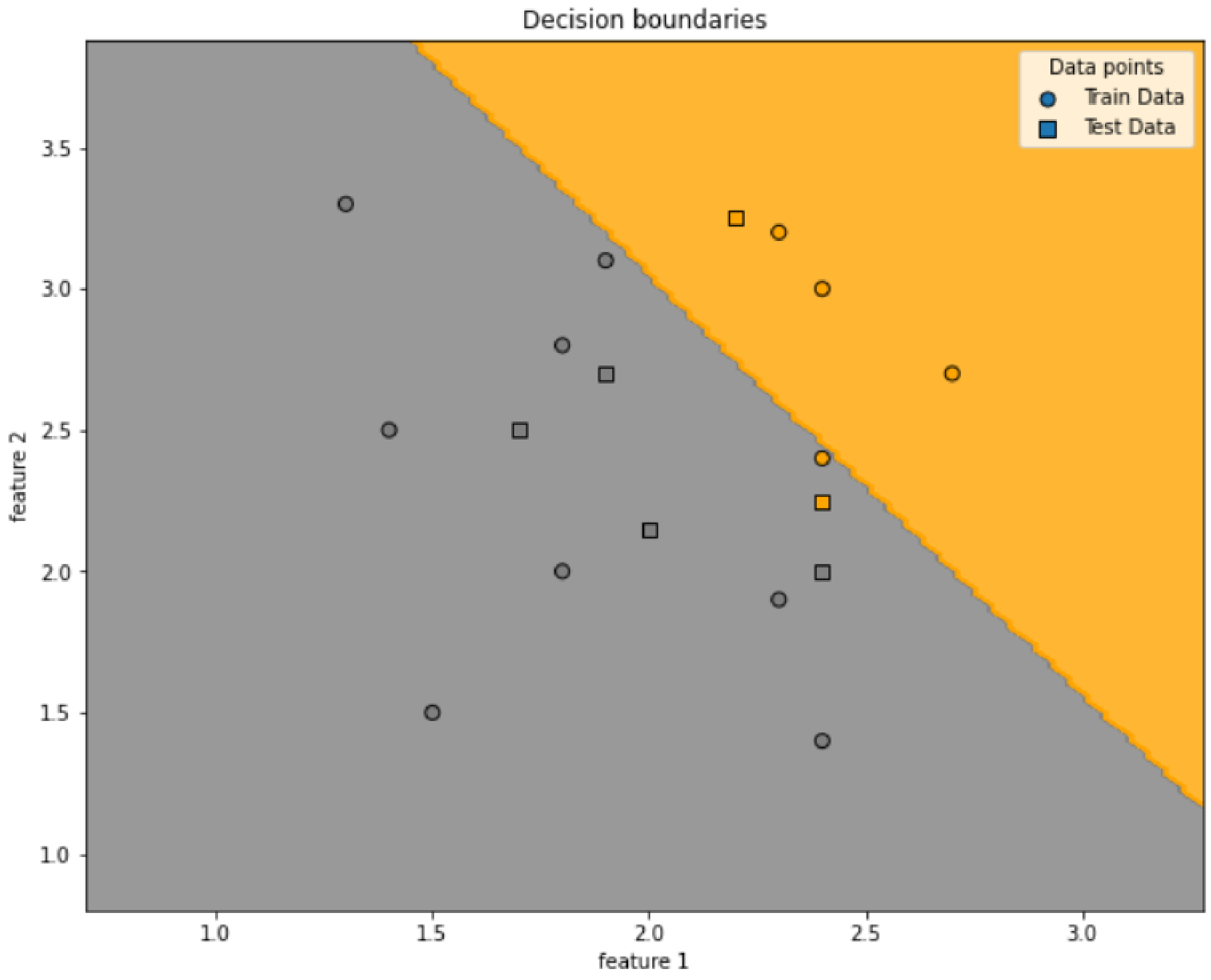


```
Classification Report of train for feature:  2
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         8
           1       0.33      1.00      0.50         4

    accuracy                           0.33        12
   macro avg       0.17      0.50      0.25        12
weighted avg       0.11      0.33      0.17        12

Confusion Matrix of train for feature:  2
Accuracy of Train =  91.66666666666666
```

```
Classification Report of test for feature:  2
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         4
           1       0.33      1.00      0.50         2

    accuracy                           0.33         6
   macro avg       0.17      0.50      0.25         6
weighted avg       0.11      0.33      0.17         6

Confusion Matrix of test for feature:  2
Accuracy of Test =  83.33333333333334
```







Compare and analyze SVM and Perceptron results:

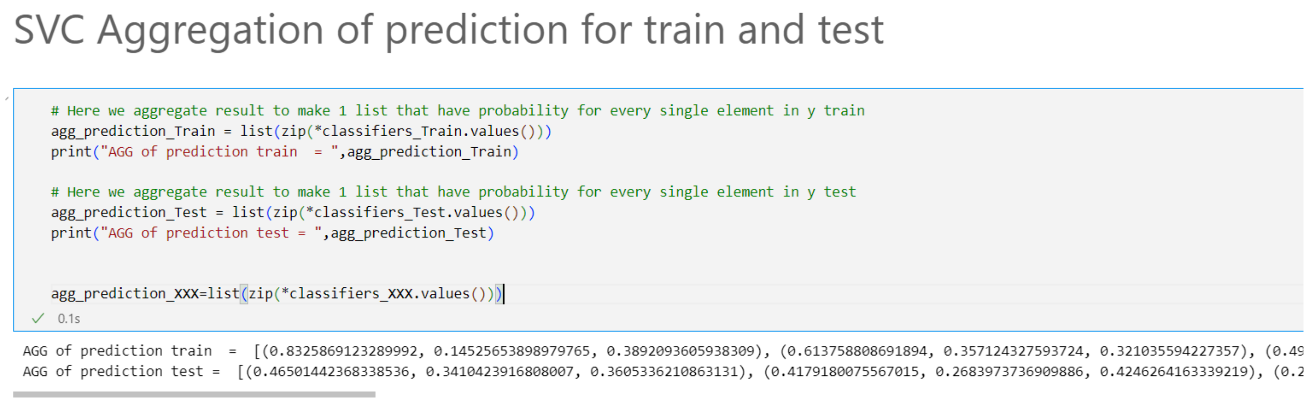
SVM	Perceptron
feature: 0 Accuracy of Train = 83.33333333333334	feature: 0 Accuracy of Train = 91.66666666666666
feature: 0 Accuracy of Test = 66.66666666666666	feature: 0 Accuracy of Test = 66.66666666666666
feature: 1 Accuracy of Train = 100.0	feature: 1 Accuracy of Train = 100.0
feature: 1 Accuracy of Test = 83.33333333333334	feature: 1 Accuracy of Test = 66.66666666666666
feature: 2 Accuracy of Train = 100.0	feature: 2 Accuracy of Train = 91.66666666666666
feature: 2 Accuracy of Test = 83.33333333333334	feature: 2 Accuracy of Test = 83.33333333333334

From this table we see results the performance of SVC is more accuracy than perceptron

----

(c) Aggregating result to make list for 3 predictions probability:

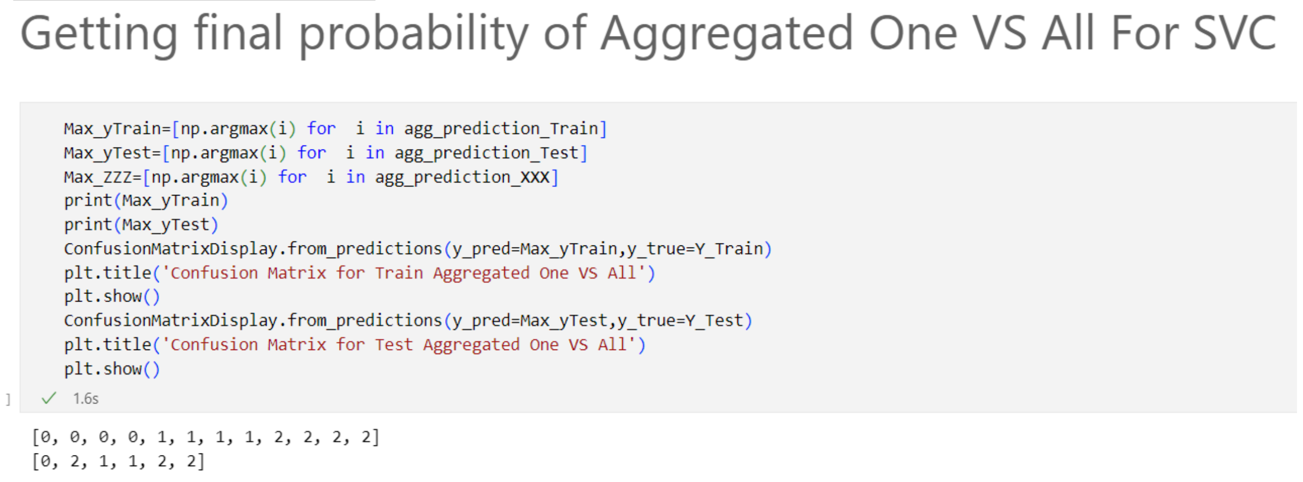
First we get SVC Aggregation of prediction for train and test:



SVC Aggregation of prediction for train and test

```
# Here we aggregate result to make 1 list that have probability for every single element in y train
agg_prediction_Train = list(zip(*classifiers_Train.values()))
print("AGG of prediction train  = ",agg_prediction_Train)

# Here we aggregate result to make 1 list that have probability for every single element in y test
agg_prediction_Test = list(zip(*classifiers_Test.values()))
print("AGG of prediction test = ",agg_prediction_Test)


agg_prediction_XXX=list(zip(*classifiers_XXX.values()))
```
✓ 0.1s

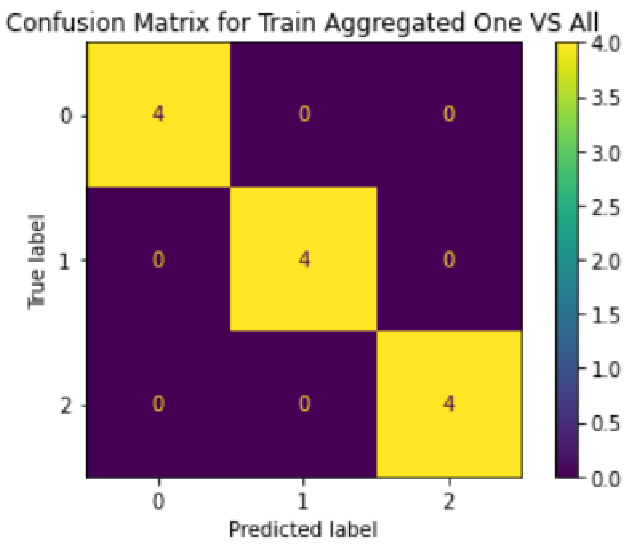
```
AGG of prediction train  =  [(0.8325869123289992, 0.14525653898979765, 0.3892093605938309), (0.613758808691894, 0.357124327593724, 0.321035594227357), (0.49
AGG of prediction test =  [(0.46501442368338536, 0.3410423916808007, 0.3605336210863131), (0.4179180075567015, 0.2683973736909886, 0.4246264163339219), (0.2
```

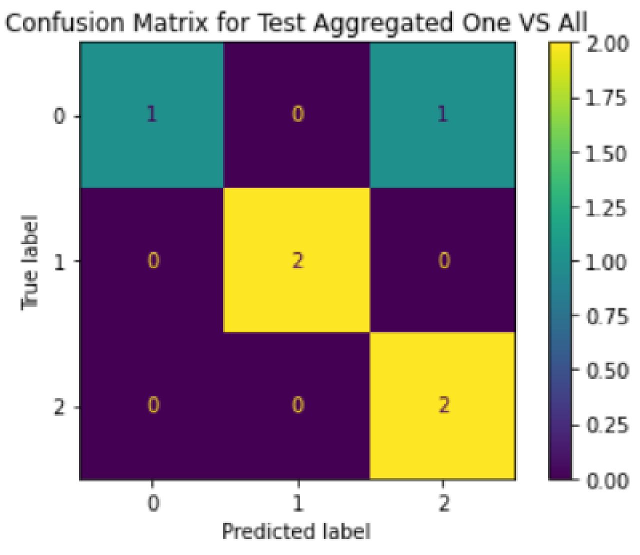


Getting final probability of Aggregated One VS All For SVC

```
Max_yTrain=[np.argmax(i) for  i in agg_prediction_Train]
Max_yTest=[np.argmax(i) for  i in agg_prediction_Test]
Max_ZZZ=[np.argmax(i) for  i in agg_prediction_XXX]
print(Max_yTrain)
print(Max_yTest)
ConfusionMatrixDisplay.from_predictions(y_pred=Max_yTrain,y_true=Y_Train)
plt.title('Confusion Matrix for Train Aggregated One VS All')
plt.show()
ConfusionMatrixDisplay.from_predictions(y_pred=Max_yTest,y_true=Y_Test)
plt.title('Confusion Matrix for Test Aggregated One VS All')
plt.show()
```
✓ 1.6s

```
[0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2]
[0, 2, 1, 1, 2, 2]
```



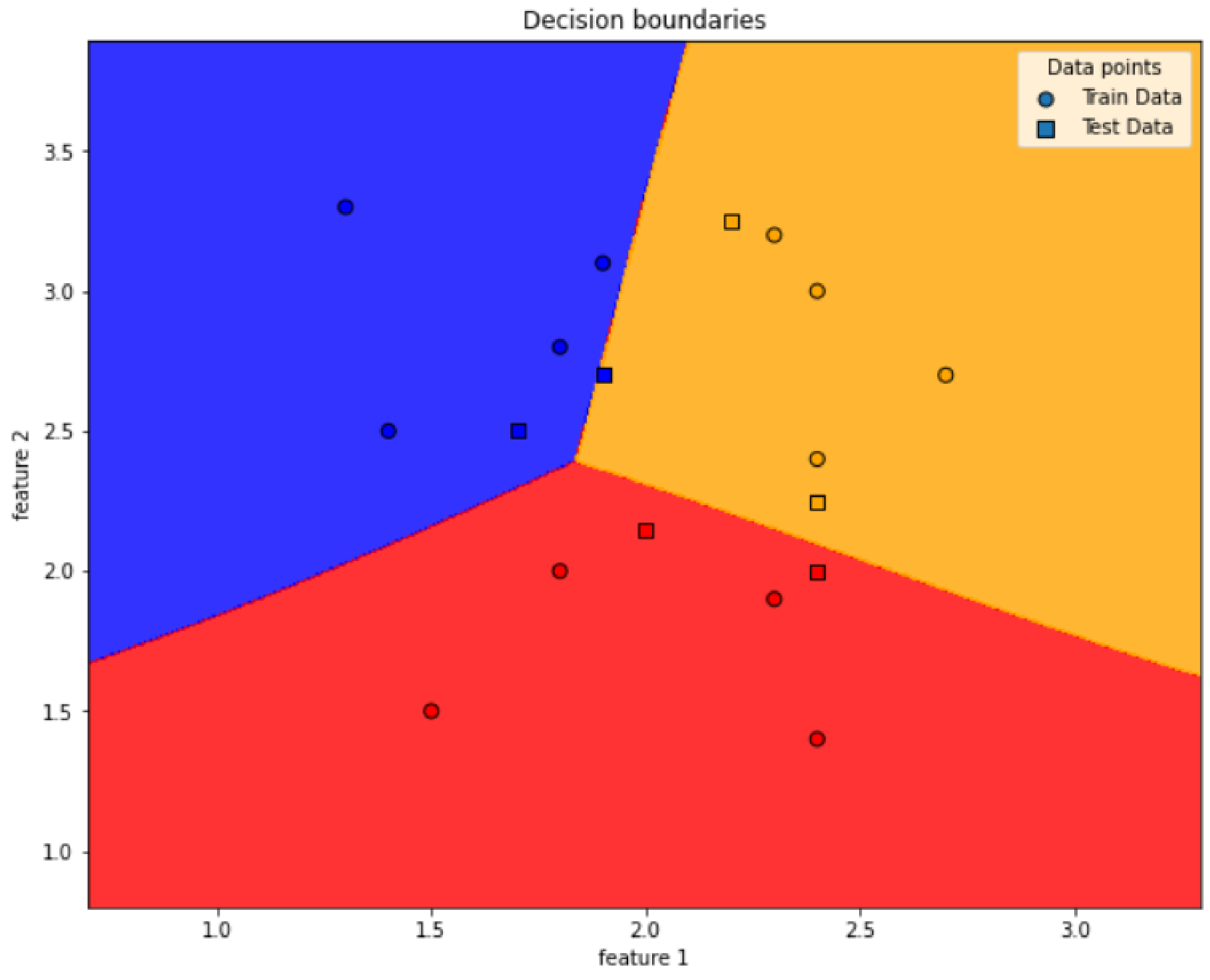


We want to aggregate result so we used zip method that aggregate all three results into one list each element contains 3 numbers

At last get our final decision using np,argmax

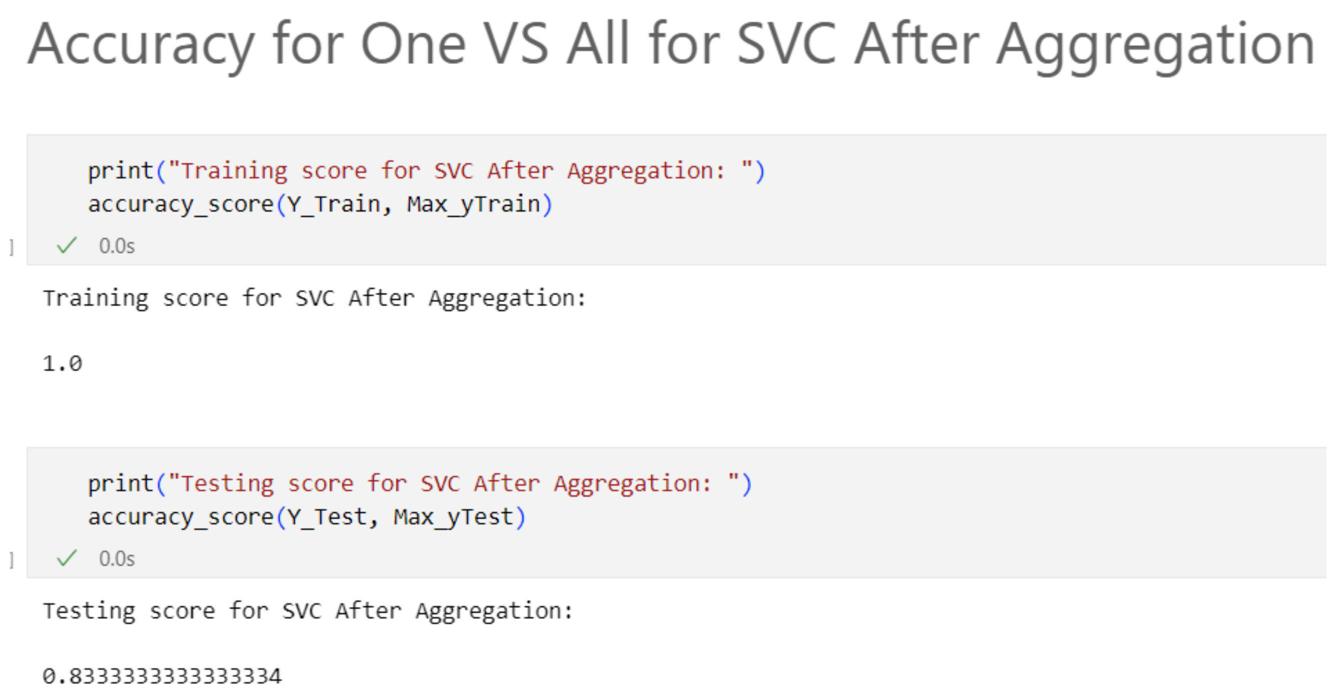
And we get confusion matrix for Train and Test aggregated

Now we plot the final aggregation Plot:



Now we get Accuracy for One VS All for SVC After Aggregation

And Training and Testing scores for SVC After Aggregation



Accuracy for One VS All for SVC After Aggregation

```python
print("Training score for SVC After Aggregation: ")
accuracy_score(Y_Train, Max_yTrain)
```

✓ 0.0s

```
Training score for SVC After Aggregation:

1.0
```

```python
print("Testing score for SVC After Aggregation: ")
accuracy_score(Y_Test, Max_yTest)
```

✓ 0.0s

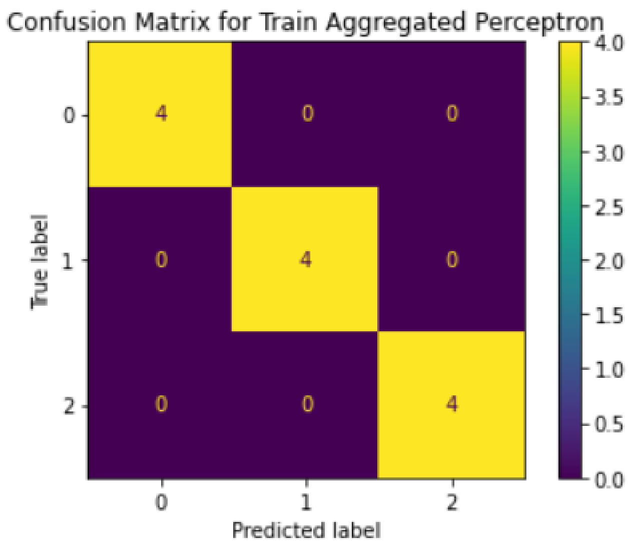
```
Testing score for SVC After Aggregation:

0.8333333333333334
```

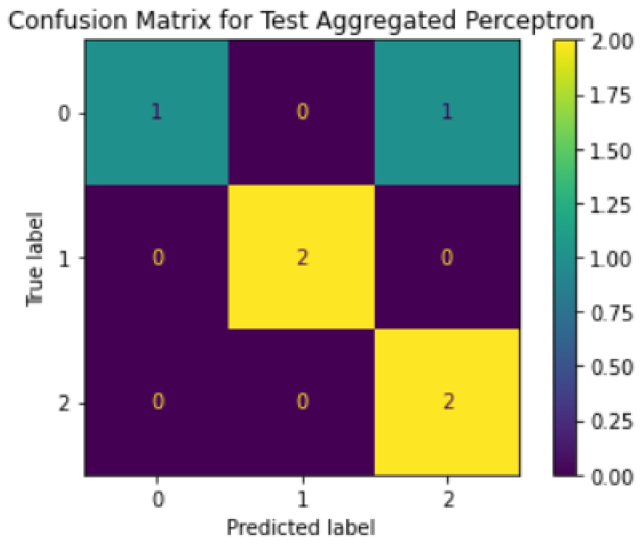
Here we can see

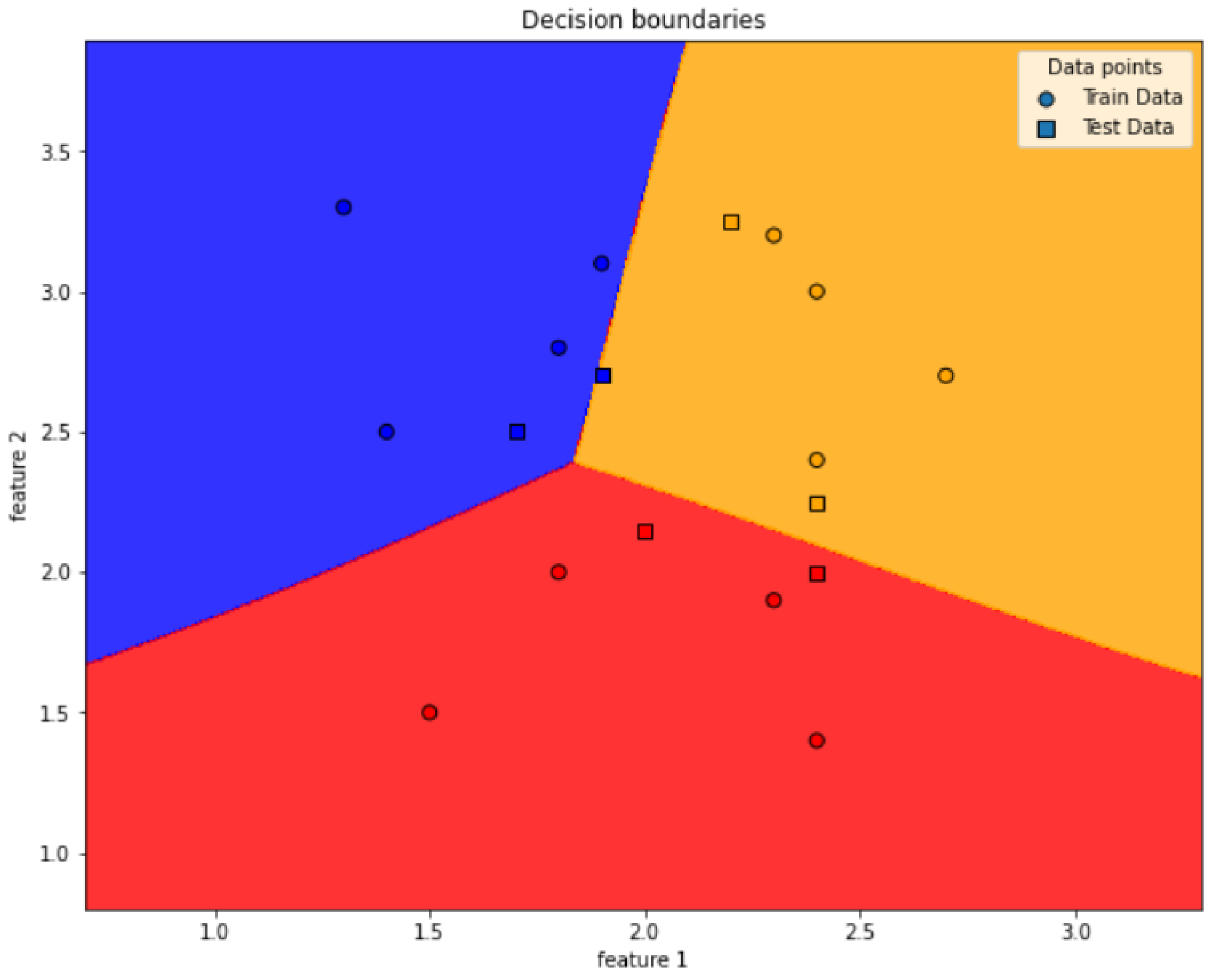
1. Training score for SVC After Aggregation = 100%
2. Testing score for SVC After Aggregation = 83.333334%

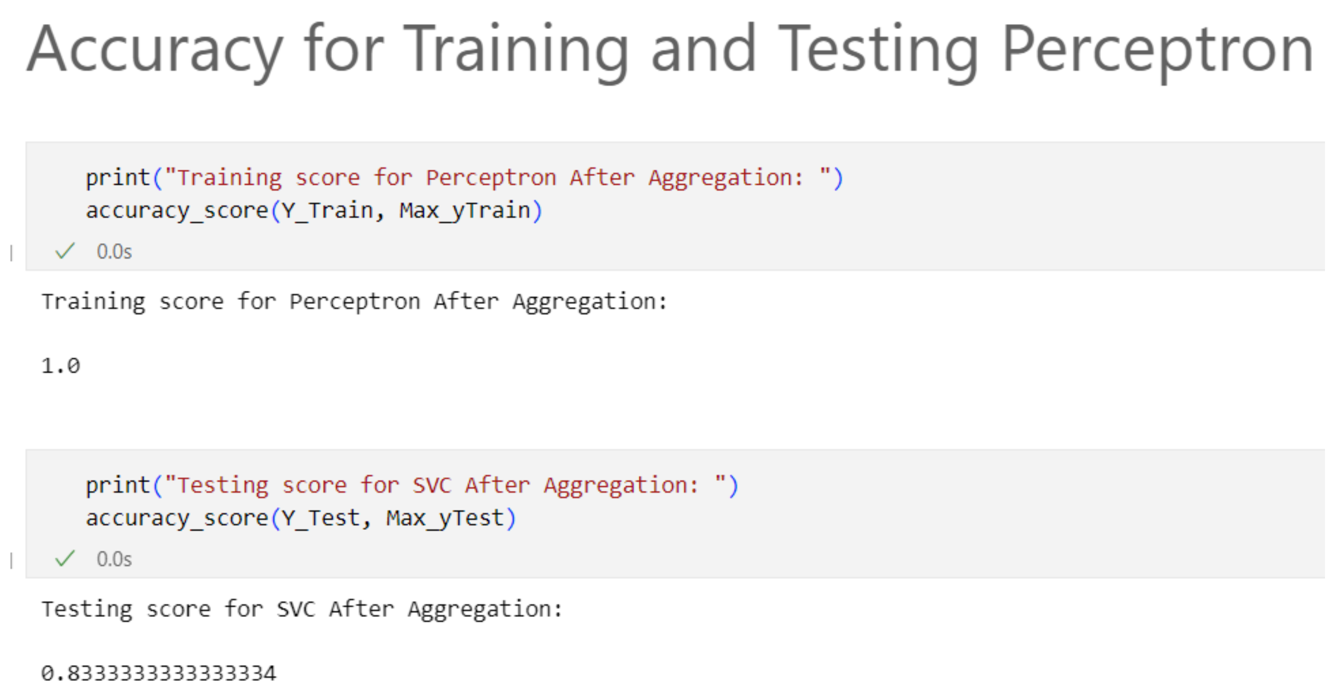
Aggregation of Perceptron:

Same as SVC Aggregation of prediction for train and test









Accuracy for Training and Testing Perceptron

```
print("Training score for Perceptron After Aggregation: ")
accuracy_score(Y_Train, Max_yTrain)
```

✓ 0.0s

```
Training score for Perceptron After Aggregation:

1.0
```

```
print("Testing score for SVC After Aggregation: ")
accuracy_score(Y_Test, Max_yTest)
```

✓ 0.0s

```
Testing score for SVC After Aggregation:

0.8333333333333334
```

Here we can see

1. Training score for Perceptron After Aggregation = 100%

2. Testing score for Perceptron After Aggregation = 83.333334%

(d)

Reason of performance for default and aggregated SVC

- The Accuracy of Train in SVC with linear kernel = 91.66666666666666%

- The Accuracy of Train in SVC with linear kernel = 100%

- The Accuracy of for SVC After Aggregation for training = 100%

- The Accuracy of for SVC After Aggregation for testing = 8333333333333334%

- The reason for this performance because the parameter of regularization C=1 in default SVC for all classifiers

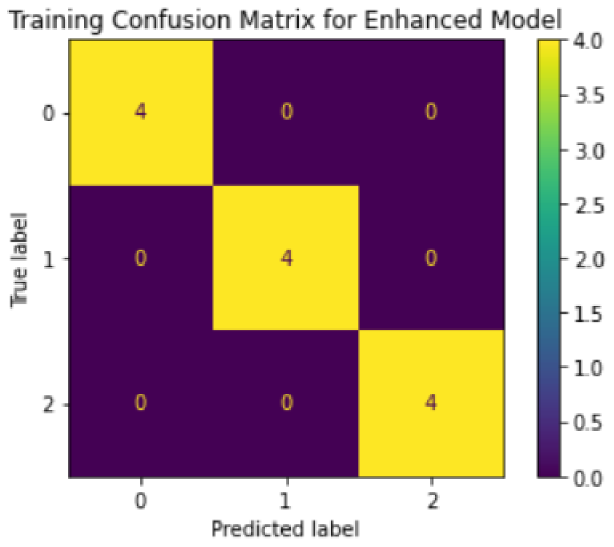
The regularization is a technique used to prevent overfitting of a model to the training data. Overfitting occurs when a model becomes too complex and starts to fit the noise in the data

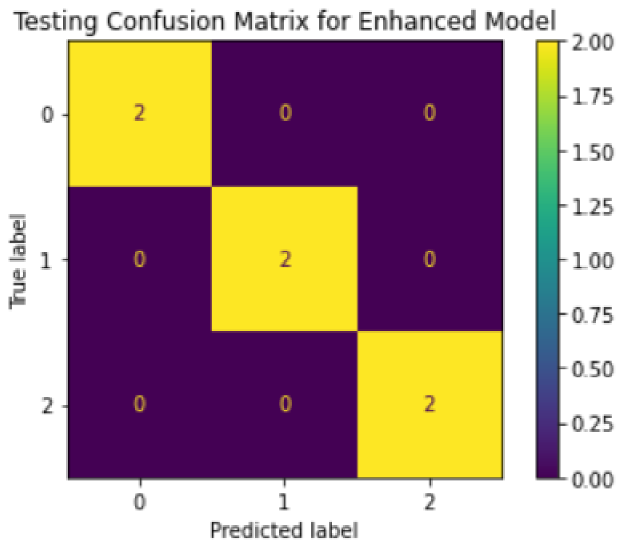
But in aggregated SVC every single class has it's own regularization parameter so when we aggregate it the result will have high accuracy

So we Decided to Refine Default SVC with following attributes

- we train SVC with appropiate parameter
- we use different higher regularization parameter where C = 100
- after that we are evaluating it's performance by getting accuracy of train and test
- Obtaining confusion metrices and decision surfaces for both training and testing

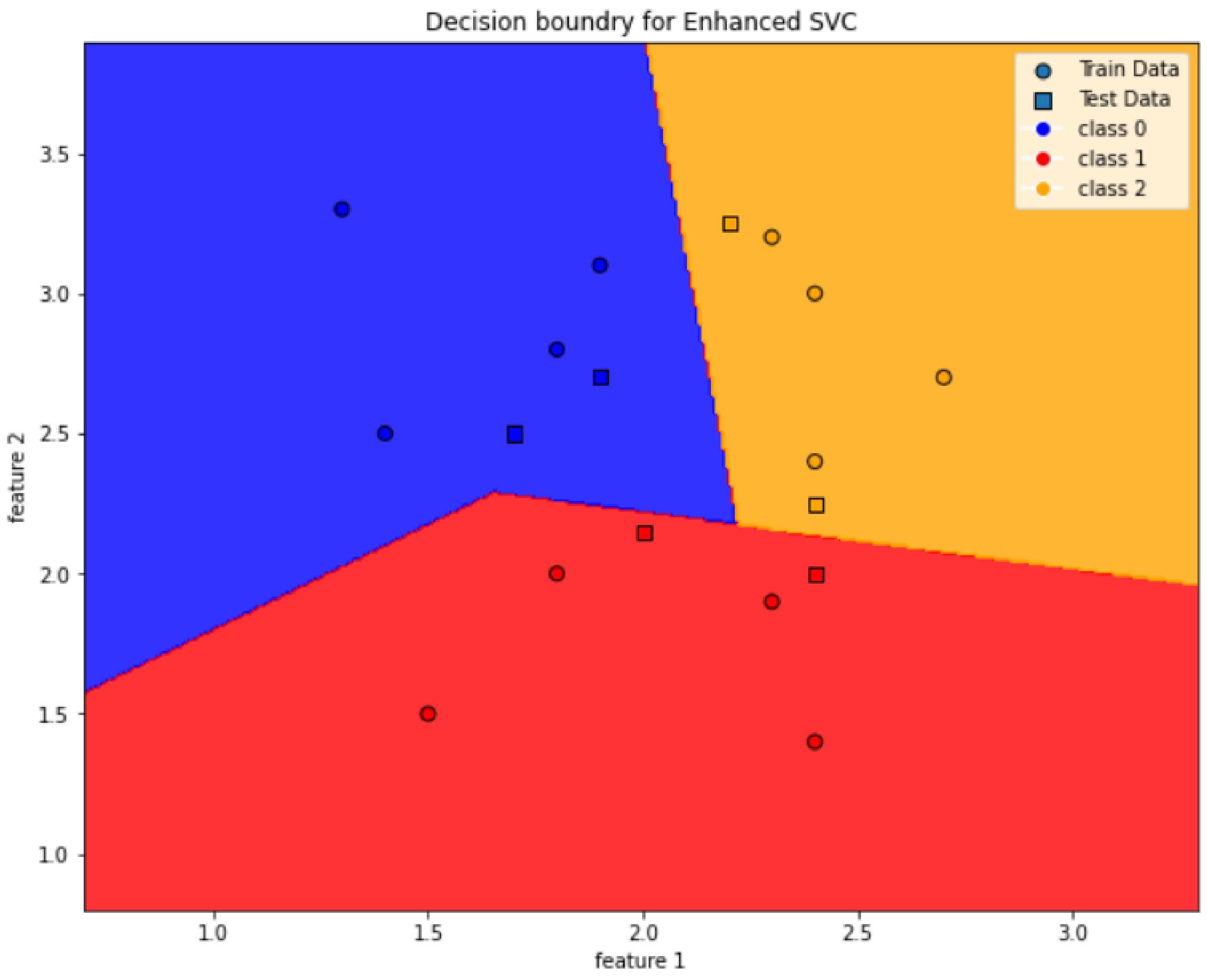
After building the model the results were as follows:





```
Classification Report of test for enhanced model
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         4
           2       1.00      1.00      1.00         4

    accuracy                           1.00        12
   macro avg       1.00      1.00      1.00        12
weighted avg       1.00      1.00      1.00        12
```

```
Classification Report of test for enhanced model
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00         2

    accuracy                           1.00         6
   macro avg       1.00      1.00      1.00         6
weighted avg       1.00      1.00      1.00         6
```



```
Accuracy of Train for SVC linear kernel using C=100 :  100.0
Accuracy of Test for SVC linear kernel using C=100 :  100.0
```

Compare results with the default SVM:

we see that when we change C to C=100 the SVC linear kernel is higher than SVC linear kernel with default C=1

The impact of parameter selection:

We can improve the accuracy by changing the regularization parameter C to the most fit number

……….

Task 2

2. Use sklearn to import train_test_split, LabelEncoder, KNeighborsClassifier, accuracy_score

And also use other python packages like numpy, matplotlib.pyplot, pandas

Scikit-learn offers a variety of tools for many different tasks, including model selection, evaluation, and data preprocessing.

train_test_split:- It splits the dataset into training and testing sets. It helps in assessing the performance of a machine learning model on unseen data.

LabelEncoder:-. It enables machine learning models to use categorical data by giving each class a distinct number.

KNeighborsClassifier: This straightforward yet efficient technique gives new instances names based on the class labels of their k-nearest neighbours in the training set, it is a straightforward and efficient technique that gives labels to new cases.

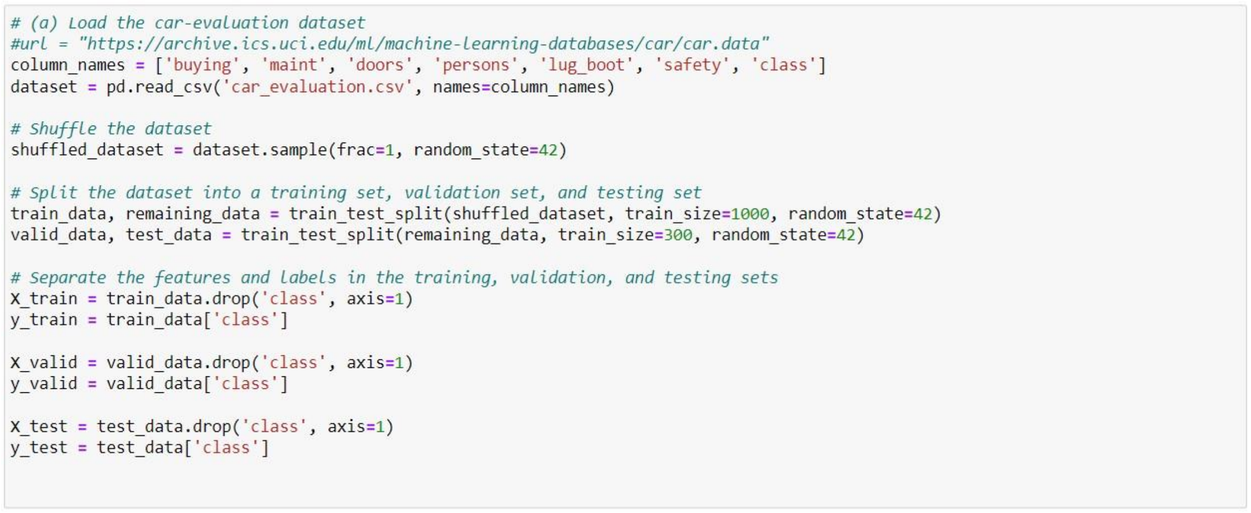
accuracy_score:- it determines how accurate a classification model is. It aids in assessing a categorization model's effectiveness.

NumPy:- it offers strong array and matrix operations. It is often used in machine learning for numerical calculations and data manipulation.

matplotlib.pyplot :- for making many kinds of plots, charts, and visualisations is provided by the pyplot module. It is frequently used to analyse patterns or trends and visualise data.

Pandas:- DataFrames, that make working with structured data simple. Pandas is frequently used for involving data preprocessing, cleaning, and exploration.

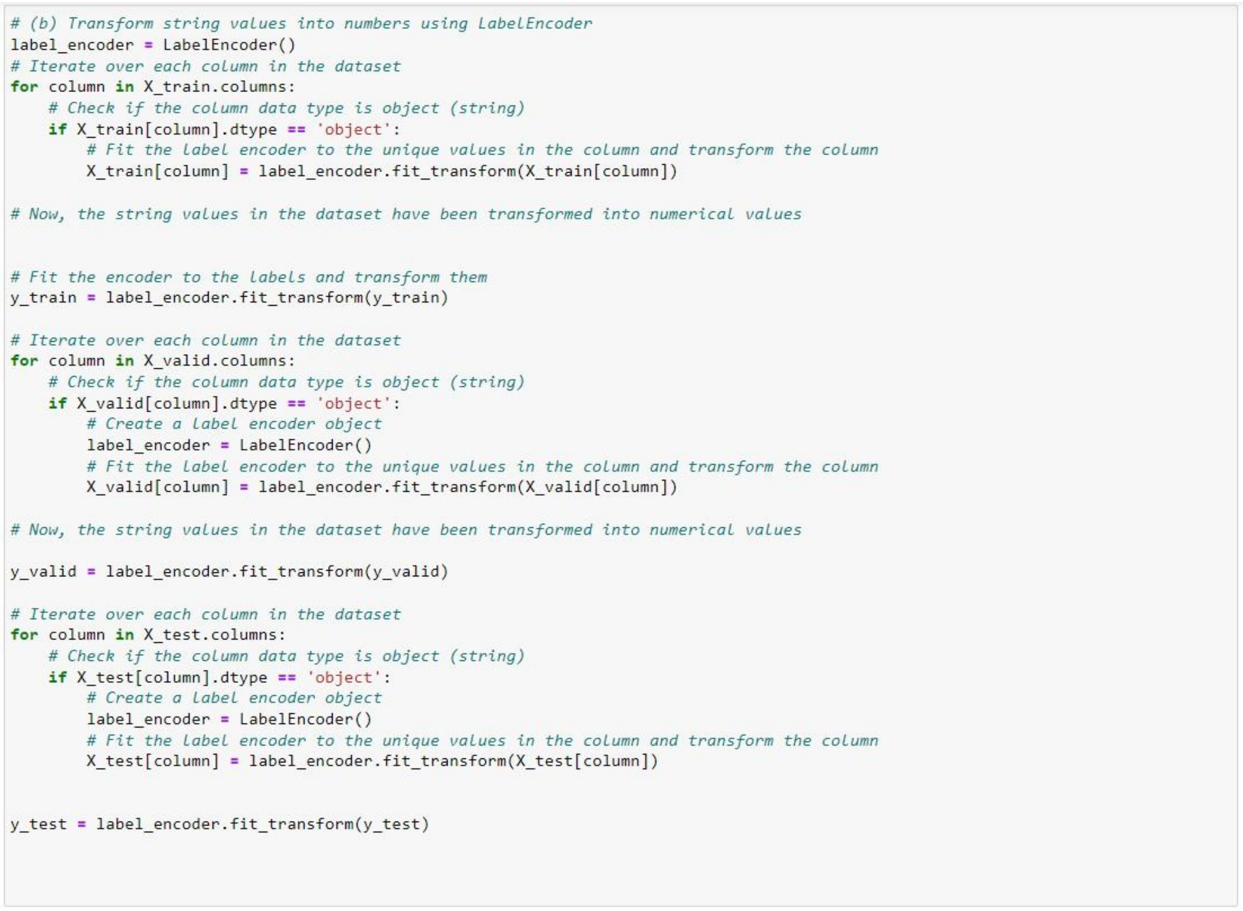
(a)



```
# (a) Load the car-evaluation dataset
#url = "https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data"
column_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
dataset = pd.read_csv('car_evaluation.csv', names=column_names)

# Shuffle the dataset
shuffled_dataset = dataset.sample(frac=1, random_state=42)

# Split the dataset into a training set, validation set, and testing set
train_data, remaining_data = train_test_split(shuffled_dataset, train_size=1000, random_state=42)
valid_data, test_data = train_test_split(remaining_data, train_size=300, random_state=42)

# Separate the features and labels in the training, validation, and testing sets
X_train = train_data.drop('class', axis=1)
y_train = train_data['class']

X_valid = valid_data.drop('class', axis=1)
y_valid = valid_data['class']

X_test = test_data.drop('class', axis=1)
y_test = test_data['class']
```

1. First we Load the car-evaluation dataset ` url = "https://archive.ics.uci.edu/ml/machinelearning-databases/car/car.data"`.
2. And we define the columns by` column_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']`.
3. Then we read the dataset by `dataset = pd.read_csv(url, names=column_names)`.
4. We Shuffle the dataset ` shuffled_dataset = dataset.sample(frac=1, random_state=42)`
   - sample(): Rows from the dataset are sampled at random using this technique.
   - frac=1: The number of rows to return is specified by the frac argument. When frac=1 in this context, we signify that we wish to sample all rows, or the complete dataset.
   - random_state=42: The reproducible random seed is set by the random_state argument. Every time the code is executed with the same seed, it makes sure the shuffle takes place in the same random order.
5. Split the dataset into a training set, validation set, and testing set
   - shuffled dataset splits are used. In the first split, 1000 samples are divided into a training set (train_data) and the remaining samples are divided into a remaining set (remaining_data). The remaining set is divided again in the second split, yielding a testing set (test_data) with the remaining samples and a validation set (valid_data) with 300 samples.

(b)

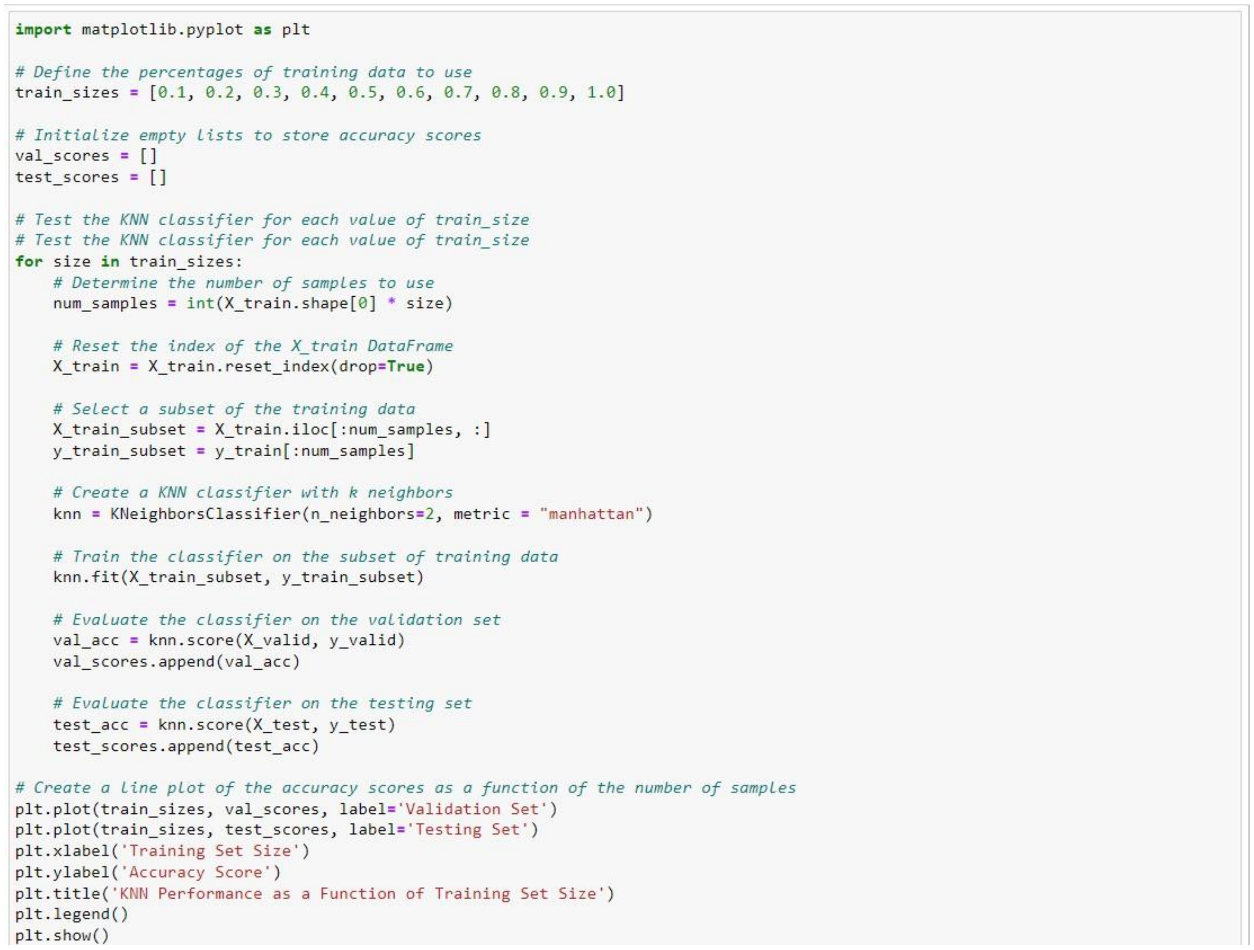


```
# (b) Transform string values into numbers using LabelEncoder
label_encoder = LabelEncoder()
# Iterate over each column in the dataset
for column in X_train.columns:
    # Check if the column data type is object (string)
    if X_train[column].dtype == 'object':
        # Fit the label encoder to the unique values in the column and transform the column
        X_train[column] = label_encoder.fit_transform(X_train[column])

# Now, the string values in the dataset have been transformed into numerical values


# Fit the encoder to the labels and transform them
y_train = label_encoder.fit_transform(y_train)

# Iterate over each column in the dataset
for column in X_valid.columns:
    # Check if the column data type is object (string)
    if X_valid[column].dtype == 'object':
        # Create a label encoder object
        label_encoder = LabelEncoder()
        # Fit the label encoder to the unique values in the column and transform the column
        X_valid[column] = label_encoder.fit_transform(X_valid[column])

# Now, the string values in the dataset have been transformed into numerical values

y_valid = label_encoder.fit_transform(y_valid)

# Iterate over each column in the dataset
for column in X_test.columns:
    # Check if the column data type is object (string)
    if X_test[column].dtype == 'object':
        # Create a label encoder object
        label_encoder = LabelEncoder()
        # Fit the label encoder to the unique values in the column and transform the column
        X_test[column] = label_encoder.fit_transform(X_test[column])


y_test = label_encoder.fit_transform(y_test)
```

Transform string values into numbers using LabelEncoder

After creating a LabelEncoder object, the X_train DataFrame's columns are iterated over to see if their data types are of the object (string) variety. If an object data type is detected for a column, the LabelEncoder is fitted to the column's unique values before the column is transformed with the fit_transform method.

(c)

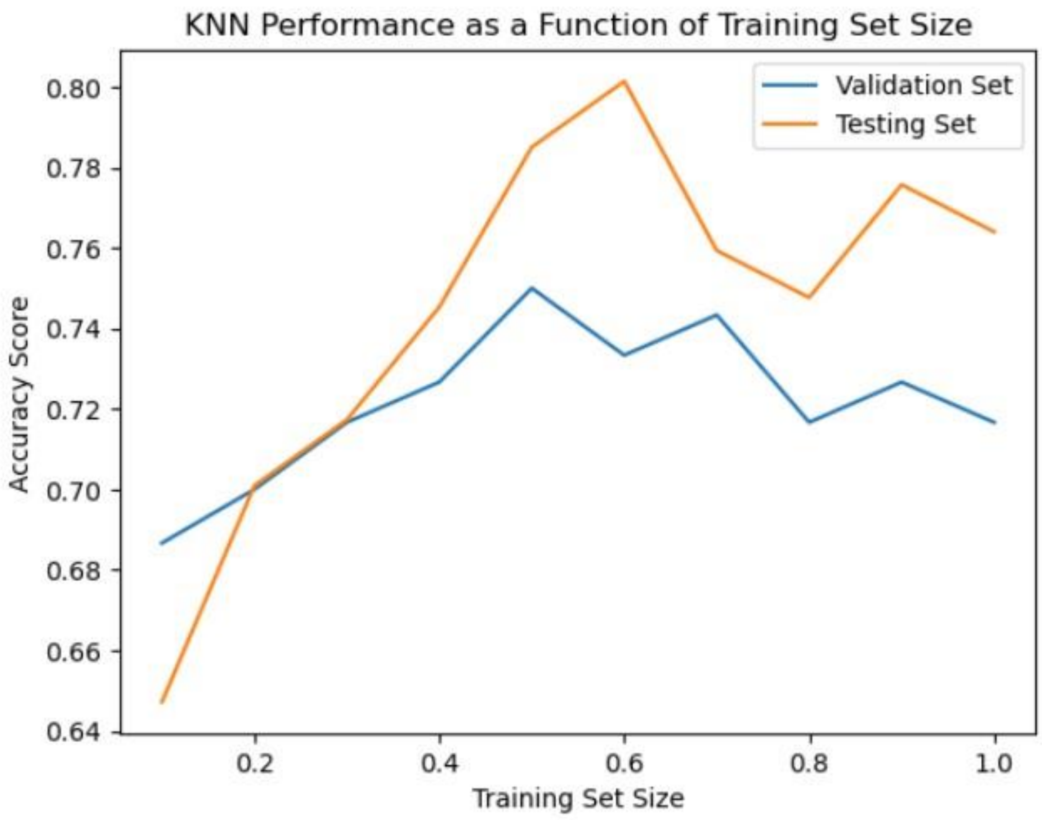


```
import matplotlib.pyplot as plt

# Define the percentages of training data to use
train_sizes = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

# Initialize empty lists to store accuracy scores
val_scores = []
test_scores = []

# Test the KNN classifier for each value of train_size
# Test the KNN classifier for each value of train_size
for size in train_sizes:
    # Determine the number of samples to use
    num_samples = int(X_train.shape[0] * size)

    # Reset the index of the X_train DataFrame
    X_train = X_train.reset_index(drop=True)

    # Select a subset of the training data
    X_train_subset = X_train.iloc[:num_samples, :]
    y_train_subset = y_train[:num_samples]

    # Create a KNN classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=2, metric = "manhattan")

    # Train the classifier on the subset of training data
    knn.fit(X_train_subset, y_train_subset)

    # Evaluate the classifier on the validation set
    val_acc = knn.score(X_valid, y_valid)
    val_scores.append(val_acc)

    # Evaluate the classifier on the testing set
    test_acc = knn.score(X_test, y_test)
    test_scores.append(test_acc)

# Create a line plot of the accuracy scores as a function of the number of samples
plt.plot(train_sizes, val_scores, label='Validation Set')
plt.plot(train_sizes, test_scores, label='Testing Set')
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy Score')
plt.title('KNN Performance as a Function of Training Set Size')
plt.legend()
plt.show()
```

train_sizes = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]: This line defines a list train_sizes containing different fractions representing the desired training set sizes. The fractions range from 10% to 100% (1.0).

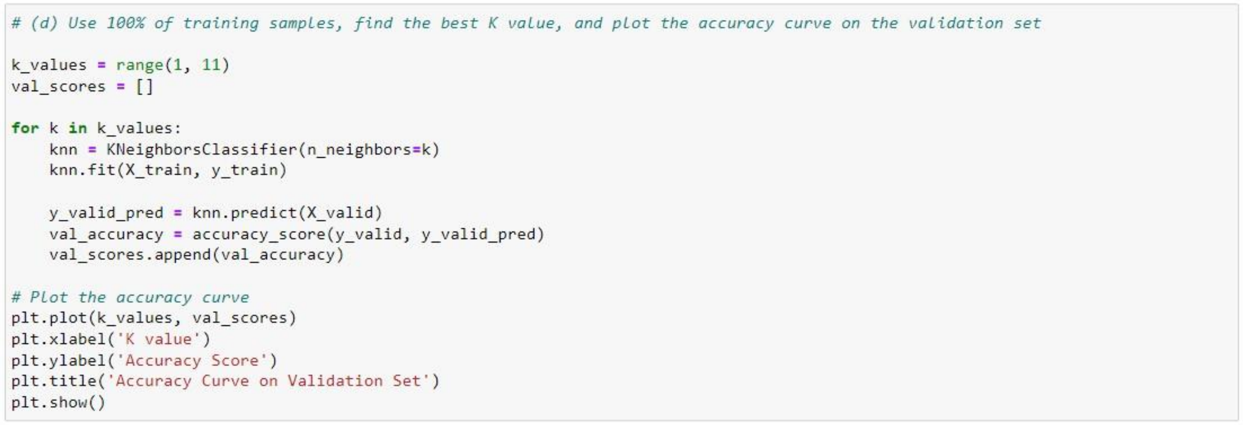
val_scores = [] and test_scores = []: These empty lists will be used to store the accuracy scores for the validation set and testing set, respectively, for each training set size.

it generates a line plot of accuracy scores as a function of the number of samples using plt.plot. He labels the x-axis as “training set size,” the y-axis as “accuracy score,” and the title as “KNN performance as a function of training set size.” It also adds a legend using plt. A legend that displays the accuracy scores for the validation set and the test set as two separate lines on the plot.



The higher the training, the greater the accuracy.

(d) Use 100% of training samples, try to find the best K value, and

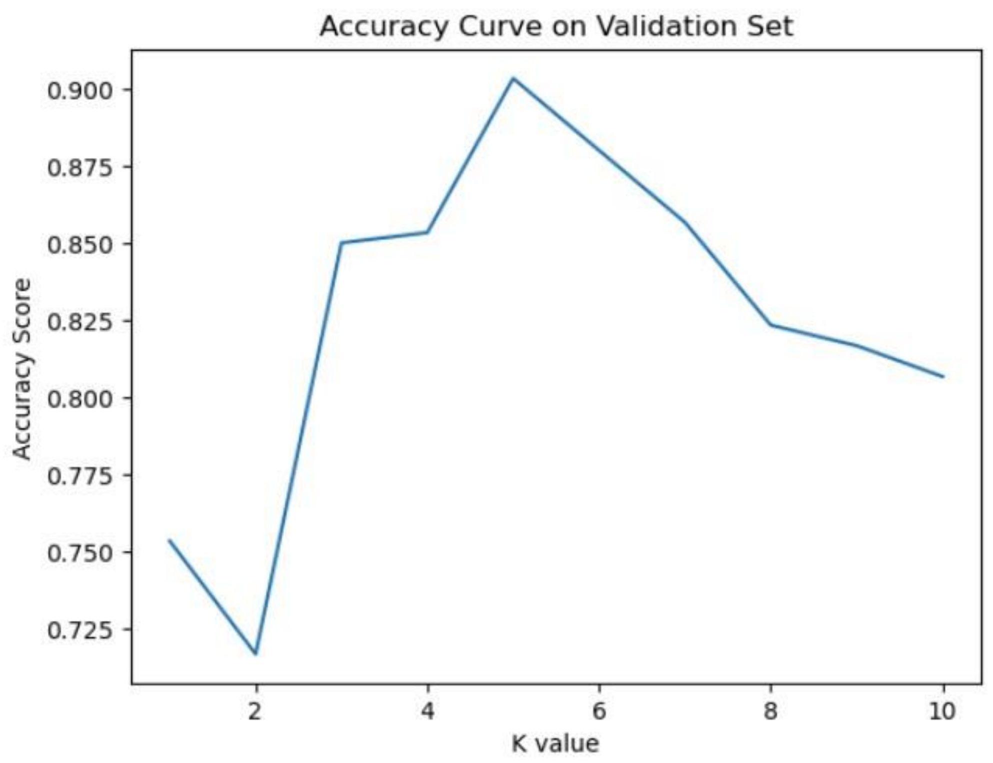


```
# (d) Use 100% of training samples, find the best K value, and plot the accuracy curve on the validation set

k_values = range(1, 11)
val_scores = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    y_valid_pred = knn.predict(X_valid)
    val_accuracy = accuracy_score(y_valid, y_valid_pred)
    val_scores.append(val_accuracy)

# Plot the accuracy curve
plt.plot(k_values, val_scores)
plt.xlabel('K value')
plt.ylabel('Accuracy Score')
plt.title('Accuracy Curve on Validation Set')
plt.show()
```

For each value of k from 1 to 10, inclusive, a K-Nearest Neighbors (KNN) classifier is trained on the training set, and its performance is assessed on the validation set. The accuracy scores as a function of k are then shown on a line.

The values of k to test, which range from 1 to 10, are contained in the k_values range object. The accuracy scores for each run of the KNN classifier are first stored in the val_scores list, which is initially empty.

Then, the function iterates through each value in k_values. It builds a KNN classifier with k neighbors for each value and trains the classifier using the full training set. The labels of the validation set are then predicted using the trained classifier, and the accuracy score is determined using the scikit-learn accuracy_score function. The val_scores list is then updated with the accuracy score.



we try K from 1 to 10.then we train model and then make predictions on the validation set. We calculate the accuracy score using the predicted labels and the true labels of the validation set.then we plot the accuracy curve using plt.plot()