Applied Machine Learning

Assignment 1

Names:

Nada Mohamed Zakaria

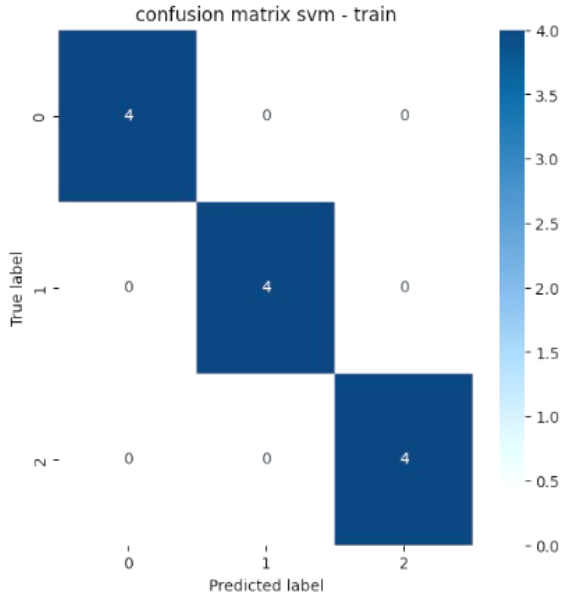
Dina Ibrahim Mohammady

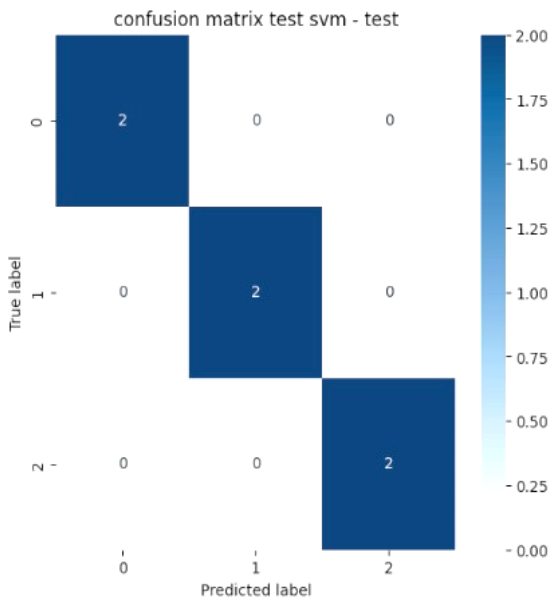
Sema Abdelnasser Mosaad

SVM and Perceptron

Creating and Training SVM Model

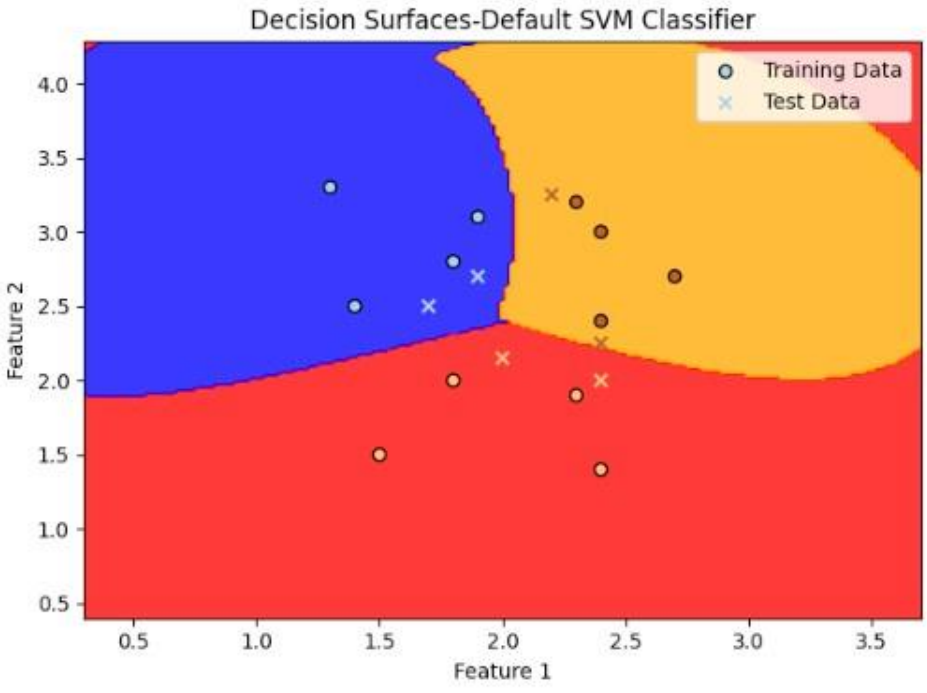
Getting confusion matrix for training and testing and calculating confusion matrix for both train and test





Decision Boundary on SVM

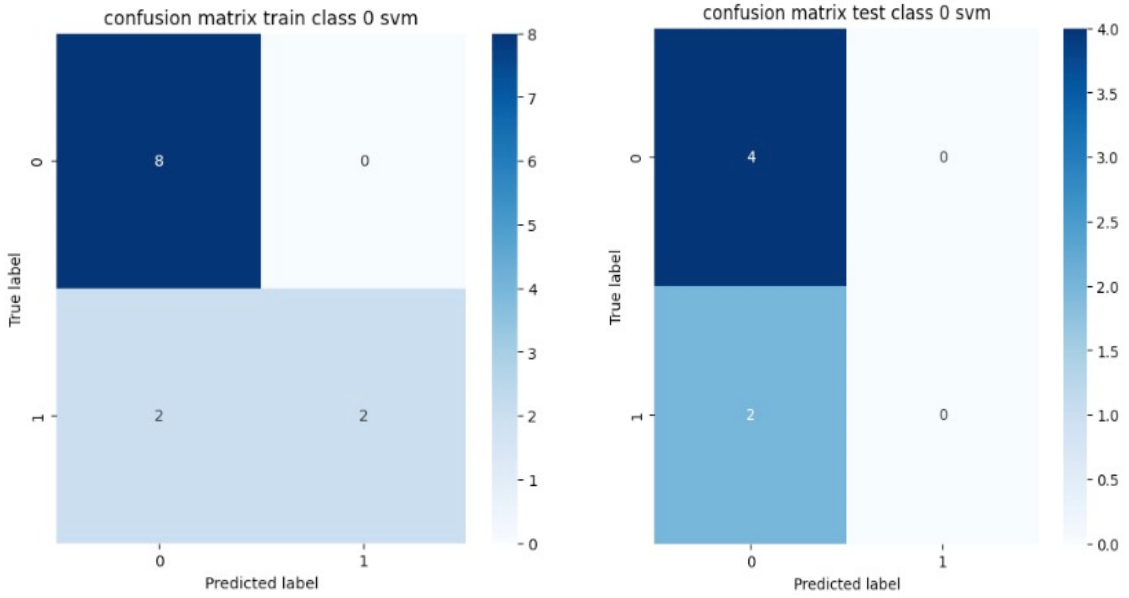
Plotting the decision surfaces and confusion matrices using the defined function.

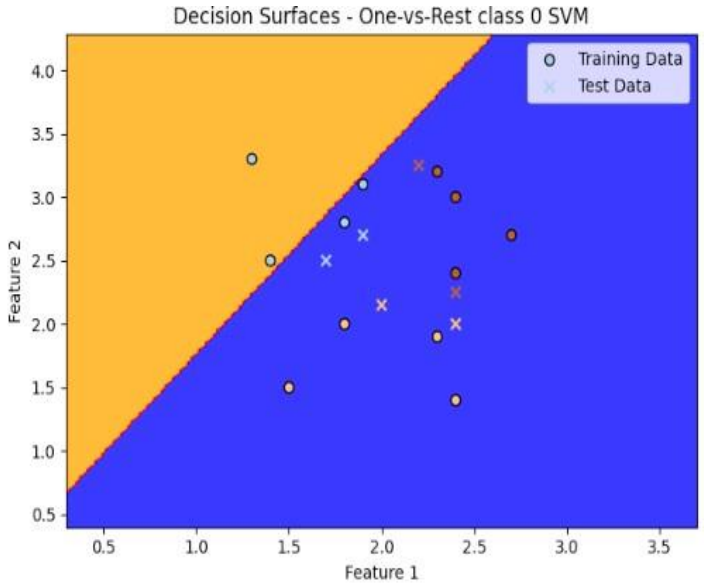


One VS Rest SVM

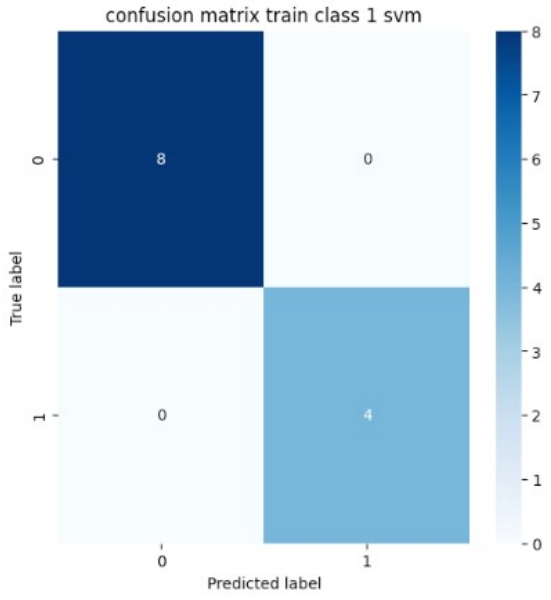
Creating and training SVM classifiers for each class.
Predicting the class labels for the training and testing data using the perceptron classifiers

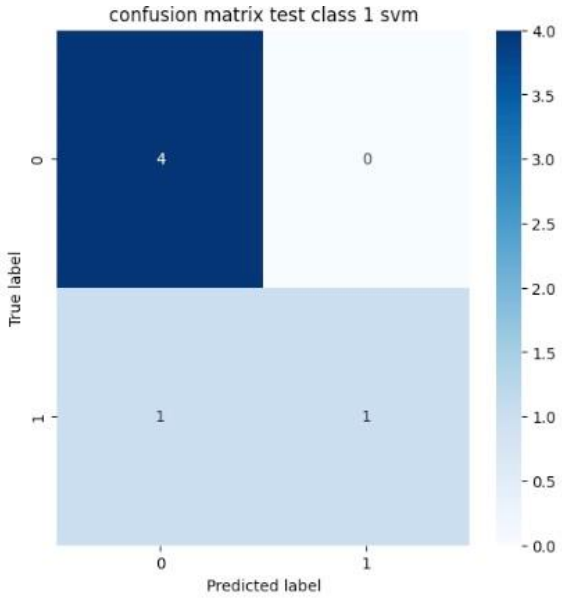
For class 0:

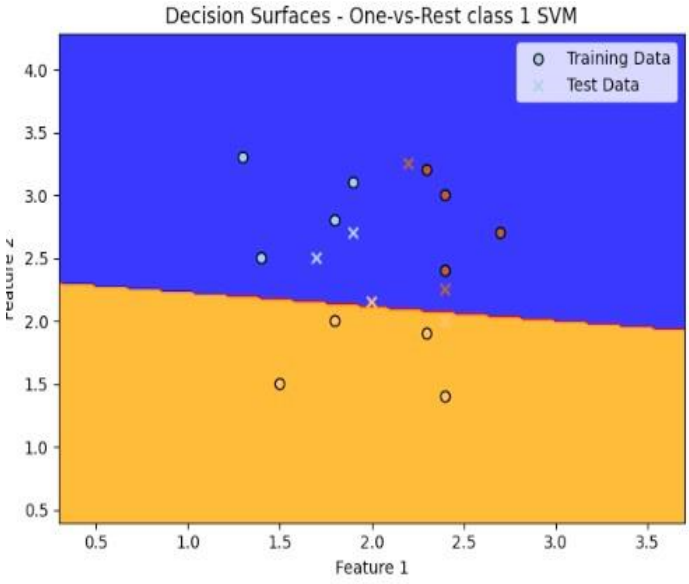




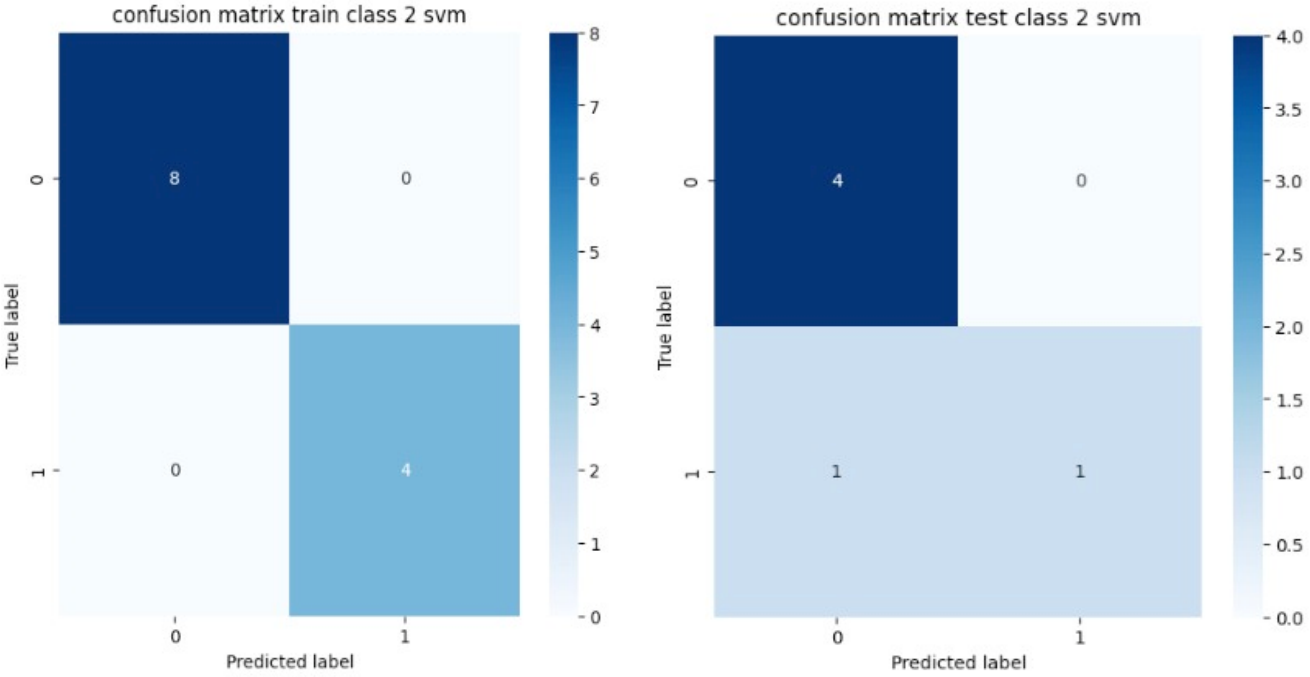
For class 1:

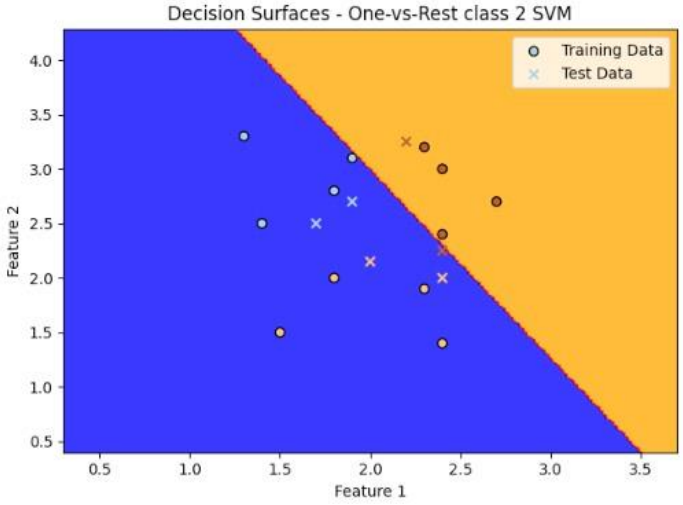

confusion matrix train class 1 svm
True label
Predicted label
8
0
0
4


confusion matrix test class 1 svm
True label
Predicted label
4
0
1
1


Decision Surfaces - One-vs-Rest class 1 SVM
Training Data
Test Data
Feature 1

For class 2:

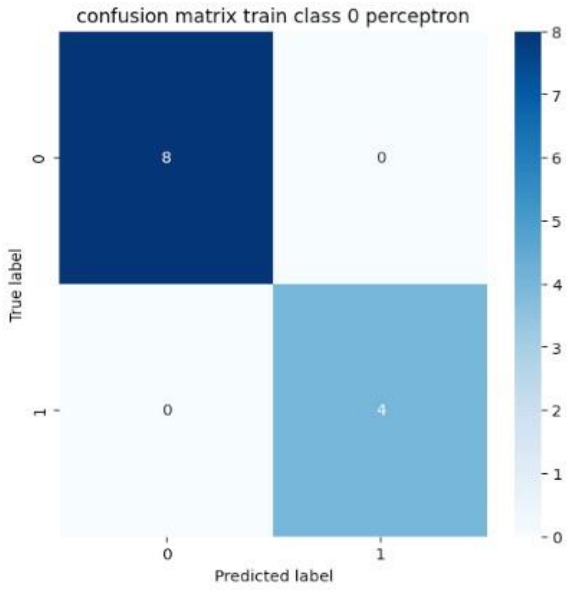

confusion matrix train class 2 svm
True label
Predicted label
0
1
8
0
0
4
confusion matrix test class 2 svm
True label
Predicted label
4
0
1
1

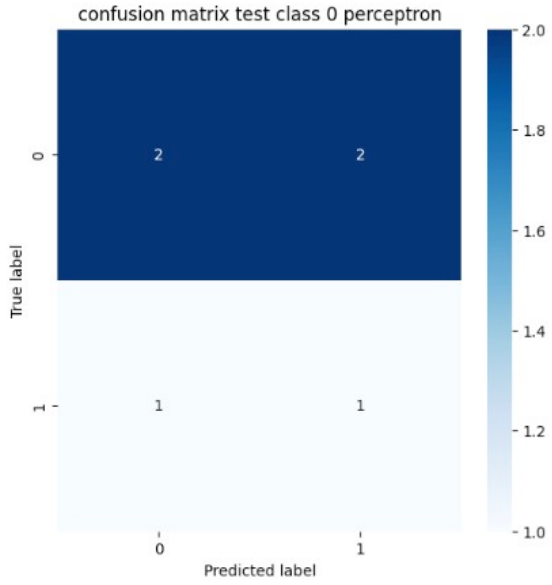

Decision Surfaces - One-vs-Rest class 2 SVM
Training Data
Test Data
Feature 1
Feature 2

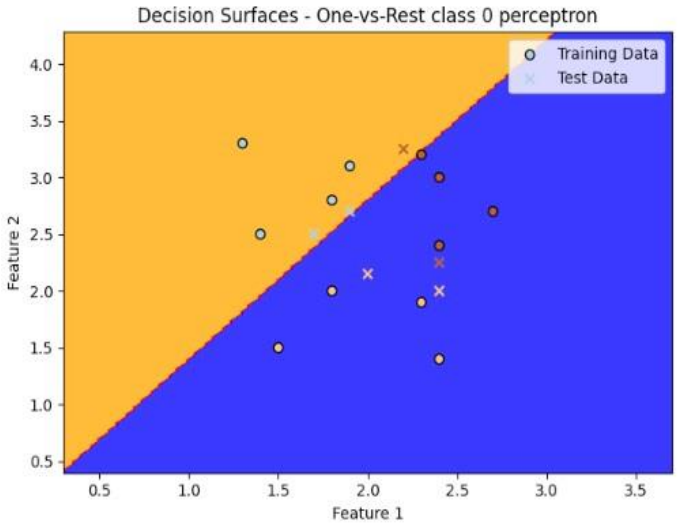
1. **Perceptron**

Creating and training perceptron classifiers for each class.
Predicting the class labels for the training and testing data using the perceptron classifiers

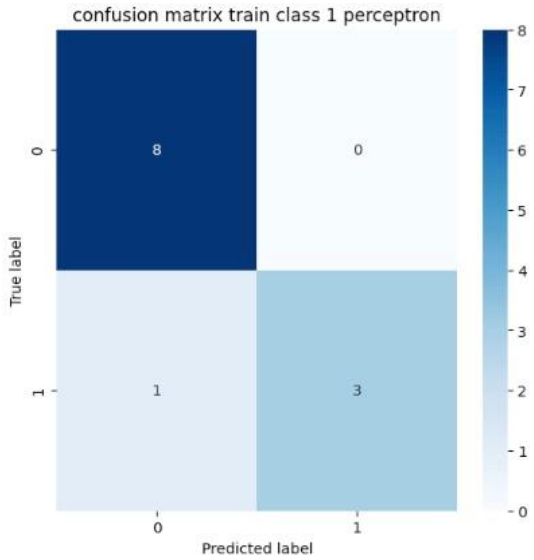
For class 0:

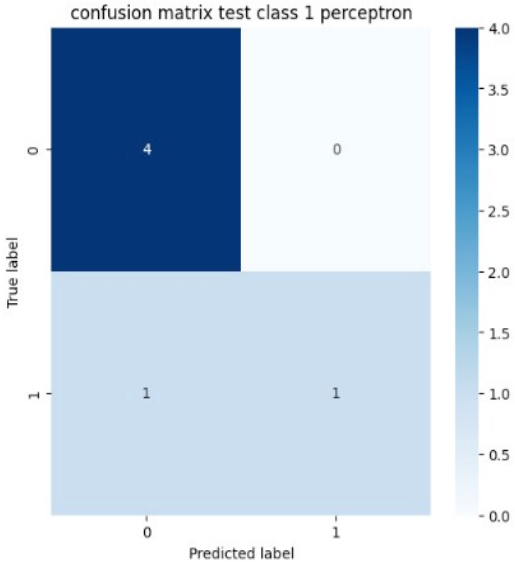


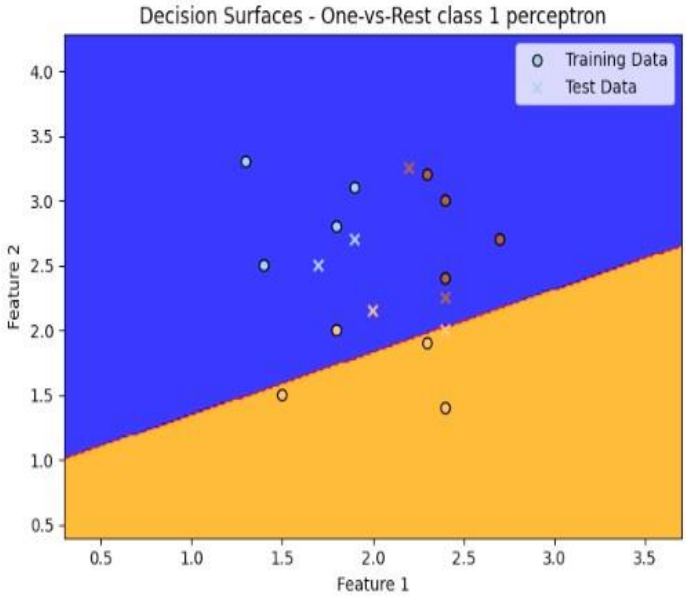




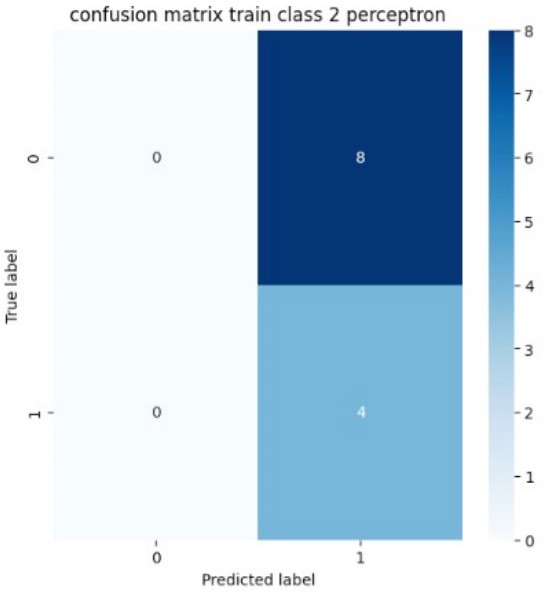
For class 1:

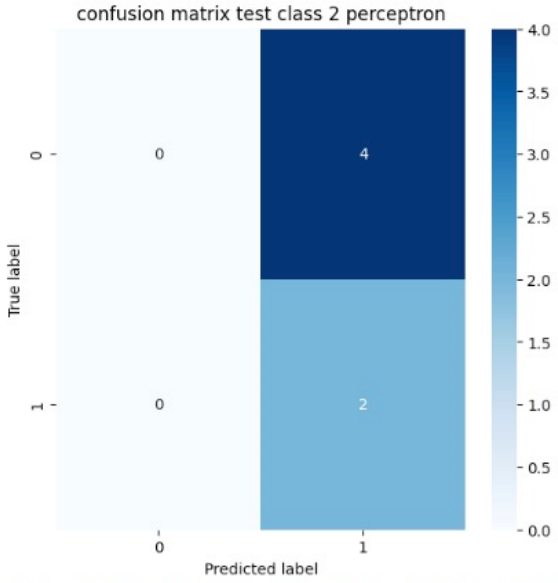

confusion matrix train class 1 perceptron
True label
Predicted label
8
0
1
3
0
1
0
1
8
7
6
5
4
3
2
1
0

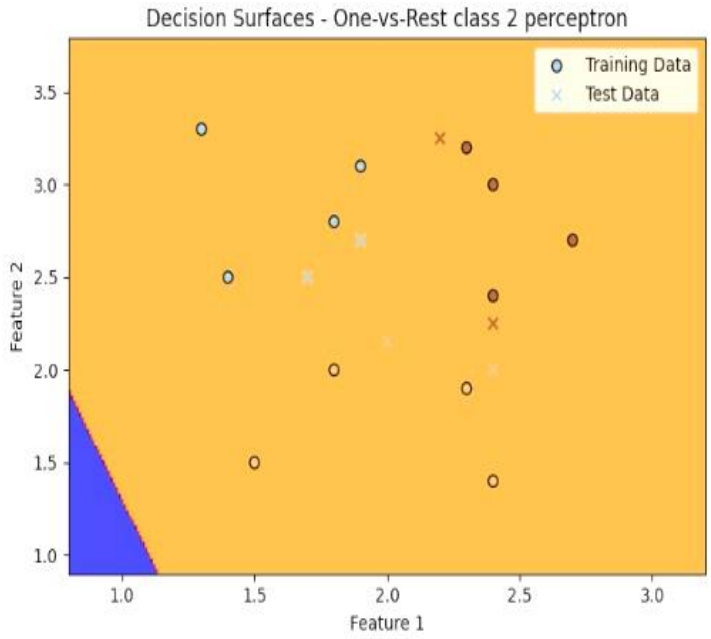

confusion matrix test class 1 perceptron
True label
Predicted label
4
0
1
1
0
1
0
1
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0


Decision Surfaces - One-vs-Rest class 1 perceptron
Training Data
Test Data
Feature 1
Feature 2
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

For class 2:


confusion matrix train class 2 perceptron
True label
Predicted label
0
1
0
8
0
4


confusion matrix test class 2 perceptron
True label
Predicted label
0
1
0
4
0
2


Decision Surfaces - One-vs-Rest class 2 perceptron
Training Data
Test Data
Feature 1
Feature 2

Comparison and analyzing between one vs rest SVM and Perceptron

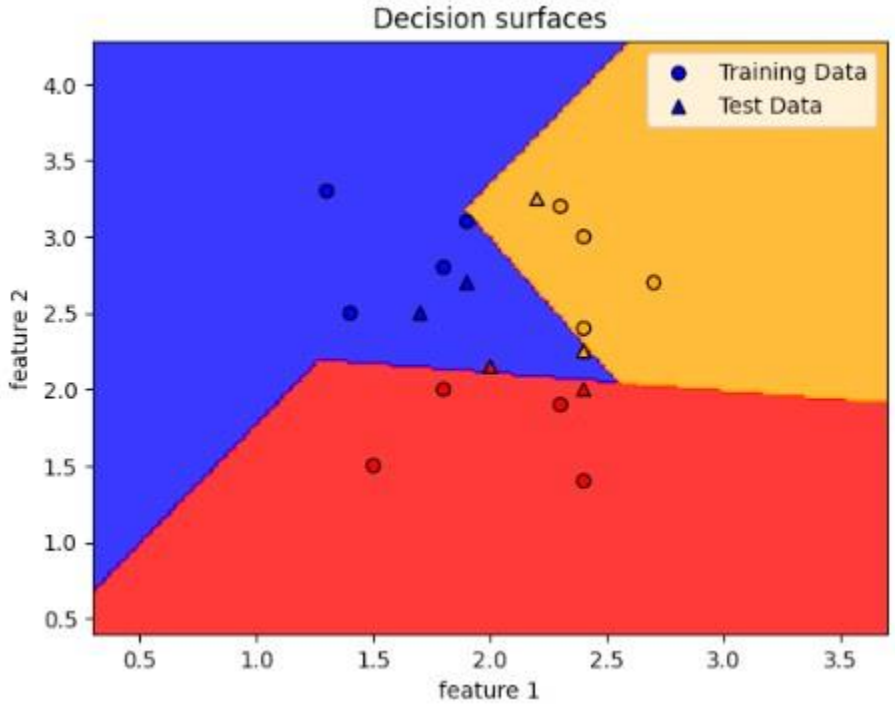
	Class	SVM (one vs rest)	Perceptron (one vs rest)
Accuracy	0	• Training: 83.33% • Testing: 66.67%	• Training: 100% • Testing: 66.67%
	1	• Training: 100% • Testing: 83.33%	• Training: 91.67% • Testing: 83.33%
	2	• Training:100% • Testing:83.33%	• Training: 33.33% • Testing: 83.33%

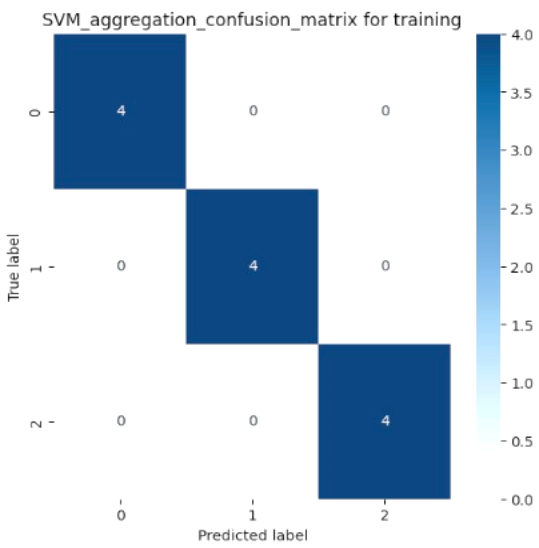
(Training and testing accuracy for each class of SVM & Perceptron)

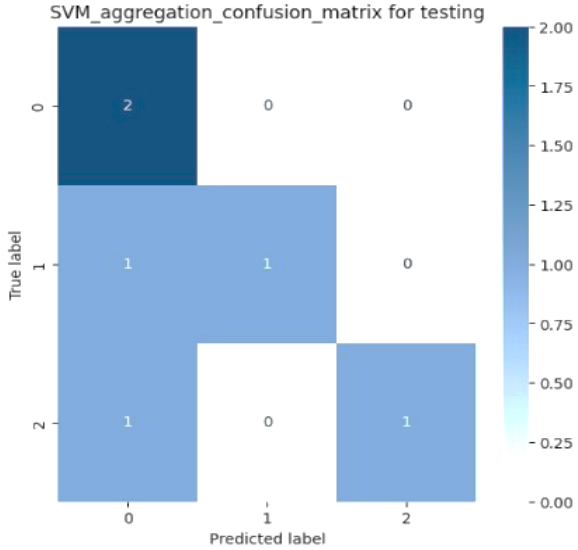
The perceptron model showed mixed performance across different classes, with overfitting on some classes and relatively lower accuracy on the training set for others. The SVM, on the other hand, demonstrated better overall performance, achieving higher accuracies and better generalization for all three classes.

Aggregating results from the one-vs-rest strategy for SVM and Perceptron.

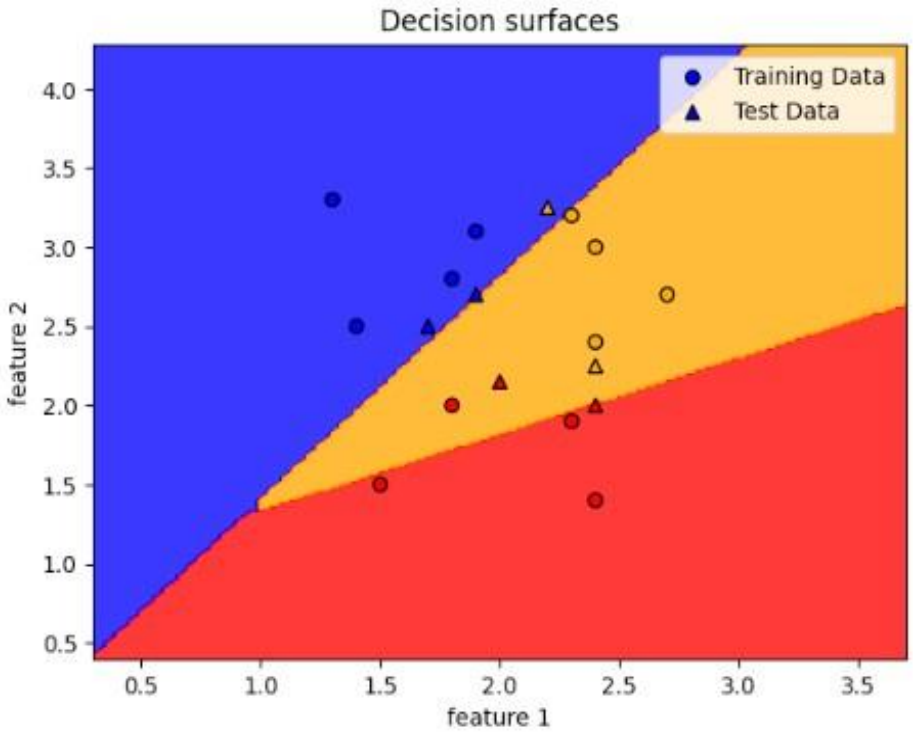
For SVM:

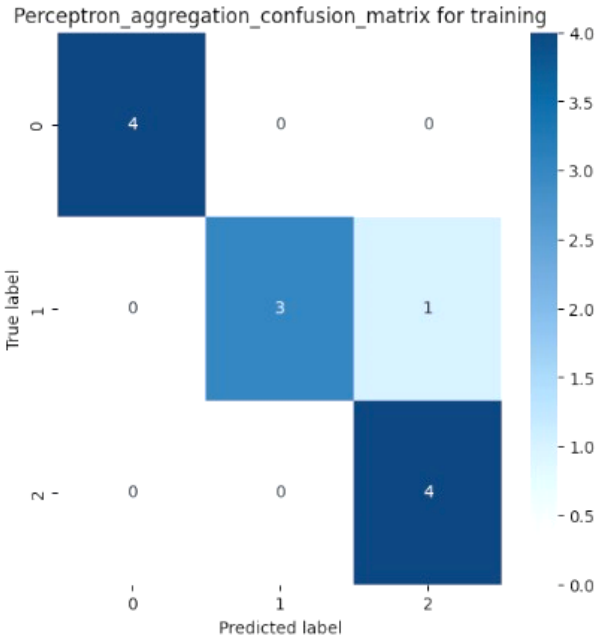


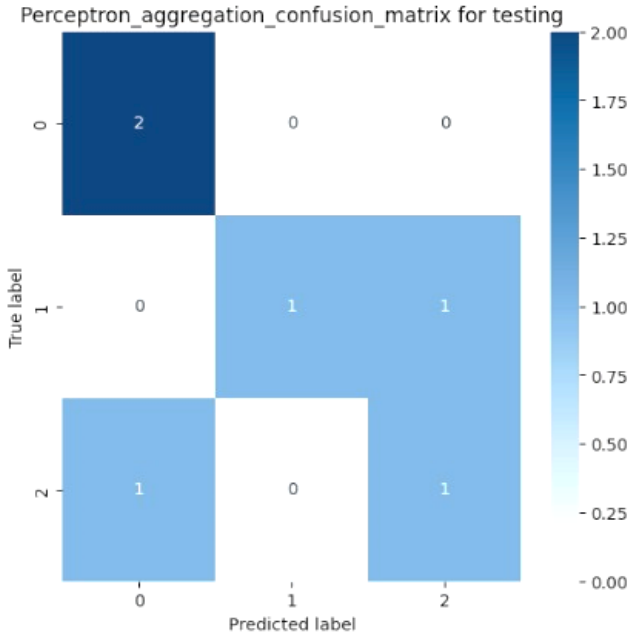




For Perceptron







Analyzing the performance of aggregated SVM and Perceptron, and comparing with the default SVM

	SVM default	SVM Aggregation	Perceptron Aggregation
Accuracy	• **Training:100%** • **Testing:100%**	• **Training:100%** • **Testing:66.67%**	• **Training:91.67%** • **Testing:66.67%**

In Training, SVM Aggregation outperforms Perceptron Aggregation, However, in Testing they have the same accuracy.

- **One vs Rest Aggregation using Perceptron:**

  The model performs well on the training data but struggles to generalize to unseen data, possibly due to overfitting or difficulty in capturing underlying patterns.

- **One vs Rest Aggregation using SVM:**

  The model achieves perfect training accuracy but faces similar challenges in generalization as the perceptron-based approach.

- **Default SVM Classifier:**

  The default SVM classifier performs exceptionally well on both the training and test data, demonstrating its ability to learn and generalize effectively

  Overall, the default SVM classifier stands out as the best-performing model, achieving perfect accuracy on both training and test data. The one vs rest aggregation approaches, whether using perceptron or SVM, show lower test accuracy, indicating difficulties in generalization. Further investigation into these models' limitations and potential improvements may be necessary.

The reason why default SVM performance id different than aggregated performance of SVM

Training Methodology:

Default SVM: A single SVM classifier that has been trained using its default hyperparameters and settings.

Multiple SVM classifiers that may have been trained using various subsets of the training data or using various hyperparameters make up aggregate SVM. To get the final forecast, various classifiers' predictions are combined.

Prediction Method:

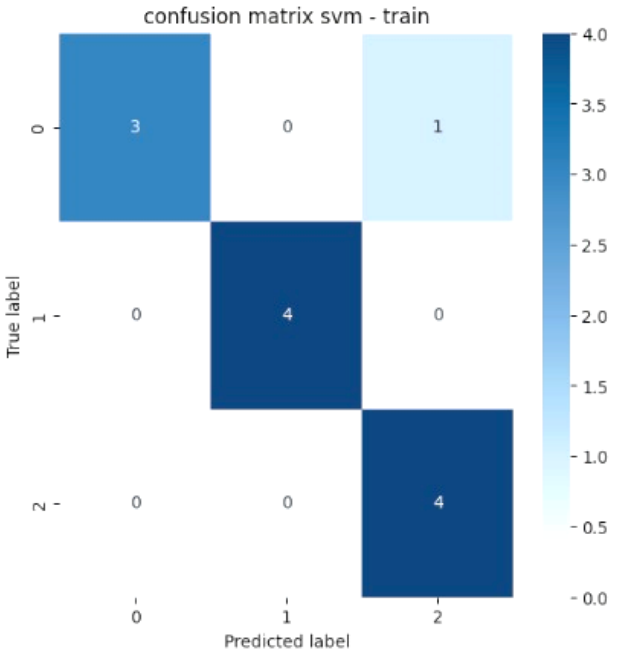
SVM by default: Separately predicts class labels for training and testing sets of data.

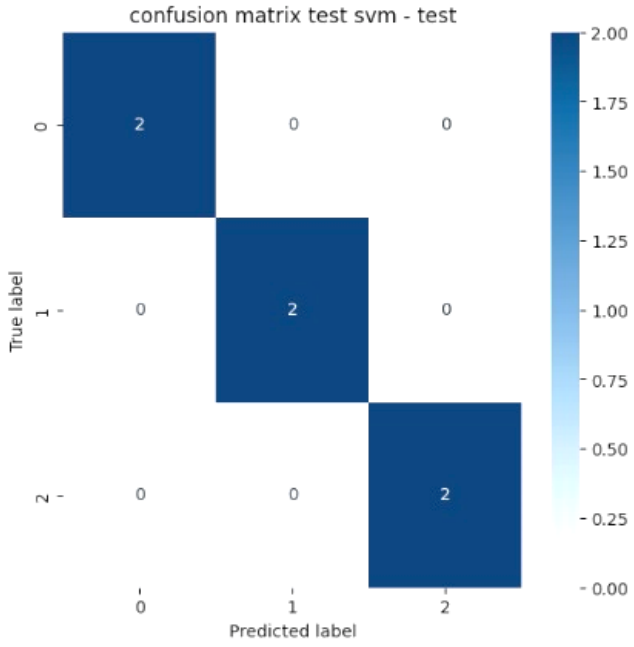
Aggregated SVM: Uses each individual SVM classifier to forecast the class labels for a mesh grid, then combines those predictions to produce the final prediction.

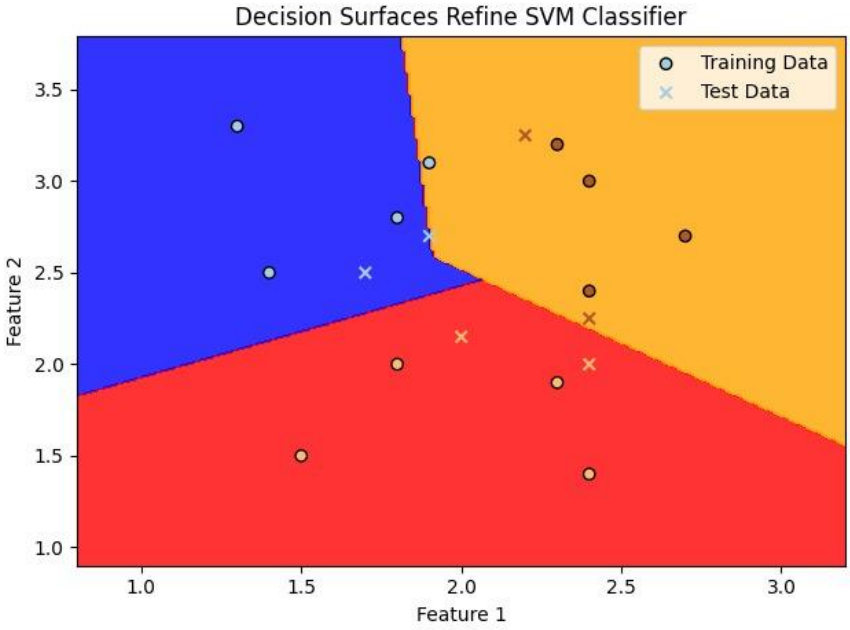
On both the training and testing sets of data, the default SVM achieves 100% accuracy, raising the possibility of overfitting. In contrast, the combined SVM achieves a precision of 100% on the testing data, however, there was only 66.67% accuracy, which suggests some overfitting.

It would be beneficial to look at the exact hyperparameters, data preprocessing, and training/validation techniques utilized for each SVM model in order to better understand the performance disparity.

After refining the SVM model







Evaluation of performance of Refine SVM

The model appears to perform very well on both the training and test data. The high accuracy on the test data suggests that the model generalized well to unseen instances and was able to classify them perfectly.

However, it's important to note that the model achieved perfect accuracy on the test data, which can sometimes be an indication of overfitting. Overfitting occurs when the model becomes too specialized to the training data and fails to generalize well to new, unseen data. To further evaluate the model's performance, it would be beneficial to assess its performance on additional unseen data or perform cross-validation.

Overall, the model seems to be performing well, but further evaluation and consideration of different metrics would be beneficial to obtain a more comprehensive assessment.

Comparing results of refining SVM with default SVM

	Default SVM	After refining the model
Accuracy	• **Training:100%** • **Testing:100%**	• **Training:91.666%** • **Testing:100%**

- The default SVM model achieved perfect accuracy on both the training and test datasets, with an accuracy of 100% for both. This indicates that the default model was able to perfectly separate the classes in the data.

- When experimenting with different parameter settings, the training accuracy dropped to 91.67% while the test accuracy remained at 100%. This suggests that the model might have overfit the training data when using different parameter values.

- **The impact of parameter selection** was evident when different parameter values were used, resulting in a drop in training accuracy. Careful consideration and experimentation with parameter values are essential to avoid overfitting and find the best configuration for a machine learning model.

K_Nearest Neighbors Classification

On a "car evaluation dataset," this programme applies K-Nearest Neighbours (KNN) classification. It preprocesses the data, divides it into training, validation, and testing sets, and assesses the effectiveness of the KNN algorithm using various parameters.

Data Preparation:
The code starts by importing the necessary libraries, including pandas, scikit-learn modules (train_test_split, LabelEncoder, KNeighborsClassifier, accuracy_score), matplotlib, and numpy. These libraries are essential for data manipulation, model building, and result visualization.

```
#import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report
```

Data Loading and Preprocessing:
The code reads the data from a CSV file ("car_evaluation.csv") into a pandas DataFrame. It assigns column names to the DataFrame to provide meaningful labels for each feature. The data is then shuffled randomly using the `sample` function to avoid any biases.

	buying Price	maintenance cost	number of doors	number of persons	lug_boot	safety	decision
599	high	high	4	2	med	high	unacc
1201	med	low	2	4	med	med	acc
628	high	high	5more	2	big	med	unacc
1498	low	high	5more	4	med	med	acc
1263	med	low	4	more	med	low	unacc
...	...	...	...	...	...	...	...
1130	med	med	3	more	med	high	vgood
1294	med	low	5more	more	big	med	good
860	high	low	5more	more	med	high	acc
1459	low	high	4	2	small	med	unacc
1126	med	med	3	more	small	med	acc

1728 rows × 7 columns

Train-Validation-Test Split:

The data is split into three sets: training, validation, and testing. The `train_test_split` function from scikit-learn is used to perform the split. The training set consists of 1000 samples, the validation set size depends on the specified proportion in the loop, and the testing set contains 428 samples.

```
Training set shape: (1000, 6)
Validation set shape: (300, 6)
Testing set shape: (428, 6)
```

Label Encoding:

To work with categorical variables, the code utilizes the `LabelEncoder` from scikit-learn. The target variable (decision) is encoded into numerical values for all three sets: training, validation, and testing. Additionally, the categorical features ('buying Price', 'maintenance cost', 'number of doors', 'number of persons', 'lug_boot', 'safety') are also encoded into numerical values.

An Example of labeling (X_Train)

	buying Price	maintenance cost	number of doors	number of persons	lug_boot	safety
1304	1	3	0	0	0	0
319	3	2	3	2	1	2
1028	2	0	2	0	2	0
563	0	0	0	2	1	0
1300	1	3	0	0	1	2
...	...	...	...	...	...	...
709	0	2	2	0	0	2
95	3	3	3	1	1	0
1095	2	2	0	1	0	1
1603	1	2	3	1	2	2
748	0	2	3	2	2	2

1000 rows × 6 columns

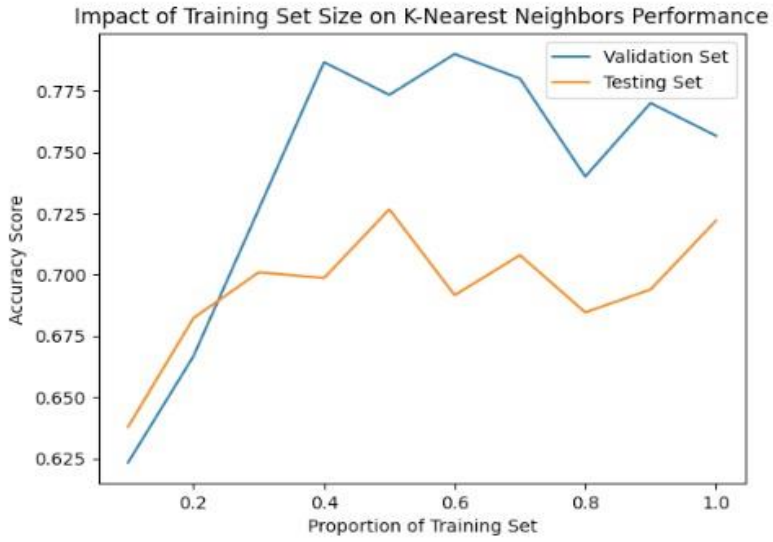
Model Training and Evaluation:

The code proceeds to train the KNN classifier using different proportions of the training set. For each proportion, the specified number of training samples is used to fit the model. The validation set is used to evaluate the model's performance by calculating the accuracy score using the `accuracy_score` function from scikit-learn. The accuracy scores for both the validation and testing sets are stored.

```
Validation accuracy for  0.1     0.6233333333333333
Test       accuracy for  0.1     0.6378504672897196
Validation accuracy for  0.2     0.6666666666666666
Test       accuracy for  0.2     0.6822429906542056
Validation accuracy for  0.3     0.7266666666666667
Test       accuracy for  0.3     0.7009345794392523
Validation accuracy for  0.4     0.7866666666666666
Test       accuracy for  0.4     0.6985981308411215
Validation accuracy for  0.5     0.7733333333333333
Test       accuracy for  0.5     0.7266355140186916
Validation accuracy for  0.6     0.79
Test       accuracy for  0.6     0.6915887850467289
Validation accuracy for  0.7     0.78
Test       accuracy for  0.7     0.7079439252336449
Validation accuracy for  0.8     0.74
Test       accuracy for  0.8     0.6845794392523364
Validation accuracy for  0.9     0.77
Test       accuracy for  0.9     0.6939252336448598
Validation accuracy for  1.0     0.7566666666666667
Test       accuracy for  1.0     0.7219626168224299
```

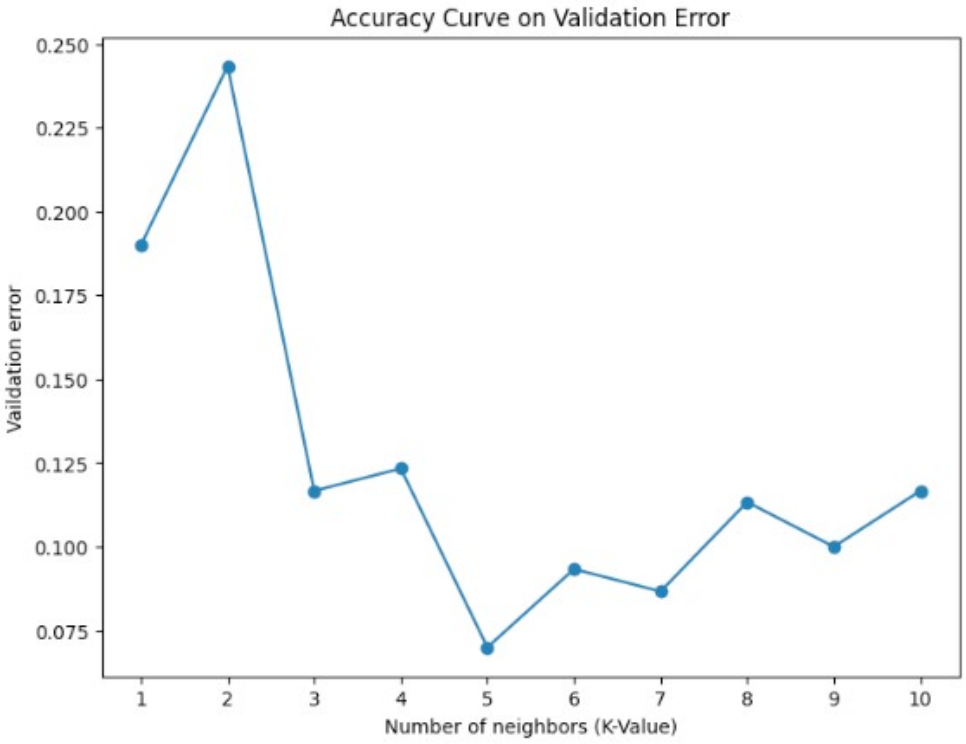
Validation Set Size Impact:

The code plots a graph to visualize the impact of the training set size on the KNN performance. The x-axis represents the proportion of the training set used, while the y-axis shows the accuracy scores. The graph helps understand how the model's performance changes as the size of the training set varies.



K-Value Selection:

The code then focuses on finding the optimal value of K (number of neighbors) for the KNN algorithm. It trains the KNN classifier using different values of K (ranging from 1 to 10) and calculates the accuracy score for each value. Another graph is plotted to illustrate the validation error (1 - accuracy) for each K value, providing insights into the impact of K on the model's performance.



Overall, the code provides a comprehensive implementation of the KNN algorithm for classification tasks. It demonstrates the effect of the training set size and the selection of the K value on the model's accuracy.

Impact of Number of Training Samplest

By varying the number of training samples from 10% to 100% of the training set, we observed the following:

1. As the number of training samples increased, the accuracy scores on both the validation and testing sets generally improved.
2. With a small number of training samples the model may not have enough data to learn meaningful patterns, leading to lower accuracy scores.
3. there might be a point of diminishing returns, where adding more training samples does not significantly improve the model's performance.

Finding the best K value

By varying the number of K_value from 1 to 10 of, we observed the following:

1. The rise in K_value was generally accompanied by a decrease in validation error.

2. The best accuracy was observed when the value equaled 5, which had the fewest validation errors.