ELG 5255: Applied Machine Learning

Assignment:1

BY: Group 5

Anas Elbattra

Ahmed Badawy

Esraa Fayad

Part 1→ Support Vector Machines

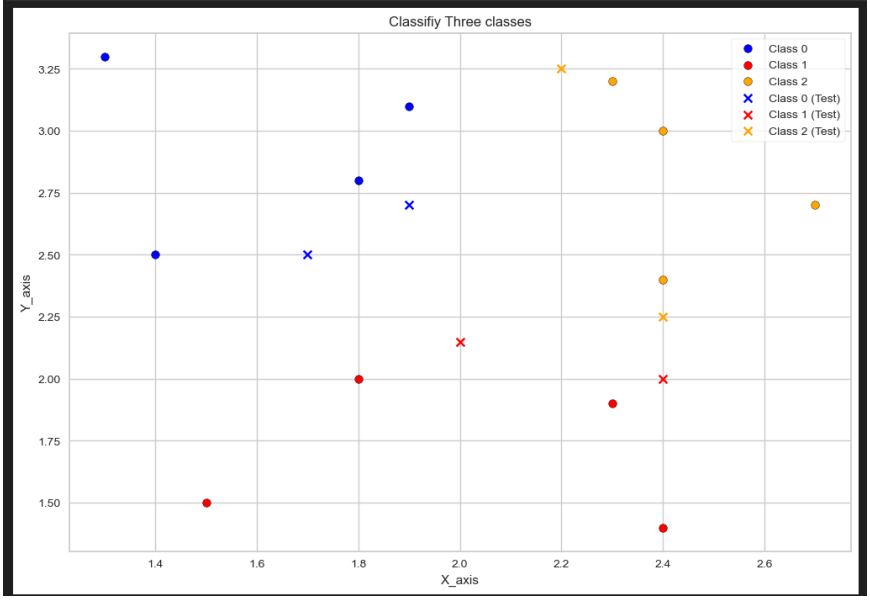
We used a scatter plot to show the distribution of data **before processing for the three classes.**

```
def define_marker (x_tr,y_train , x_test,y_test):

    unique_values = np.unique(y_train)

    for i, color in zip(range(0,len(unique_values)), colors):
        idx = np.where(y_train == i)
        plt.scatter(x_tr[idx, 0], x_tr[idx, 1], c=color, label='Class ' + str(i), edgecolors='black', marker='o')

    for i, color in zip(range(0,len(unique_values)), colors):
        idx = np.where(y_test == i)
        plt.scatter(x_test[idx, 0], x_test[idx, 1], c=color, label='Class ' + str(i) + ' (Test)', edgecolors='black', marker='x')
```

```
def classify_Three_classes():

    plt.rcParams['figure.figsize'] = (12, 8)
    define_marker(X_Train,y_Train,X_Test,y_Test)
    plt.xlabel('X_axis')
    plt.ylabel('Y_axis')
    plt.title('Classifiy Three classes')
    plt.legend(frameon=True)

    plt.show()
```
[5]

```
classify_Three_classes()
```

```
C:\Users\DEBI\AppData\Local\Temp\ipykernel_5800\1601434719.py:11: UserWarning: You passed a edgecolor/edgecolors ('black') for an un
  plt.scatter(x_test[idx, 0], x_test[idx, 1], c=color, label='Class ' + str(i) + ' (Test)', edgecolors='black', marker='x')
```



a.

- Applying the default SVM classifier

SVC () → For classification problems.

```
Default_classifier = SVC()
Default_classifier.fit(X_Train, y_Train)

y_Pred_test = Default_classifier.predict(X_Test)
y_pred_train = Default_classifier.predict(X_Train)
```

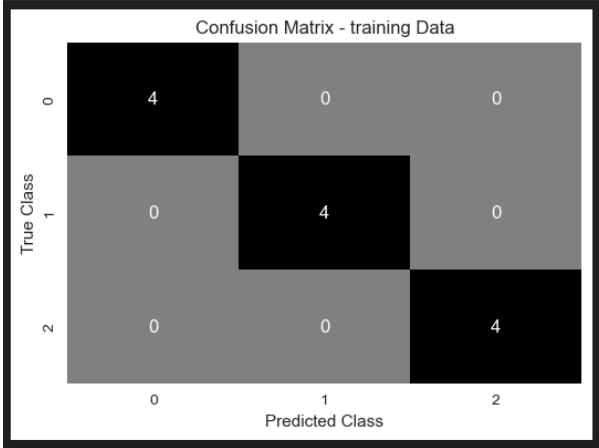
- Confusion matrices for training and test datasets

```
def conf_matrix (x,y,title,show_report=False):
    cm = confusion_matrix(x, y)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap=[ 'grey','black'], cbar=False)
    plt.title(f'Confusion Matrix - {title} Data')
    plt.xlabel('Predicted Class')
    plt.ylabel('True Class')
    plt.show()
    if not show_report:
        print("Accuracy: ", accuracy_score(x, y))

    if show_report:
        report = classification_report(x, y)
        print("Classification Report:")
        print(report)
        plt.show()
```

- The conf_matrix () function calculates and displays a confusion matrix for predicted and true class labels, and prints the accuracy and classification report based on the show_report parameter.

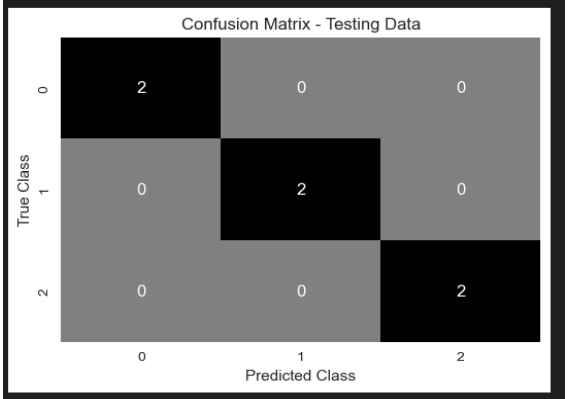
- **The Output of Confusion matrix for training and test sets.**

```
conf_matrix(y_Train, y_pred_train,"training")
```




```
Accuracy:  1.0
```

```
y_pred = Default_classifier.predict(X_Test)
conf_matrix(y_Test, y_pred,"Testing")
```




```
Accuracy:  1.0
```

- Decision surfaces / boundaries for multi-class classification by using blue, red, and orange colors to plot each class.
- Training data points should be marked different than test data points for visualization of decision surfaces.

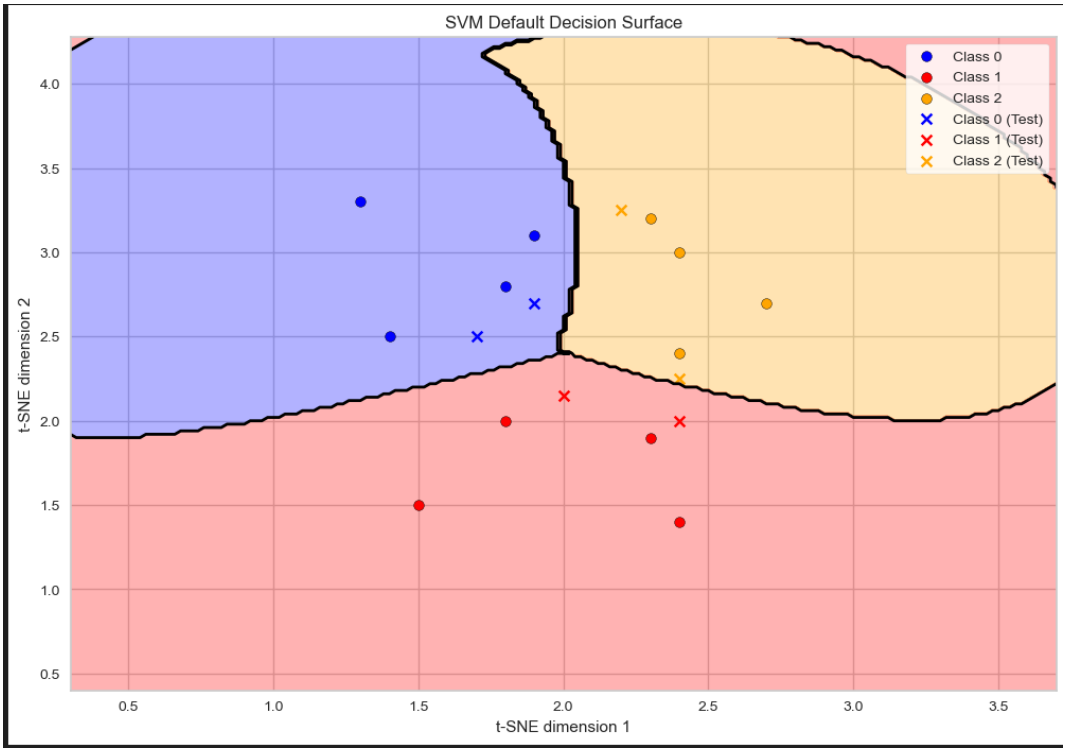
```
colors = ['blue', 'red', 'orange']
```

```
def plot_decision_boundary_All_data(models, X_train, y_train,X_test,y_test,title,lb=None ):


    x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
    y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
    h = 0.02

    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    if len(models) > 1:
        stacked_predictions = np.vstack([model.predict(np.c_[xx.ravel(), yy.ravel()]) for model in models])
        Z = np.argmax(stacked_predictions, axis=0)
        Z = Z.reshape(xx.shape)

    else:

        Z =  models[0].predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)


    plt.contourf(xx, yy, Z,levels=[-.5,.1,1.5,3], colors=colors, alpha=0.3)
    plt.contour(xx, yy, Z, levels=1, colors='black', linewidths=2)
    plt.xlabel('t-SNE dimension 1')
    plt.ylabel('t-SNE dimension 2')

    define_marker(X_train,y_train,X_test,y_test)


    plt.title(title)
    plt.legend(frameon=True)
    plt.show()
```

- (plot_decision_boundary_All_data) Function -→ Takes in multiple models, training and testing data, a title, and an optional label. The function plots the decision boundary for the given models using the training data and overlays it on a scatter plot of the training and testing data.
- The xx and yy variables are created using np.meshgrid to create a grid of points that cover the range of the training data.
- The if statement checks if there is more than one model and if so, it stacks the predictions of each model and takes the argmax to determine the predicted class for each point on the grid. If there is only one model, it simply makes predictions for each point on the grid.
- The plt.contourf function is used to fill in the regions between the decision boundaries with different colors based on the predicted class.
- The plt.contour function is used to draw the decision boundaries themselves. The plt.xlabel and plt.ylabel functions set the labels for the x and y axes, respectively.

The output of decision boundary for all data –- Default SVM

```
plot_decision_boundary_All_data([Default_classifier],X_Train,y_Train,X_Test,y_Test, 'SVM Default Decision Surface')
```



b. One-vs-Rest→ SVM

- convert three classes to two classes

```
def binary_classification (Y_train):

        y_binary0 =np.where((Y_train == 1) | (Y_train == 2), 0, 1)
        y_binary1 =np.where((Y_train == 0) | (Y_train == 2), 0, 1)
        y_binary2 =np.where((Y_train == 0) | (Y_train == 1), 0, 1)


        return y_binary0,y_binary1,y_binary2
```

The function (binary_classification) creates three binary classification targets based on the original target variable values. The first binary target (y_binary0) is created by setting the values to 0 if the original target variable values are 1 or 2, and 1 otherwise. The second binary target (y_binary1) is created by setting the values to 0 if the original target variable values are 0 or 2, and 1 otherwise. The third binary target (y_binary2) is created by setting the values to 0 if the original target variable values are 0 or 1, and 1 otherwise.

- To classify each class with the other two classes

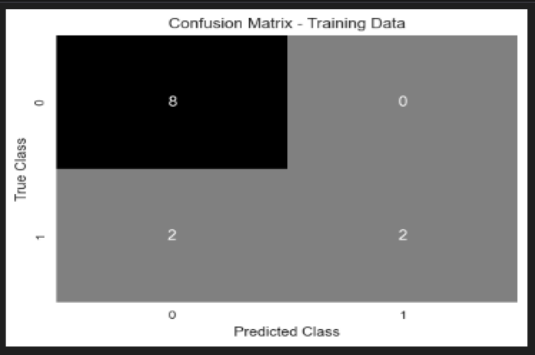
```
def One_Vs_rest(X_Train,Y_tain,X_test,Y_test,class_num,model):
    yTrain_binary=binary_classification(Y_tain)[class_num]
    yTest_binary=binary_classification(Y_test)[class_num]



    model=model.fit(X_Train, yTrain_binary)
    y_Train_pred = model.predict(X_Train)
    y_pred = model.predict(X_Test)

    conf_matrix(yTrain_binary, y_Train_pred,"Training")

    conf_matrix(yTest_binary, y_pred,"Testing")



    return y_Train_pred,y_pred,model
```

One_Vs_rest function takes five parameters: X_Train, Y_tain, X_test, Y_test, class_num, and model. The function first calls the binary_classification function to create a binary target variable for the specified class_num. It then fits the model to the training data (X_Train and yTrain_binary) and makes predictions on both the training and testing data. The function also calls a conf_matrix function to generate confusion matrices for both the training and testing data. The conf_matrix function is not shown in this code snippet, so it is unclear what it does or how it works.Finally, the function returns the predicted values for the training and testing data, as well as the fitted model.

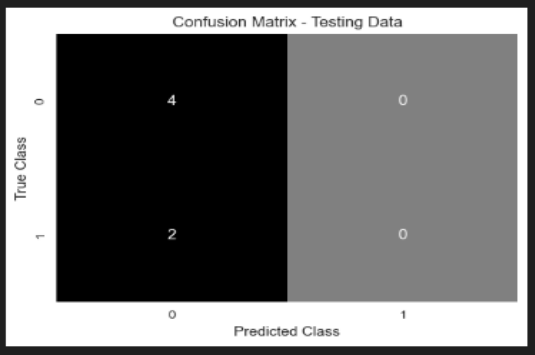
Confusion matrices for both training and test datasets-----**Zero with other**

run One_Vs_rest fOR eash two classes

```
train_0,test_0,model_0 = One_Vs_rest(X_Train,y_Train,X_Test,y_Test,0,SVC(kernel='linear'))
```



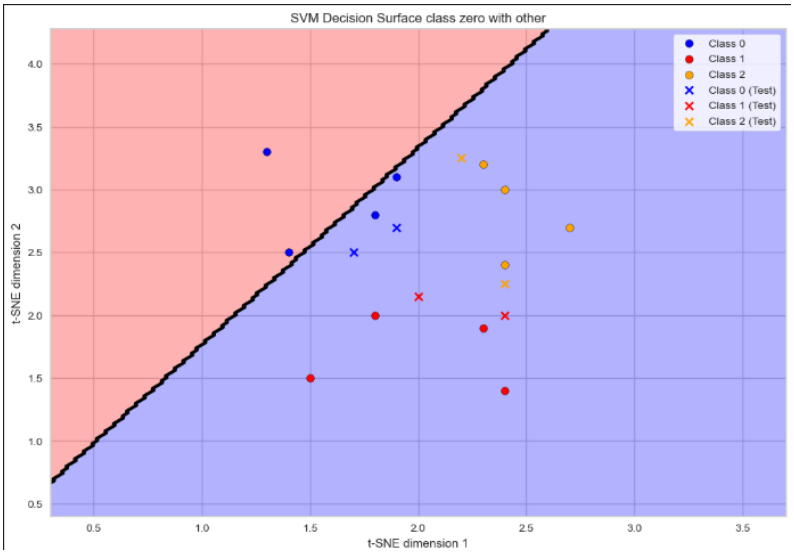

```
Accuracy:  0.8333333333333334
```




```
Accuracy:  0.6666666666666666
```

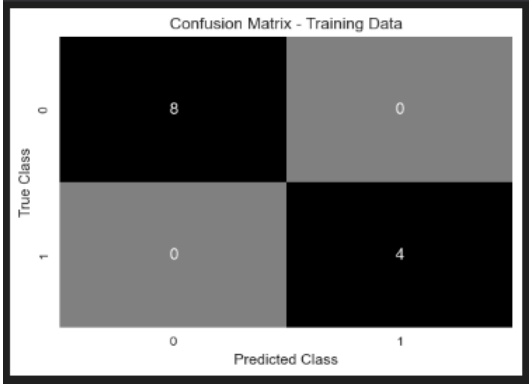
Output for SVM

```
plot_decision_boundary_All_data([model_0],X_Train, y_Train,X_Test,y_Test,"SVM Decision Surface class zero with other")
```

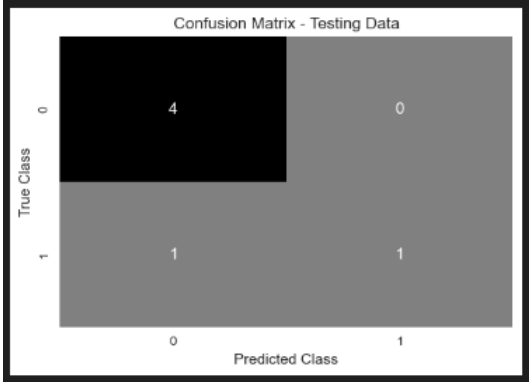


Confusion matrices for both training and test datasets-----**ONE with other**

```
train_1,test_1,model_1=One_Vs_rest(X_Train,y_Train,X_Test,y_Test,1,SVC(kernel='linear'))
```



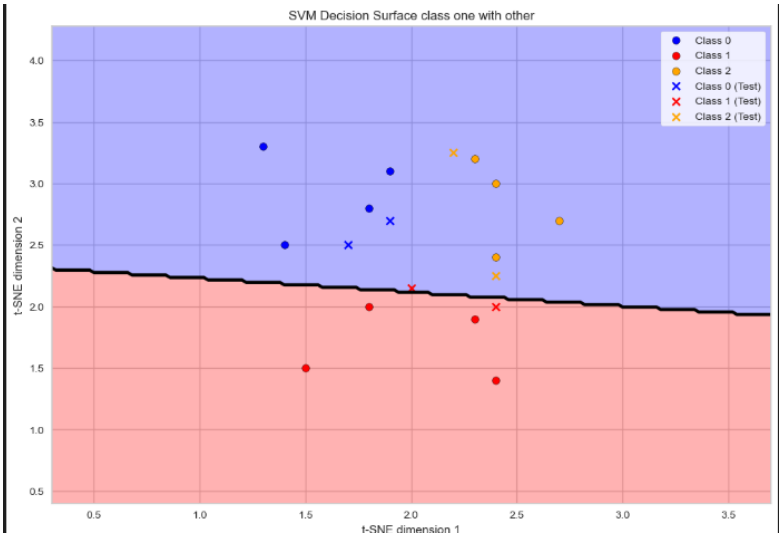

```
Accuracy:  1.0
```




```
Accuracy:  0.8333333333333334
```

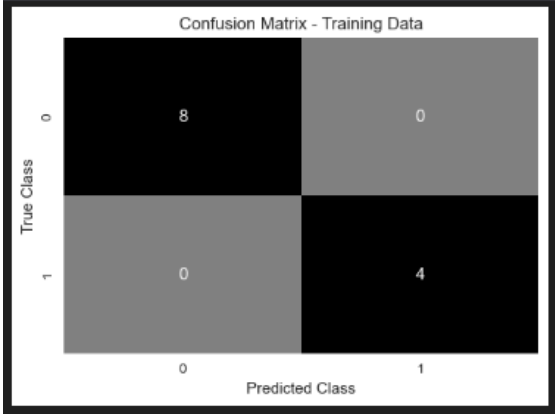
The output for SVM --- **ONE with other**

```
plot_decision_boundary_All_data([model_1],X_Train, y_Train,X_Test,y_Test,"SVM Decision Surface class one with other")
```

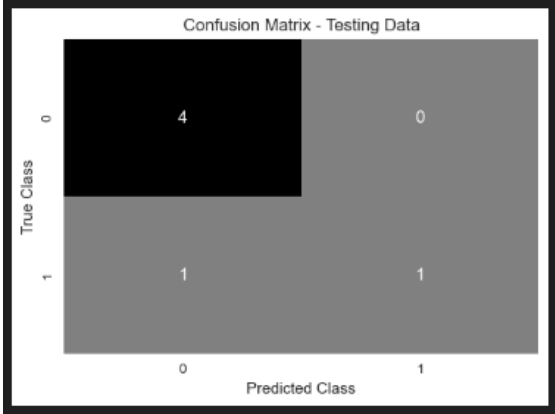


Confusion matrices for both training and test datasets-----**Two with other**

```
train_2,test_2,model_3=One_Vs_rest(X_Train,y_Train,X_Test,y_Test,2,SVC(kernel='linear'))
```



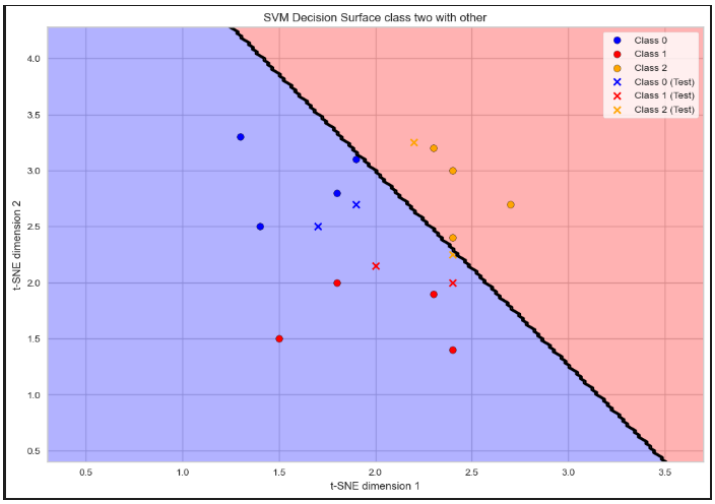

```
Accuracy:  1.0
```




```
Accuracy:  0.8333333333333334
```

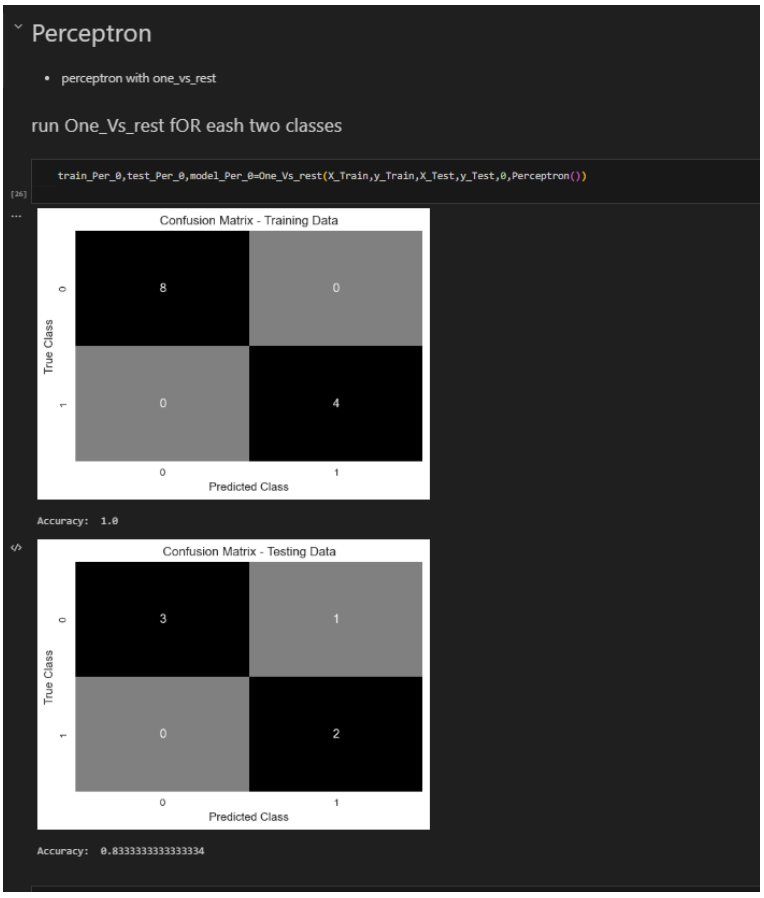
The output for SVM --- **TWO with other**

```
plot_decision_boundary_All_data([model_3],X_Train, y_Train,X_Test,y_Test,"SVM Decision Surface class two with other")
```



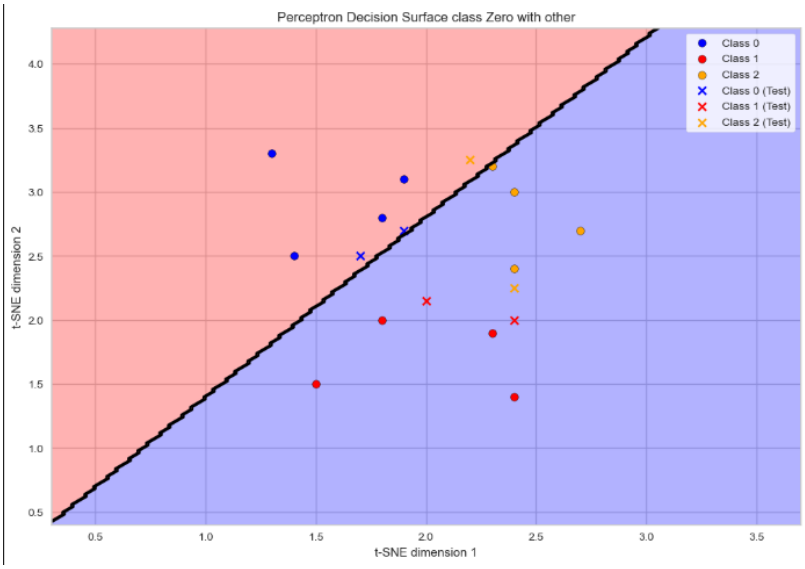
One-vs-Rest →Perceptron

Confusion matrices for both training and test datasets-----**Zero with other**



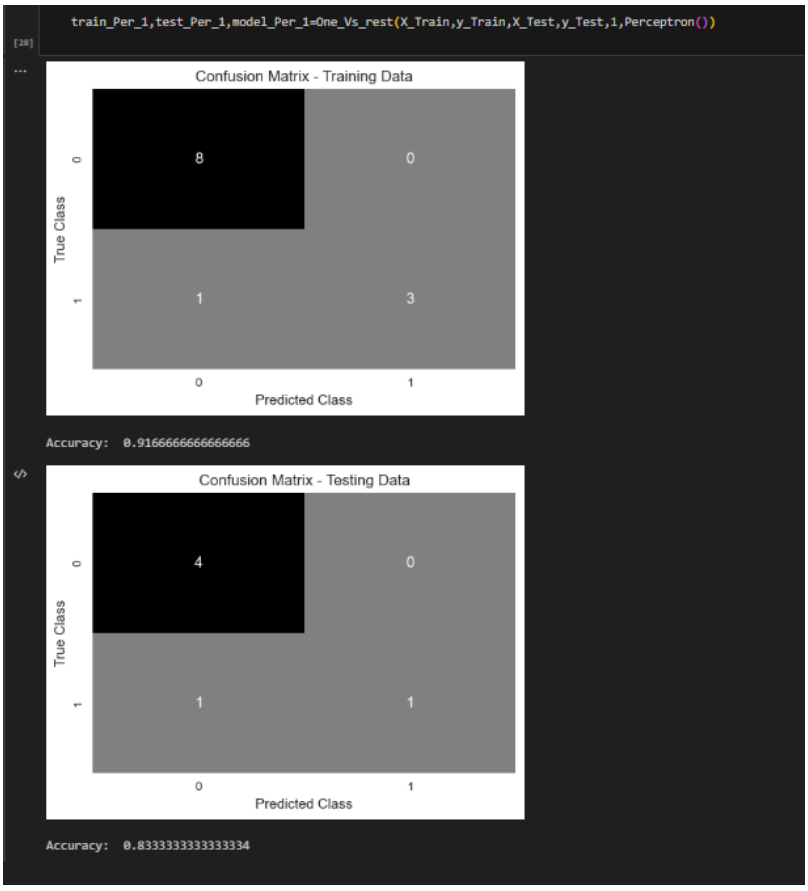
The output for perceptron--- **Zero with other**

```
plot_decision_boundary_All_data([model_Per_0],X_Train, y_Train,X_Test,y_Test,"Perceptron Decision Surface class Zero with other")
```



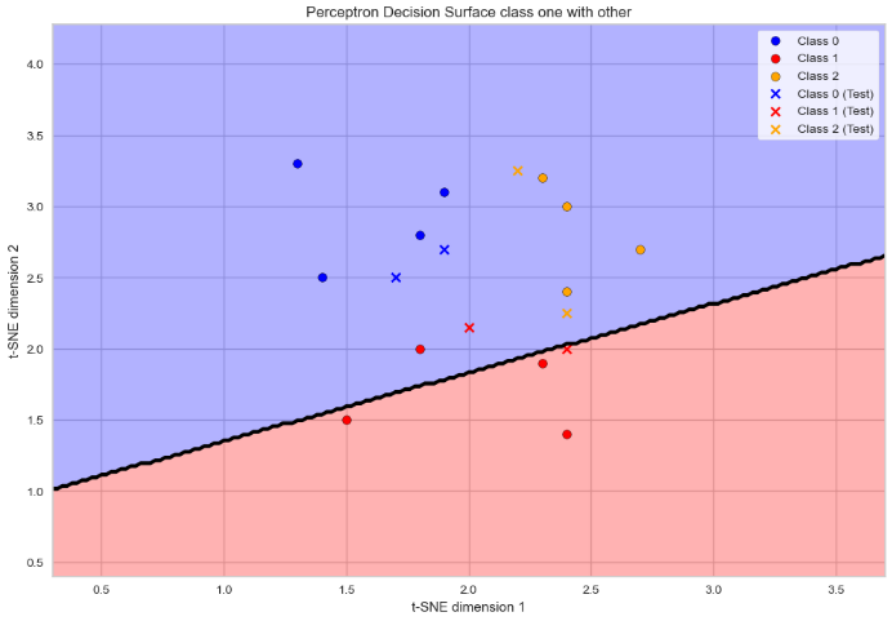
Confusion matrices for both training and test datasets-----**ONE with other**

```
train_Per_1,test_Per_1,model_Per_1=One_Vs_rest(X_Train,y_Train,X_Test,y_Test,1,Perceptron())
```



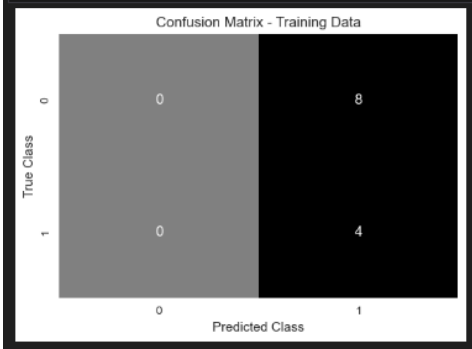
The output for perceptron--- **ONE with other**

```
plot_decision_boundary_All_data([model_Per_1],X_Train, y_Train,X_Test,y_Test,"Perceptron Decision Surface class one with other")
```

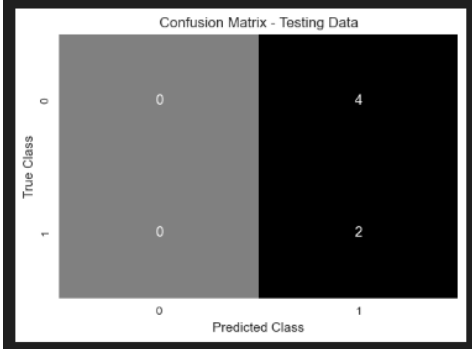


Confusion matrices for both training and test datasets-----**TWO with other**

```
train_Per_2,test_Per_2,model_Per_3=One_Vs_rest(X_Train,y_Train,X_Test,y_Test,2,Perceptron())
```

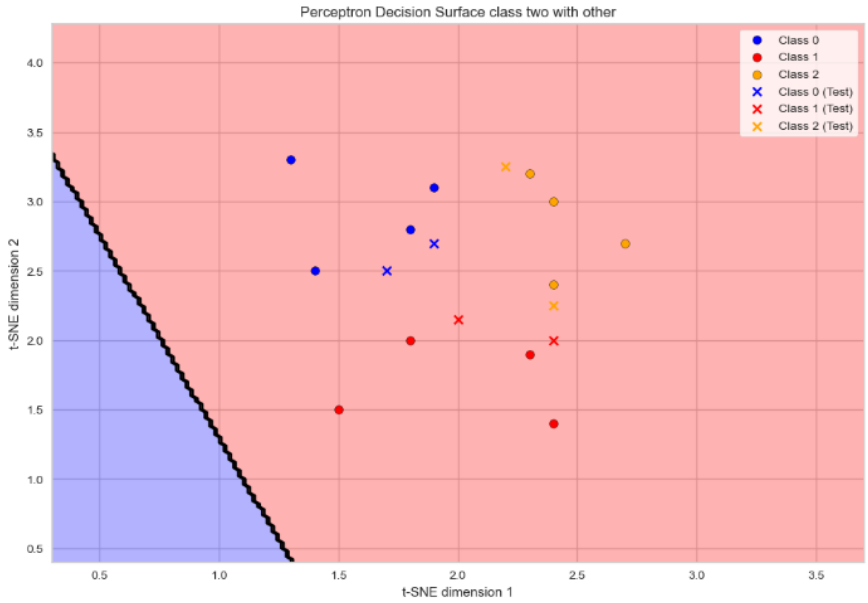



```
Accuracy:  0.3333333333333333
```




```
Accuracy:  0.3333333333333333
```

```
plot_decision_boundary_All_data([model_Per_3],X_Train, y_Train,X_Test,y_Test,"Perceptron Decision Surface class two with other")
```



c. Aggregation step (the one-vs-rest strategy for SVM).

Function concat

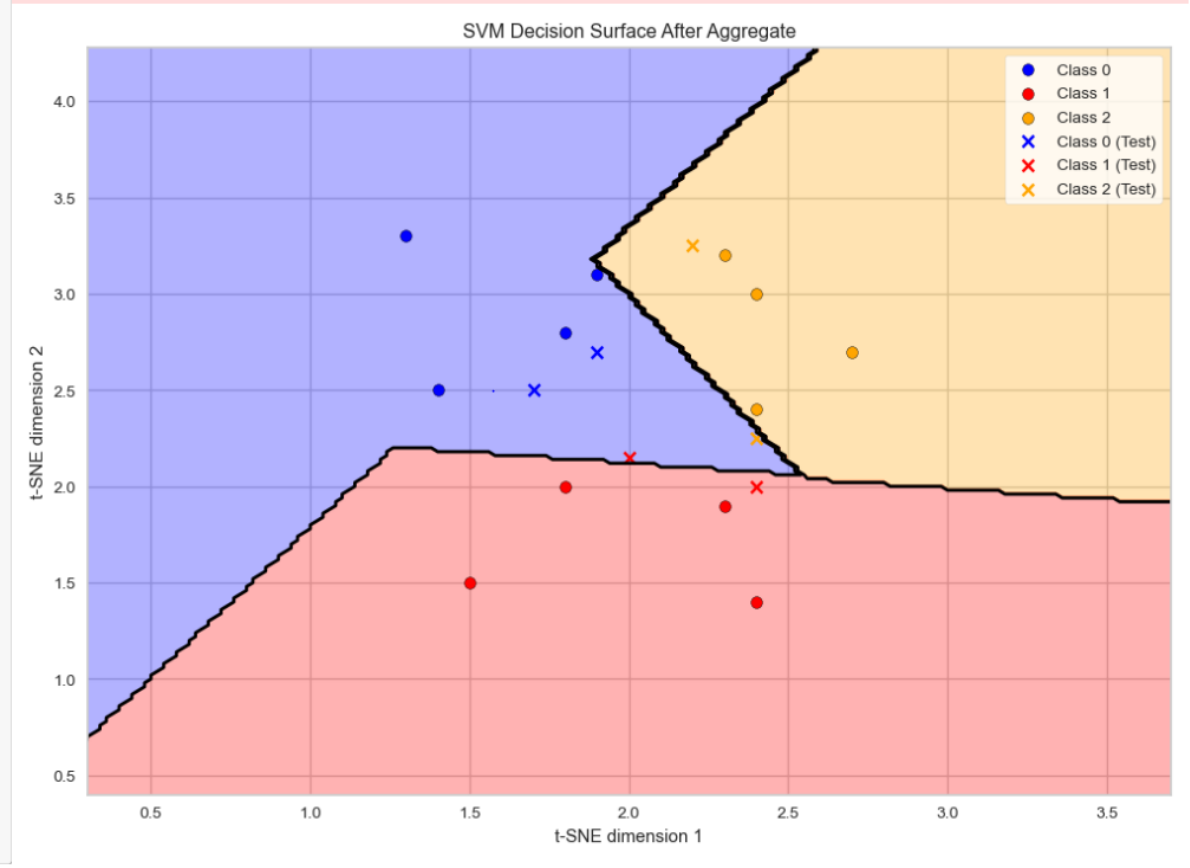
- To aggregate three list

```
def concat(l1, l2, l3):
    combined_array = np.vstack((l1, l2, l3))
    result = np.argmax(combined_array, axis=0)
    return result
```

Aggregate results from the one-vs-rest strategy for SVM

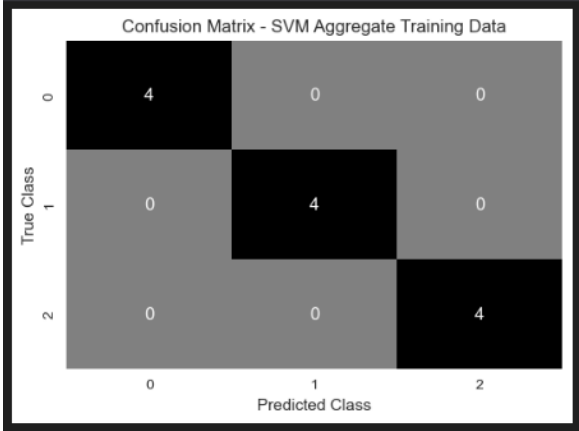
```
argmax_train=concat(train_0,train_1,train_2)
argmax_test=concat(test_0,test_1,test_2)
```

```
plot_decision_boundary_All_data([model_0,model_1,model_3],X_Train, y_Train,X_Test,y_Test,'SVM Decision Surface After Aggregate')
```



Confusion matrix for aggregated SVM ---TRAINING SET

```
conf_matrix(y_Train, argmax_train,"SVM Aggregate Training",True)
```




```
Classification Report:
              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
```

Confusion matrix for aggregated SVM ---TESTING SET

```
conf_matrix(y_Test, argmax_test,"SVM Aggregate Training testing",True)
```

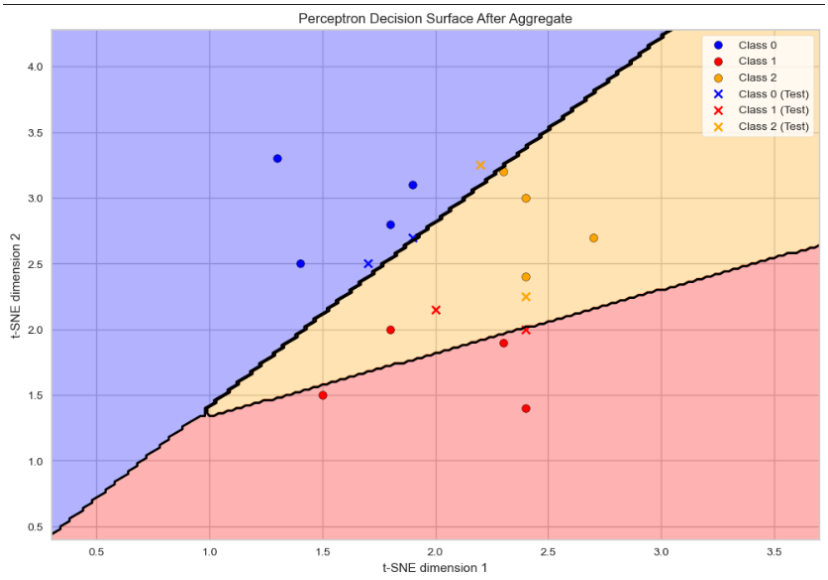

```
Classification Report:
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         2
           1       1.00      0.50      0.67         2
           2       1.00      0.50      0.67         2

    accuracy                           0.67         6
   macro avg       0.83      0.67      0.67         6
weighted avg       0.83      0.67      0.67         6
```

Aggregation step (the one-vs-rest strategy for perceptron).

Aggregate results from the one-vs-rest strategy for perceptron

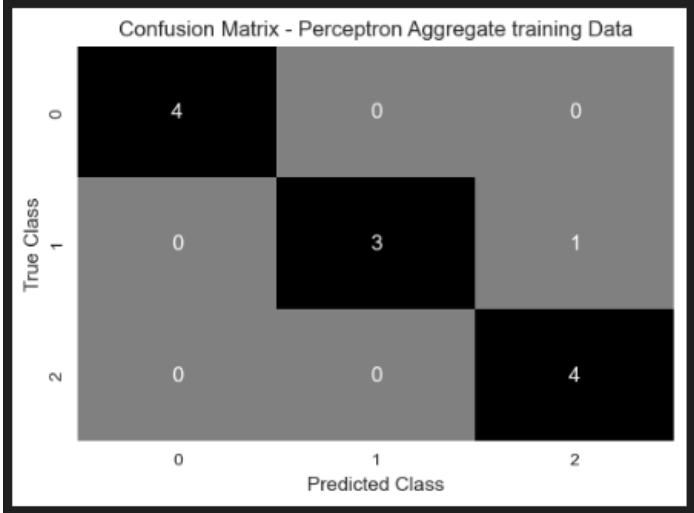
```
argmax_train_prc=concat(train_Per_0,train_Per_1,train_Per_2)
argmax_test_prc=concat(test_Per_0,test_Per_1,test_Per_2)
```

```
plot_decision_boundary_All_data([model_Per_0,model_Per_1,model_Per_3],X_Train, y_Train,X_Test,y_Test,'Perceptron Decision Surface After Aggregate')
```



Confusion matrix for aggregated perceptron---TRAINING SET

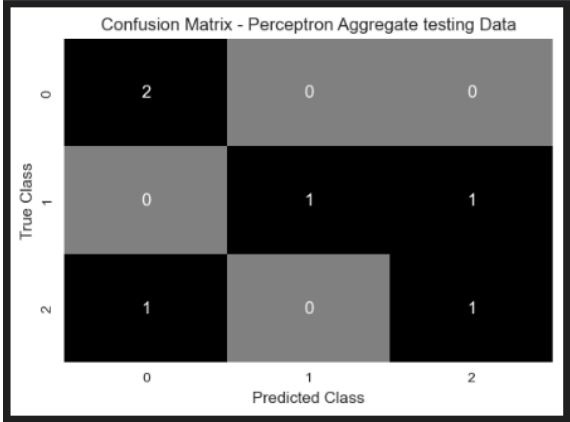
```
conf_matrix(y_Train, argmax_train_prc,"Perceptron Aggregate training")
```




```
Accuracy:  0.9166666666666666
```

Confusion matrix for aggregated perceptron---TESTING SET

```
conf_matrix(y_Test, argmax_test_prc,"Perceptron Aggregate testing",True)
```



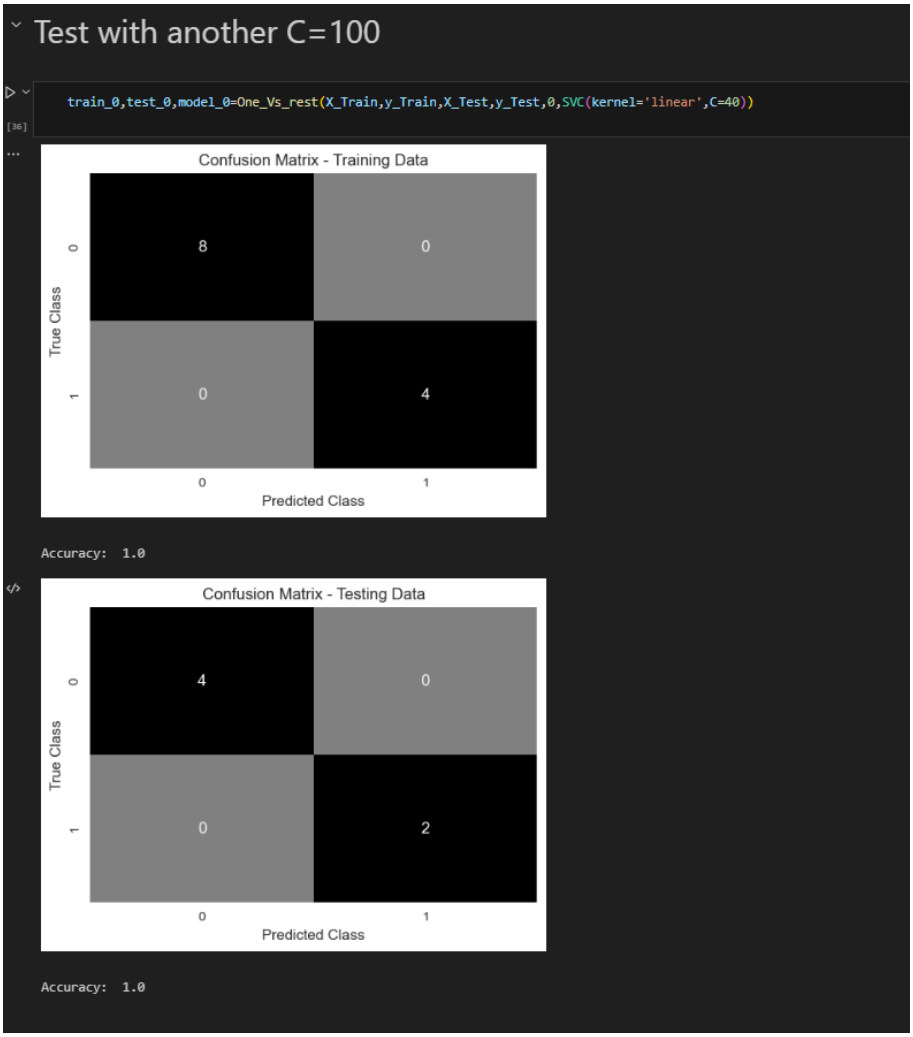

```
Classification Report:
              precision    recall  f1-score   support

           0       0.67      1.00      0.80         2
           1       1.00      0.50      0.67         2
           2       0.50      0.50      0.50         2

    accuracy                           0.67         6
   macro avg       0.72      0.67      0.66         6
weighted avg       0.72      0.67      0.66         6
```

d.

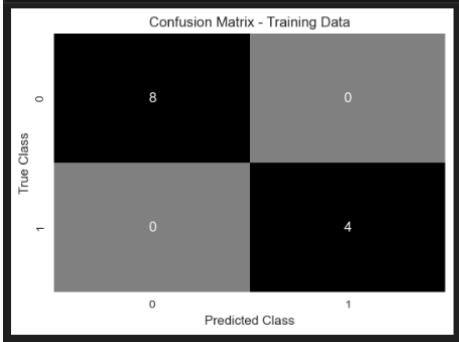
- The reason to why SVM performance in section (a) is different from the aggregated performance of SVM in section (c)?

  ✓ Because we use in one Vs rest SVM With Kernal Linear to classify between each two classes but in a we use Default SVM that more complex than linear and give more Accuracy but when we use SVM (Linear) in One vs Rest with C=100 we get 100 % Accuracy.

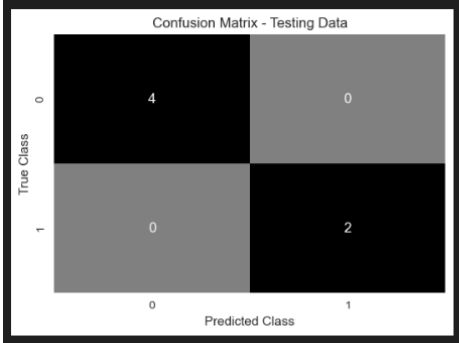
Refine the default SVM by selecting the appropriate parameter



```
train_1,test_1,model_1=One_Vs_rest(X_Train,y_Train,X_Test,y_Test,1,SVC(kernel='linear',C=40))
```

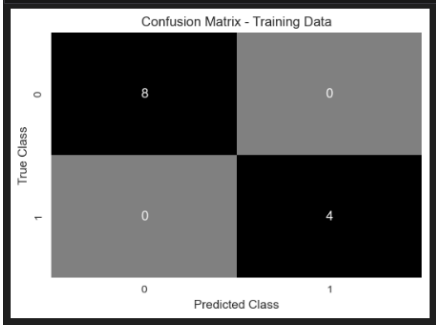



```
Accuracy:  1.0
```

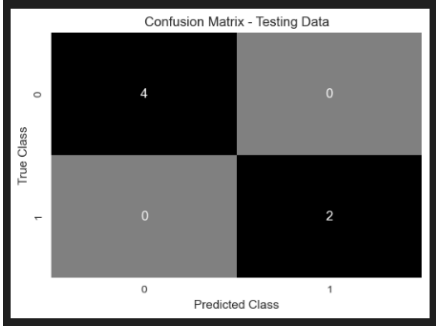



```
Accuracy:  1.0
```

```
train_1,test_1,model_1=One_Vs_rest(X_Train,y_Train,X_Test,y_Test,1,SVC(kernel='linear',C=40))
```

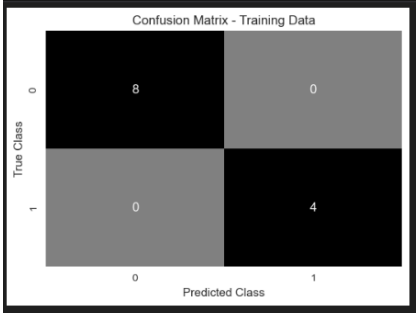



```
Accuracy:  1.0
```

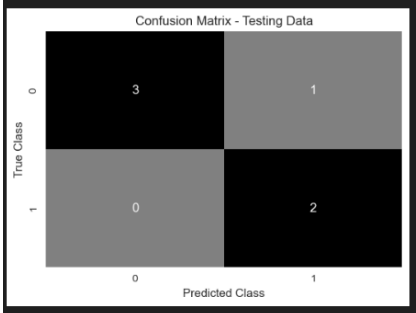



```
Accuracy:  1.0
```

```
train_2,test_2,model_3=One_Vs_rest(X_Train,y_Train,X_Test,y_Test,2,SVC(kernel='linear',C=40))
```



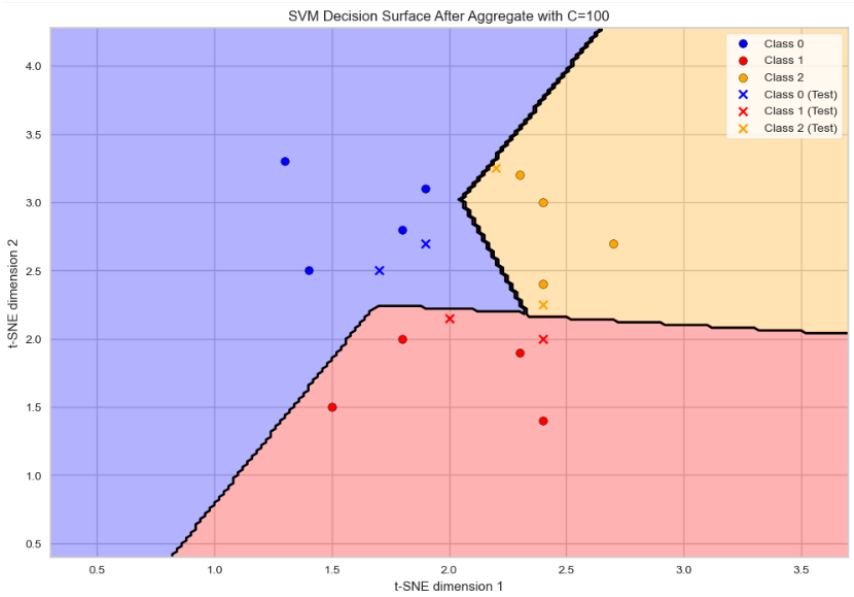

```
Accuracy:  1.0
```



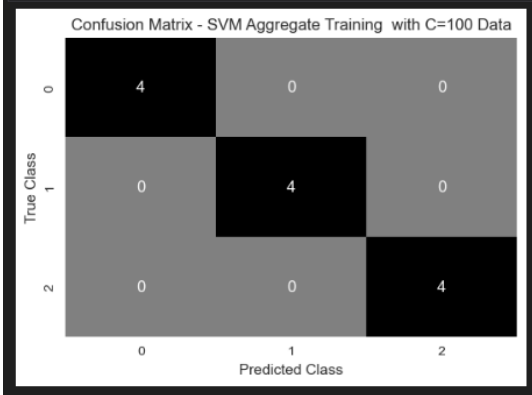

```
Accuracy:  0.8333333333333334
```

```
argmax_train=concat(train_0,train_1,train_2)
argmax_test=concat(test_0,test_1,test_2)
```

```
plot_decision_boundary_All_data([model_0,model_1,model_3],X_Train, y_Train,X_Test,y_Test,'SVM Decision Surface After Aggregate with C=100')
```

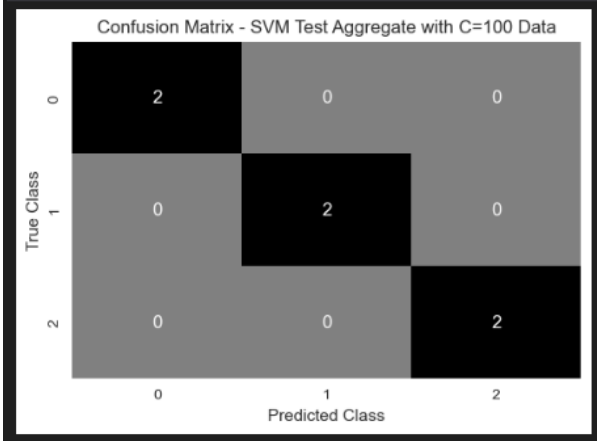


```
conf_matrix(y_Train, argmax_train,"SVM Aggregate Training  with C=100",True)
```




```
Classification Report:
              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
```

```
conf_matrix(y_Test, argmax_test,"SVM Test Aggregate with C=100",True)
```




```
Classification Report:
              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
```

Part 2 → KNeighborsClassifier

Firstly, we named the columns with these names and we considered the buying_decision as our target column.

```
header=['buying_price','maint_cost','car_doors','car_seaters','boot_space','safety','buying_decision']
df.columns=header
df.head()
```

✓ 0.0s

The Output

	buying_price	maint_cost	car_doors	car_seaters	boot_space	safety	buying_decision
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

Requirements of the task

a. We shuffled the dataset and split the dataset into a
   1) A training set with **1000** samples
   2) Validation set with **300** samples
   3) Testing set with **428** samples.

```
shuffled_df = df.sample(frac=1, random_state=42)

train_data, remaining_data = train_test_split(shuffled_df, train_size=1000, random_state=42)
valid_data, test_data = train_test_split(remaining_data, train_size=300, random_state=42)
```

✓ 0.0s

b- Transforming the string values into numbers with LabelEncoder.

```
def encoding (df):
    encoder = LabelEncoder()
    for i in header:
        df[i] = encoder.fit_transform(df[i])
```

Make Label Encoding for testing, validation, and training.

```
encoding(train_data)
encoding(valid_data)
encoding(test_data)
```

✓ 0.0s

The Output of Training Data encoding

```
train_data
```

✓ 0.0s

	buying_price	maint_cost	car_doors	car_seaters	boot_space	safety	buying_decision
1299	1	3	0	0	1	1	2
1302	1	3	0	0	0	1	2
320	3	2	3	2	1	0	0
595	0	0	2	0	2	2	2
1499	1	0	3	1	1	0	3
...	...	...	...	...	...	...	...
945	2	3	3	0	2	1	2
4	3	3	0	0	1	2	2
665	0	2	0	1	0	0	0
488	0	3	2	0	2	0	2
338	3	1	0	1	1	0	0

1000 rows × 7 columns

c- We used different numbers of training samples to show the impact of a number of training samples as required.

We used 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100% of the training set for 10 separate KNN classifiers.

```
def mult_KNN():
    validation_scores=[]
    testing_scores=[]
    max_valid_score_index = -1
    max_test_score_index = -1

    for i in training_proportions:
        num_train_samples = int(i * len(train_data))

        target_Xtrain = train_data.iloc[:num_train_samples, :-1]
        target_Ytrain = train_data.iloc[:num_train_samples, -1]

        y_valid_pred,y_test_pred=knn(target_Xtrain,target_Ytrain,X_valid_data,X_test_data,k=2)
        validation_acc = accuracy_score(y_valid_data, y_valid_pred)
        validation_scores.append(validation_acc)
        if(validation_acc == max(validation_scores)):
            max_valid_score_index = training_proportions.index(i)

        testing_acc = accuracy_score(y_test_data, y_test_pred)
        testing_scores.append(testing_acc)
        if(testing_acc == max(testing_scores)):
            max_test_score_index = training_proportions.index(i)

    print("valid",max(validation_scores))
    print("test",max(testing
    print("max_valid_index",
    print("max_test_index", max_test_score_index)

    plt.plot(training_proportions, validation_scores, label='Validation')
    plt.plot(training_proportions, testing_scores, label='Testing')
    plt.xlabel('Training Set Proportion')
    plt.ylabel('Accuracy Score')
    plt.title("Training set for 10 separate KNN classifiers and show their performance")
    plt.legend()
    plt.show()

mult_KNN()
```

(variable) max_test_score_index: int

- Inside the loop, calculate the number of training samples based on the current proportion.
- Extract the target features (**target_Xtrain**) and target labels (**target_Ytrain**) from the **train_data** DataFrame up to the calculated number of samples.
- Calculate the accuracy scores for the validation set by comparing the predicted labels (**y_valid_pred**) with the true labels (**y_valid_data**) using the **accuracy_score ()** function. Append the score to the **validation_scores** list.
- It shows 10 scores for each training sample in the graph.

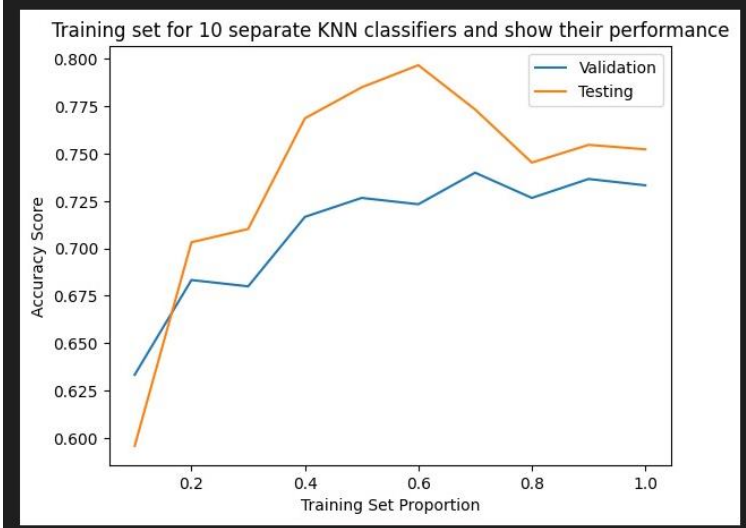
d- Using 100% of training samples, try to find the best K value, and show the accuracy curve on the validation set when K varies from 1 to 10.

```
def Check_k():
    K_proportions = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    validation_scores=[]
    testing_scores=[]

    for i in K_proportions:
        y_valid_pred,y_test_pred=knn(x_train_data,y_train_data,X_valid_data,X_test_data,k=i)
        validation_acc = accuracy_sco(variable) y_test_data: Any )
        validation_scores.append(vali
        testing_acc = accuracy_score(y_test_data, y_test_pred)
        testing_scores.append(testing_acc)
    plt.plot(K_proportions, validation_scores, label='Validation')
    plt.xlabel('K Set Proportion')
    plt.ylabel('Accuracy Score')
    plt.legend()
    plt.title("The accuracy curve on the validation set when K varies from 1 to 10")
    plt.show()


Check_k()
```

e- Conclusions from the experiments of questions (c) and (d).

We noticed that the optimal number of training data is 600 because achieve the highest accuracy

```
valid 0.74
test 0.7967289719626168
max_valid_index 6
max_test_index 5
```



We noticed that the optimal k for validating the set is 5 because achieve the highest accuracy