uOttawa

(Group_10) Assignment_1

Prepared for

Prof: Murat SIMSEK
TA: Samhita Kuili

Faculty of Engineering, University of u Ottawa

ELG5255: Applied Machine Learning

Prepared by

Yousef Abd Al Haleem Ahmed Shindy

Esraa Mahmoud Said Ahmed

Esmael Alahmady Ebrahim Ezz

Problems

Problem_1

a)

- ✓ **Train the model using the dataset.**

```
svm = SVC()
svm=svm.fit(X_Train , Y_Train)
y_pred = svm.predict(X_Train)
#plotting the data without the decision boundary
plot_tsne_decision([svm], X_Train, Y_Train, X_Test, Y_Test, "Scatter Plot")
#plotting the data with the decision boundary
plot_tsne_decision([svm], X_Train, Y_Train, X_Test, Y_Test,"Decision Surface for Multi-Class Classification",boole=True)
```

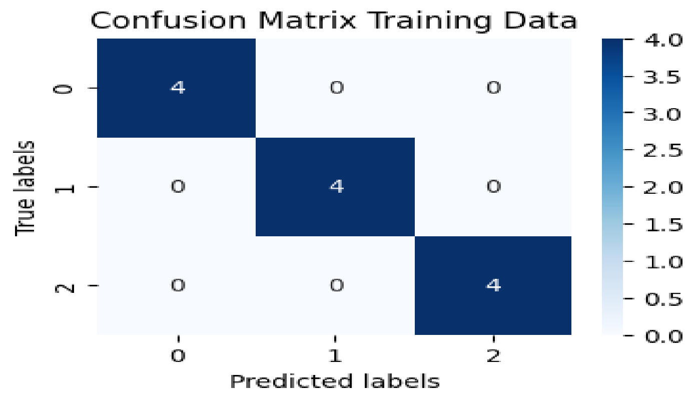
- ✓ **obtain confusion matrices for training dataset.**

➢ To obtain confusion matrix for training & testing datasets through the entire assignment, we define a **confusion_mx** function to calculate confusion matrix and plot its ‘labels.

```
def confusion_mx (y_test,y_pred,label):
    cm = confusion_matrix(y_test,y_pred)
    plt.figure(figsize=(4, 3))
    sns.heatmap(cm, annot=True,fmt='g', cmap='Blues')
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.title(f'Confusion Matrix {label}')
    plt.show()
```

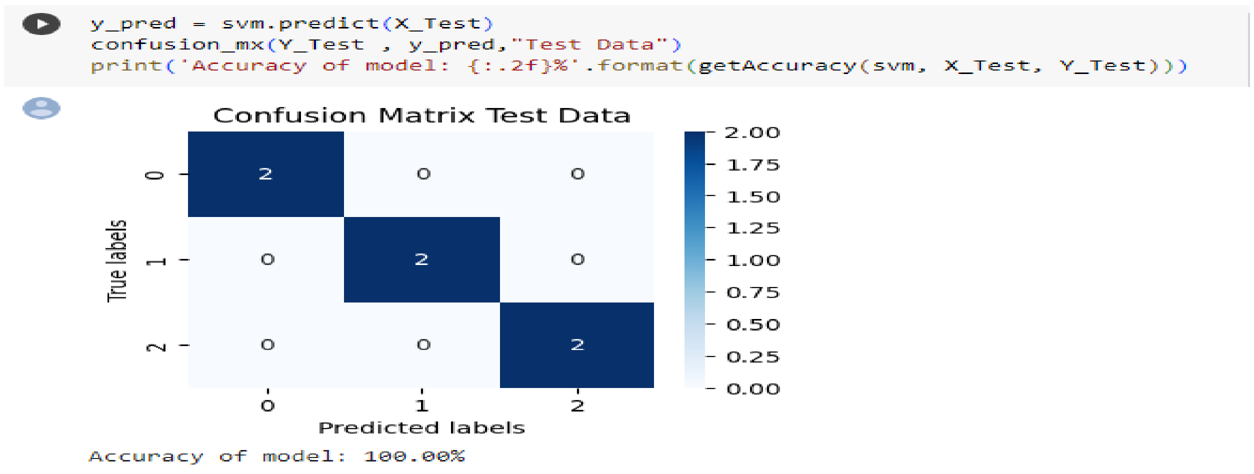
- ✓ **obtain confusion matrices for training dataset.**

```
y_pred = svm.predict(X_Train)
confusion_mx(Y_Train , y_pred,"Training Data")
print('Accuracy of model: {:.2f}%'.format(getAccuracy(svm, X_Test, Y_Test)))
```



```
Accuracy of model: 100.00%
```

✓ **obtain confusion matrices for testing dataset.**



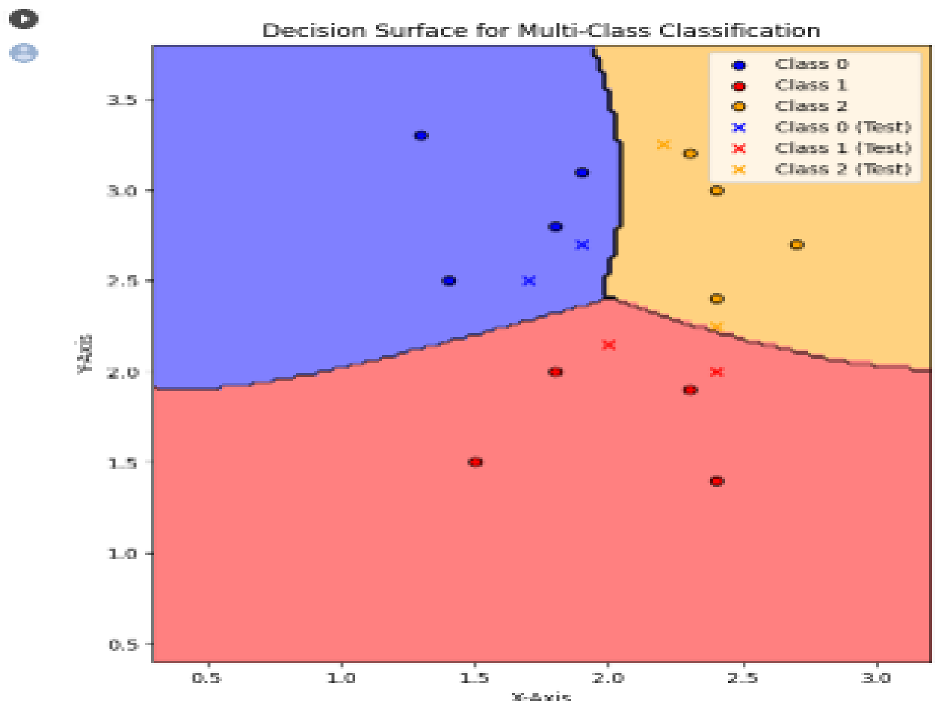
```
y_pred = svm.predict(X_Test)
confusion_mx(Y_Test , y_pred,"Test Data")
print('Accuracy of model: {:.2f}%'.format(getAccuracy(svm, X_Test, Y_Test)))
```

	0	1	2
0	2	0	0
1	0	2	0
2	0	0	2

```
Accuracy of model: 100.00%
```

➢ To visualize decision surfaces for multi-class classification we create a fun called **plot_tsne_decision** which takes list of models as a parameter in case of multi-class aggregation we iterate over the list to use the fitted models to plot the decision boundary.

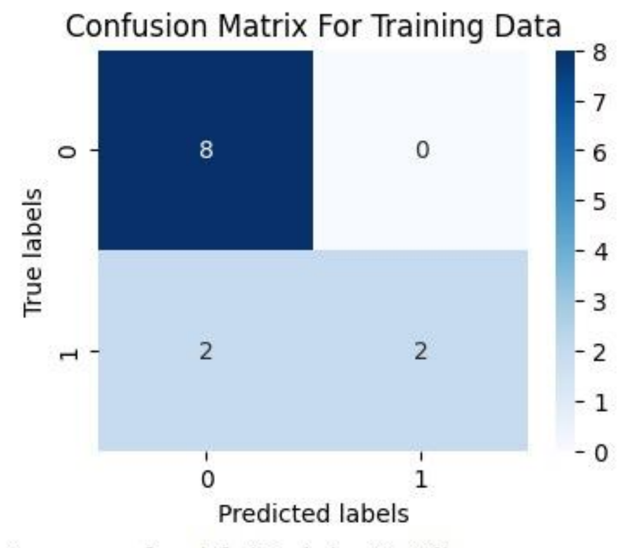
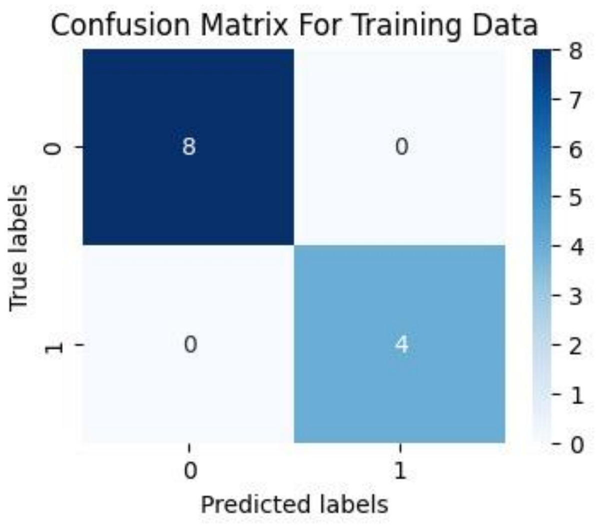
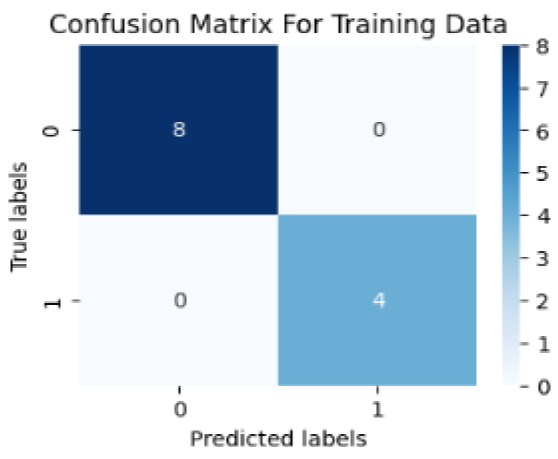
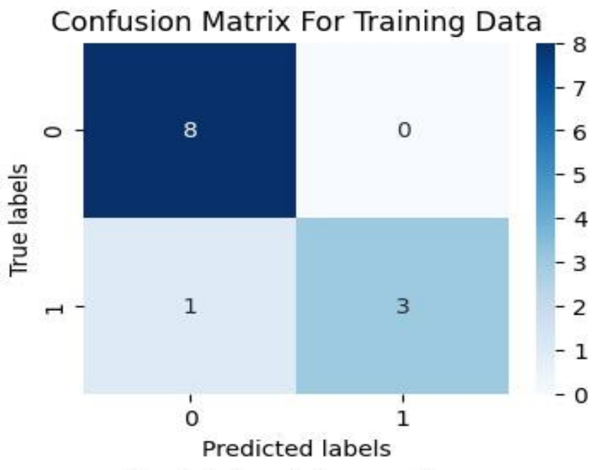
✓ **visualize decision surfaces for multi-class classification, using blue, red, and orange colors to plot each class.**

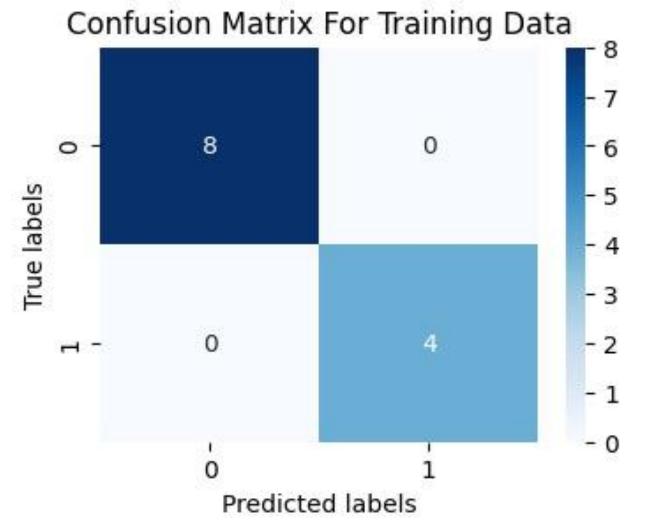
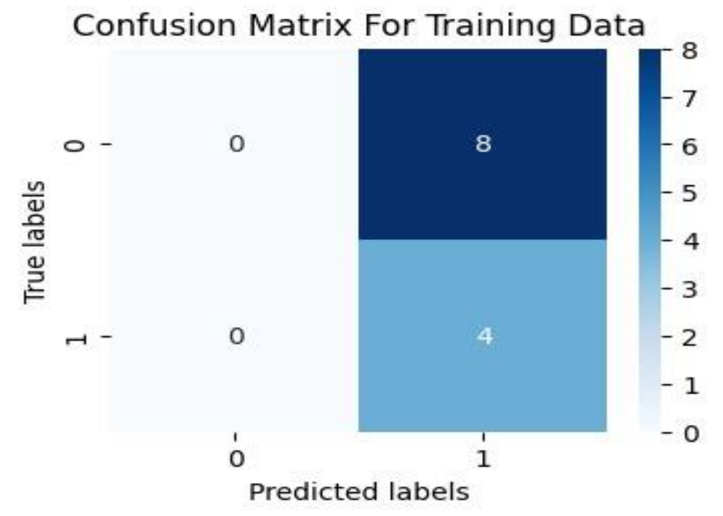


b)

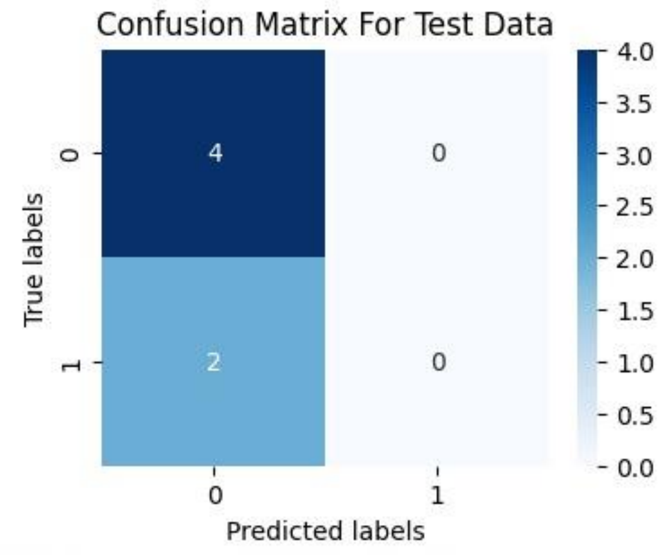
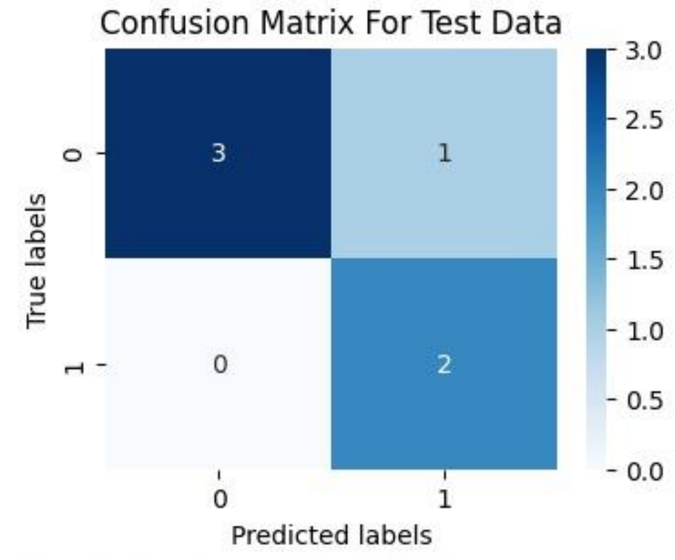
- To apply One-vs-Rest on SVM & Perceptron, we create **ove_class** function which takes main class and the model, encode target data for binary classification, fit the model for training data and plot the confusion matrix and do the same for the test data and return the model to use when plotting aggregated models results.

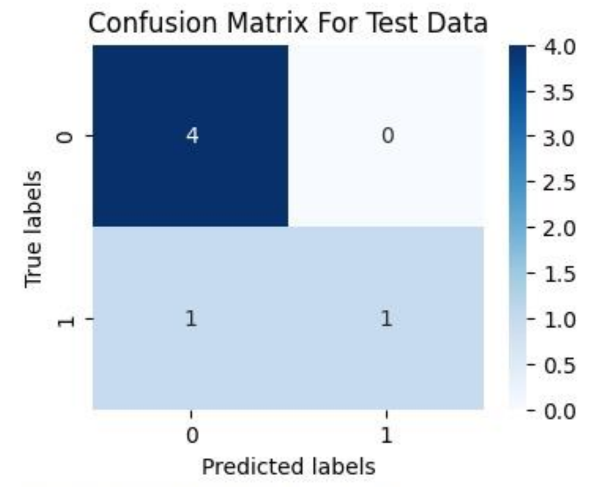
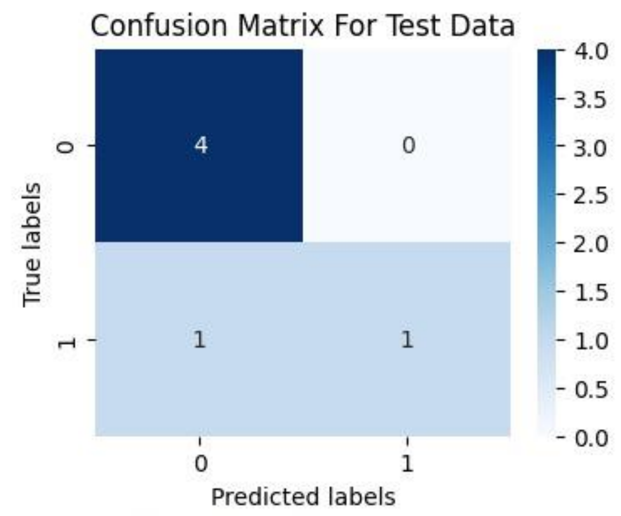
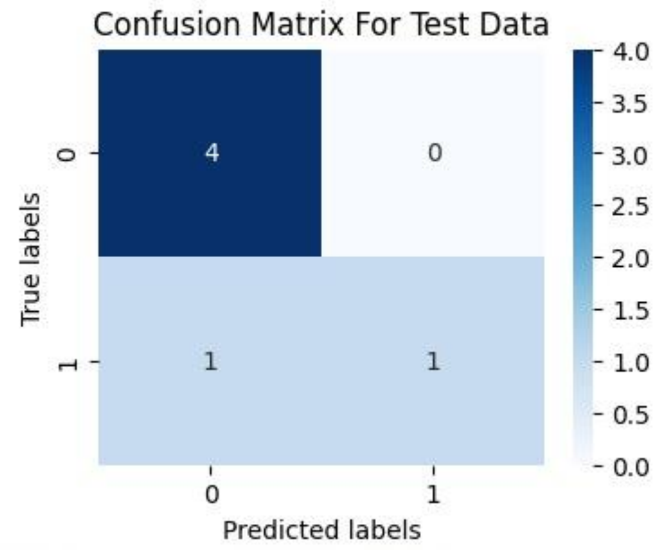
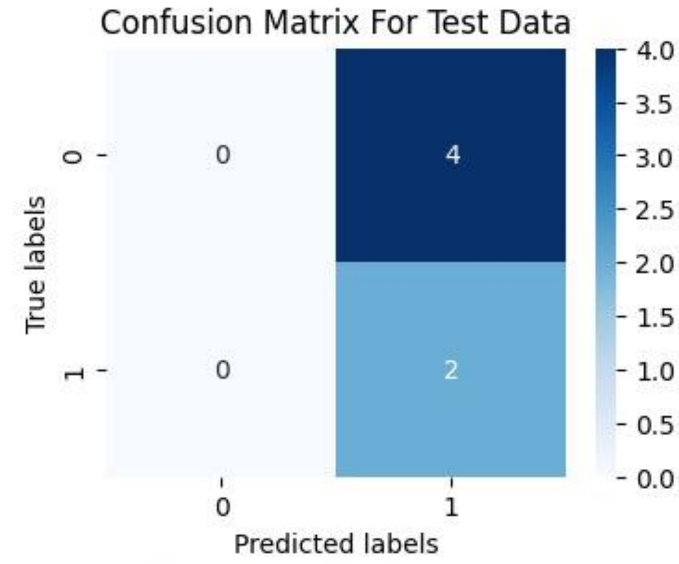
✓ **Obtain confusion matrices for training dataset for SVM compared to Perceptron.**

	SVM_Training	**Perceptron_Training**
Class **0** vs Rest	 `Accuracy of model 'Train': 83.33%`	 `Accuracy of model 'Train': 100.00%`
Class **1** vs Rest	 `Accuracy of model 'Train': 100.00%`	 `Accuracy of model 'Train': 91.67%`

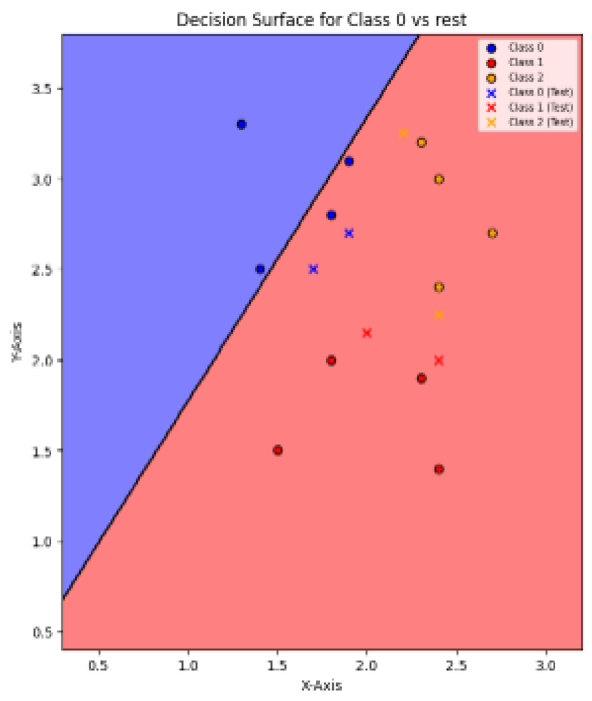
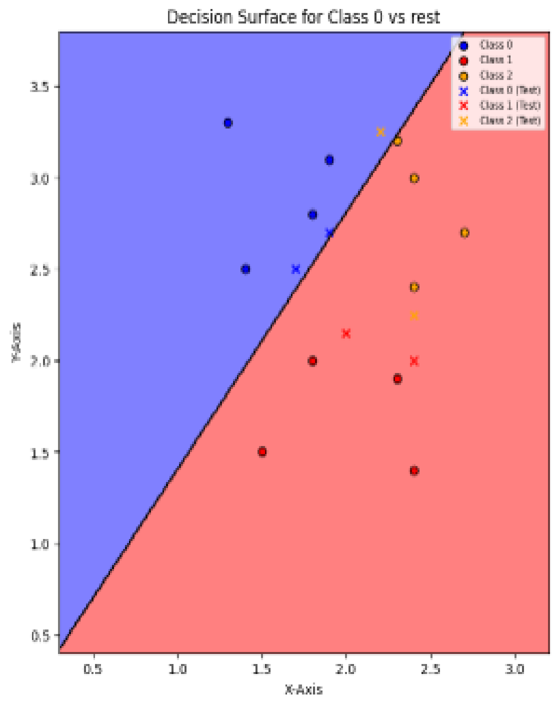
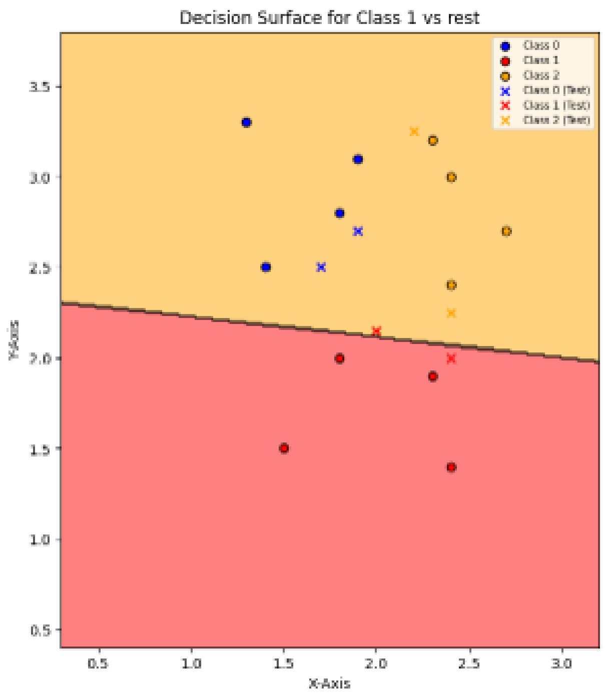
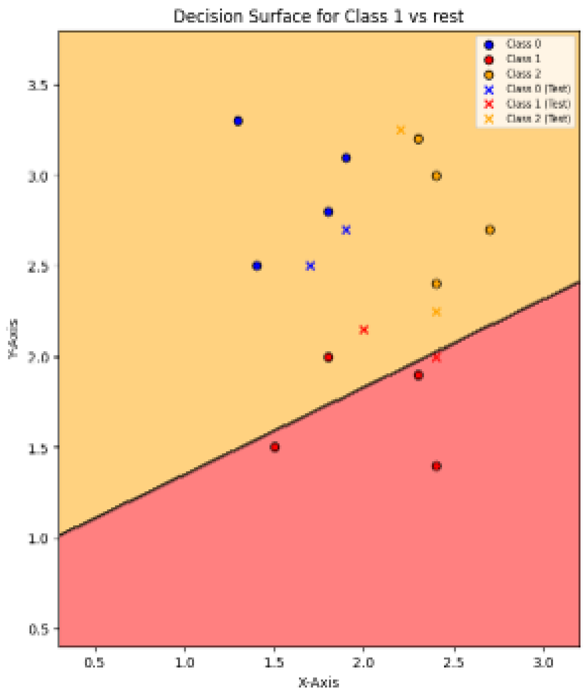
Class **2** vs Rest	 `Accuracy of model 'Train': 100.00%`	 `Accuracy of model 'Train': 33.33%`

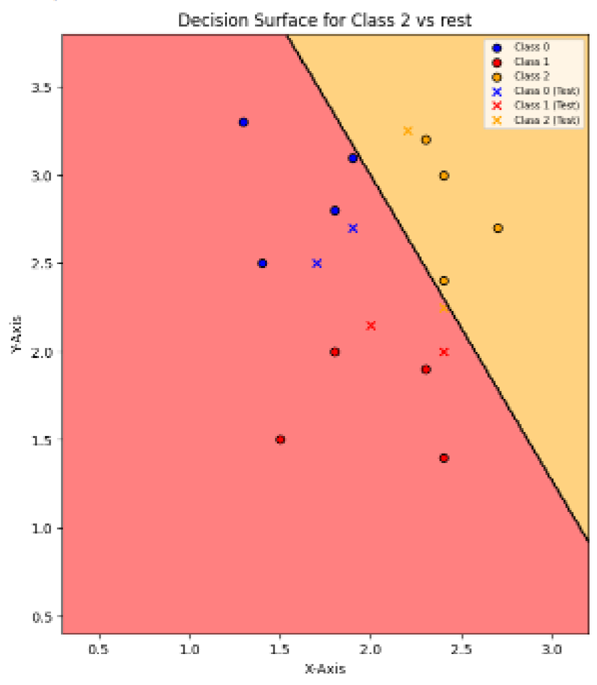
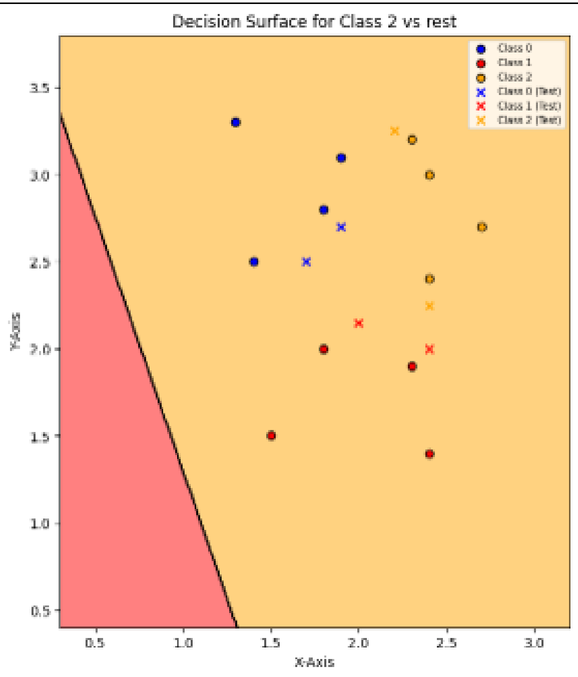
✓ **Obtain confusion matrices for testing dataset for SVM compared to Perceptron.**

	SVM_Testing	**Perceptron_Testing**
Class **0** vs Rest	 `Accuracy of model 'Test': 66.67%`	 `Accuracy of model 'Test': 83.33%`

Class **1** vs Rest	 `Accuracy of model 'Test': 83.33%`	 `Accuracy of model 'Test': 83.33%`
Class **2** vs Rest	 `Accuracy of model 'Test': 83.33%`	 `Accuracy of model 'Test': 33.33%`

✓ **Visualize decision surfaces using the color scheme mentioned earlier.**

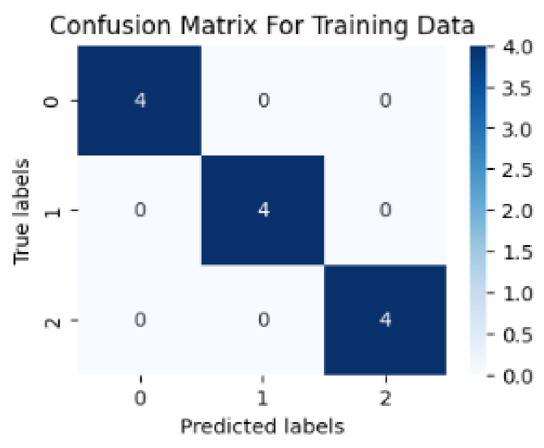
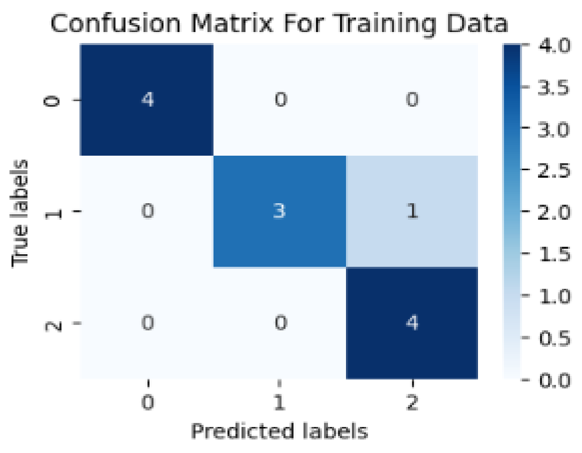
	SVM	**Perceptron**
Class **0** vs Rest		
Class **1** vs Rest		

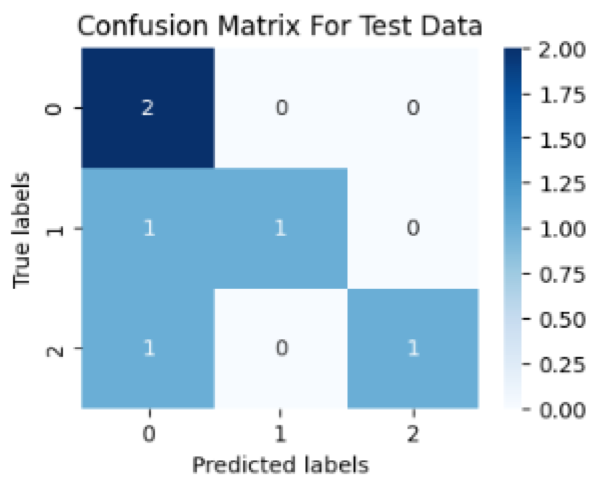
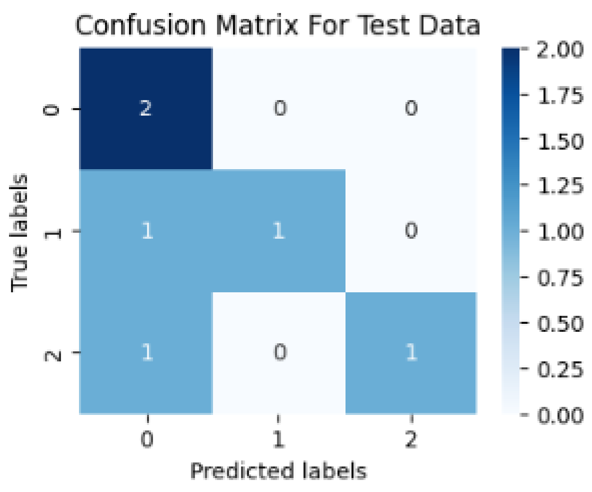
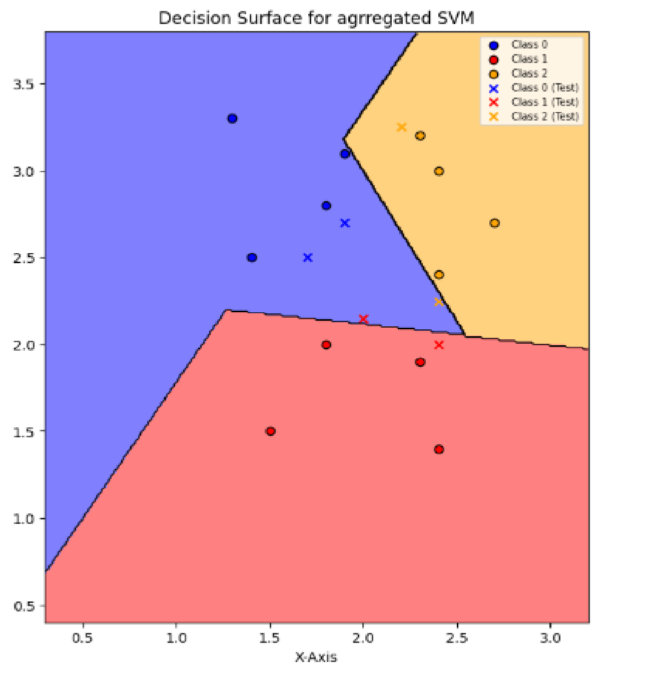
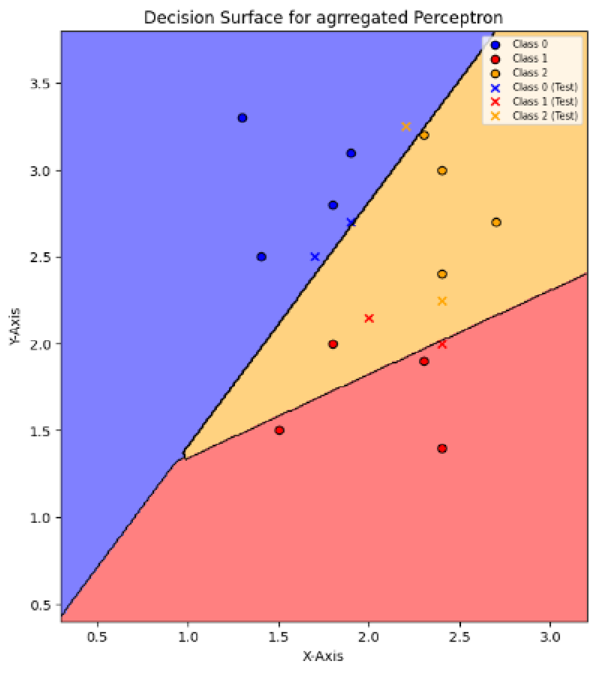
Class **2** vs Rest		

c)

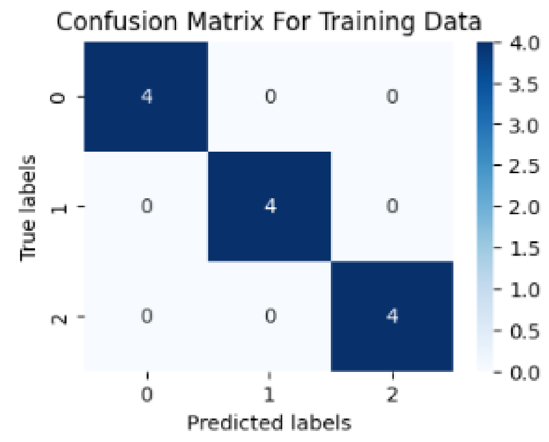
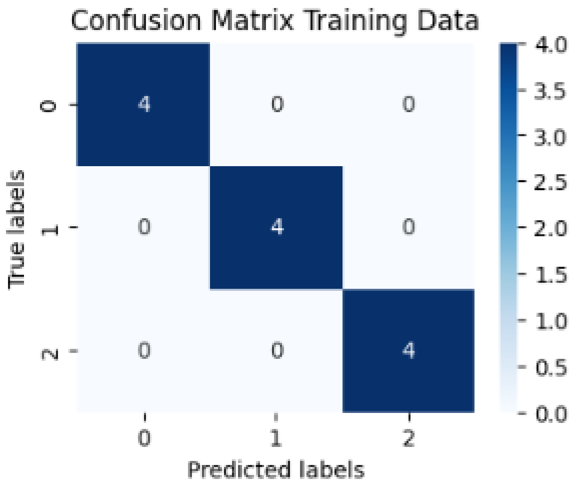
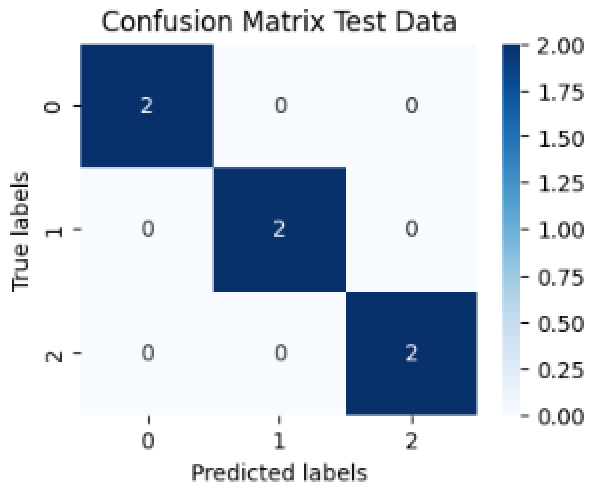
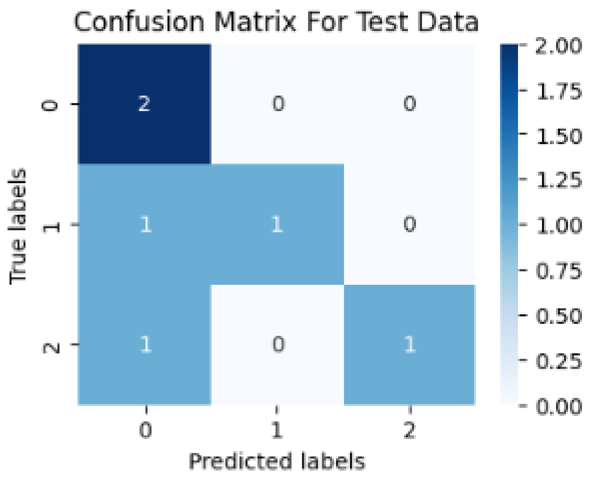
- To Aggregate the results from the one-vs-rest strategy for SVM and Perceptron, we define **aggre** function which take 3 lists and combining the results using **argmax** function to return the aggregated results.

- ✓ **Calculate the confusion matrix and visualize the decision surface for the aggregated results.**

	SVM	**Perceptron**
Training Result Aggregation	 `Accuracy of model: 100.0 %`	 `Accuracy of model: 91.66666666666666 %`

Testing Result Aggregation	 `Accuracy of model: 66.66666666666666 %`	 `Accuracy of model: 66.66666666666666 %`
Decision Boundary aggregation		

- ✓ **Analyze performance and compare with section (a), we notice a decrease in accuracy by 34% after aggregation.**

	SVM_Default In Section (a)	**SVM_After Aggregation In Section (c)**
Training	 `Accuracy of model: 100.0 %`	 `Accuracy of model: 100.00%`
Testing	 `Accuracy of model: 100.00%`	 `Accuracy of model: 66.66666666666666 %`

Confusion Matrix For Training Data (Section (a)):

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

Confusion Matrix Training Data (Section (c)):

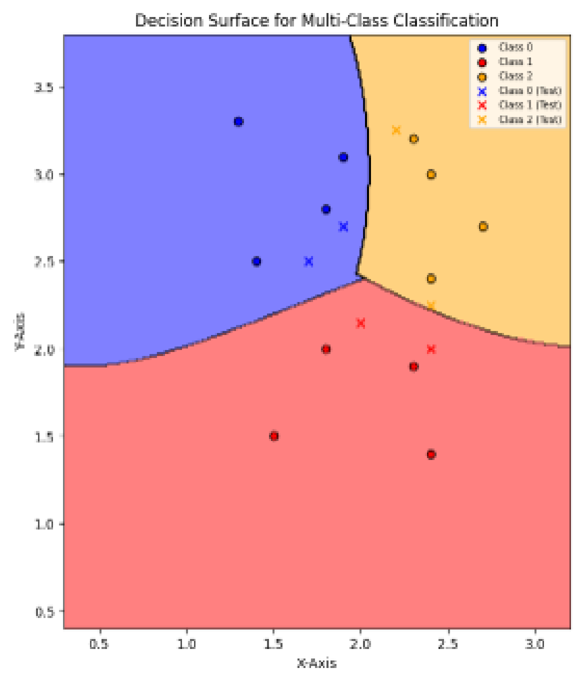
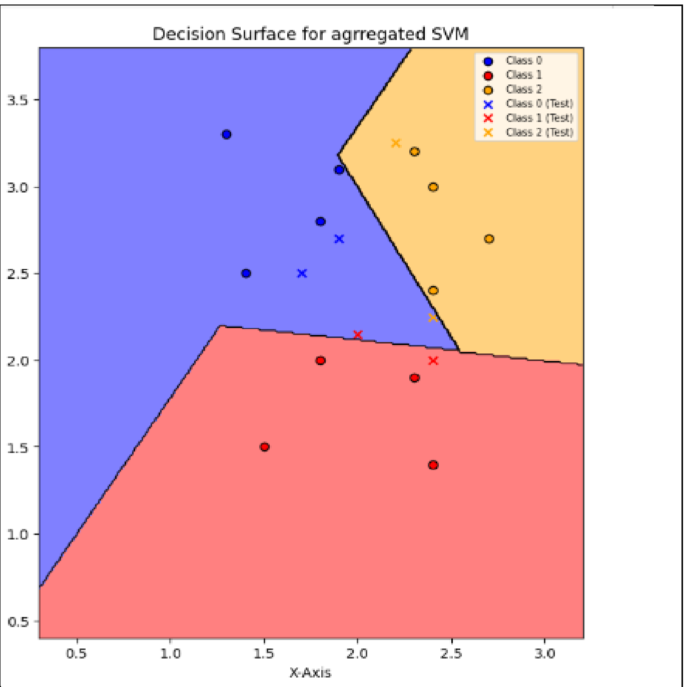
	0	1	2
0	4	0	0
1	0	4	0
2	0	0	4

Confusion Matrix Test Data (Section (a)):

	0	1	2
0	2	0	0
1	0	2	0
2	0	0	2

Confusion Matrix For Test Data (Section (c)):

	0	1	2
0	2	0	0
1	1	1	0
2	1	0	1

Decision Boundary		

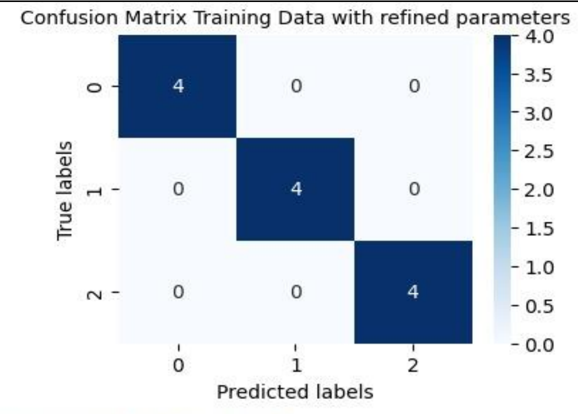
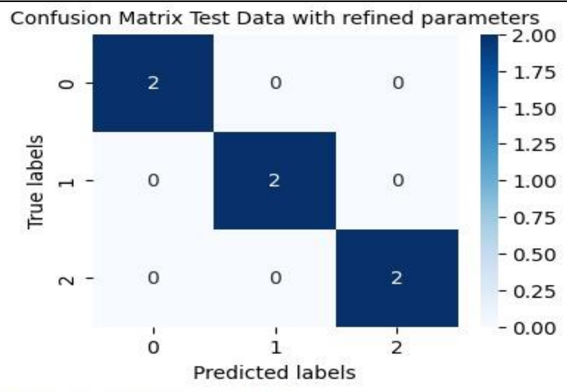
d)

- ✓ **Determine the reason why SVM performance in section (a) is different than aggregated performance of SVM in section (c).**

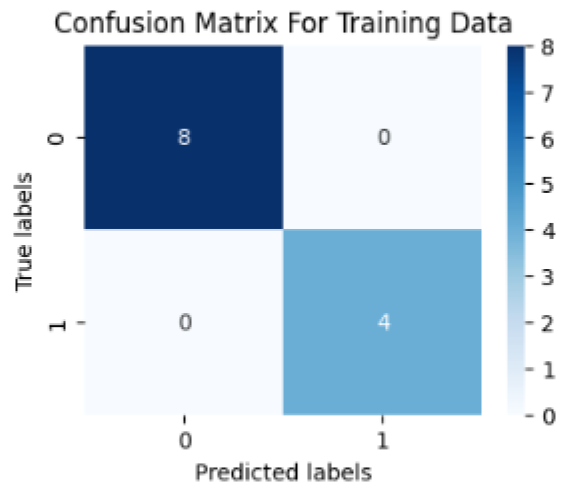
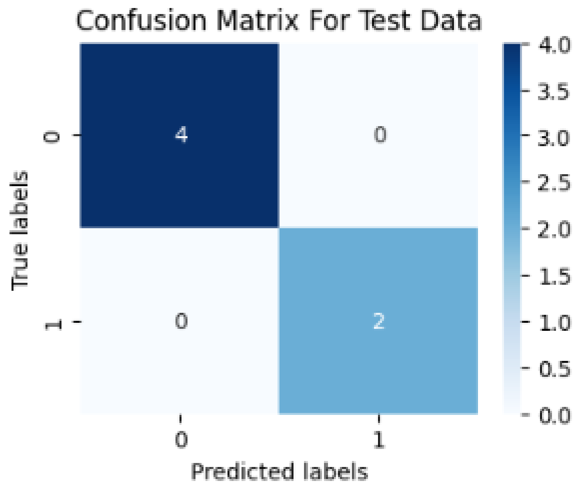
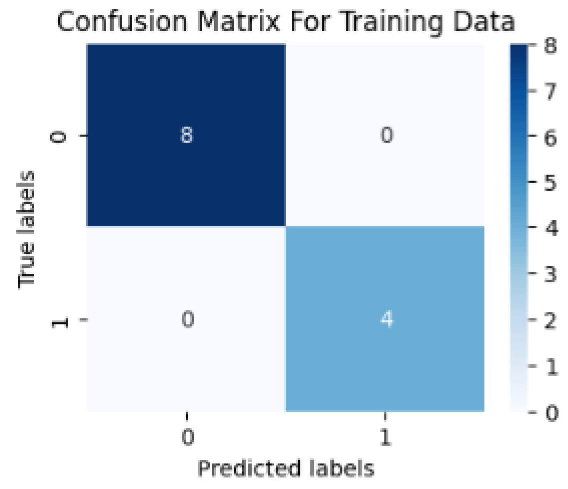
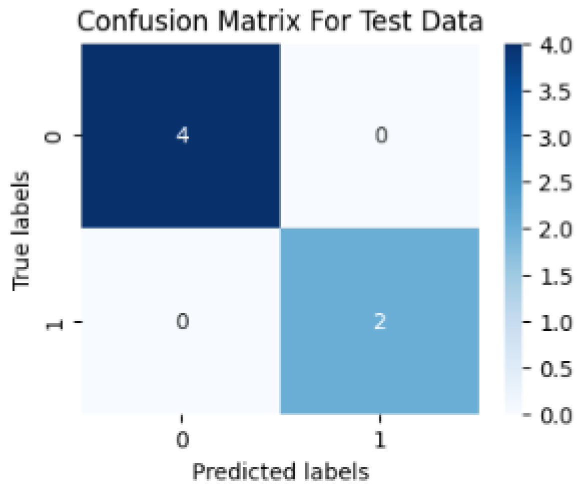
  That's because the default SVM uses the default kernel as rbf and we change section (b) and (c) the kernel to linear.

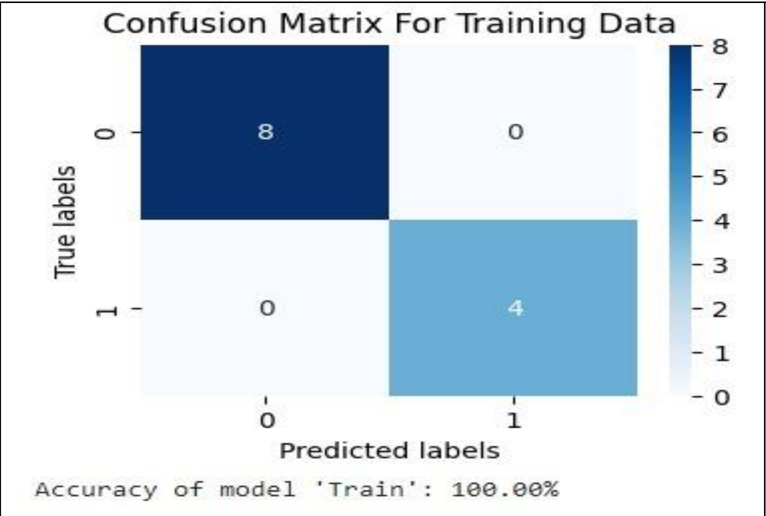
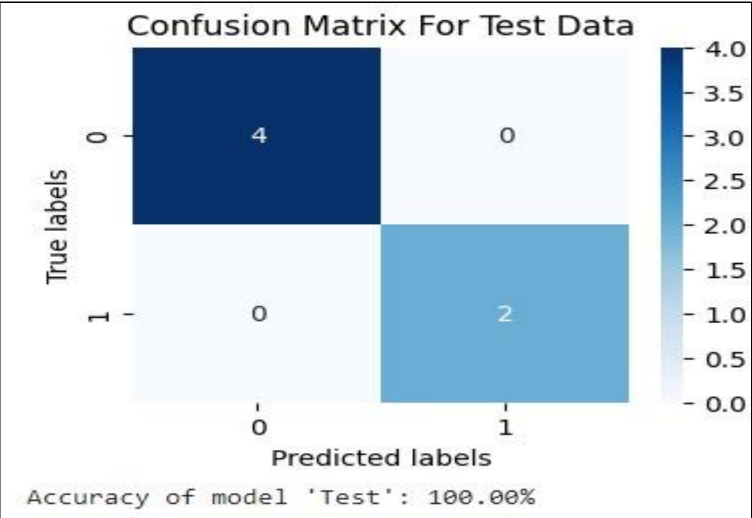
- ✓ **Refine the default SVM by selecting the appropriate parameter.**

  We refine the default by changing the c parameter to 5 instead of 1.

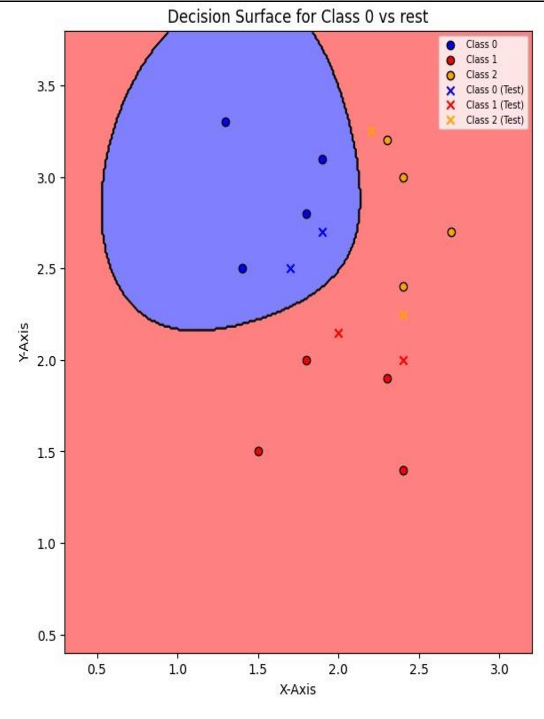
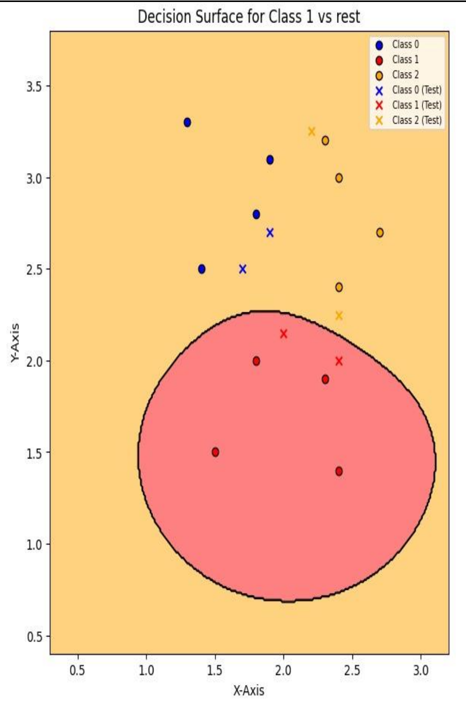
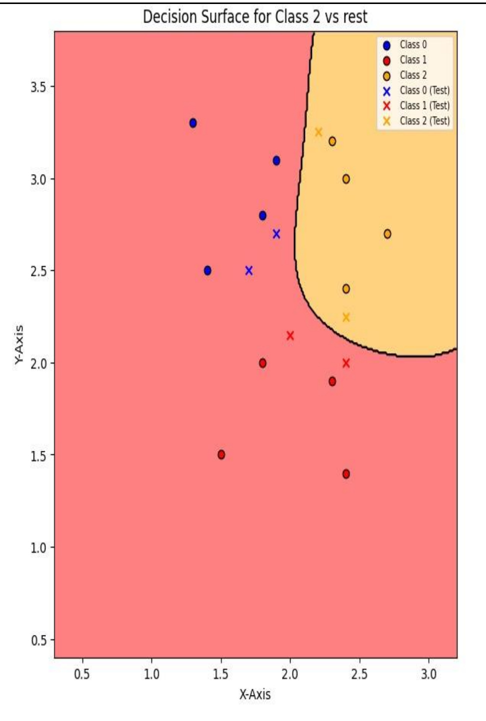
Training	**Testing**
 `Accuracy of model: 100.00%`	 `Accuracy of model: 100.00%`

✓ **Obtain confusion matrices for multi-class classification for SVM After refining the parameters.**

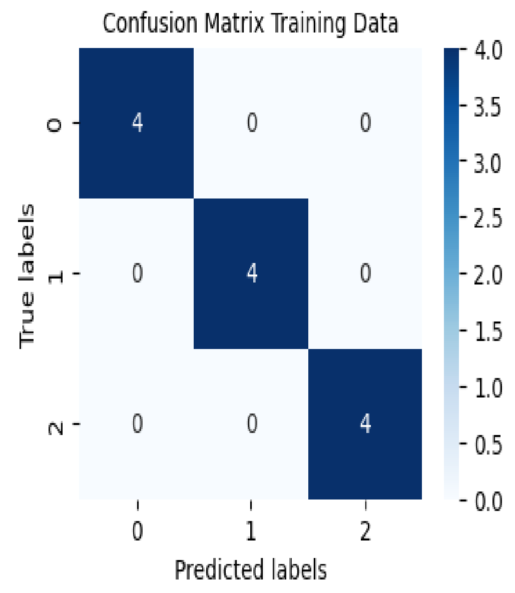
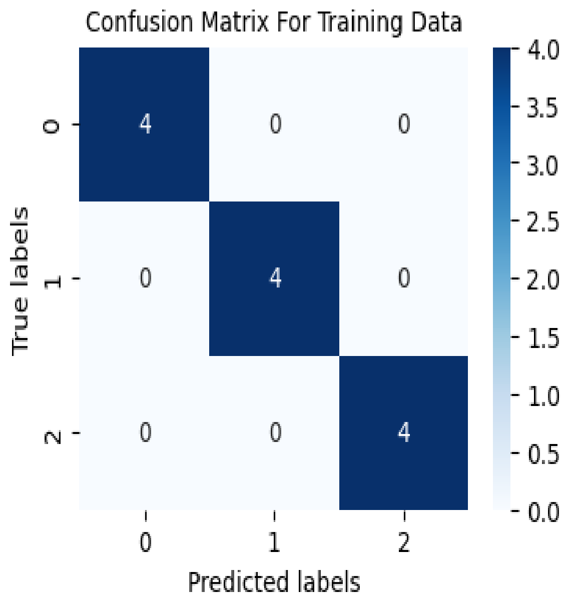
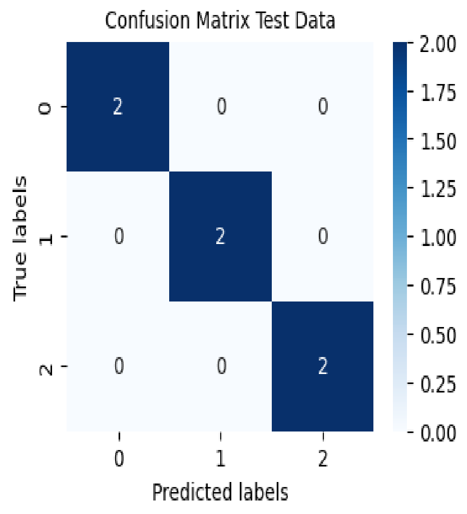
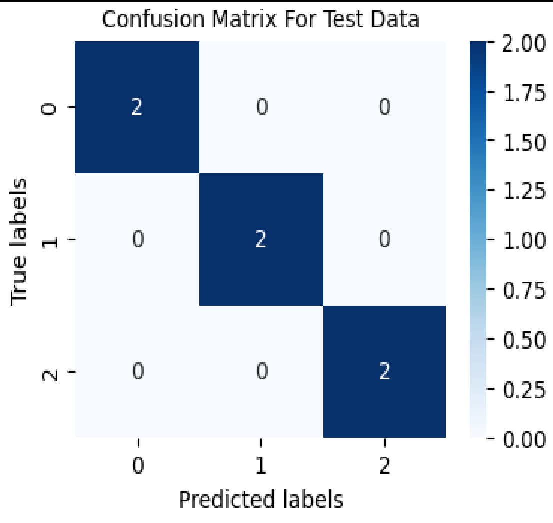
	SVM_Training	**SVM_Testing**
Class **0** vs Rest	 `Accuracy of model 'Train': 100.00%`	 `Accuracy of model 'Test': 100.00%`
Class **1** vs Rest	 `Accuracy of model 'Train': 100.00%`	 `Accuracy of model 'Test': 100.00%`

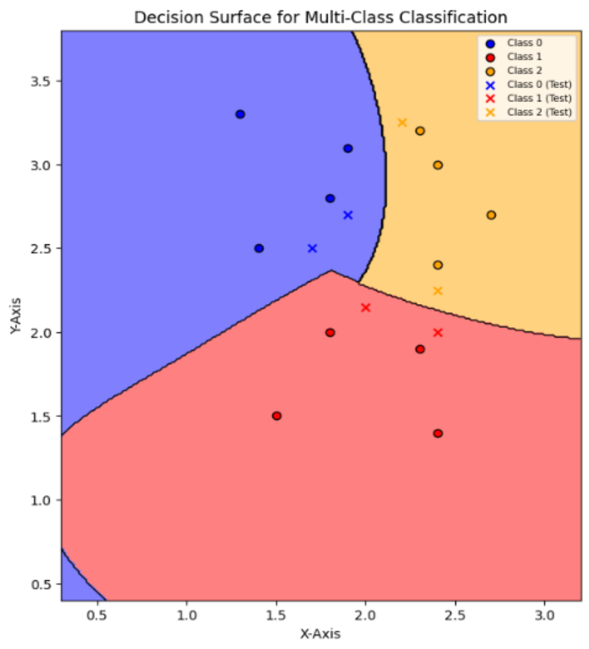
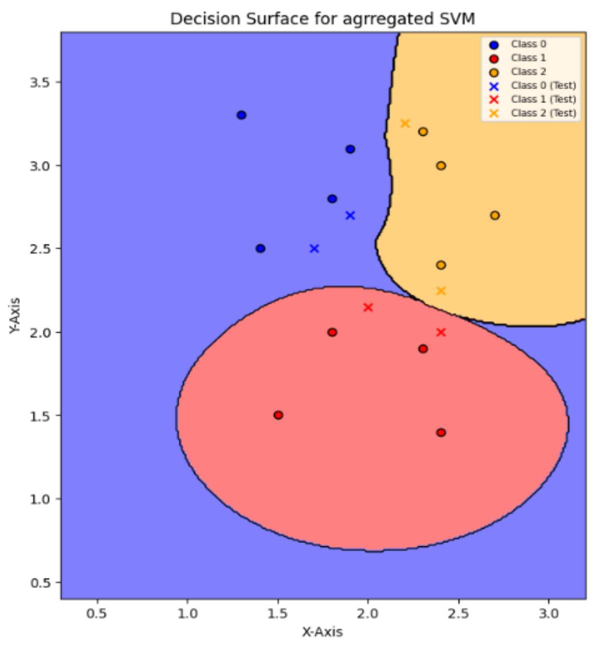
Class **2** vs Rest	 `Accuracy of model 'Train': 100.00%`	 `Accuracy of model 'Test': 100.00%`

- ✓ **Obtain decision surfaces for multi-class classification for SVM After refining the parameters.**

Class_0 after refining	Class_1 after refining	Class_2 after refining
		

- ✓ **Compare results with the default SVM and discuss the impact of parameter selection.**

	SVM_Default After refining	**SVM_After Aggregation After refining**
Training	 `Accuracy of model: 100.00%`	 `Accuracy of model: 1.0%`
Testing	 `Accuracy of model: 100.00%`	 `Accuracy of model: 1.0%`

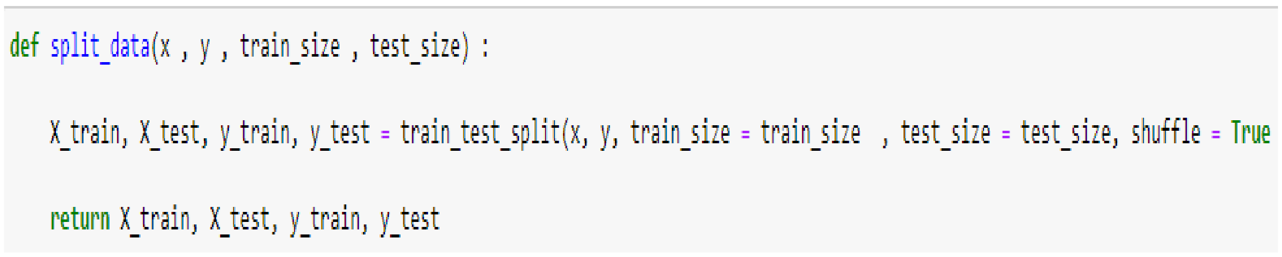
Decision Boundary		

Problem_2

a)

- ✓ **shuffle the dataset and split the dataset into a training set with 1000 samples and a validation set with 300 samples and a testing set with 428 samples.**

- ➢ We coded a function that takes the X and Y and the train and test needed sizes and shuffles the data then split depend on the train and test sizes returns X, Y as train and test sets We used this function to split our data into 1000 row for the training set, 300 rows for the validation set, and 428 rows for the test set

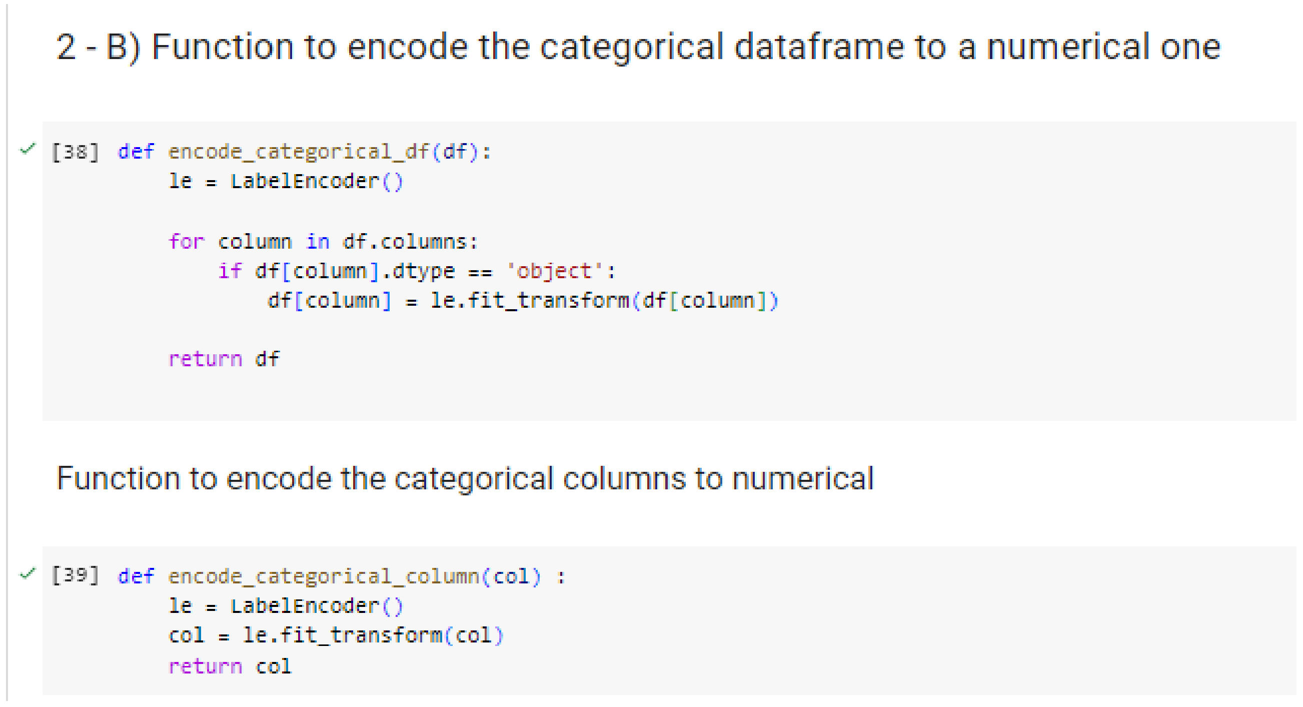


```
def split_data(x , y , train_size , test_size) :

    X_train, X_test, y_train, y_test = train_test_split(x, y, train_size = train_size  , test_size = test_size, shuffle = True

    return X_train, X_test, y_train, y_test
```

b)

- ✓ **transform the string values into numbers.**

  - ➢ Then we create 2 functions, the first **encode_categorical_df** for encoding the whole columns in the data frame that go through it and we used this to encode all the X sets, the second **encode_categorical_column** for encoding a particular column like the Y sets.



2 - B) Function to encode the categorical dataframe to a numerical one

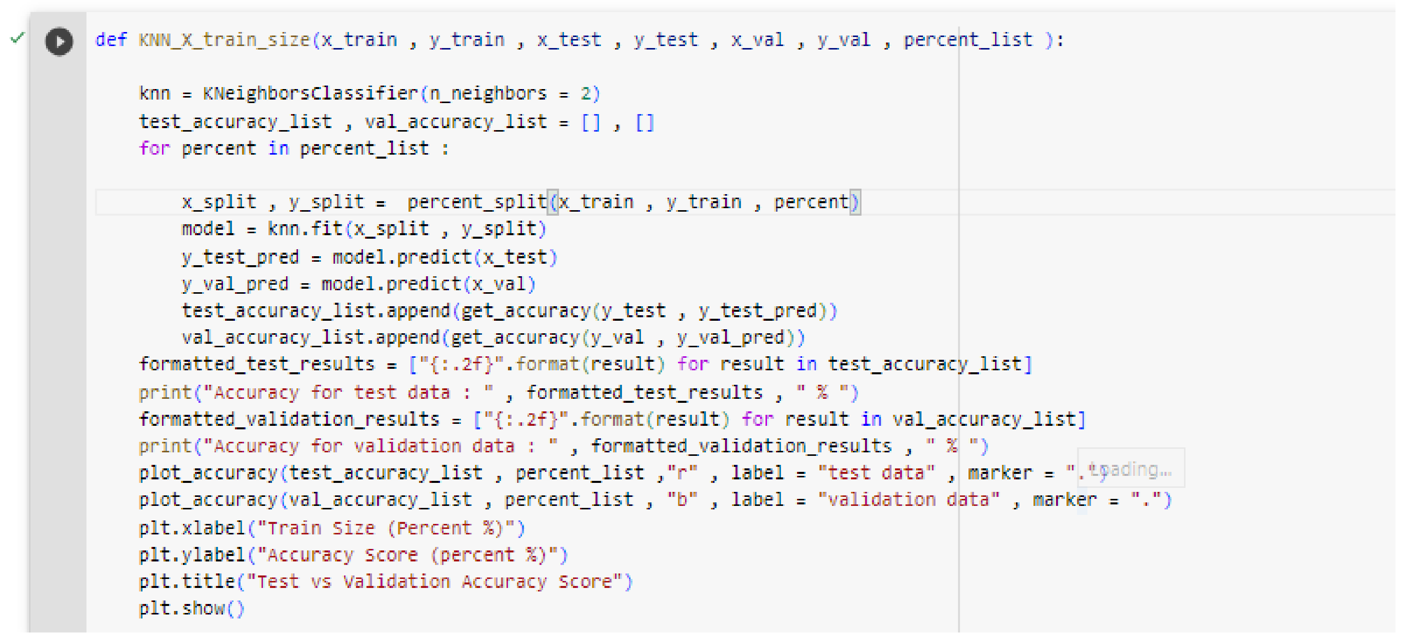
```
[38] def encode_categorical_df(df):
         le = LabelEncoder()

         for column in df.columns:
             if df[column].dtype == 'object':
                 df[column] = le.fit_transform(df[column])

         return df
```

Function to encode the categorical columns to numerical

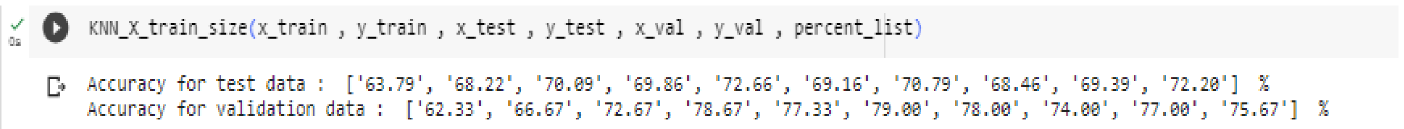
```
[39] def encode_categorical_column(col) :
         le = LabelEncoder()
         col = le.fit_transform(col)
         return col
```

c)

- ➢ In this point we first instantiate a KNN classifier with K = 2 and for a given set of train size from 10 to 100 percent we split our data depends on this percent by a function then train the model on this portion then get the validation and test predictions and print the accuracy for each portion of the train data then we plot the validation and testing accuracy of each portion vs the train data percent.

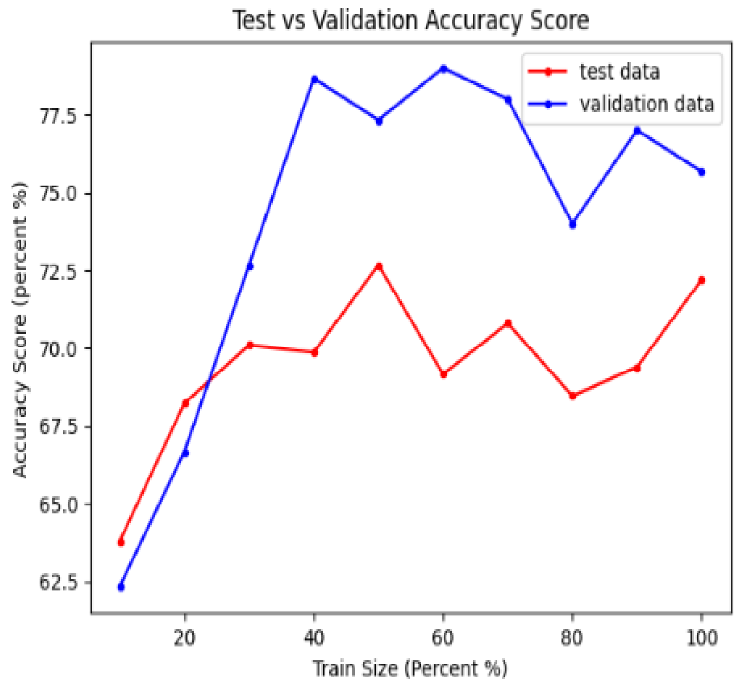


```
def KNN_X_train_size(x_train , y_train , x_test , y_test , x_val , y_val , percent_list ):

    knn = KNeighborsClassifier(n_neighbors = 2)
    test_accuracy_list , val_accuracy_list = [] , []
    for percent in percent_list :

        x_split , y_split =  percent_split(x_train , y_train , percent)
        model = knn.fit(x_split , y_split)
        y_test_pred = model.predict(x_test)
        y_val_pred = model.predict(x_val)
        test_accuracy_list.append(get_accuracy(y_test , y_test_pred))
        val_accuracy_list.append(get_accuracy(y_val , y_val_pred))
    formatted_test_results = ["{:.2f}".format(result) for result in test_accuracy_list]
    print("Accuracy for test data : " , formatted_test_results , " % ")
    formatted_validation_results = ["{:.2f}".format(result) for result in val_accuracy_list]
    print("Accuracy for validation data : " , formatted_validation_results , " % ")
    plot_accuracy(test_accuracy_list , percent_list ,"r" , label = "test data" , marker = ".")
    plot_accuracy(val_accuracy_list , percent_list , "b" , label = "validation data" , marker = ".")
    plt.xlabel("Train Size (Percent %)")
    plt.ylabel("Accuracy Score (percent %)")
    plt.title("Test vs Validation Accuracy Score")
    plt.show()
```



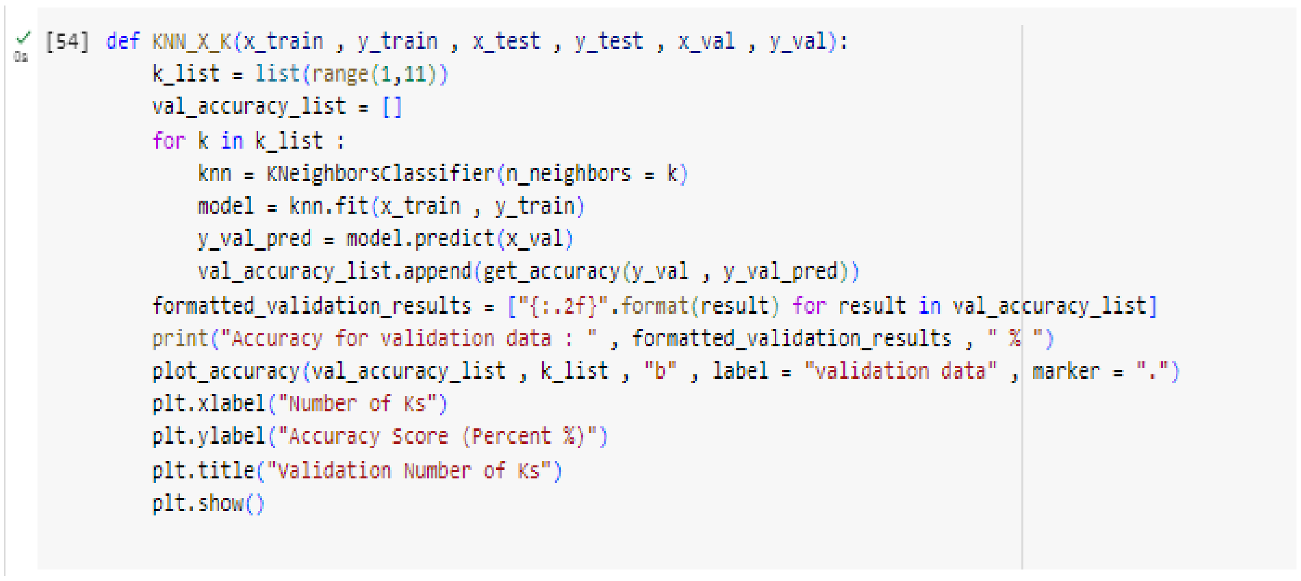
```
KNN_X_train_size(x_train , y_train , x_test , y_test , x_val , y_val , percent_list)
```

```
Accuracy for test data :  ['63.79', '68.22', '70.09', '69.86', '72.66', '69.16', '70.79', '68.46', '69.39', '72.20']  %
Accuracy for validation data :  ['62.33', '66.67', '72.67', '78.67', '77.33', '79.00', '78.00', '74.00', '77.00', '75.67']  %
```



d)

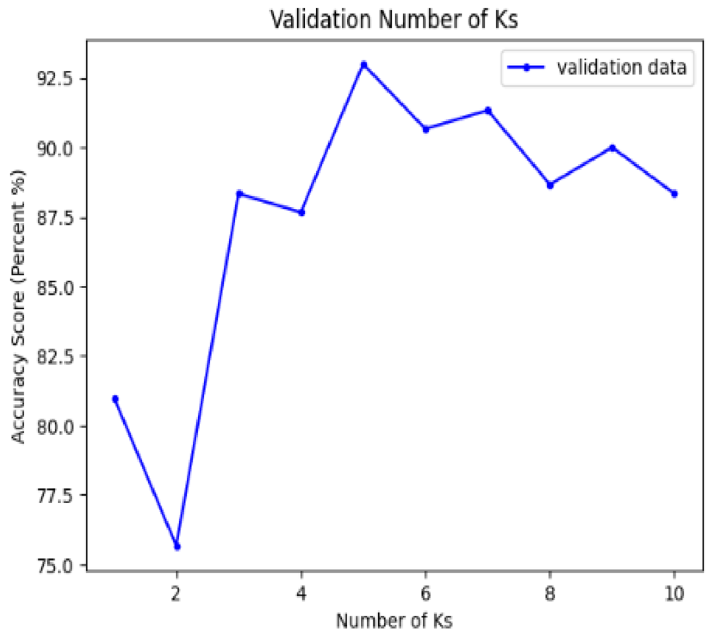
- ✓ **find the best K value, and show the accuracy curve on the validation set when K varies from 1 to 10.**



```
[54] def KNN_X_K(x_train , y_train , x_test , y_test , x_val , y_val):
         k_list = list(range(1,11))
         val_accuracy_list = []
         for k in k_list :
             knn = KNeighborsClassifier(n_neighbors = k)
             model = knn.fit(x_train , y_train)
             y_val_pred = model.predict(x_val)
             val_accuracy_list.append(get_accuracy(y_val , y_val_pred))
         formatted_validation_results = ["{:.2f}".format(result) for result in val_accuracy_list]
         print("Accuracy for validation data : " , formatted_validation_results , " % ")
         plot_accuracy(val_accuracy_list , k_list , "b" , label = "validation data" , marker = ".")
         plt.xlabel("Number of Ks")
         plt.ylabel("Accuracy Score (Percent %)")
         plt.title("Validation Number of Ks")
         plt.show()
```

- ➢ Another trial but this time we set the train data size fixed to 100 percent but we tried a different number of Ks from 1 to 10 and for each trial we get the accuracy for each K and we plotted the final result of validation and testing accuracy vs the numbers of Ks.

```
Accuracy for validation data :  ['81.00', '75.67', '88.33', '87.67', '93.00', '90.67', '91.33', '88.67', '90.00', '88.33']  %
```



e)

- According to the results we noticed that changing the number of Ks is more effective for both validation and testing accuracy than changing the train data size but in comparing the both functions accuracy alone we found that in changing the train data size there is a clear contrast between the accuracy and it reaches its best between 60 and 100 percent of the train data size and for the different numbers of Ks we found that in the validation data which consists of 300 rows the optimal K is 5 .