Group26_HW1

Group member:

Beshay, Sandy

Fana, Phoebe

Ali, Muhammad

Problem 1:

A)

- First, we import the important libraries we use.
- Then we used the training and testing data.
- Then we built a three function for drawing.

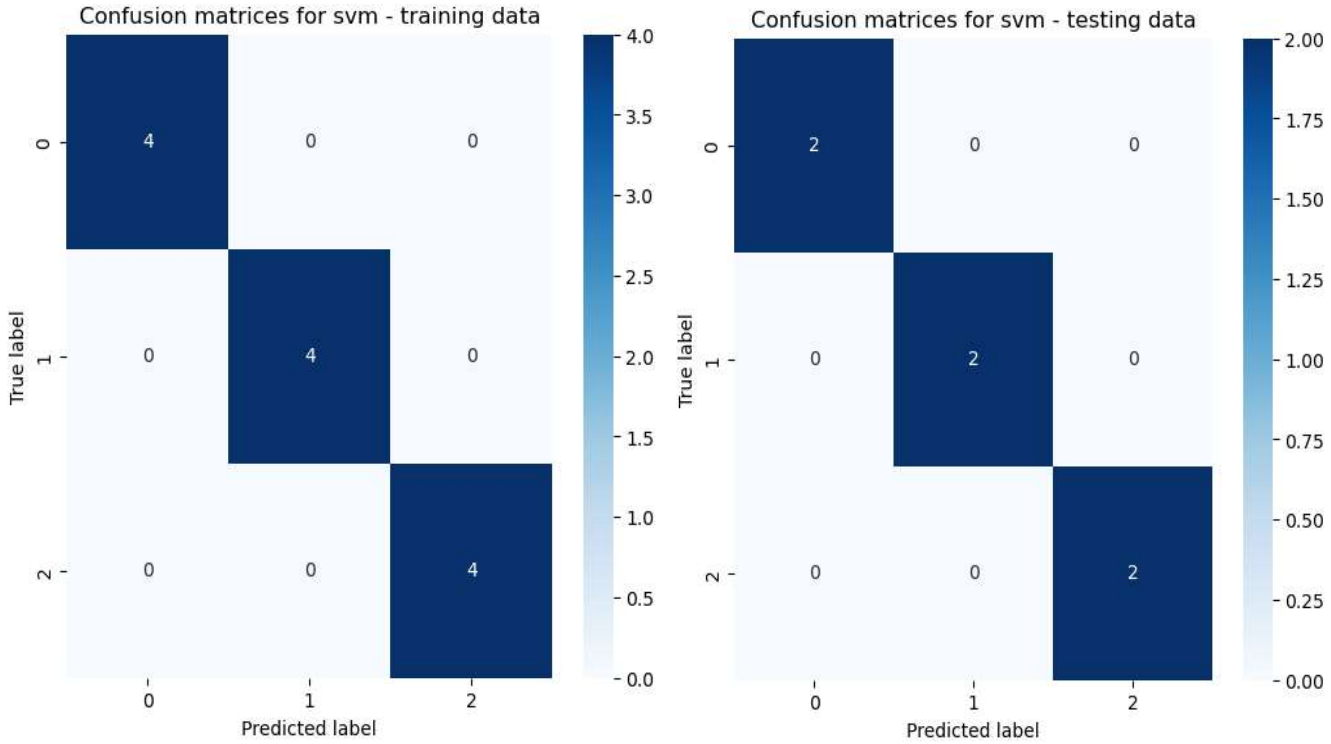
(Confusion matrices, decision surfaces, decision surfaces aggregated one vs rest)

```python
#Define a function to plot confusion matrices
def plot_confusion_matrix(confusion_matrix, title):
    plt.figure(figsize=(6, 6))
    sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")
    plt.title(title)
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.show()
```

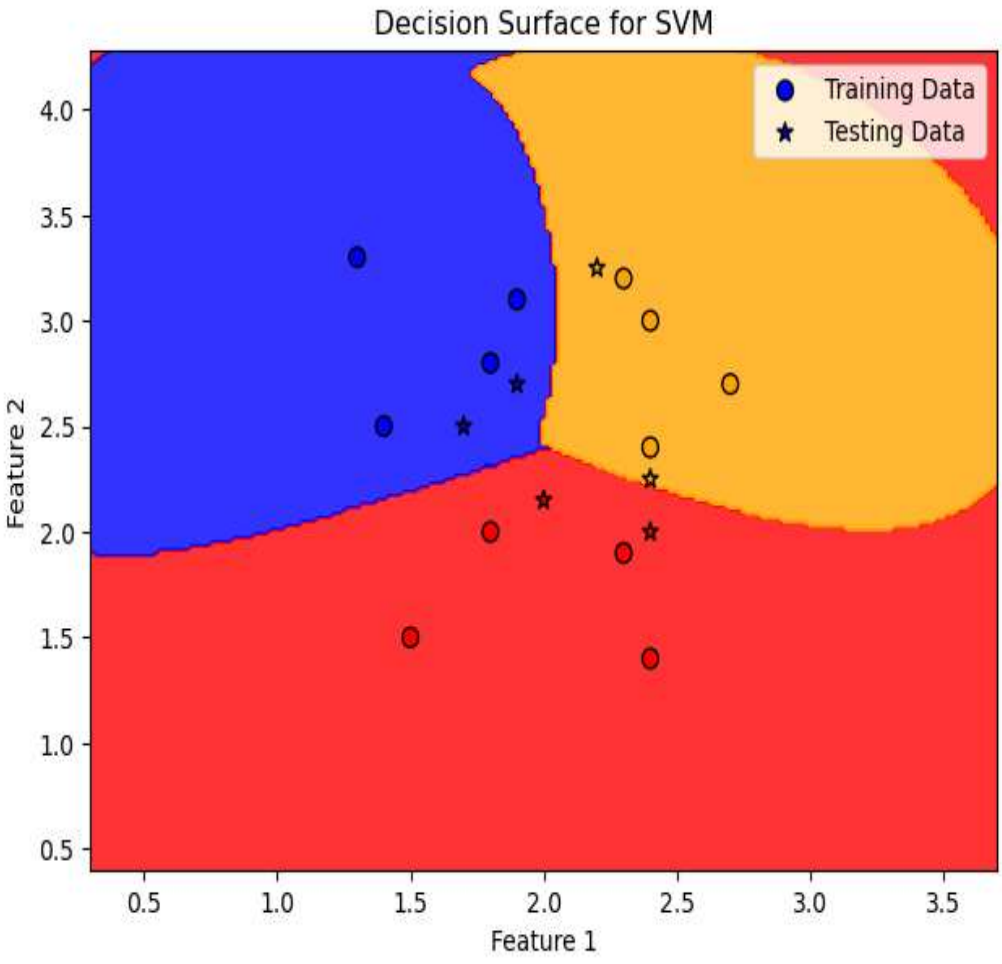
```python
#Define a function to plot decision surfaces
def plot_decision_surface(estimator, X_train, y_train, X_test, y_test,name):
    x_min, x_max = x_train[:, 0].min() - 1, x_train[:, 0].max() + 1
    y_min, y_max = x_train[:, 1].min() - 1, x_train[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))

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

    colors = ['blue', 'red', 'orange']
    color_map = ListedColormap(colors)
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=color_map)
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=color_map, edgecolor='k', s=50, marker='o', label='Training Data')
    plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=color_map, edgecolor='k', s=50, marker='*', label='Testing Data')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title(name)
    plt.show()
```

```python
#Define a function to plot decision surfaces for aggregted one vs rest
def plot_decision_surface_ag(models, X_train, y_train, X_test, y_test,name):
    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
    xx, yy = np.meshgrid(np.arange(x_min , x_max, 0.02), np.arange(y_min, y_max, 0.02))

    outputs = []
    for model in models:
        pred = model.predict(np.c_[xx.ravel(), yy.ravel()])
        outputs.append(pred)
    z_agregated = np.argmax(outputs, axis=0)
    Z = z_agregated.reshape(xx.shape)

    colors = ['blue', 'red', 'orange']
    color_map = ListedColormap(colors)
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=color_map)
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=color_map, edgecolor='k', s=50, marker='o', label='Training Data')
    plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=color_map, edgecolor='k', s=50, marker='*', label='Testing Data')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title(name)
    plt.show()
```

- Then we trained the SVM model, and the accuracy was 100%.
- Then we obtain confusion matrices for training and testing datasets.



- Then we visualize decision surfaces for multi-class classification.



B)

- We label training and testing data for one vs rest as binary classifier.

```
#label training and testing data for one vs rest as binary classifier
y_train_0 = np.where(y_train == 0, 1, 0)
y_train_1 = np.where(y_train == 1, 1, 0)
y_train_2 = np.where(y_train == 2, 1, 0)

y_test_0 = np.where(y_test == 0, 1, 0)
y_test_1 = np.where(y_test == 1, 1, 0)
y_test_2 = np.where(y_test == 2, 1, 0)
```

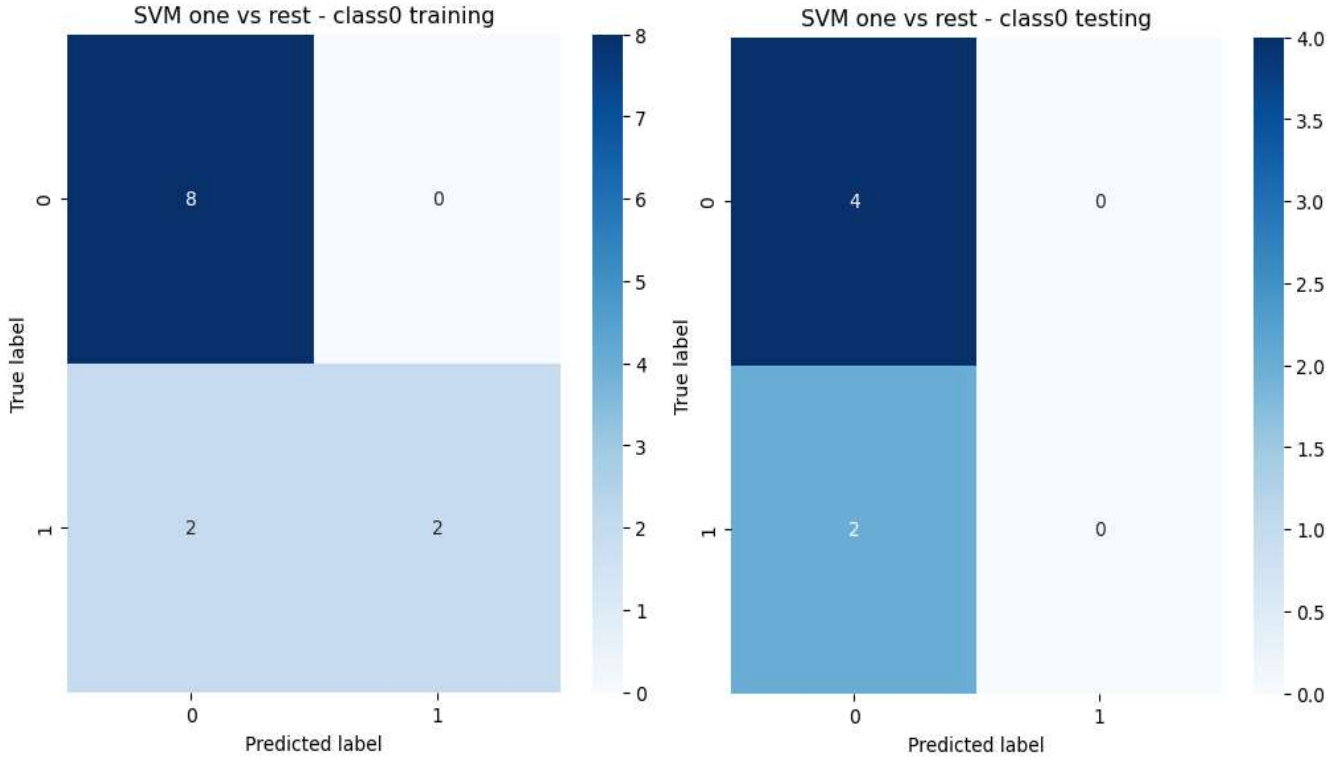
- Then we use the one-vs-rest strategy for SVM.

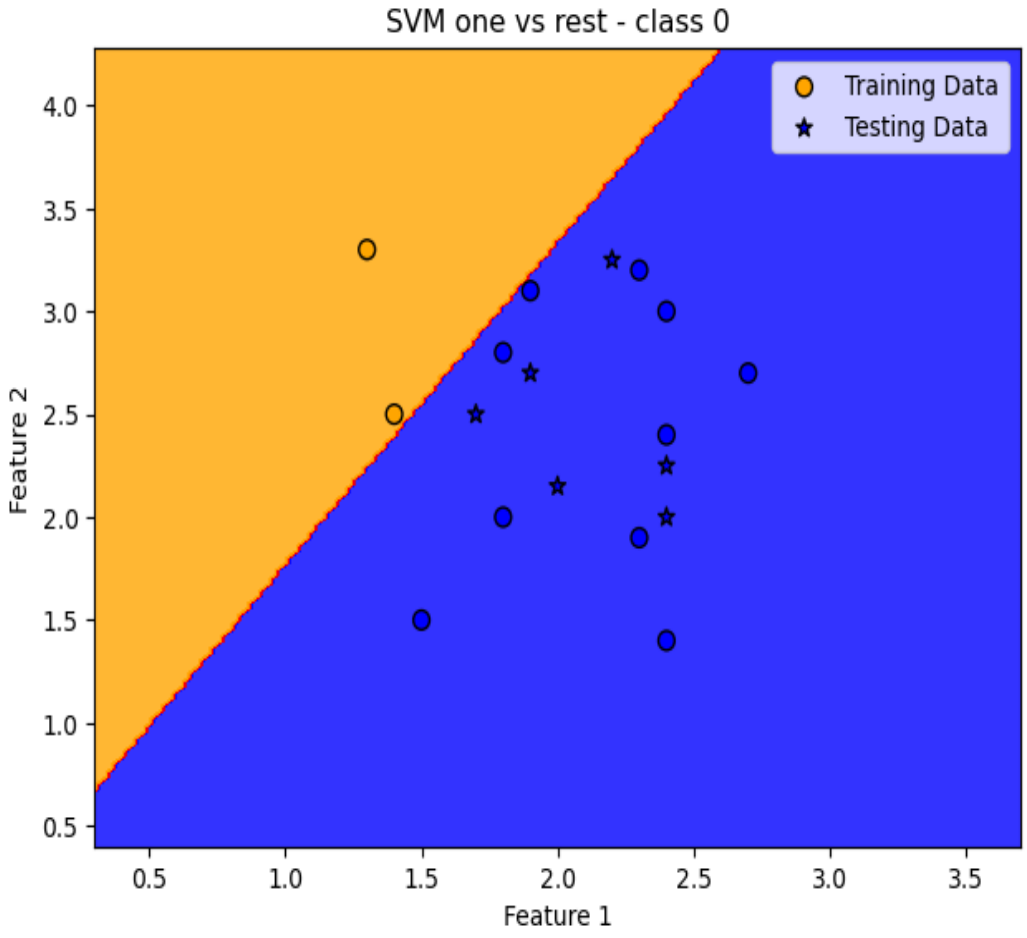
```
#one-vs-rest strategy for SVM
svm0 = SVC(kernel="linear")
svm1 = SVC(kernel="linear")
svm2 = SVC(kernel="linear")


svm0.fit(x_train, y_train_0)
sp_train0= svm0.predict(x_train)
sp_test0= svm0.predict(x_test)
sacc_train0= accuracy_score(y_train_0, sp_train0)
sacc_test0= accuracy_score(y_test_0, sp_test0)
scm_train0=confusion_matrix(y_train_0, sp_train0)
scm_test0=confusion_matrix(y_test_0, sp_test0)

svm1.fit(x_train, y_train_1)
sp_train1= svm1.predict(x_train)
sp_test1= svm1.predict(x_test)
sacc_train1= accuracy_score(y_train_1, sp_train1)
sacc_test1= accuracy_score(y_test_1, sp_test1)
scm_train1=confusion_matrix(y_train_1, sp_train1)
scm_test1=confusion_matrix(y_test_1, sp_test1)

svm2.fit(x_train, y_train_2)
sp_train2= svm2.predict(x_train)
sp_test2= svm2.predict(x_test)
sacc_train2= accuracy_score(y_train_2, sp_train2)
sacc_test2= accuracy_score(y_test_2, sp_test2)
scm_train2=confusion_matrix(y_train_2, sp_train2)
scm_test2=confusion_matrix(y_test_2, sp_test2)
```

- Then we get accuracy, Confusion matrices, decision surfaces for each class (0,1,2).

- The result of class0:

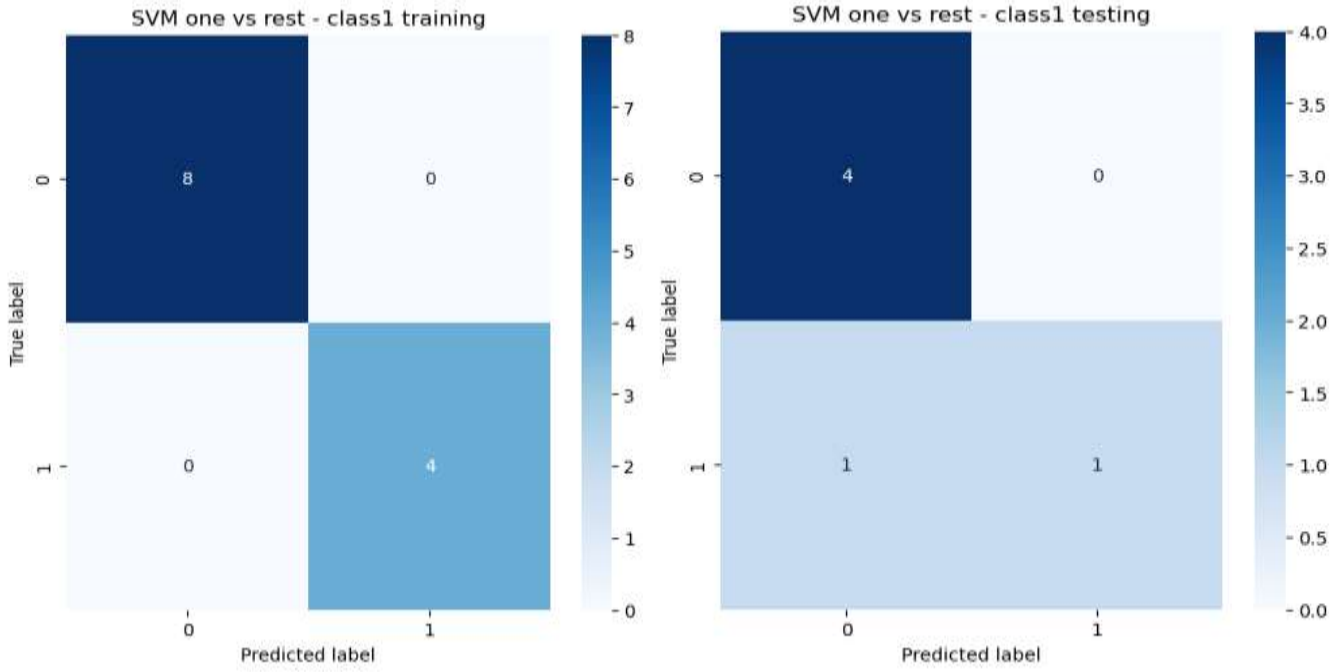
```
SVM one vs rest - class0 trainig accurcy :  83.33333333333334 %
SVM one vs rest - class0 testing accurcy :  66.66666666666666 %
```

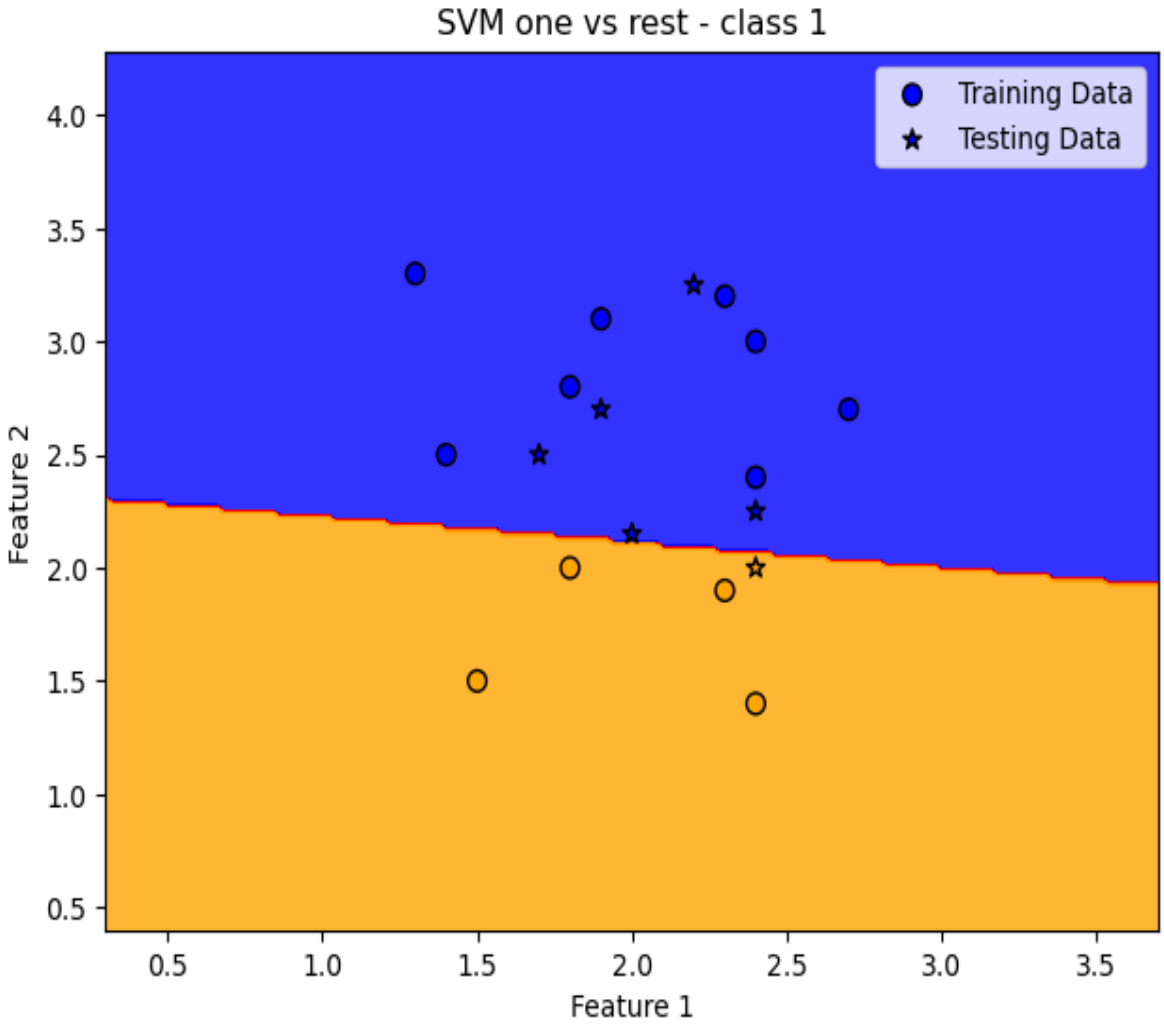




- The result of class1:

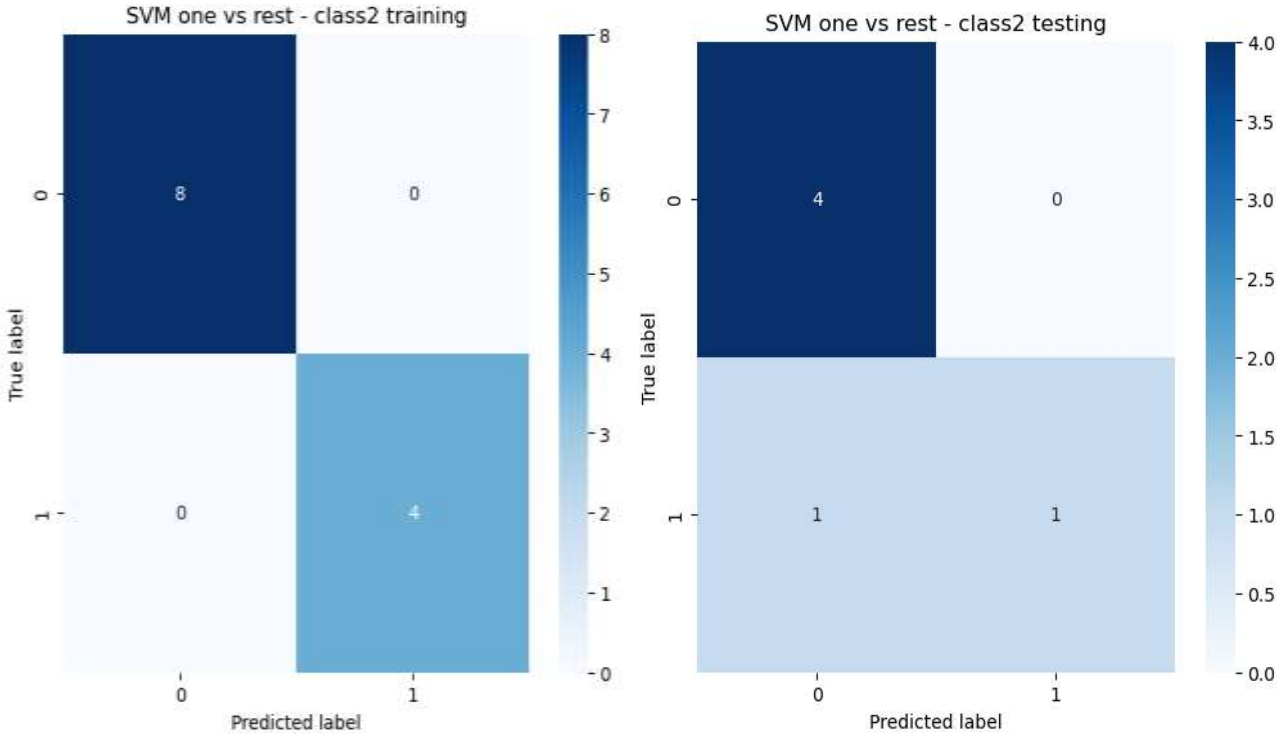
```
SVM one vs rest - class1 trainig accurcy :  100.0 %
SVM one vs rest - class1 testing accurcy :  83.33333333333334 %
```

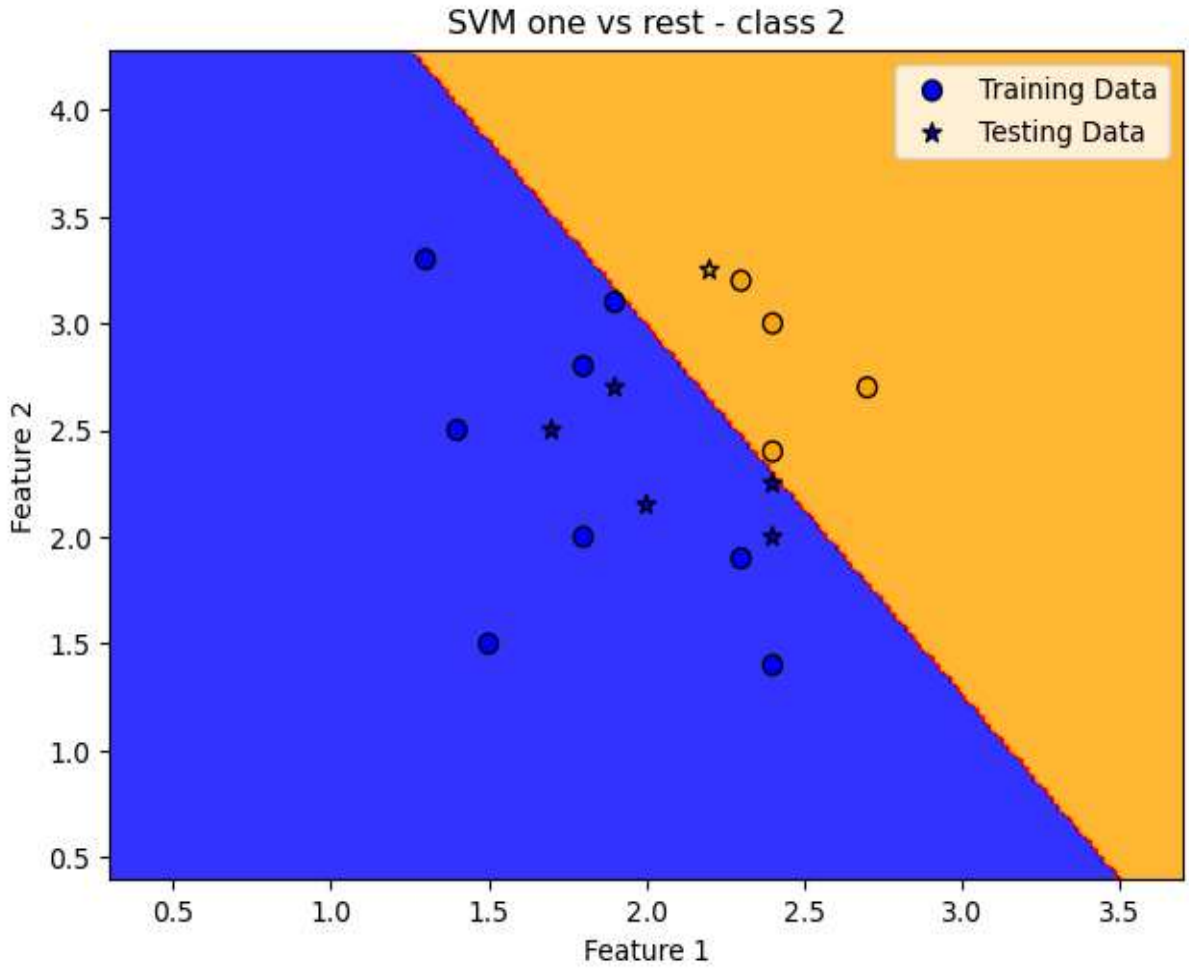




- The result of class2:

```
SVM one vs rest - class1 trainig accurcy :  100.0 %
SVM one vs rest - class1 testing accurcy :  83.33333333333334 %
```





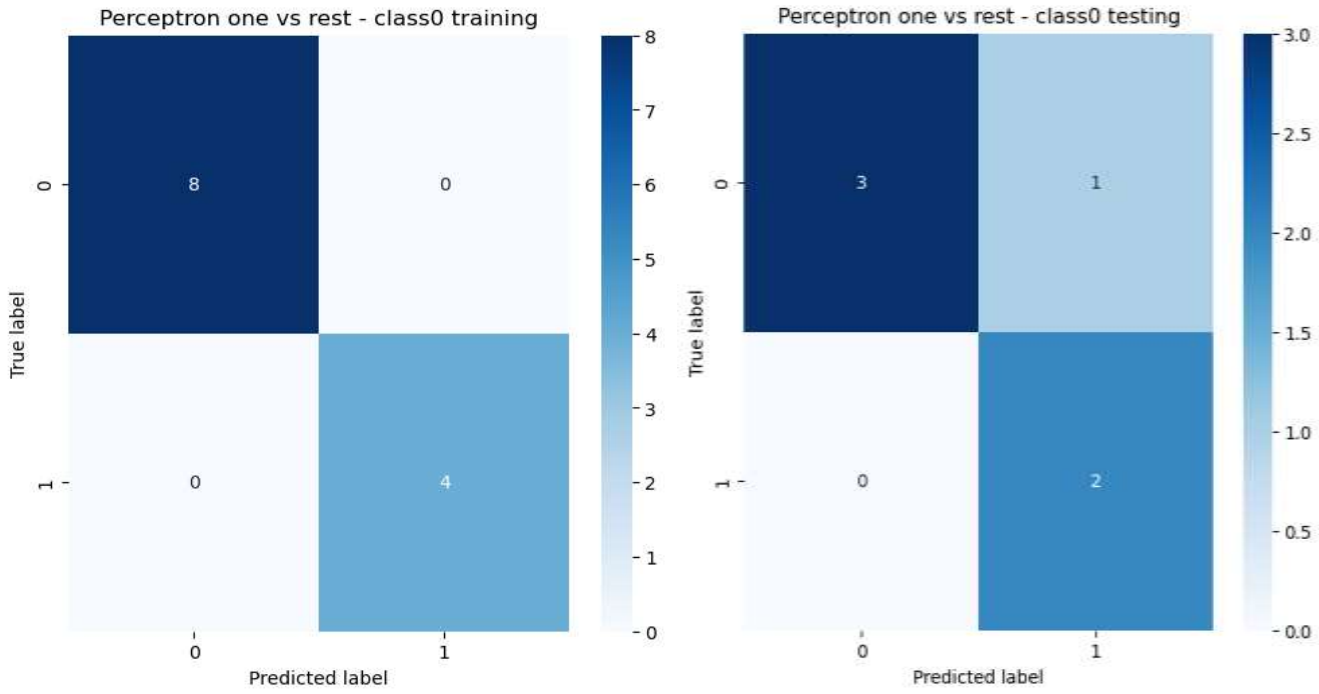
- Then we repeat all this steps with Perceptron.
- Then we use the one-vs-rest strategy for Perceptron.

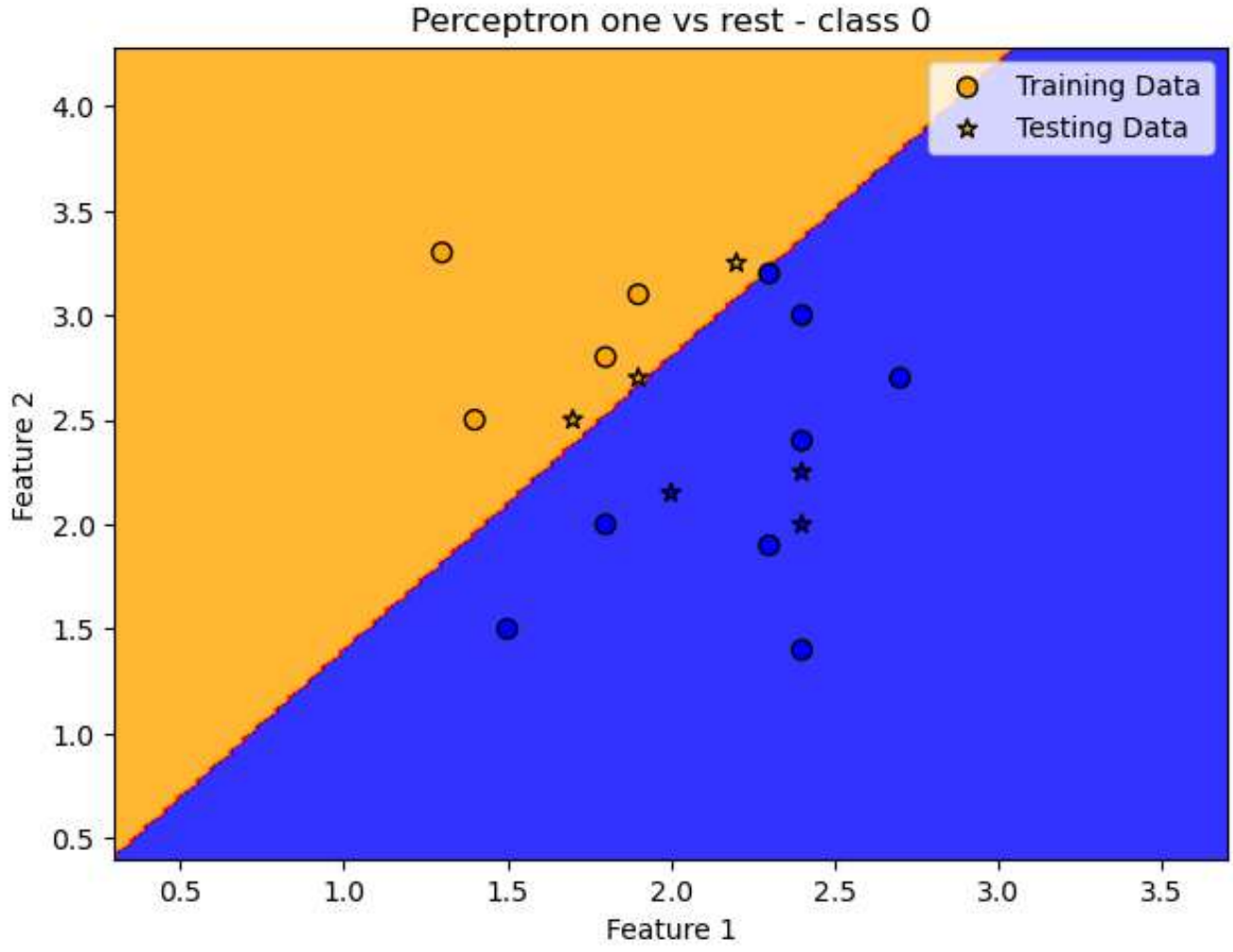
```
#one-vs-rest strategy for Perceptron
per0=Perceptron()
per1=Perceptron()
per2=Perceptron()

per0.fit(x_train, y_train_0)
pp_train0= per0.predict(x_train)
pp_test0= per0.predict(x_test)
pacc_train0= accuracy_score(y_train_0, pp_train0)
pacc_test0= accuracy_score(y_test_0, pp_test0)
pcm_train0=confusion_matrix(y_train_0, pp_train0)
pcm_test0=confusion_matrix(y_test_0, pp_test0)

per1.fit(x_train, y_train_1)
pp_train1= per1.predict(x_train)
pp_test1= per1.predict(x_test)
pacc_train1= accuracy_score(y_train_1, pp_train1)
pacc_test1= accuracy_score(y_test_1, pp_test1)
pcm_train1=confusion_matrix(y_train_1, pp_train1)
pcm_test1=confusion_matrix(y_test_1, pp_test1)

per2.fit(x_train, y_train_2)
pp_train2= per2.predict(x_train)
pp_test2= per2.predict(x_test)
pacc_train2= accuracy_score(y_train_2, pp_train2)
pacc_test2= accuracy_score(y_test_2, pp_test2)
pcm_train2=confusion_matrix(y_train_2, pp_train2)
pcm_test2=confusion_matrix(y_test_2, pp_test2)
```

- Then we get accuracy, Confusion matrices, decision surfaces for each class (0,1,2)

- The result of class0:

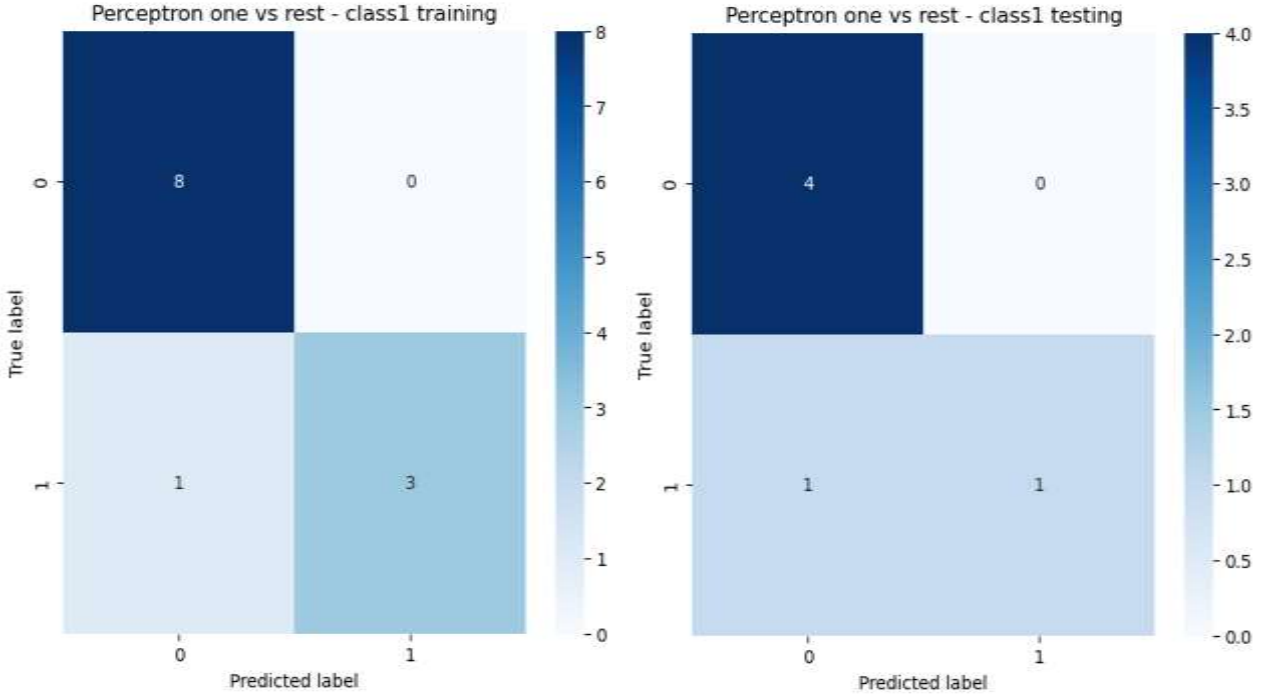
```
Perceptron one vs rest - class0 trainig accurcy :  100.0 %
Perceptron one vs rest - class0 testing accurcy :  83.33333333333334 %
```

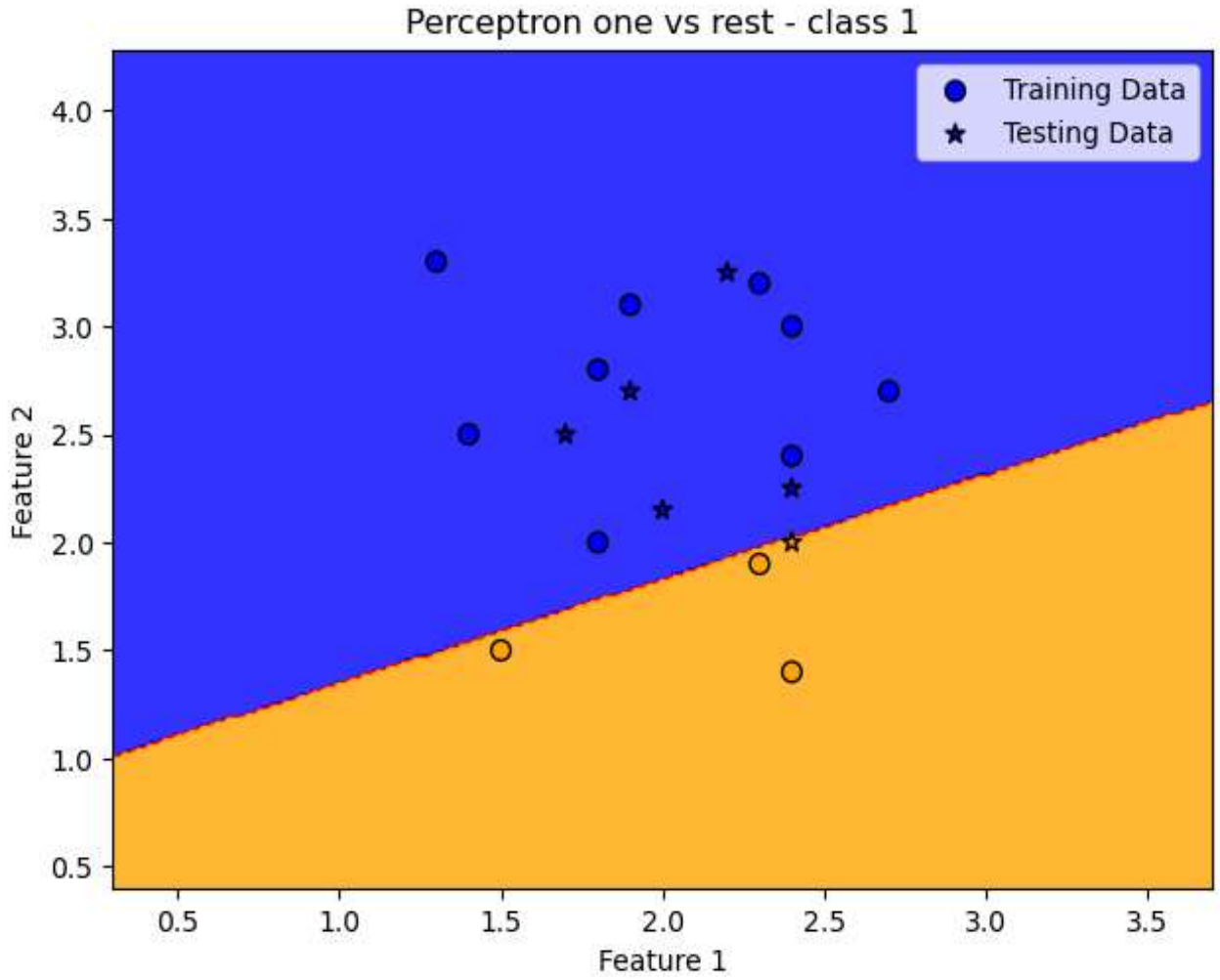




- The result of class1:

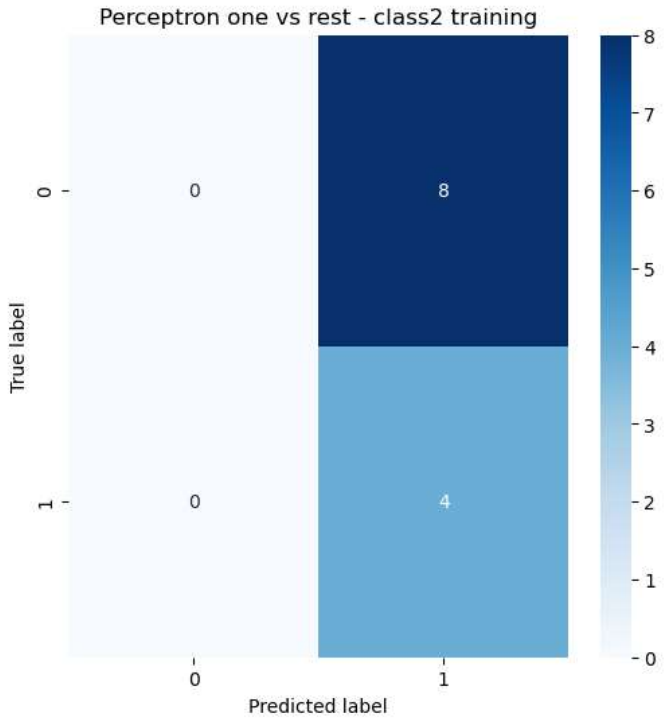
```
Perceptron one vs rest - class1 trainig accurcy :  91.66666666666666 %
Perceptron one vs rest - class1 testing accurcy :  83.33333333333334 %
```

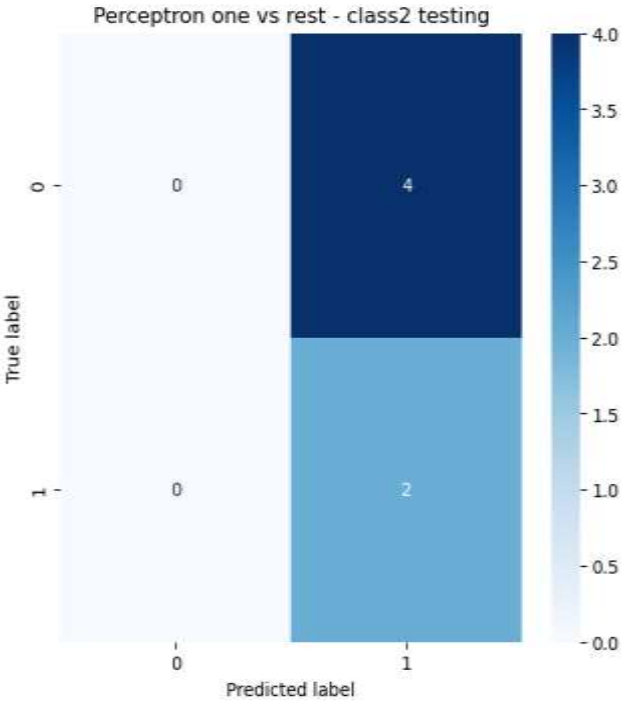


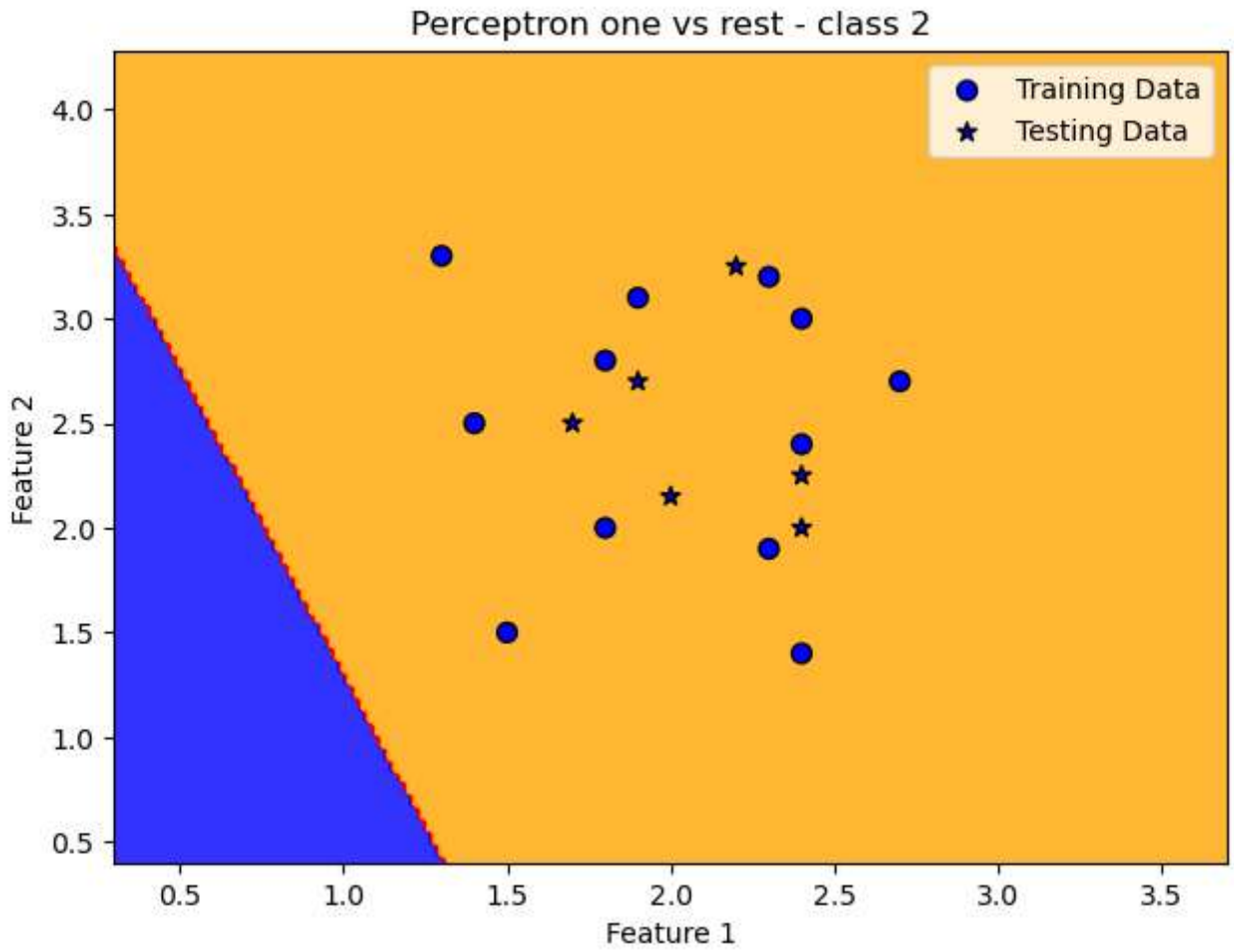


- The result of class2:

```
Perceptron one vs rest - class2 trainig accurcy :  33.33333333333333 %
Perceptron one vs rest - class2 testing accurcy :  33.33333333333333 %
```



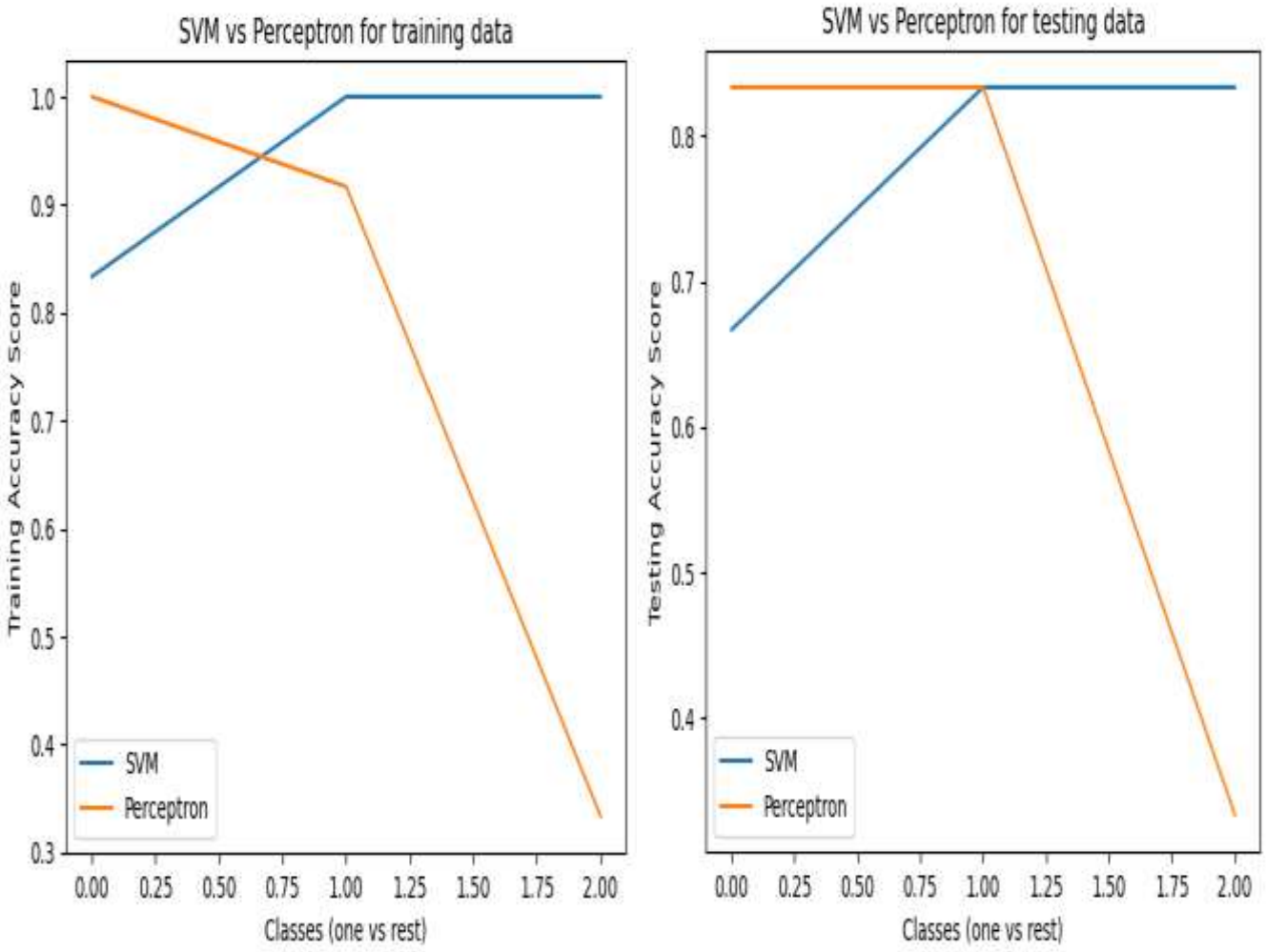




- Then we Compare and analyze SVM and Perceptron results:
  by plots that compare the accuracy between (SVM and Perceptron).

```
#Compare and analyze SVM and Perceptron results
classes=[0,1,2]
sacc_train_all=[sacc_train0,sacc_train1,sacc_train2]
sacc_test_all=[sacc_test0,sacc_test1,sacc_test2]
pacc_train_all=[pacc_train0,pacc_train1,pacc_train2]
pacc_test_all=[pacc_test0,pacc_test1,pacc_test2]

# Plot the accuracy scores for SVM and Perceptron - training data
plt.plot(classes,sacc_train_all, label='SVM')
plt.plot(classes,pacc_train_all, label='Perceptron')
plt.xlabel('Classes (one vs rest)')
plt.ylabel('Training Accuracy Score')
plt.title('SVM vs Perceptron for training data')
plt.legend()
plt.show()

# Plot the accuracy scores for SVM and perceptron - testing dataset
plt.plot(classes,sacc_test_all, label='SVM')
plt.plot(classes,pacc_test_all, label='Perceptron')
plt.xlabel('Classes (one vs rest)')
plt.ylabel('Testing Accuracy Score')
plt.title('SVM vs Perceptron for testing data')
plt.legend()
plt.show()
```

- We draw one plot for train dataset and one for test dataset accuracy prediction.



- The plots show that SVM has better results than Perceptron.

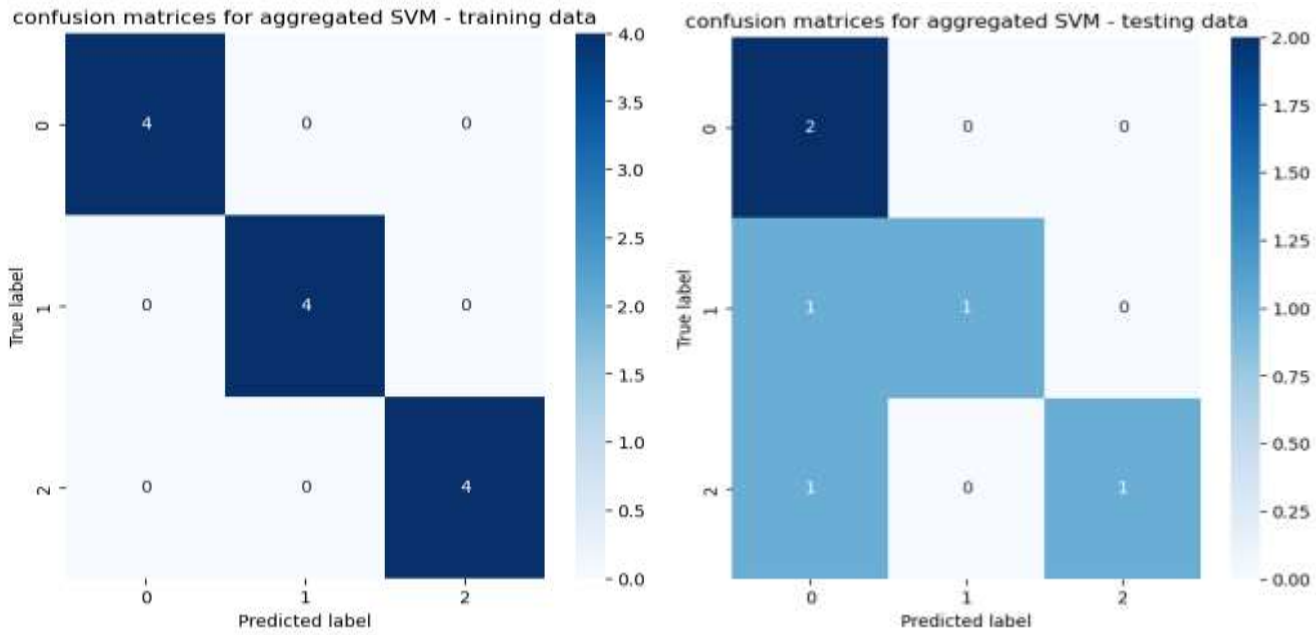
C)

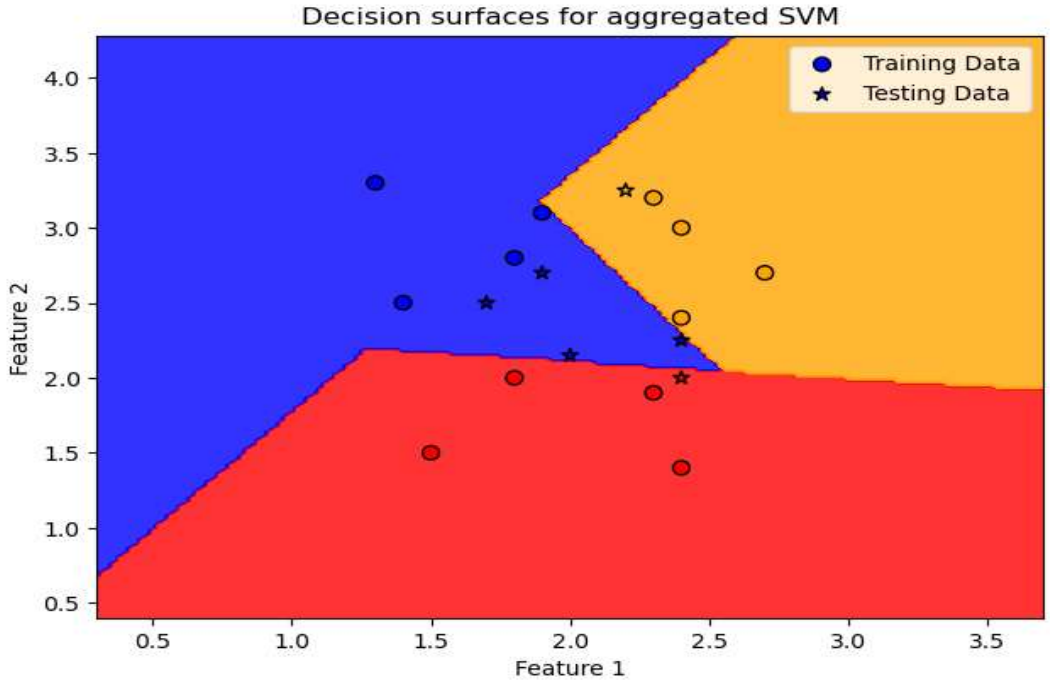
- Then we Aggregate results from the one-vs-rest strategy for SVM.

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

#aggregated train data
svm_train = [sp_train0, sp_train1, sp_train2]
svm_train_arg = np.argmax(svm_train, axis=0)
svm_train_all = svm_train_arg.tolist()

#aggregated test data
svm_test = [sp_test0, sp_test1, sp_test2]
svm_test_arg = np.argmax(svm_test, axis=0)
svm_test_all = svm_test_arg.tolist()
```

- Then we get accuracy, Confusion matrices, decision surface for aggregated svm classes.

```
aggregated SVM accurcy for training :  100.0 %
aggregated SVM accurcy for testing :  66.66666666666666 %
```



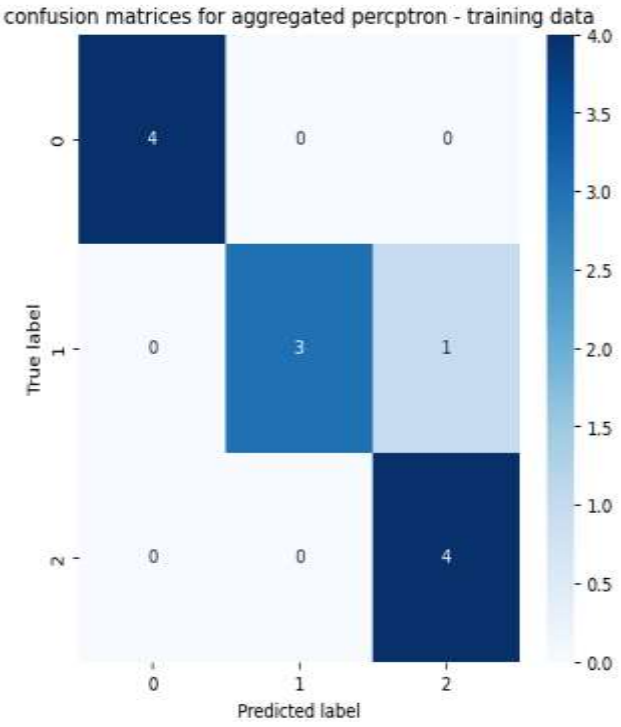


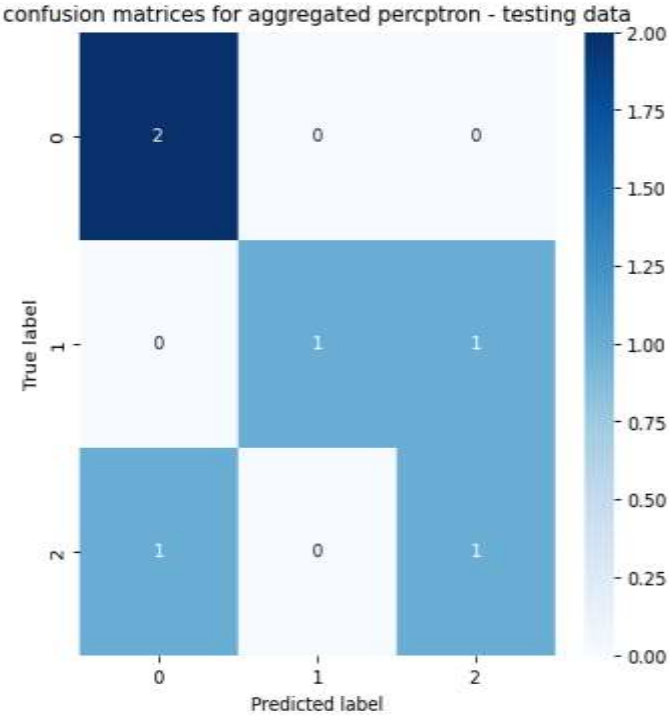
- Then we Aggregate results from the one-vs-rest strategy for perceptron.

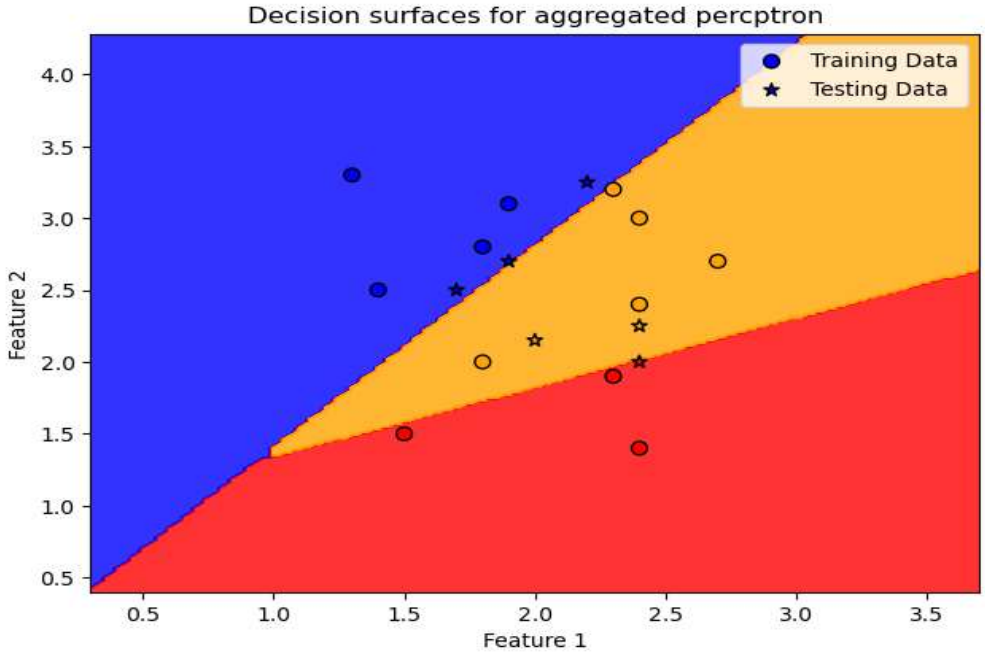
```
#Aggregate results from the one-vs-rest strategy for perceptron

#aggregated train data
per_train = [pp_train0, pp_train1, pp_train2]
per_train_arg = np.argmax(per_train, axis=0)
per_train_all = per_train_arg.tolist()

#aggregated test data
per_test = [pp_test0, pp_test1, pp_test2]
per_test_arg = np.argmax(per_test, axis=0)
per_test_all = per_test_arg.tolist()
```

- Then we get accuracy, Confusion matrices, decision surface for aggregated svm classes.

```
aggregated SVM accurcy for training :  91.66666666666666 %
aggregated SVM accurcy for testing :  66.66666666666666 %
```







- we compere the SVM aggregated classes and default SVM, then we discover that the accuracy is very close but default SVM is better than SVM aggregated classes , because default SVM accuracy is 100%.

D)

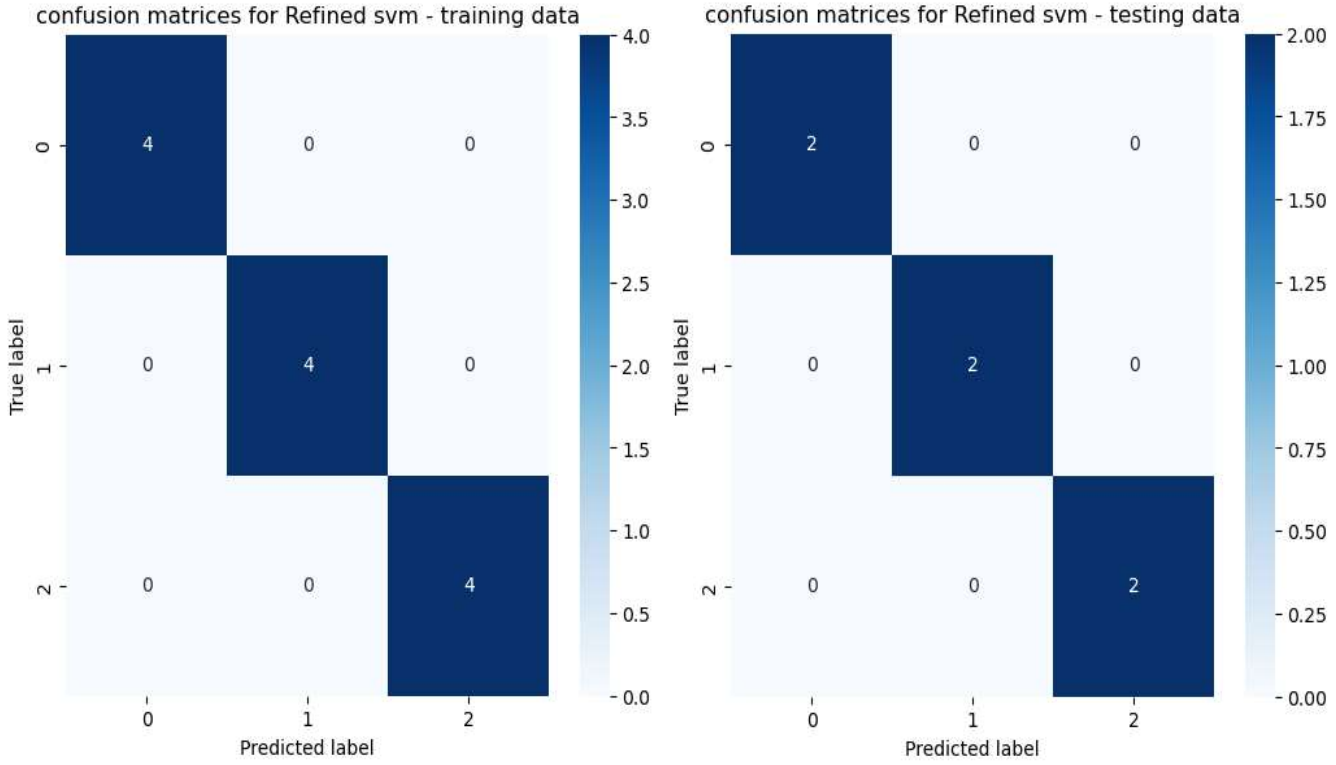
- The default model of SVM is a linear model, which means that it separates the data points by a straight line. This is a simple and efficient model, but it can be too simple for some data sets. In these cases, the aggregated performance of SVM can be better than the default model.
- Then we Refine the default SVM by selecting the appropriate parameter and train the model.

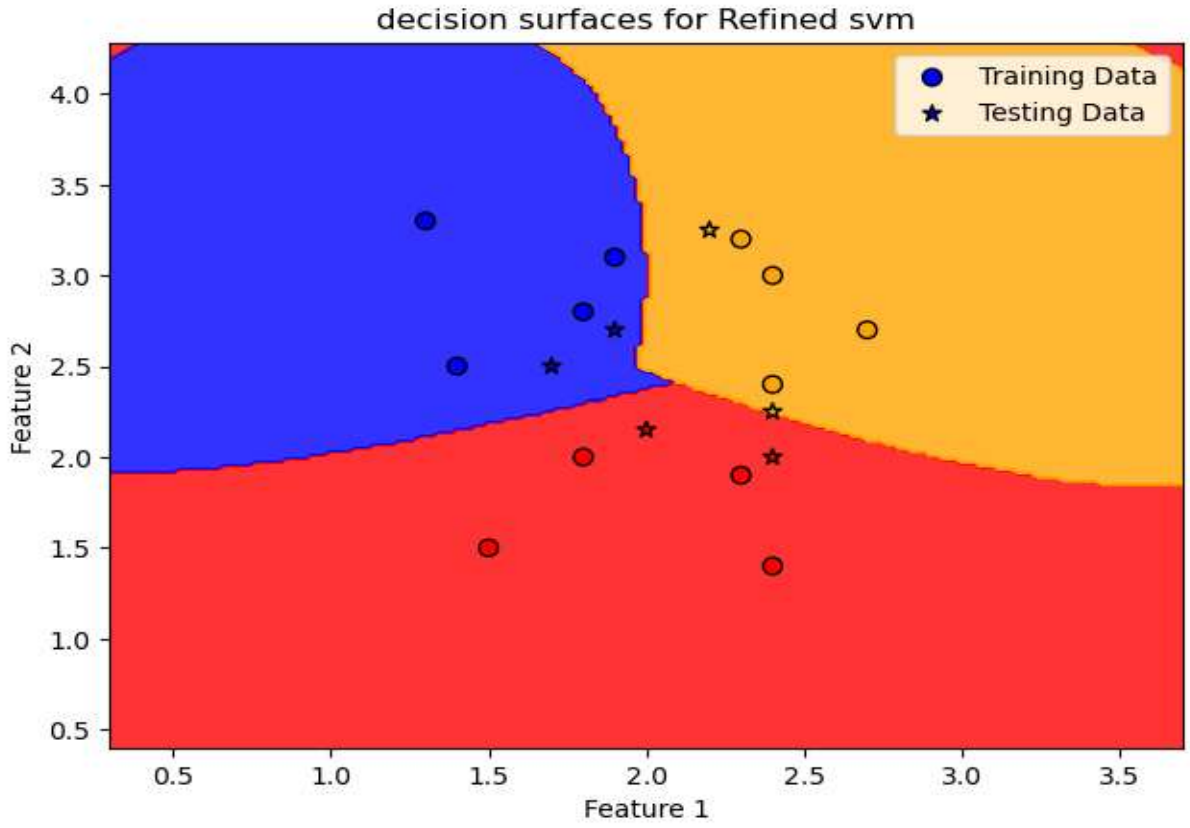
```
#Refine the default SVM by selecting the appropriate parameter

#Define SVM model with default hyperparameters
svm_svm = SVC(kernel='rbf')

#Define grid of hyperparameters to search over
param_grid = {'C': [0.1, 1, 10], 'gamma': [0.01, 0.1, 1]}

#Use GridSearchCV to find best hyperparameters
svm_cv = GridSearchCV(svm_svm, param_grid, cv=3)
svm_cv.fit(x_train, y_train)

#Get best hyperparameters and performance on test set
best_params = svm_cv.best_params_
r_pred_train = svm_cv.predict(x_train)
r_pred_test = svm_cv.predict(x_test)
raccuracy_train = accuracy_score(y_train, r_pred_train)
raccuracy_test = accuracy_score(y_test, r_pred_test)

print("Best hyperparameters:", best_params)
```

```
Best hyperparameters: {'C': 1, 'gamma': 1}
```

- Then we get accuracy, Confusion matrices, decision surface for refined SVM.

```
Accuracy on train set for refined SVM: 100.0 %
Accuracy on test set for refined SVM: 100.0 %
```





- We compere the refined SVM and default SVM, then we discover that the two model have the same accuracy , but the decision surface for refined SVM is more accurate .

Problem 2:

A)

- First, we import the important libraries we use.
- Then we load the dataset from a CSV file and define the column names.

```
# Load the dataset from a CSV file
cars_data = pd.read_csv('car_evaluation.csv',header=None)

# Define the column names
column_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'target']
cars_data.columns = column_names

cars_data.head()
```

	buying	maint	doors	persons	lug_boot	safety	target
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

- Then we shuffle and split the dataset into a training set with 1000 samples, a validation set with 300 samples, and a testing set with 428 samples.

```
#create x,y
x=cars_data.drop('target', axis=1)
y=cars_data['target']

# Split the dataset into a training set with 1000 samples, a validation set with 300 samples, and a testing set with 428 samples
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=428, random_state=42,shuffle=True)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, train_size=1000, random_state=42)
```

B)

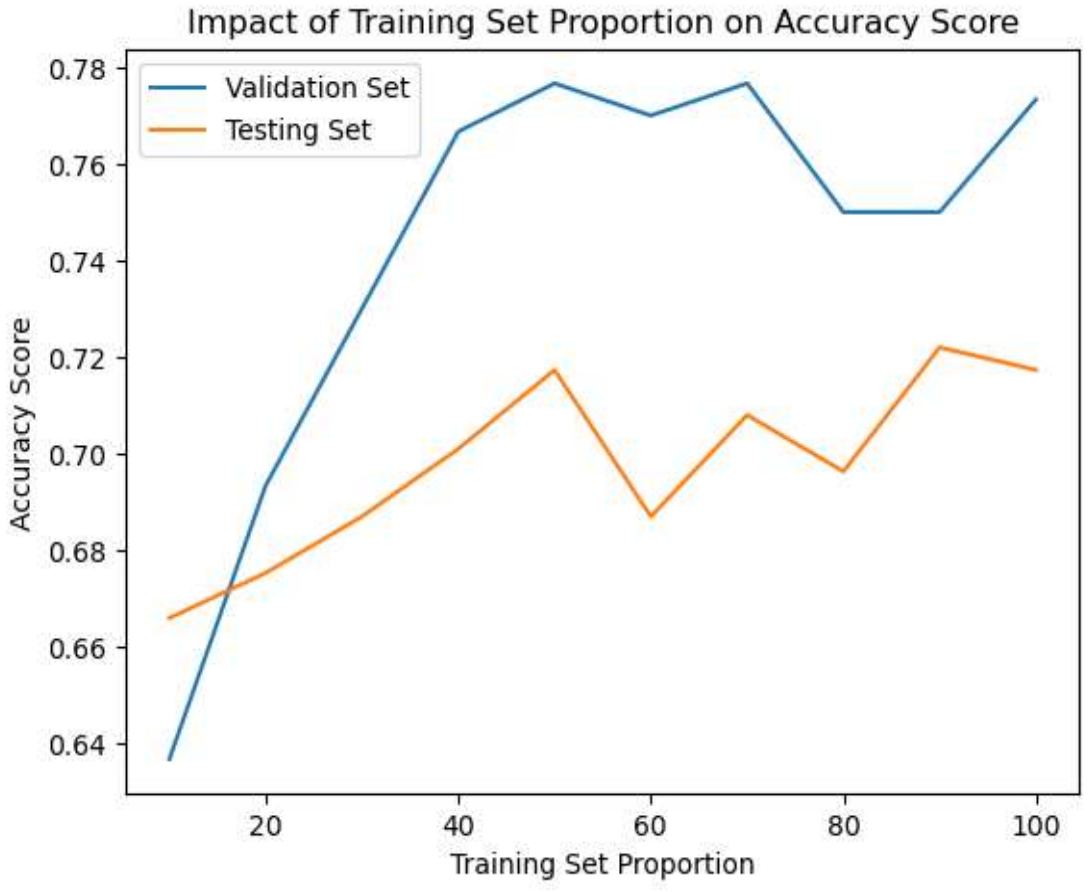
- Then we change string values into numeric by Encoder after divide the data.

```
#change string values into numeric by Encoder
le=LabelEncoder()
y_train=le.fit_transform(y_train)
y_test=le.transform(y_test)
y_val=le.fit_transform(y_val)

oe=OrdinalEncoder()
x_train=oe.fit_transform(x_train)
x_test=oe.transform(x_test)
x_val=oe.fit_transform(x_val)
```

C)

- Then we use different number of training samples to show the impact of number of training samples and Use 10%, 20%, 30%, 40%, 50%,60%, 70%, 80%, 90% and 100% of the training set for 10 separate KNN classifiers and show their performance (accuracy score) on the validation set and testing set.

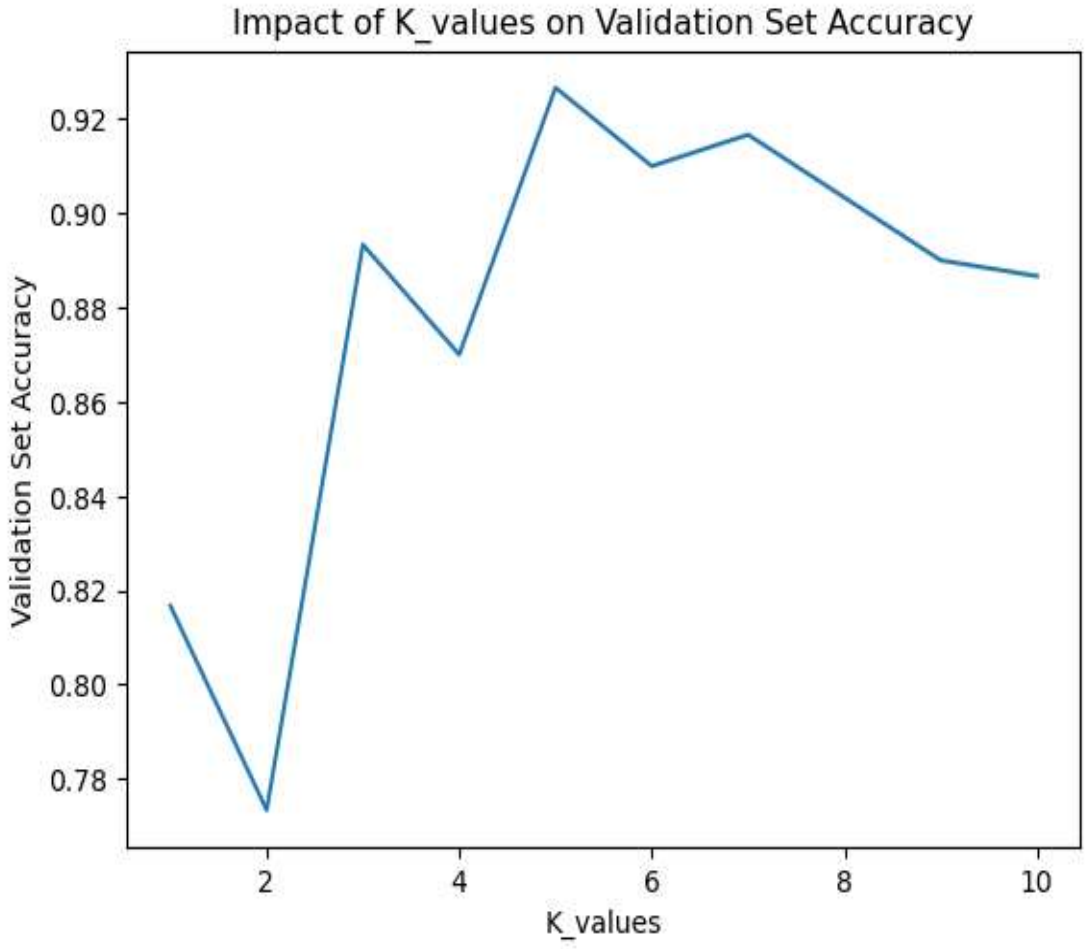
```
# Define the range of training set proportions
training_proportions = np.linspace(0.1, 1.0, 10)

# Initialize lists to store accuracy scores
validation_scores = []
testing_scores = []

# Iterate over the training set proportions
for proportion in training_proportions:
    # Calculate the number of training samples based on the proportion
    num_samples = int(len(x_train) * proportion)

    # Create subsets of the training data
    X_train_subset = x_train[:num_samples]
    Y_train_subset = y_train[:num_samples]

    # Create a KNN classifier with K=2
    knn = KNeighborsClassifier(n_neighbors=2)

    # Train the classifier
    knn.fit(X_train_subset, Y_train_subset)

    # Predict labels for the validation set and calculate accuracy
    val_predictions = knn.predict(x_val)
    val_accuracy = accuracy_score(y_val, val_predictions)
    validation_scores.append(val_accuracy)

    # Predict labels for the testing set and calculate accuracy
    test_predictions = knn.predict(x_test)
    test_accuracy = accuracy_score(y_test, test_predictions)
    testing_scores.append(test_accuracy)

# Plot the accuracy scores on validation and testing sets
plt.plot(training_proportions*100, validation_scores, label='Validation Set')
plt.plot(training_proportions*100, testing_scores, label='Testing Set')

# Set the plot labels and title
plt.xlabel('Training Set Proportion')
plt.ylabel('Accuracy Score')
plt.title('Impact of Training Set Proportion on Accuracy Score')
plt.legend()

# Show the plot
plt.show()
```



D)

- Then we use 100% of training samples, to find the best K value, and show the accuracy curve on the validation set when K varies from 1 to 10 and the best k value is 5.

```
# Define a range of K values to try
k_values = range(1, 11)

# Train KNN classifiers with different K values and evaluate on the validation set
val_scores = []
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_val)
    val_score =accuracy_score(y_val,y_pred)

    val_scores.append(val_score)

# Plot the validation set accuracy scores as a function of K
plt.plot(k_values, val_scores)
plt.xlabel('K_values')
plt.ylabel('Validation Set Accuracy')
plt.title('Impact of K_values on Validation Set Accuracy')
plt.show()
```



E)

- The conclusion:
- When the percentage of training data is increased, the accuracy increases.
- We should train the KNN model over all available k value to choose the best k value that suits our dataset, in our dataset the best k value is five.