Applied Machine Learning

7/6/2023
Assignment 1
ELG5255[EG]

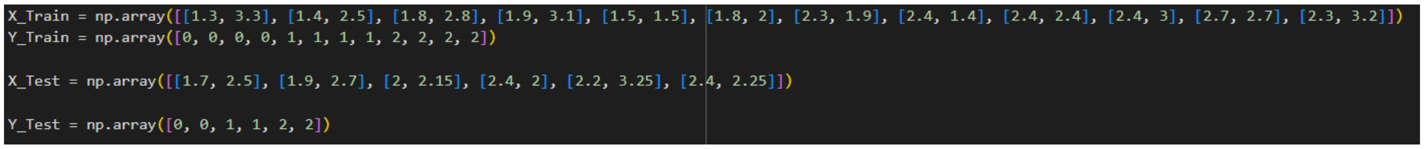
Group 4

1. Aya Metwally
2. Heba Mostafa
3. Amira Abu Issa

Problem 1:

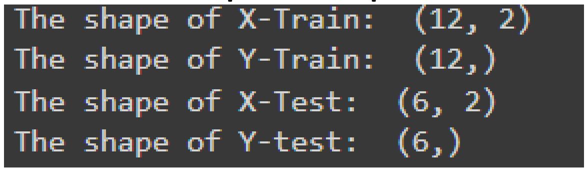
First, we created the dataset as provided in text file uploaded in Brightspace with the assignment:

Code:



```
X_Train = np.array([[1.3, 3.3], [1.4, 2.5], [1.8, 2.8], [1.9, 3.1], [1.5, 1.5], [1.8, 2], [2.3, 1.9], [2.4, 1.4], [2.4, 2.4], [2.4, 3], [2.7, 2.7], [2.3, 3.2]])
Y_Train = np.array([0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2])

X_Test = np.array([[1.7, 2.5], [1.9, 2.7], [2, 2.15], [2.4, 2], [2.2, 3.25], [2.4, 2.25]])

Y_Test = np.array([0, 0, 1, 1, 2, 2])
```

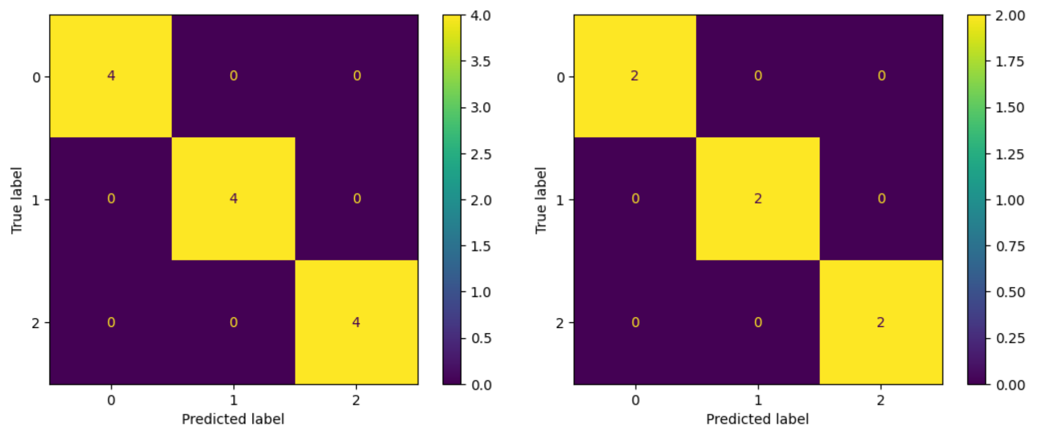
Output for shape:

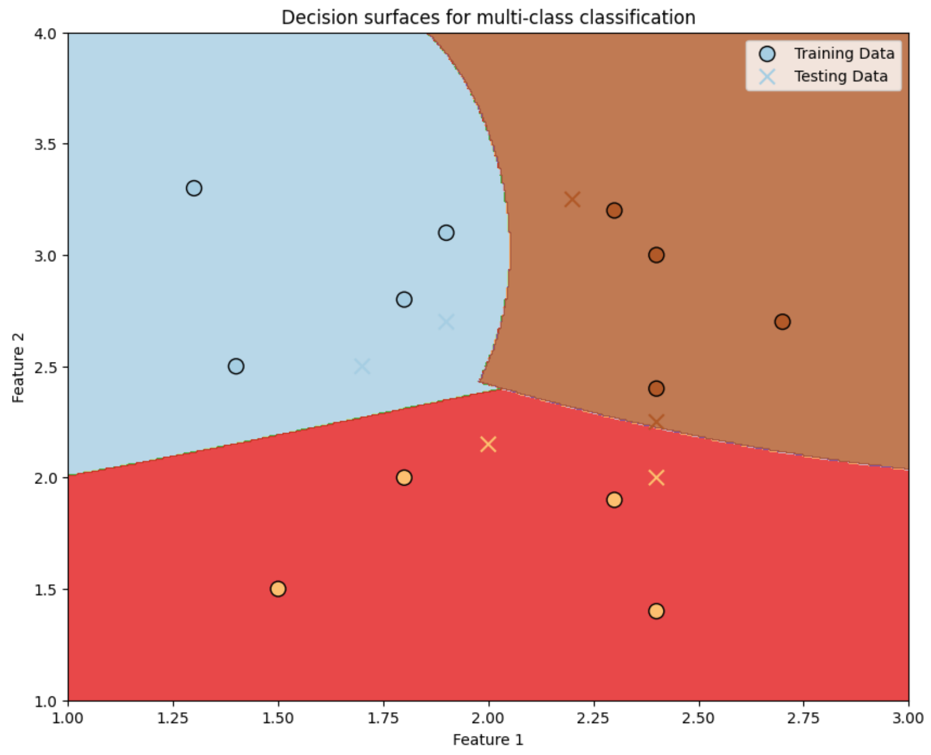


```
The shape of X-Train:  (12, 2)
The shape of Y-Train:  (12,)
The shape of X-Test:  (6, 2)
The shape of Y-test:  (6,)
```

a) Default SVM

the confusion matrix for training dataset **the confusion matrix for test dataset**

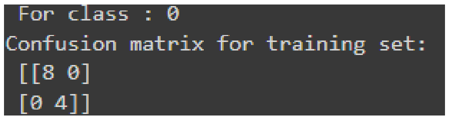




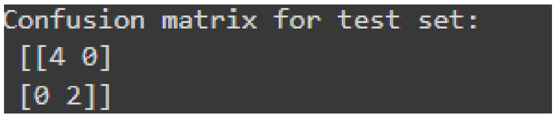
b) SVM with OVR Vs. Perceptron with OVR

SVM with OVR:

for class 0:

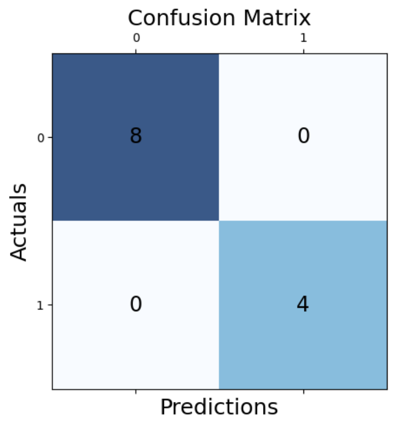


```
 For class : 0
Confusion matrix for training set:
 [[8 0]
 [0 4]]
```



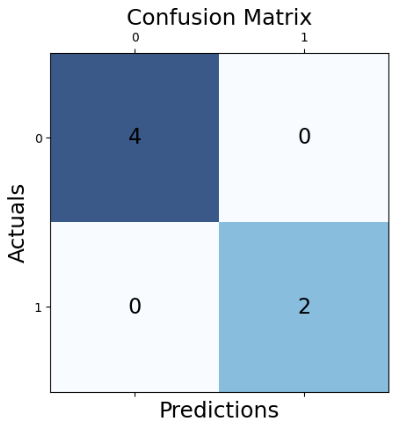
```
Confusion matrix for test set:
 [[4 0]
 [0 2]]
```

Confusion Matrix for training set:

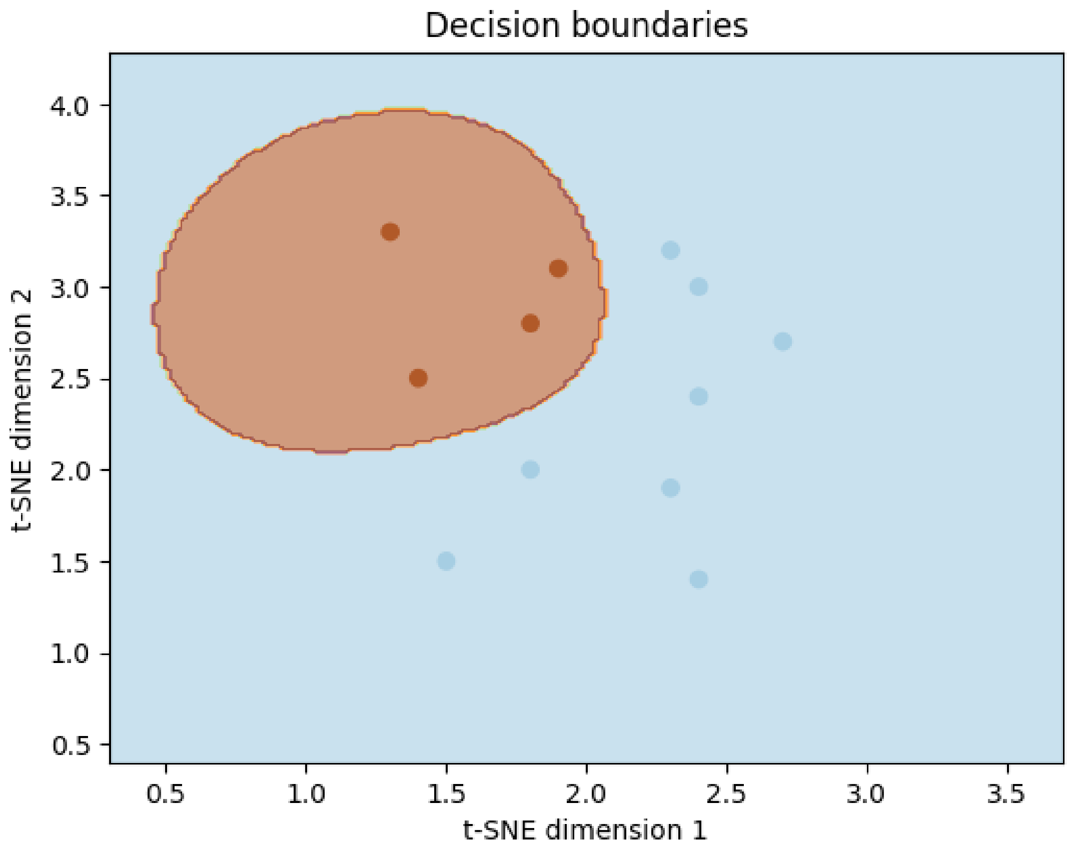


	0	1
0	8	0
1	0	4

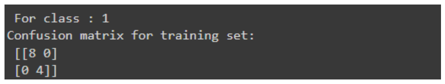
confusion matrix for test set:



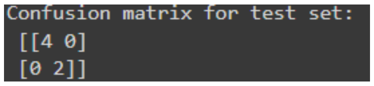
	0	1
0	4	0
1	0	2



for class 1:

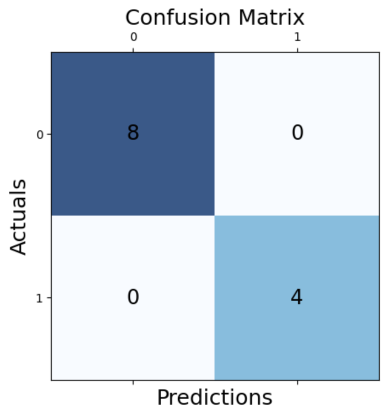


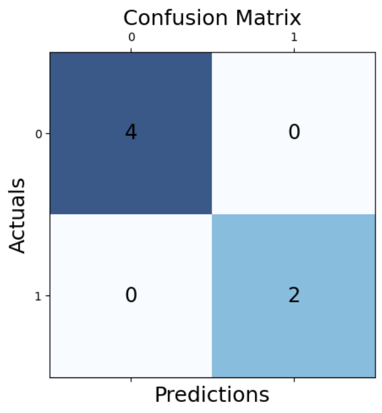
```
 For class : 1
Confusion matrix for training set:
 [[8 0]
 [0 4]]
```

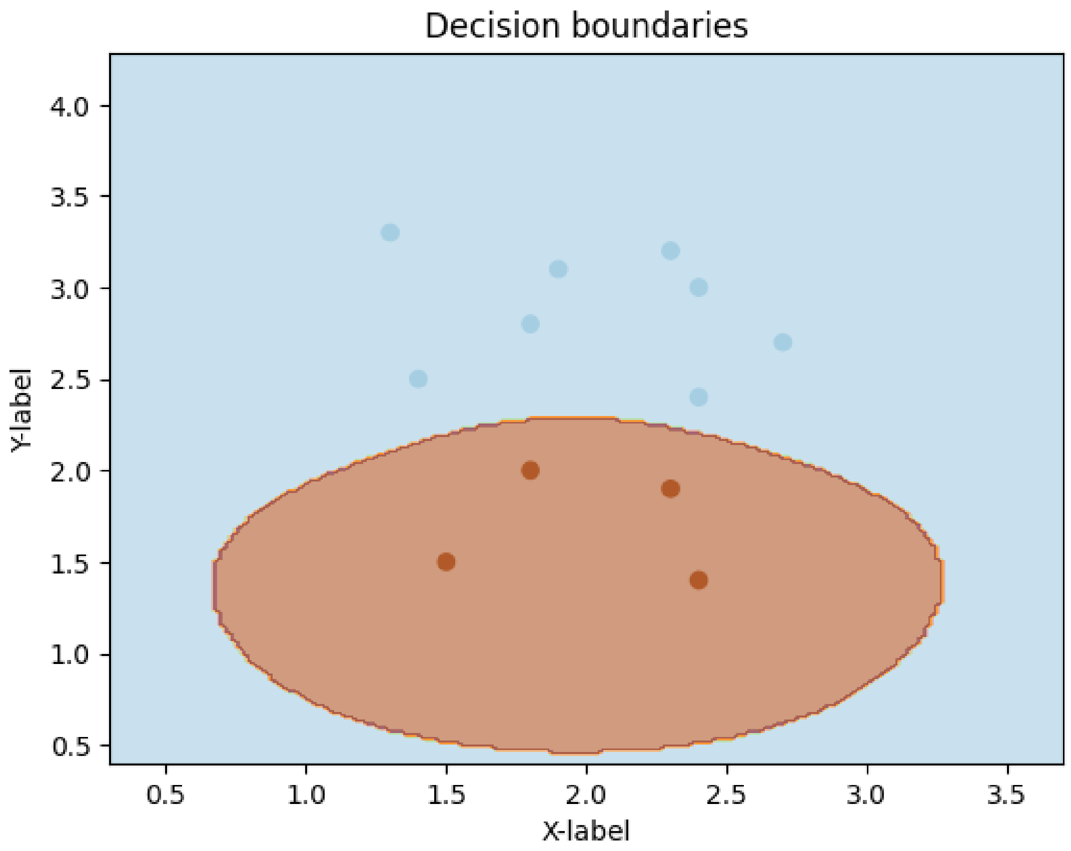


```
Confusion matrix for test set:
 [[4 0]
 [0 2]]
```

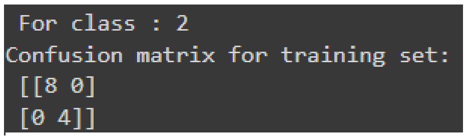
Confusion Matrix for training set: **confusion matrix for test set:**



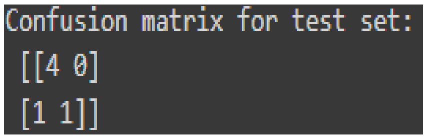




for class 2:

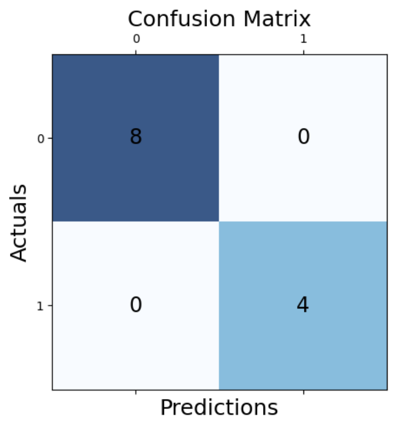


```
 For class : 2
Confusion matrix for training set:
 [[8 0]
 [0 4]]
```

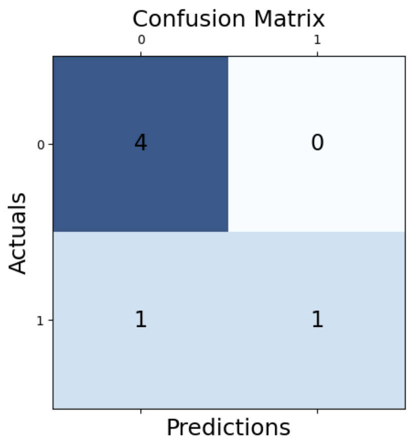


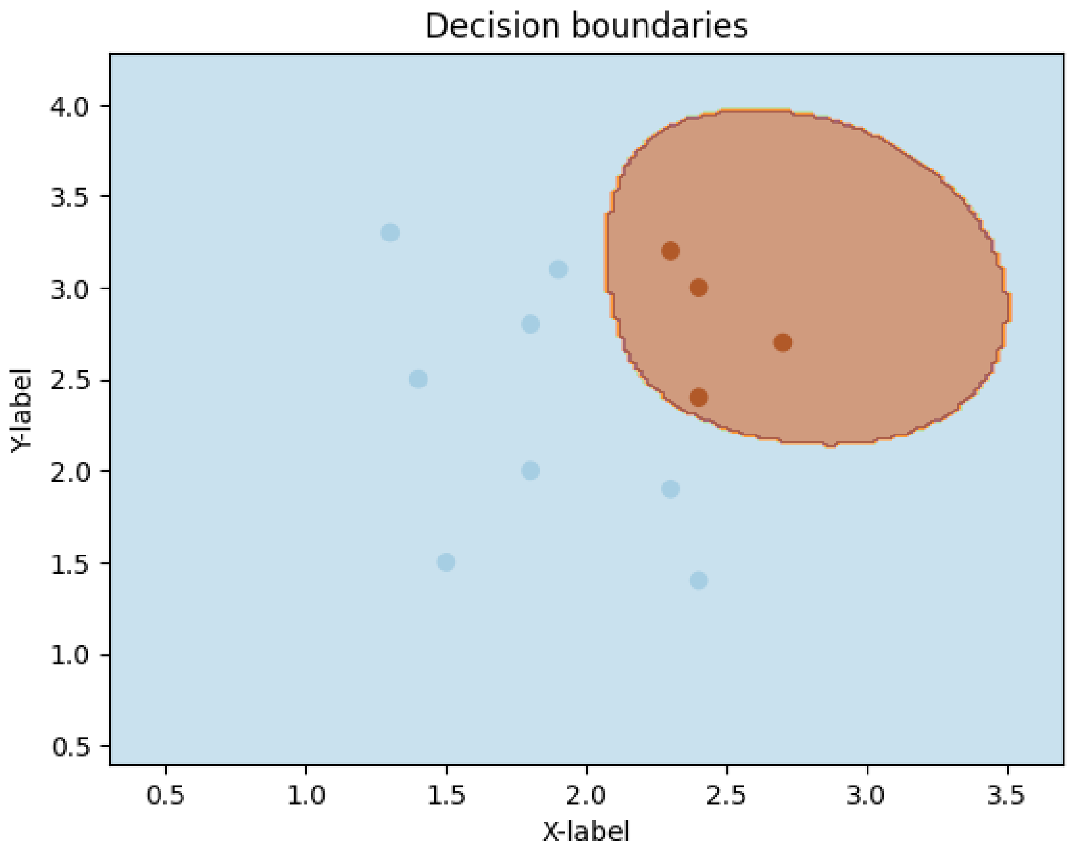
```
Confusion matrix for test set:
 [[4 0]
 [1 1]]
```

Confusion Matrix for training set:



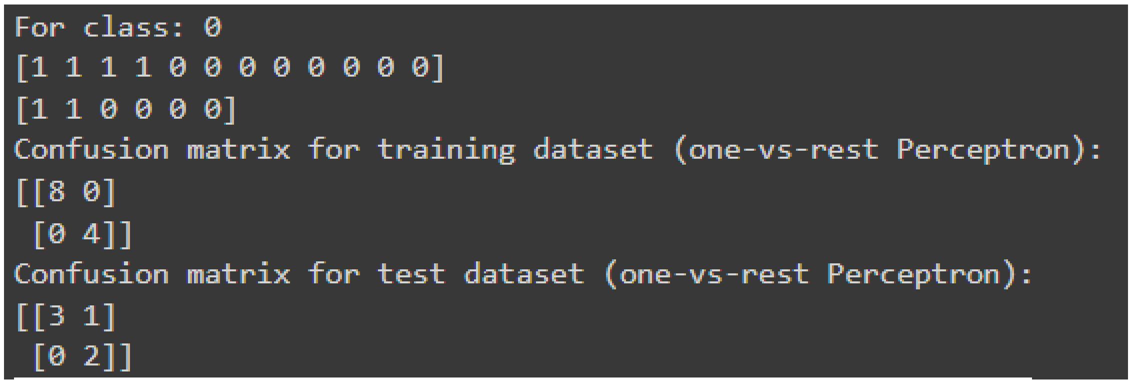
confusion matrix for test set:





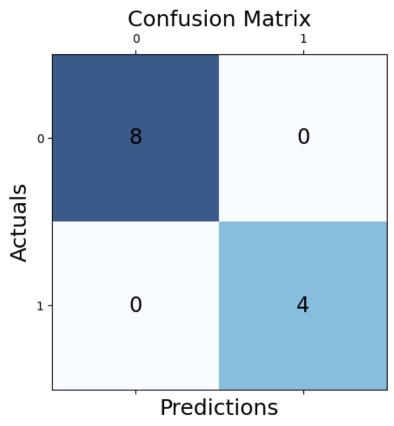
Perceptron with OVR:

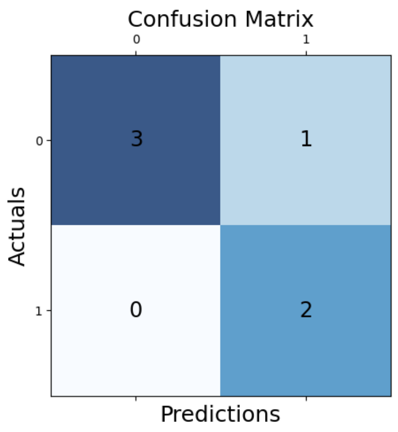
for class 0:

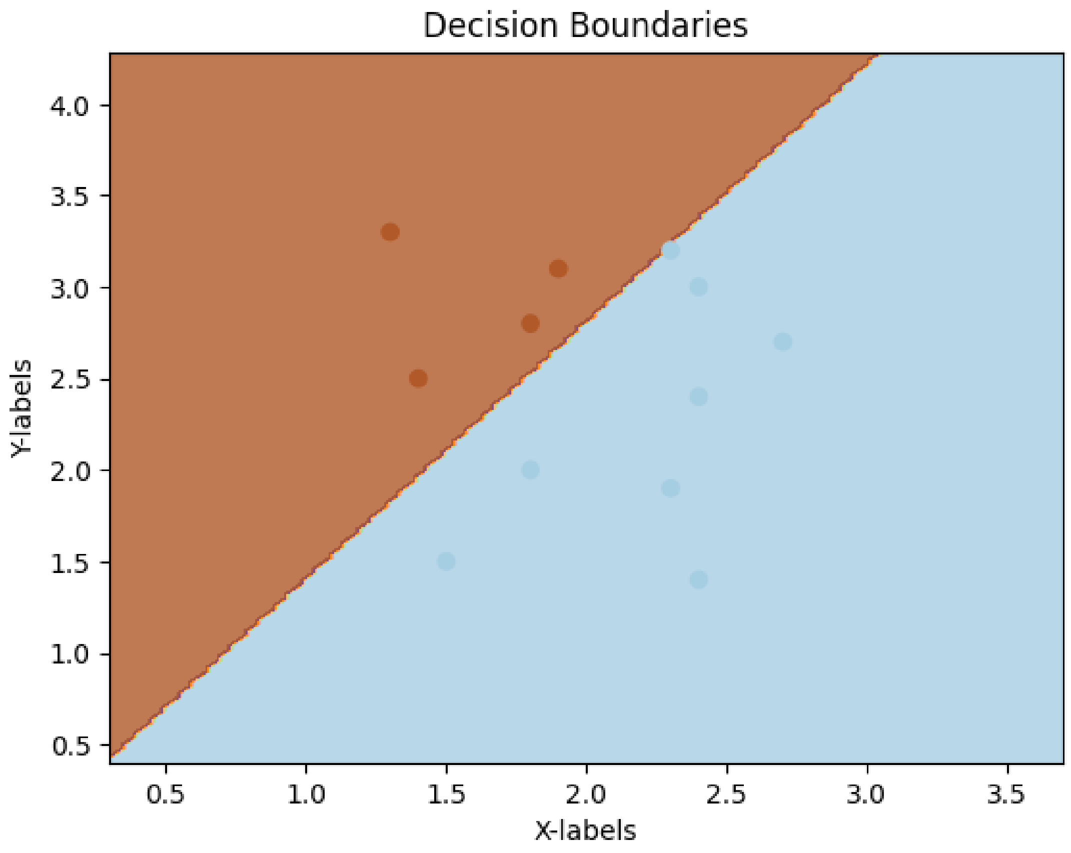


```
For class: 0
[1 1 1 1 0 0 0 0 0 0 0 0]
[1 1 0 0 0 0]
Confusion matrix for training dataset (one-vs-rest Perceptron):
[[8 0]
 [0 4]]
Confusion matrix for test dataset (one-vs-rest Perceptron):
[[3 1]
 [0 2]]
```

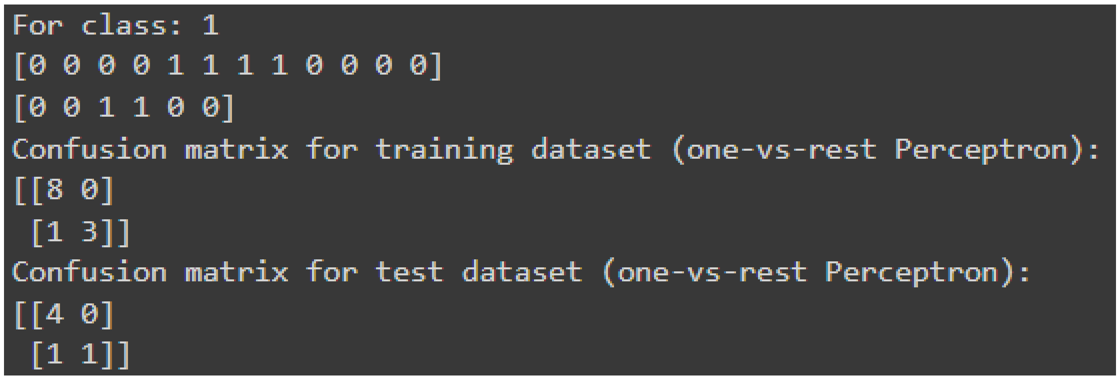
Confusion Matrix for training set: **confusion matrix for test set:**





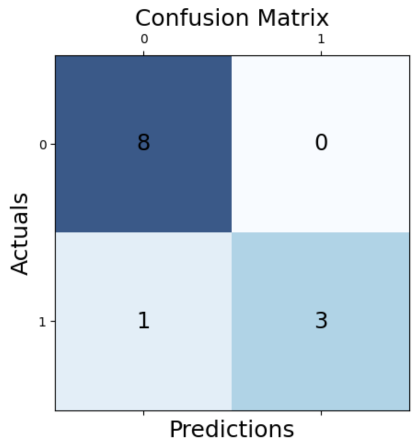


FOR CLASS 1:

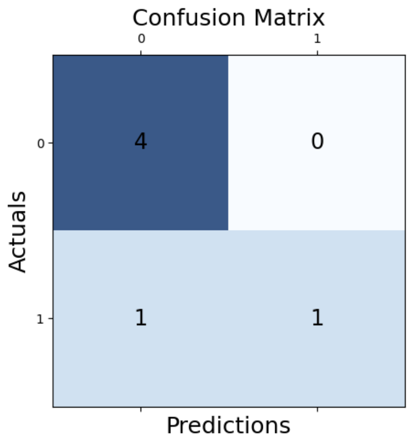


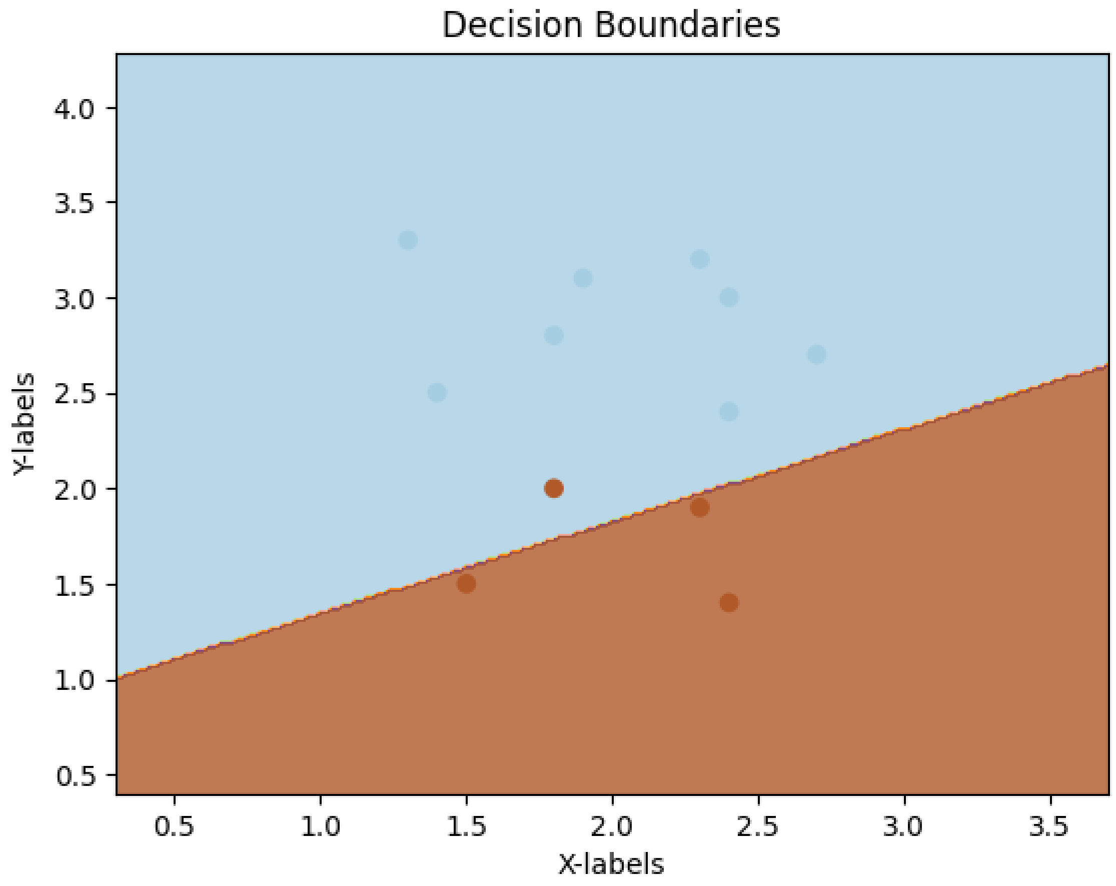
```
For class: 1
[0 0 0 0 1 1 1 1 0 0 0 0]
[0 0 1 1 0 0]
Confusion matrix for training dataset (one-vs-rest Perceptron):
[[8 0]
 [1 3]]
Confusion matrix for test dataset (one-vs-rest Perceptron):
[[4 0]
 [1 1]]
```

Confusion Matrix for training set:

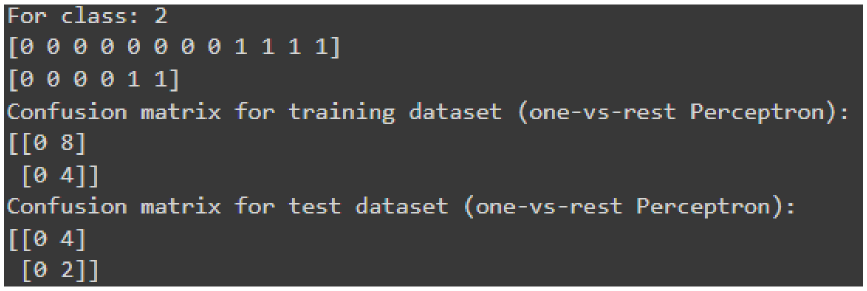


confusion matrix for test set:



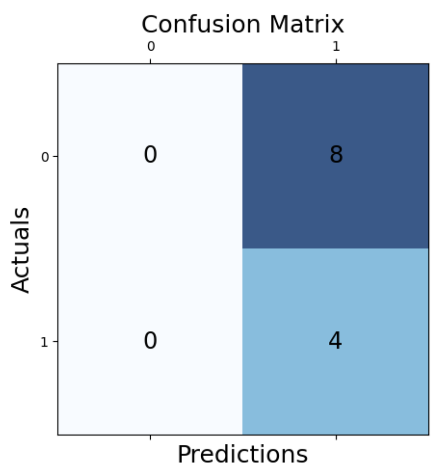


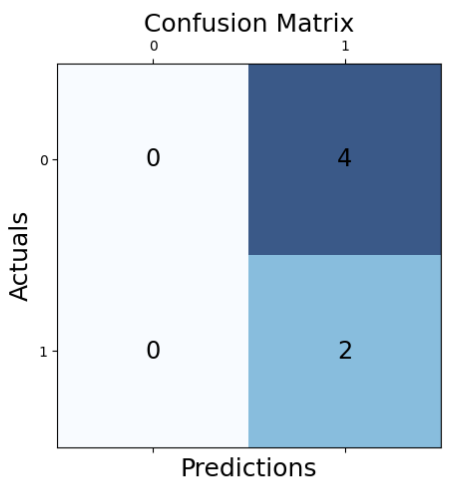
For Class 2:

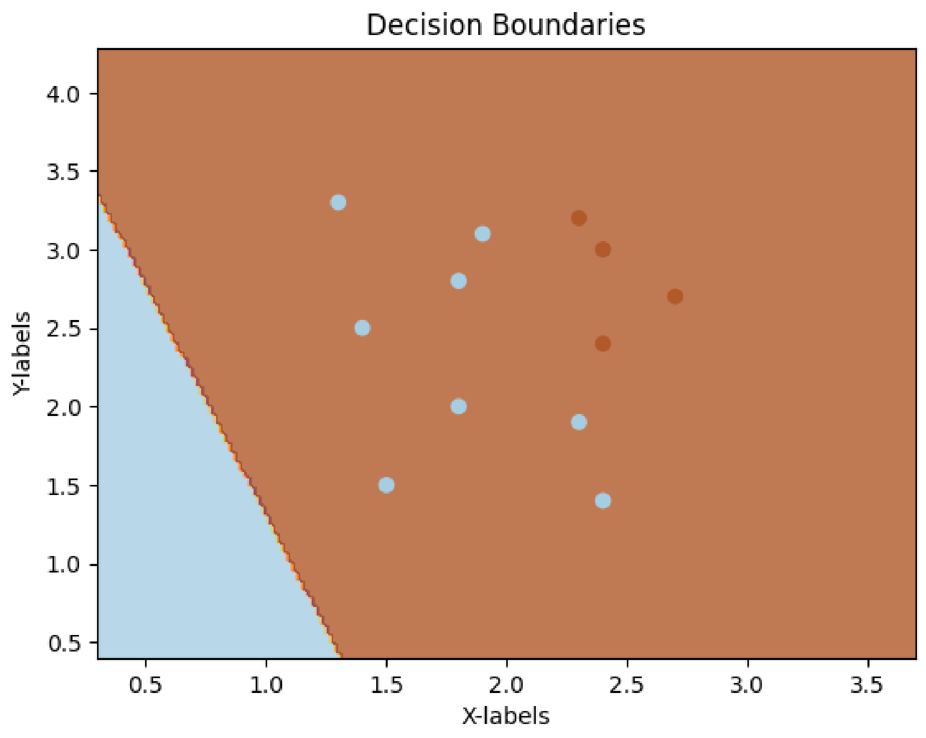


```
For class: 2
[0 0 0 0 0 0 0 0 1 1 1 1]
[0 0 0 0 1 1]
Confusion matrix for training dataset (one-vs-rest Perceptron):
[[0 8]
 [0 4]]
Confusion matrix for test dataset (one-vs-rest Perceptron):
[[0 4]
 [0 2]]
```

Confusion Matrix for training set: **confusion matrix for test set:**







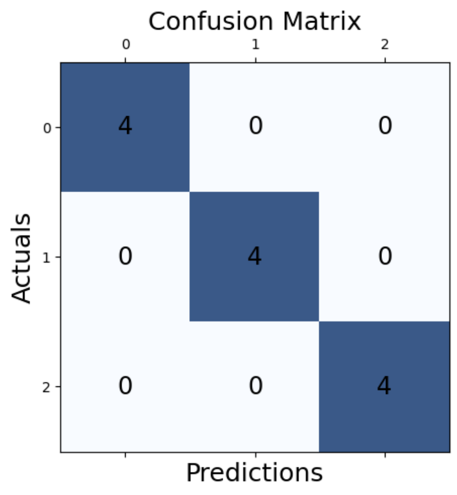
SVM and Perceptron results:

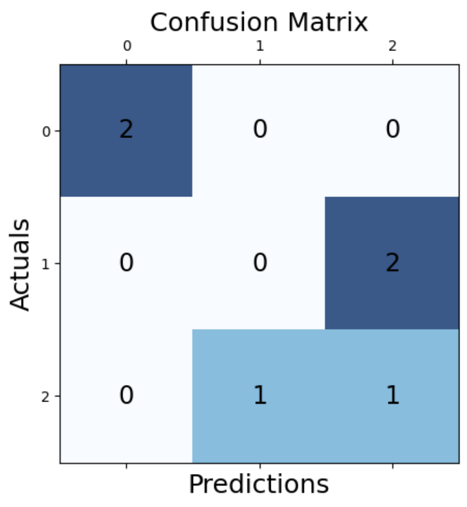
SVM seeks to find the maximum margin hyperplane that separates the classes, while Perceptron seeks to find a hyperplane that minimizes the classification error. SVM can handle non-linearly separable data by using kernel functions, whereas Perceptron can only handle linearly separable data. Our dataset are non-linearly separable data, so SVM was more accurate than perceptron. SVM predicted each binary-class well, whereas perceptron didn't predict well for class c2 as shown in decision surface (the previous figure).

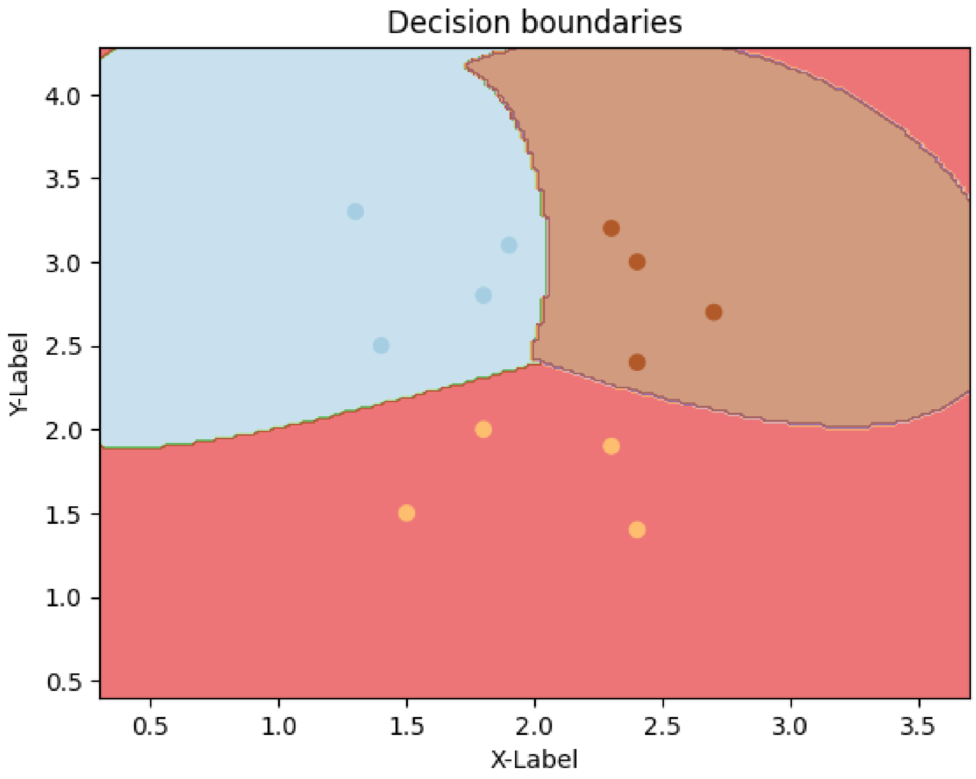
c) **Aggregation**

SVM-OVR Aggregation:

Confusion Matrix for training set: **confusion matrix for test set:**

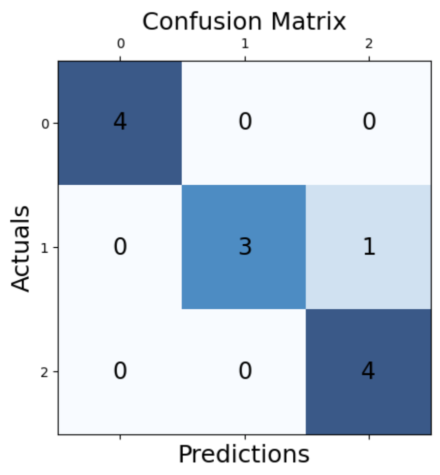






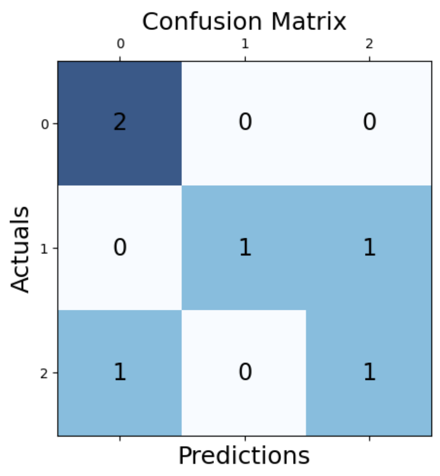
Perceptron-OVR Aggregation:

Confusion Matrix for training set: **confusion matrix for test set:**



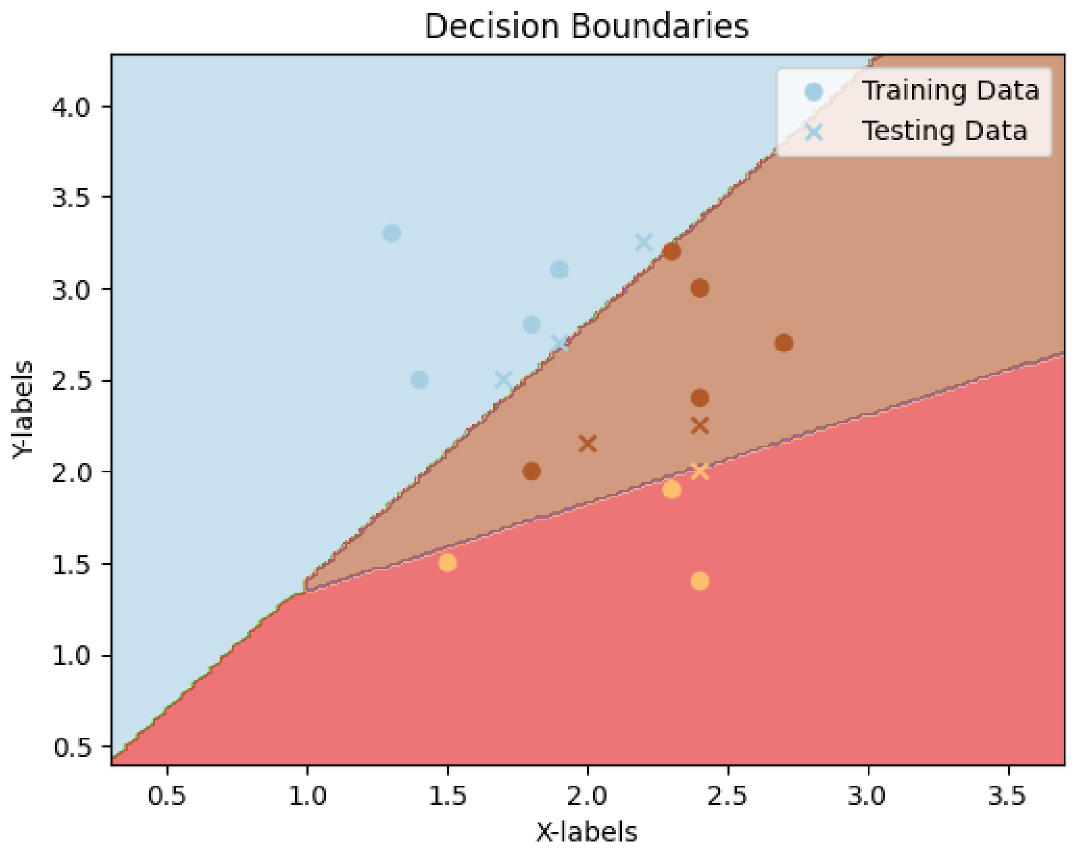
Confusion Matrix
Actuals
Predictions

Actuals \ Predictions	0	1	2
0	4	0	0
1	0	3	1
2	0	0	4



Confusion Matrix
Actuals
Predictions

Actuals \ Predictions	0	1	2
0	2	0	0
1	0	1	1
2	1	0	1

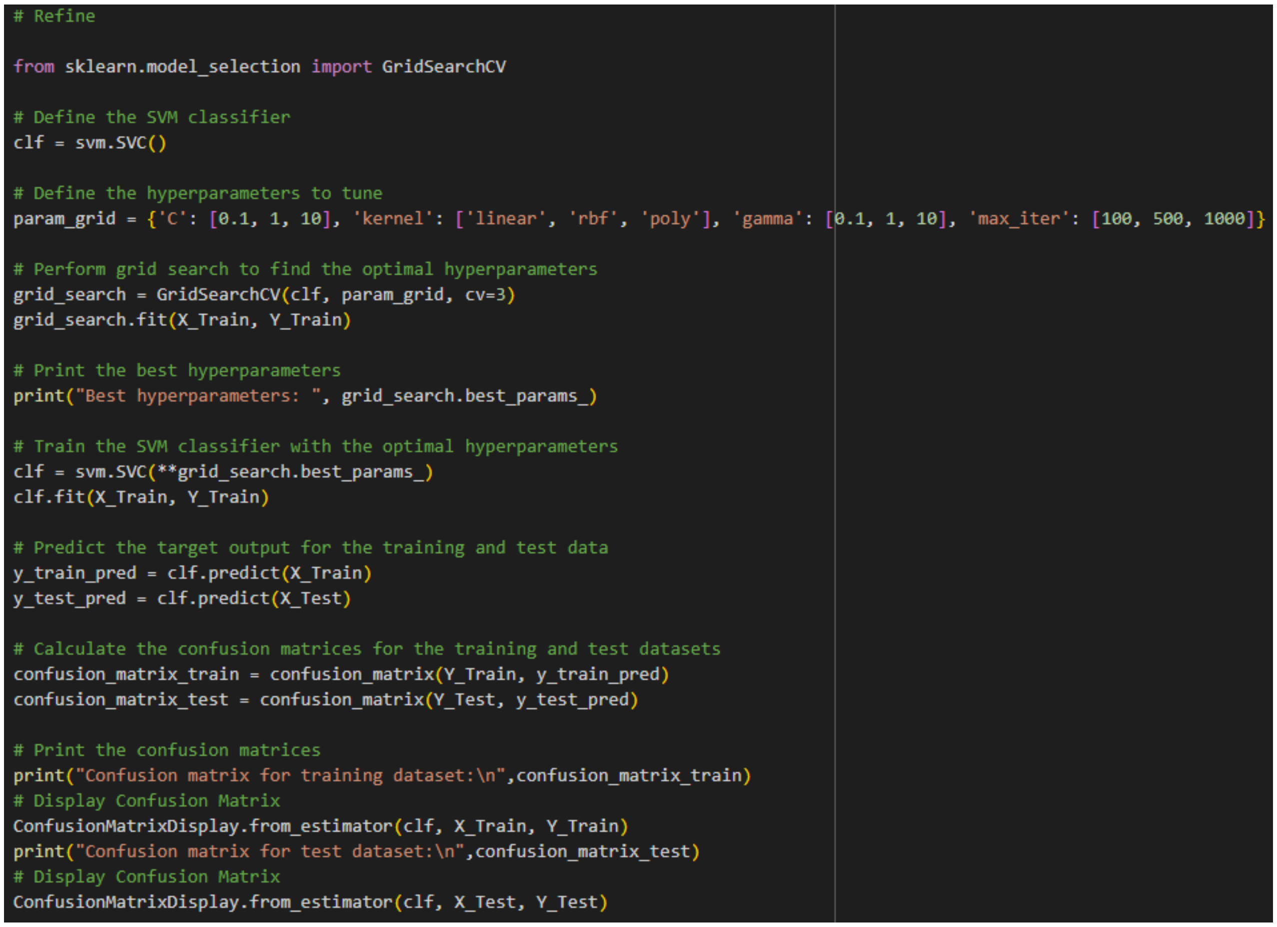


d) **Reason why SVM performance in (a) is different than aggregated performance of SVM in(c)**

The aggregation in section (a) perform well with accuracy 100% than in section (c) as: In section (a), The default SVM is a multi-classification which can handle problems with more than two classes and can provide more detailed information about the data than binary classification. In section (c), the aggregation applied for binary classifiers (c0, c1, c2), where the classes are not well-separated so the accuracy of the aggregation wasn't accurate as in multi-classification as shown in the confusion matrix.

Refine the default SVM by selecting the appropriate parameter:
we select some parameters to refine the SVM model, involves kernel, C, gamma and max_iter. This can help to improve the performance of SVM and ensure that it is well-suited to the classification problem.

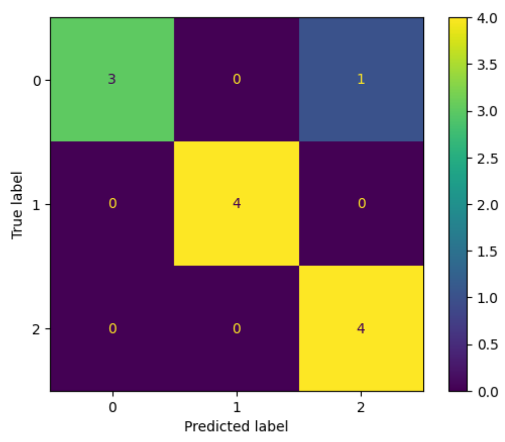
the code:



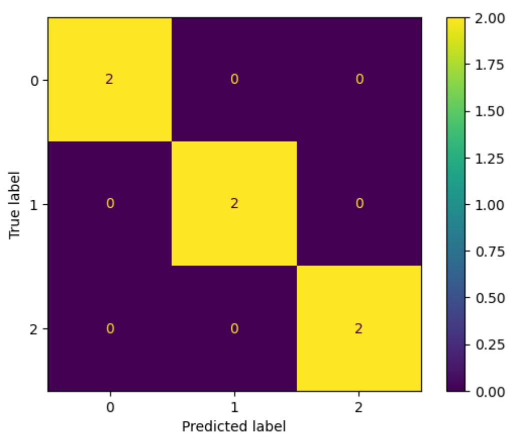
```python
# Refine

from sklearn.model_selection import GridSearchCV

# Define the SVM classifier
clf = svm.SVC()

# Define the hyperparameters to tune
param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf', 'poly'], 'gamma': [0.1, 1, 10], 'max_iter': [100, 500, 1000]}

# Perform grid search to find the optimal hyperparameters
grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X_Train, Y_Train)

# Print the best hyperparameters
print("Best hyperparameters: ", grid_search.best_params_)

# Train the SVM classifier with the optimal hyperparameters
clf = svm.SVC(**grid_search.best_params_)
clf.fit(X_Train, Y_Train)

# Predict the target output for the training and test data
y_train_pred = clf.predict(X_Train)
y_test_pred = clf.predict(X_Test)

# Calculate the confusion matrices for the training and test datasets
confusion_matrix_train = confusion_matrix(Y_Train, y_train_pred)
confusion_matrix_test = confusion_matrix(Y_Test, y_test_pred)

# Print the confusion matrices
print("Confusion matrix for training dataset:\n",confusion_matrix_train)
# Display Confusion Matrix
ConfusionMatrixDisplay.from_estimator(clf, X_Train, Y_Train)
print("Confusion matrix for test dataset:\n",confusion_matrix_test)
# Display Confusion Matrix
ConfusionMatrixDisplay.from_estimator(clf, X_Test, Y_Test)
```

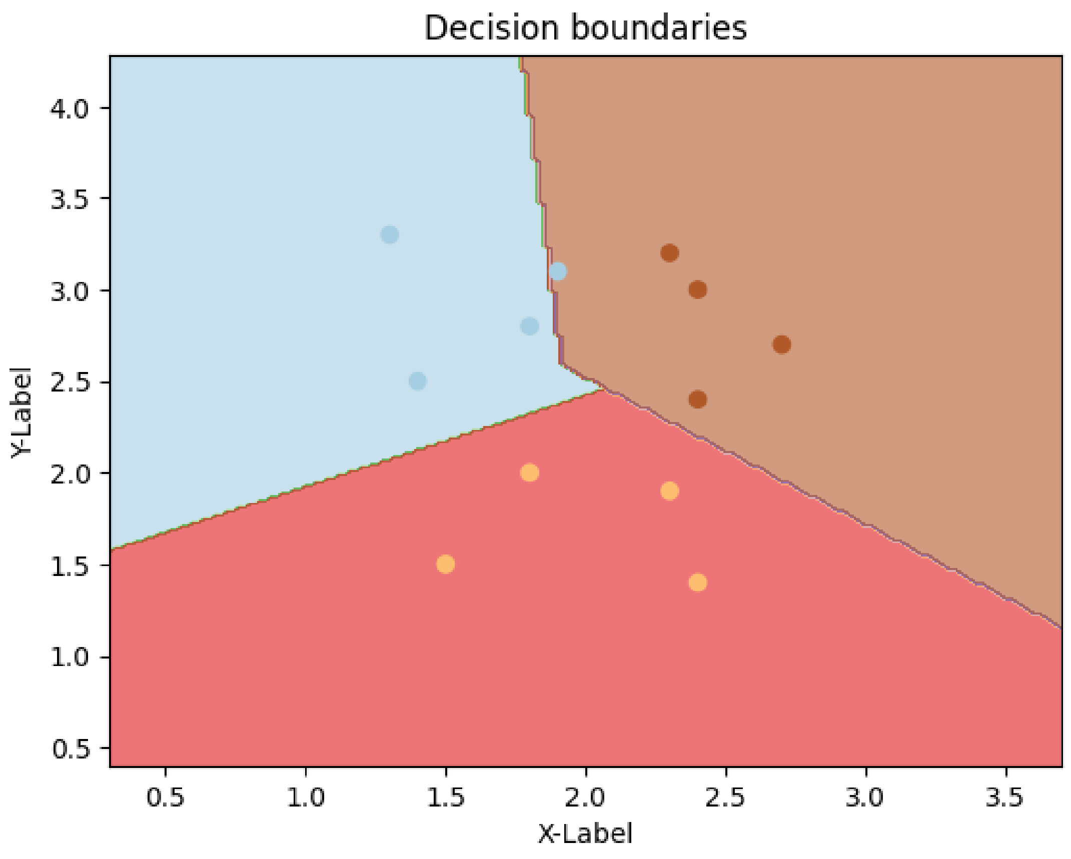
for class after refining:

Confusion Matrix for training set:



confusion matrix for test set:

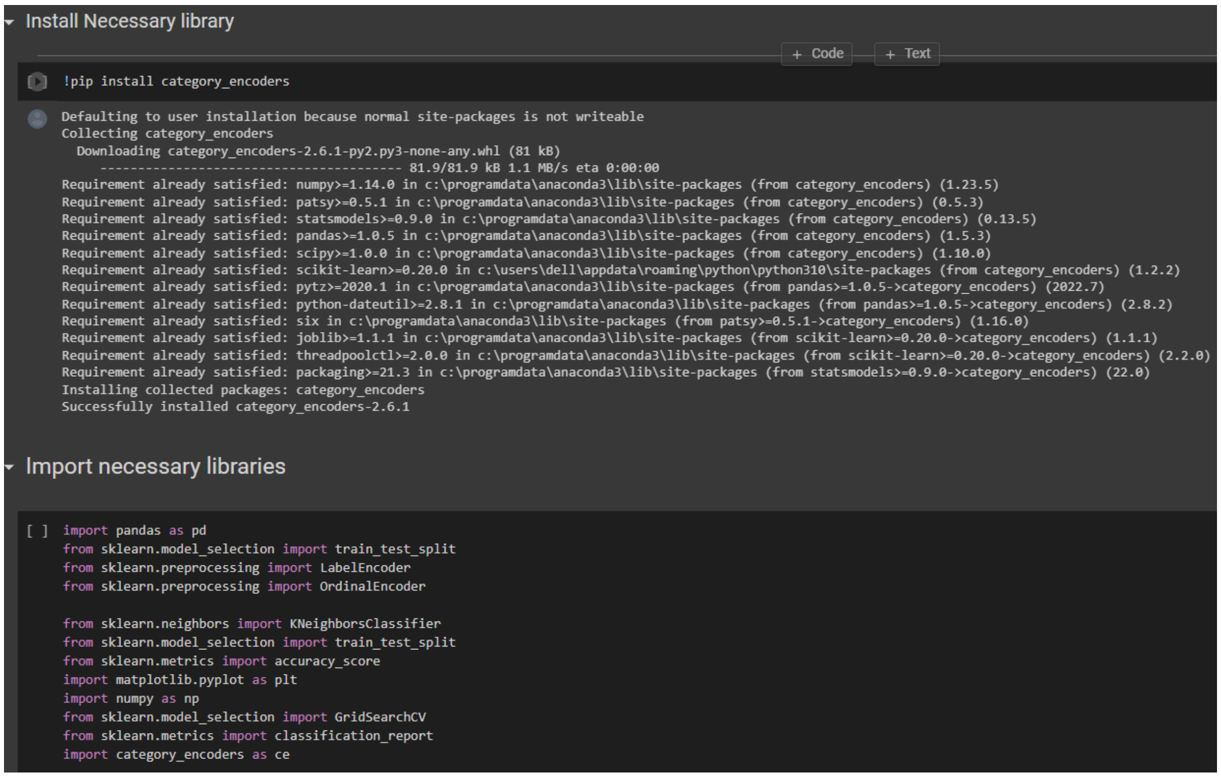




Problem 2:

we use car-evaluation dataset, downloaded from Kaggle and implement KNN classifier and answer some questions in the assignment:

firstly, we needed to Install and import libraries:



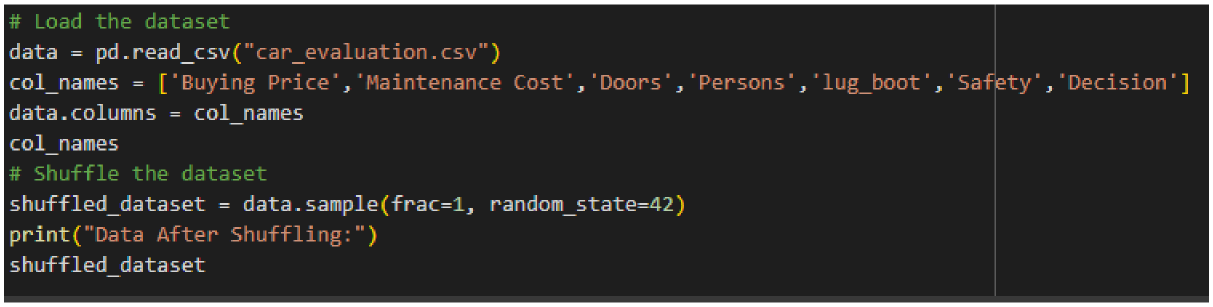
```
!pip install category_encoders
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting category_encoders
  Downloading category_encoders-2.6.1-py2.py3-none-any.whl (81 kB)
     ---------------------------------------- 81.9/81.9 kB 1.1 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.14.0 in c:\programdata\anaconda3\lib\site-packages (from category_encoders) (1.23.5)
Requirement already satisfied: patsy>=0.5.1 in c:\programdata\anaconda3\lib\site-packages (from category_encoders) (0.5.3)
Requirement already satisfied: statsmodels>=0.9.0 in c:\programdata\anaconda3\lib\site-packages (from category_encoders) (0.13.5)
Requirement already satisfied: pandas>=1.0.5 in c:\programdata\anaconda3\lib\site-packages (from category_encoders) (1.5.3)
Requirement already satisfied: scipy>=1.0.0 in c:\programdata\anaconda3\lib\site-packages (from category_encoders) (1.10.0)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\dell\appdata\roaming\python\python310\site-packages (from category_encoders) (1.2.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-packages (from pandas>=1.0.5->category_encoders) (2022.7)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\programdata\anaconda3\lib\site-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->category_encoders) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->category_encoders) (2.2.0)
Requirement already satisfied: packaging>=21.3 in c:\programdata\anaconda3\lib\site-packages (from statsmodels>=0.9.0->category_encoders) (22.0)
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.1
```

```
[ ] import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import LabelEncoder
    from sklearn.preprocessing import OrdinalEncoder

    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score
    import matplotlib.pyplot as plt
    import numpy as np
    from sklearn.model_selection import GridSearchCV
    from sklearn.metrics import classification_report
    import category_encoders as ce
```

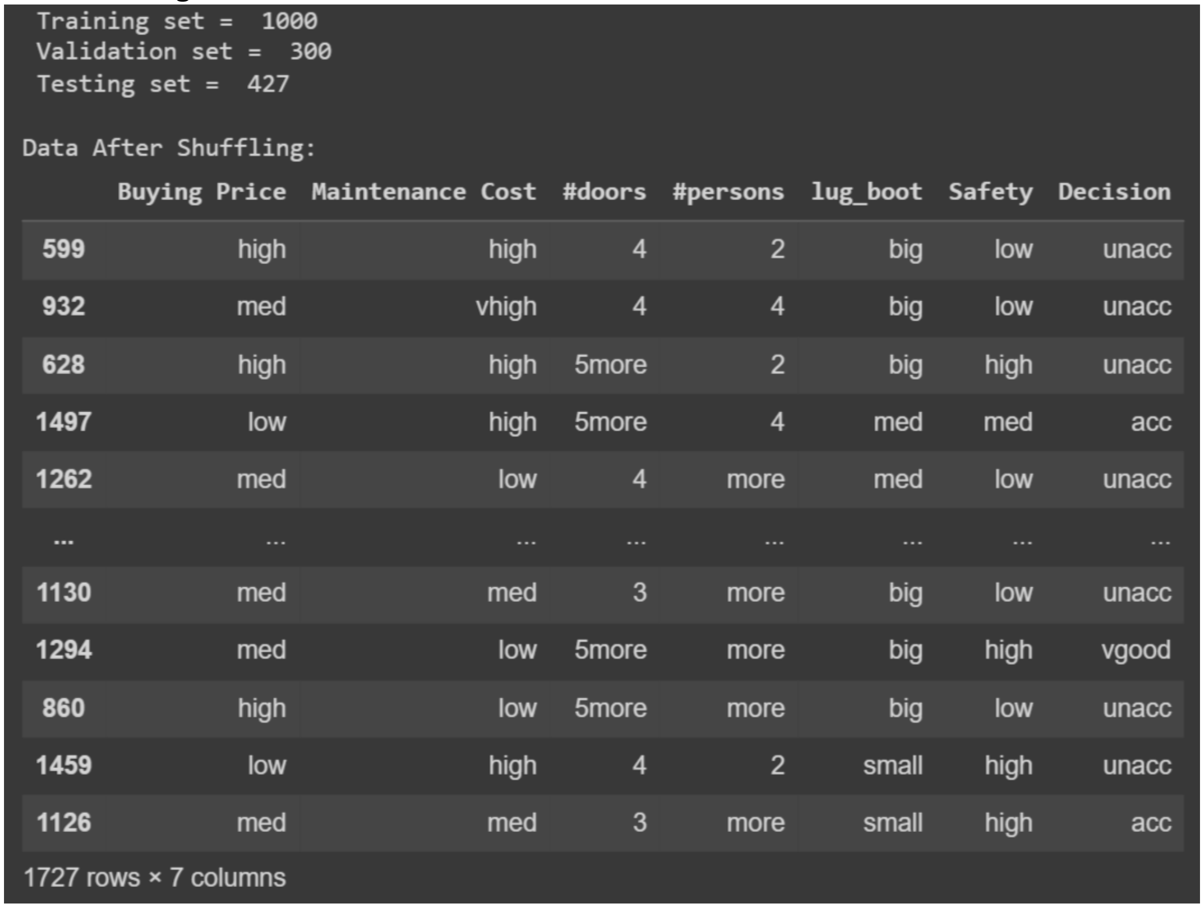
a) **shuffle the dataset and split the dataset**
Shuffling and splitting the dataset is an important step, as it helps to evaluate model performance, prevent overfitting, select the best model, and avoid bias.

The code:



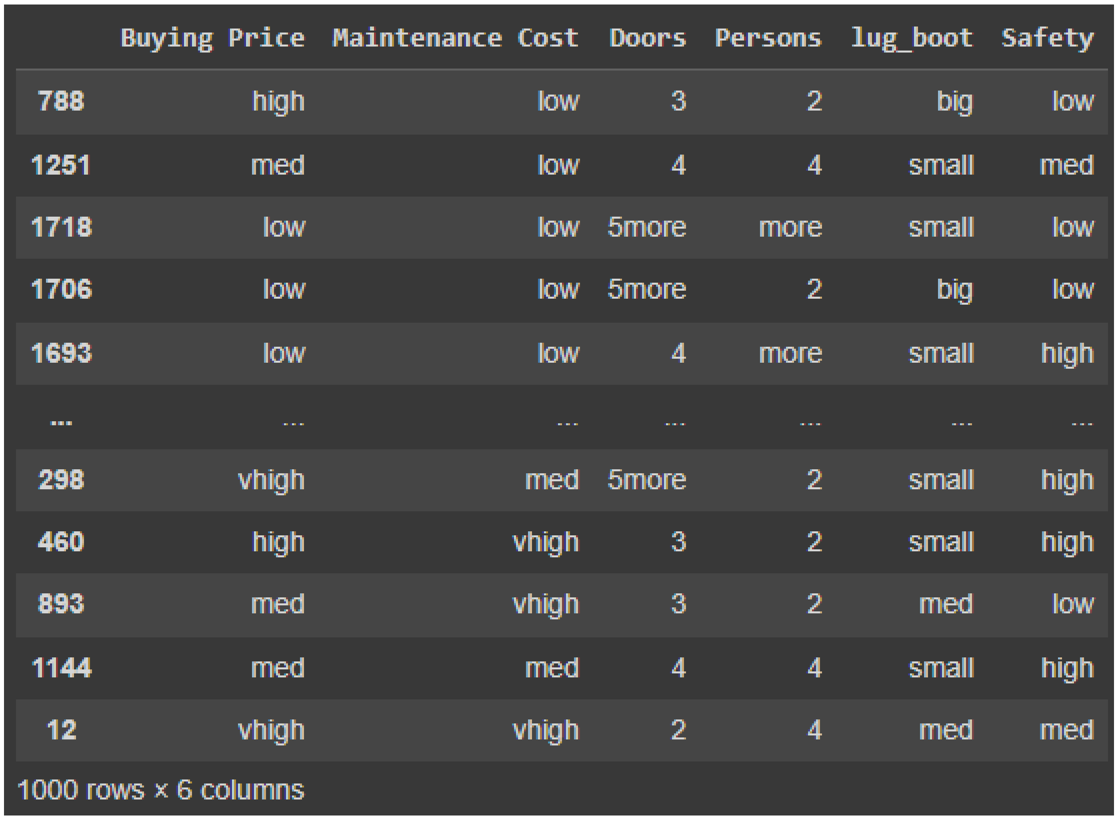
```
# Load the dataset
data = pd.read_csv("car_evaluation.csv")
col_names = ['Buying Price','Maintenance Cost','Doors','Persons','lug_boot','Safety','Decision']
data.columns = col_names
col_names
# Shuffle the dataset
shuffled_dataset = data.sample(frac=1, random_state=42)
print("Data After Shuffling:")
shuffled_dataset
```

The Output:
after shuffling dataset:



```
 Training set =  1000
 Validation set =  300
 Testing set =  427

Data After Shuffling:
      Buying Price  Maintenance Cost  #doors  #persons  lug_boot  Safety  Decision
 599          high              high       4         2       big     low     unacc
 932           med             vhigh       4         4       big     low     unacc
 628          high              high   5more         2       big    high     unacc
1497           low              high   5more         4       med     med       acc
1262           med               low       4      more       med     low     unacc
 ...           ...               ...     ...       ...       ...     ...       ...
1130           med               med       3      more       big     low     unacc
1294           med               low   5more      more       big    high     vgood
 860          high               low   5more      more       big     low     unacc
1459           low              high       4         2     small    high     unacc
1126           med               med       3      more     small    high       acc
1727 rows × 7 columns
```

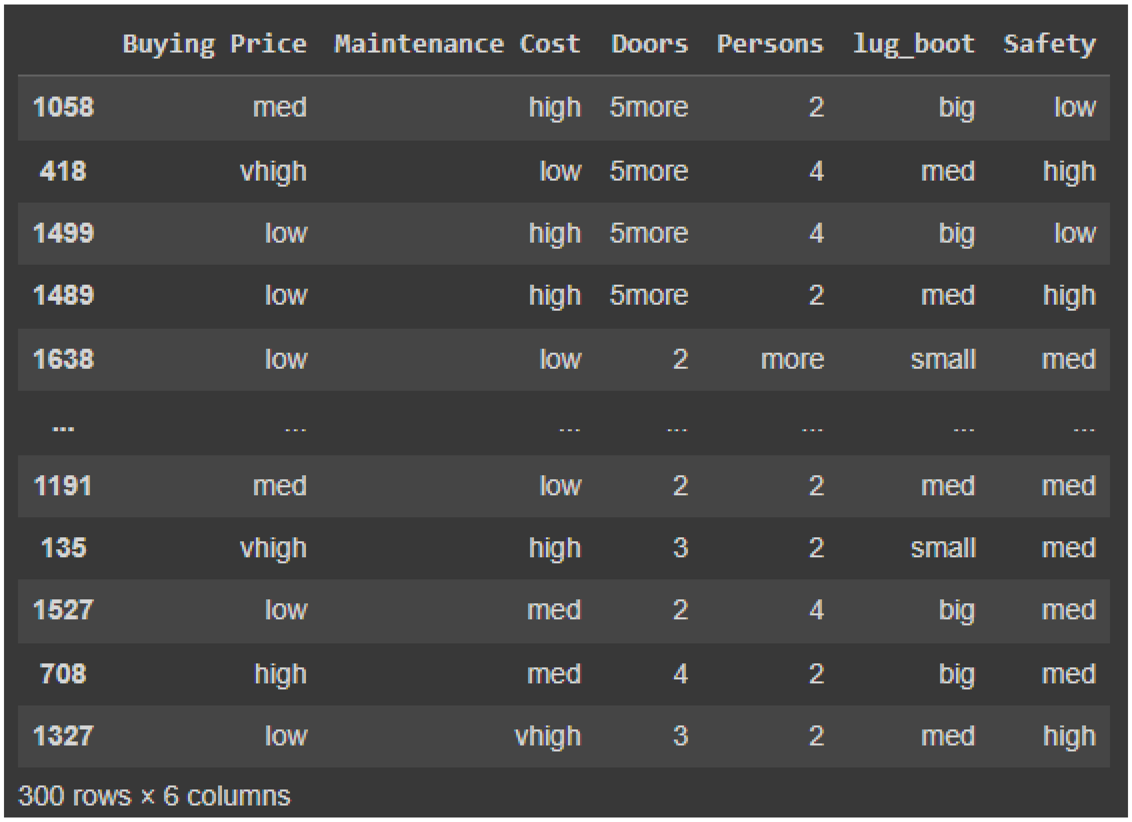
after splitting dataset:
Training set:



	Buying Price	Maintenance Cost	Doors	Persons	lug_boot	Safety
788	high	low	3	2	big	low
1251	med	low	4	4	small	med
1718	low	low	5more	more	small	low
1706	low	low	5more	2	big	low
1693	low	low	4	more	small	high
...	...	...	...	...	...	...
298	vhigh	med	5more	2	small	high
460	high	vhigh	3	2	small	high
893	med	vhigh	3	2	med	low
1144	med	med	4	4	small	high
12	vhigh	vhigh	2	4	med	med

1000 rows × 6 columns

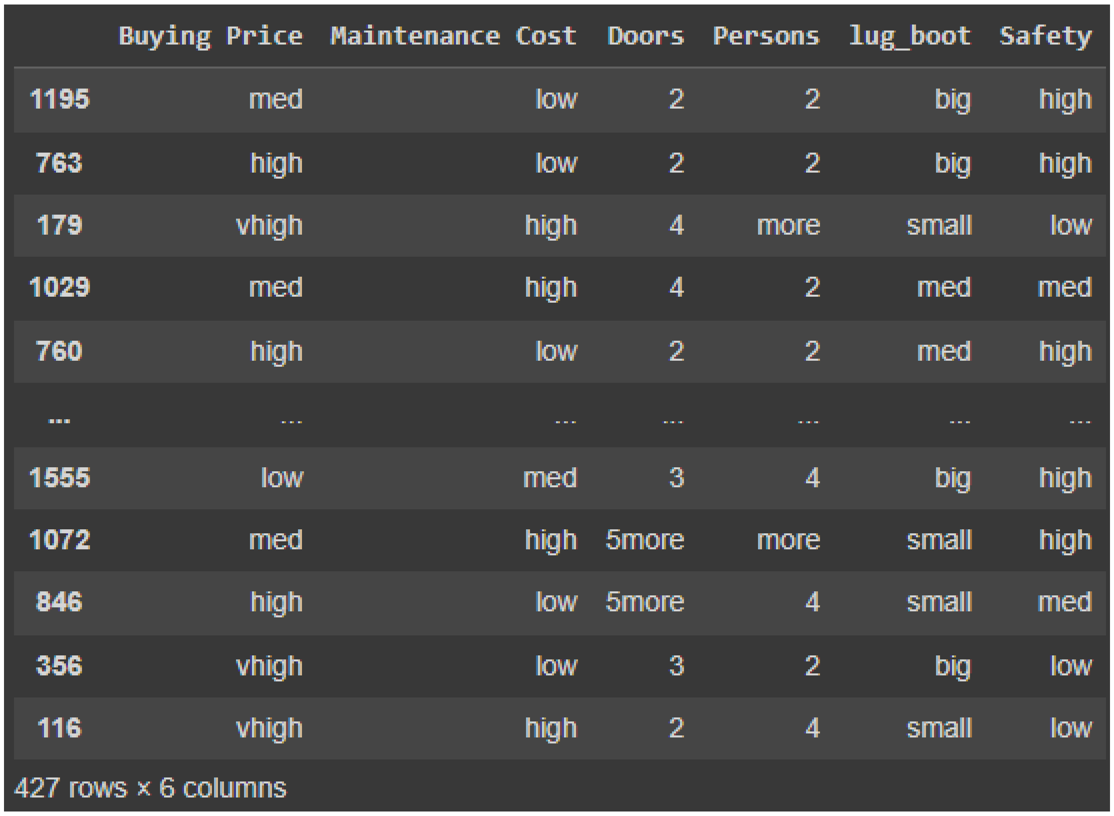
Validation set:



	Buying Price	Maintenance Cost	Doors	Persons	lug_boot	Safety
1058	med	high	5more	2	big	low
418	vhigh	low	5more	4	med	high
1499	low	high	5more	4	big	low
1489	low	high	5more	2	med	high
1638	low	low	2	more	small	med
...	...	...	...	...	...	...
1191	med	low	2	2	med	med
135	vhigh	high	3	2	small	med
1527	low	med	2	4	big	med
708	high	med	4	2	big	med
1327	low	vhigh	3	2	med	high

300 rows × 6 columns

Testing set:



	Buying Price	Maintenance Cost	Doors	Persons	lug_boot	Safety
1195	med	low	2	2	big	high
763	high	low	2	2	big	high
179	vhigh	high	4	more	small	low
1029	med	high	4	2	med	med
760	high	low	2	2	med	high
...	...	...	...	...	...	...
1555	low	med	3	4	big	high
1072	med	high	5more	more	small	high
846	high	low	5more	4	small	med
356	vhigh	low	3	2	big	low
116	vhigh	high	2	4	small	low

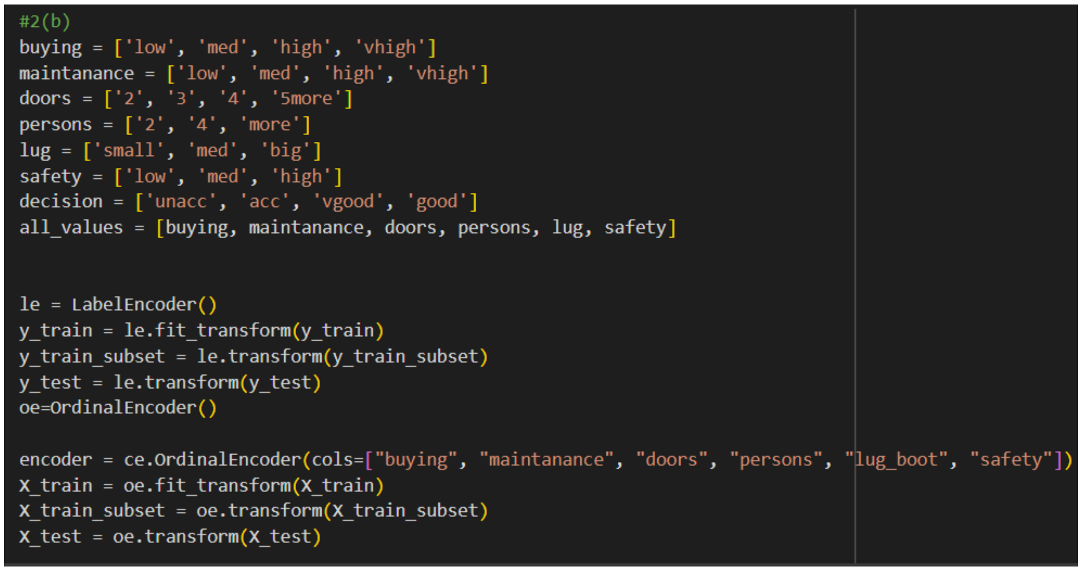
427 rows × 6 columns

b) transform the string values into numbers

K-Nearest Neighbors (KNN) is designed to work with numerical data, not categorical data. The KNN algorithm is a type of instance-based learning, where the prediction for a new data point is based on the "k" closest training examples in feature space. The distance between the instances is computed based on the numerical values of the features. So, we transformed categorical features into numerical values before they can be used with KNN.

We use a technique called "label encoding", which assigns a unique integer value to each category. We also used Ordinal encoding, which is a technique for encoding categorical variables as numerical values based on the order or rank of the categories. We also use Ordinal encoding, a technique for encoding categorical variables as numerical values based on the order or rank of the categories.

the code:

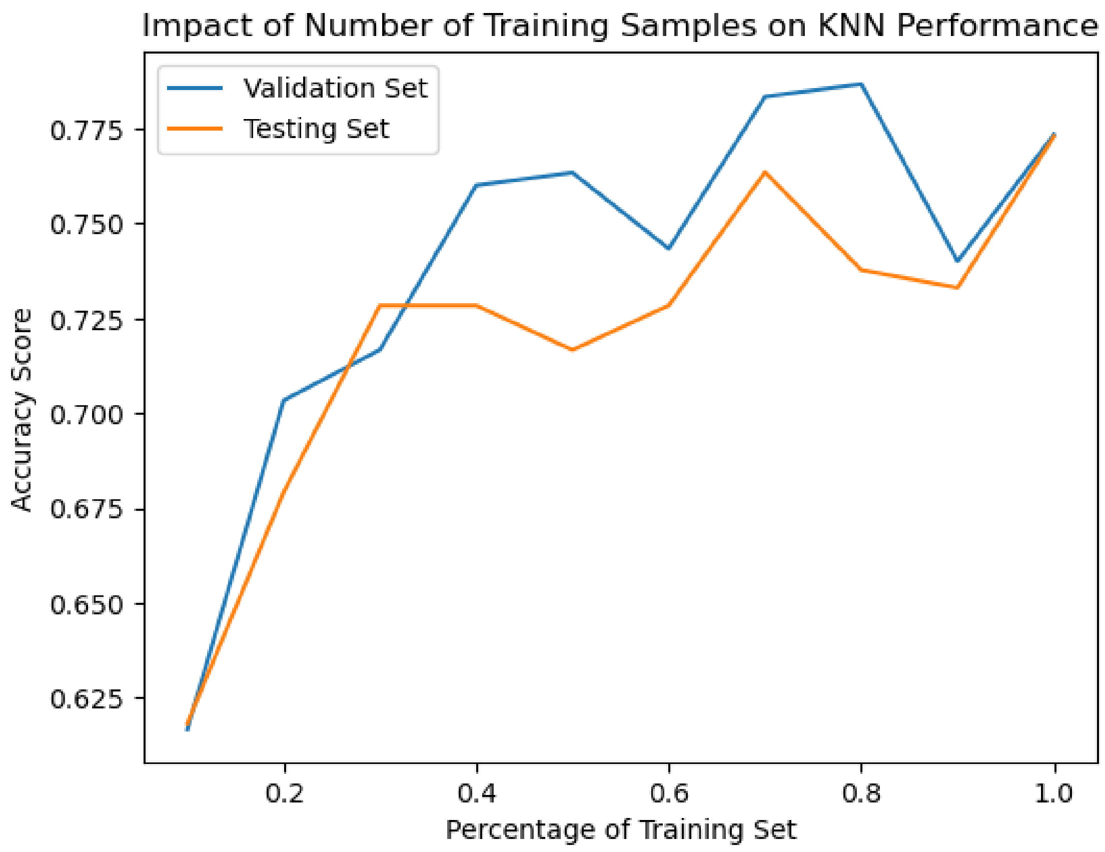


```
#2(b)
buying = ['low', 'med', 'high', 'vhigh']
maintanance = ['low', 'med', 'high', 'vhigh']
doors = ['2', '3', '4', '5more']
persons = ['2', '4', 'more']
lug = ['small', 'med', 'big']
safety = ['low', 'med', 'high']
decision = ['unacc', 'acc', 'vgood', 'good']
all_values = [buying, maintanance, doors, persons, lug, safety]


le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_train_subset = le.transform(y_train_subset)
y_test = le.transform(y_test)
oe=OrdinalEncoder()

encoder = ce.OrdinalEncoder(cols=["buying", "maintanance", "doors", "persons", "lug_boot", "safety"])
X_train = oe.fit_transform(X_train)
X_train_subset = oe.transform(X_train_subset)
X_test = oe.transform(X_test)
```

c) different number of training samples
the code:

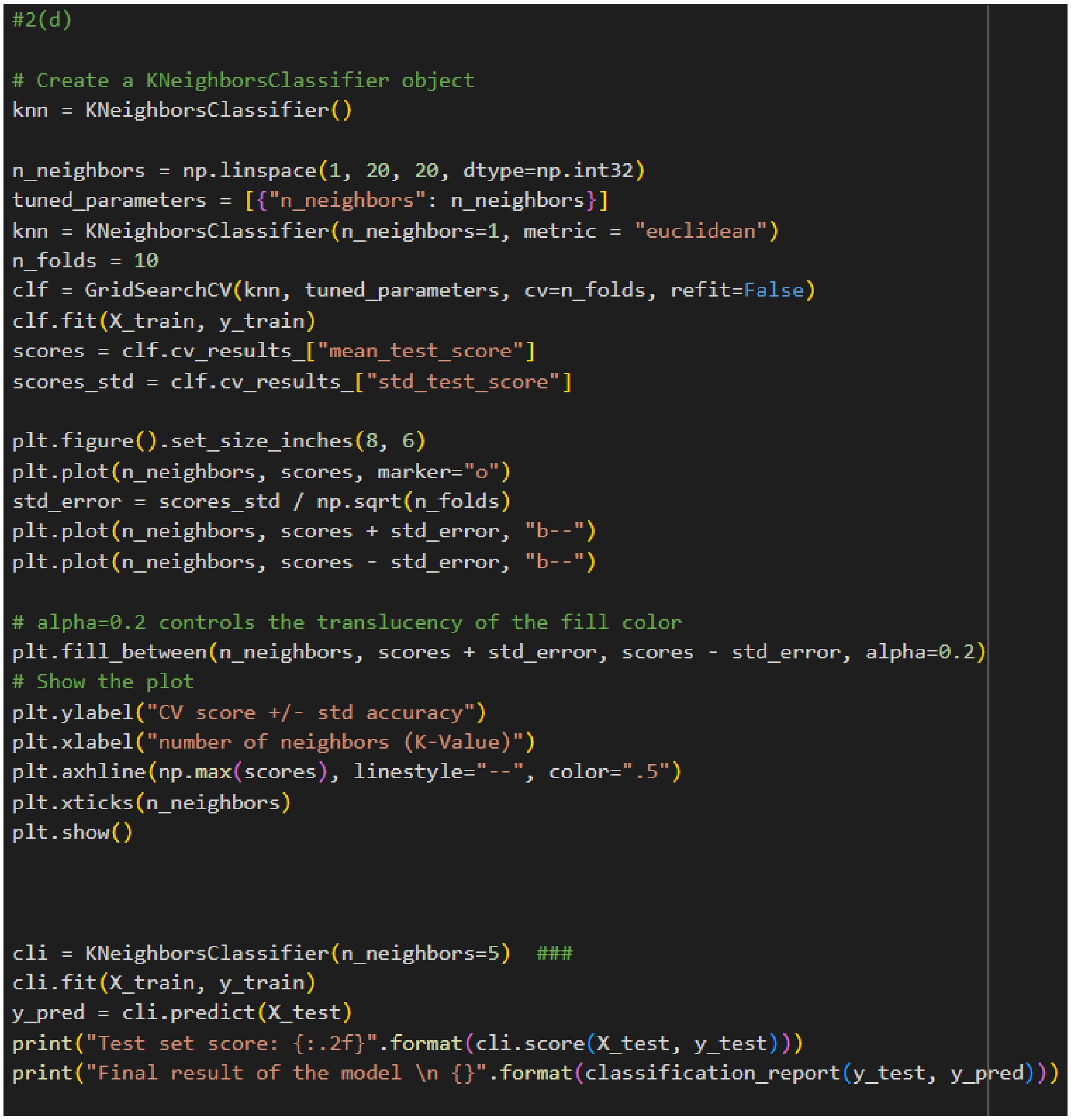
```
val_scores = []
test_scores = []
# Define a list of percentages of the training set to use for each classifier
train_percentages = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

# Iterate over each percentage and train a KNN classifier
for train_ratio in train_percentages:

    train_size = int(train_ratio * len(y_train))

    partial_X_train = X_train[:train_size]
    partial_y_train = y_train[:train_size]


    # Initialize the KNN classifier
    knn = KNeighborsClassifier(n_neighbors=2)
    knn.fit(partial_X_train, partial_y_train)
    # Train the KNN classifier on the subset of training data
    #knn.fit(X_train, y_train)
    y_valid_pred=knn.predict(X_train_subset)
    y_test_pred=knn.predict(X_test)
    # Append the accuracy scores to the respective lists
    val_accuracy = accuracy_score(y_train_subset, y_valid_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    val_scores.append(val_accuracy)
    test_scores.append(test_accuracy)

# Plot the accuracy scores for validation and testing sets
plt.plot(train_percentages, val_scores, label='Validation Set')
plt.plot(train_percentages, test_scores, label='Testing Set')
plt.xlabel('Percentage of Training Set')
plt.ylabel('Accuracy Score')
plt.title('Impact of Number of Training Samples on KNN Performance')
plt.legend()
plt.show()
```

the output:



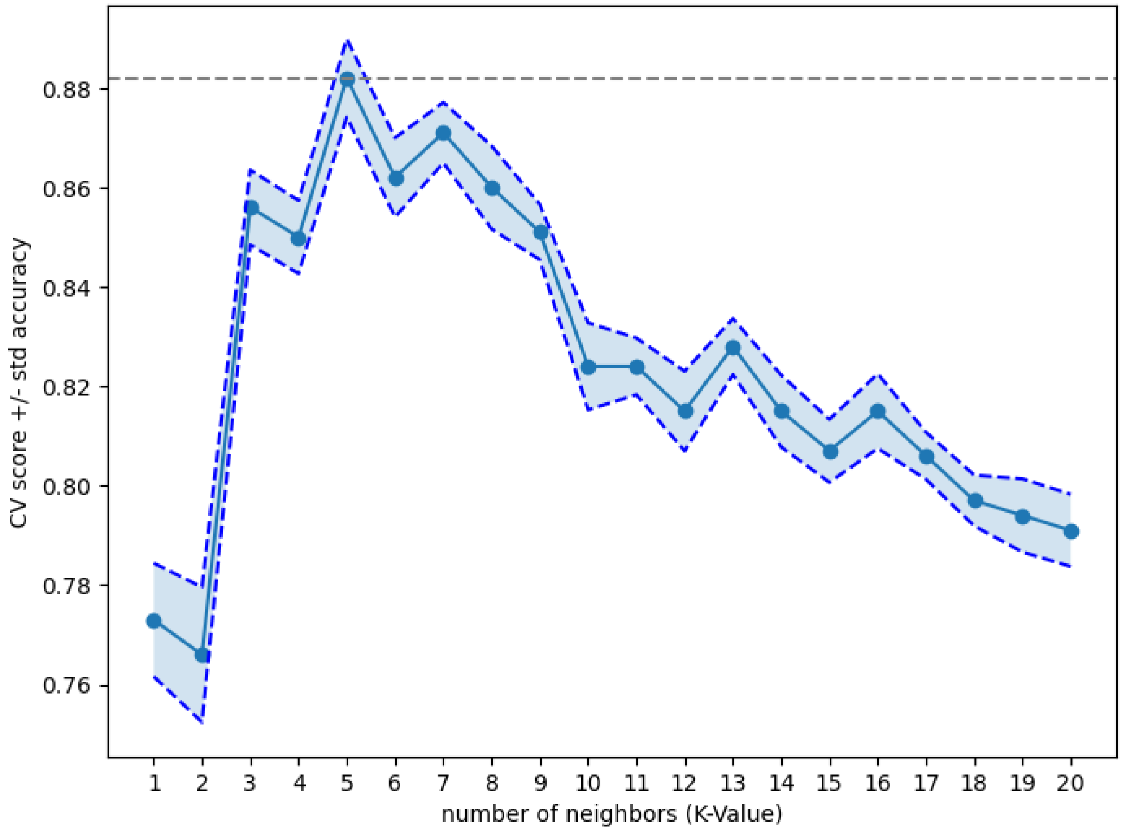
The resulting plot shows how the performance of the KNN classifier varies as a function of the number of training samples used:
We can see that as we increase the number of training samples, the accuracy of the KNN classifier on the validation set and the testing set both increases. However, the rate of improvement slows down as we approach 100% of the training set. We can also see that the accuracy on the testing set is slightly lower than the accuracy on the validation set, which is expected since the model was not trained on the testing set. Overall, this plot demonstrates the importance of having a sufficient number of training samples to achieve good performance with a KNN classifier.

d) **find the best K value**
using different values of k from 1 to 10 to train KNN classifier and evaluate its performance
the code:



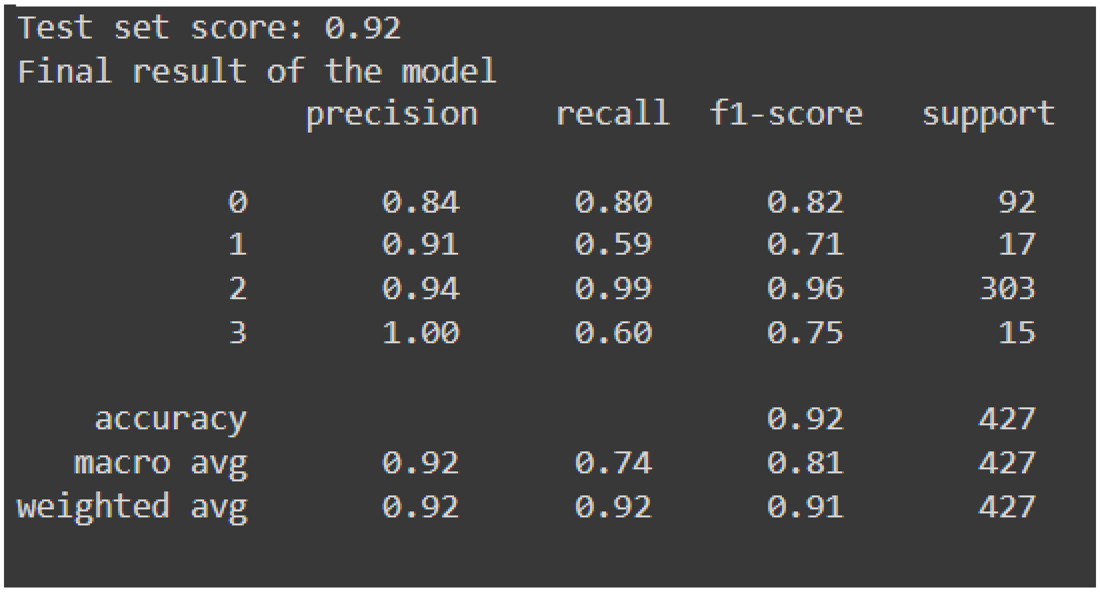
```python
#2(d)

# Create a KNeighborsClassifier object
knn = KNeighborsClassifier()

n_neighbors = np.linspace(1, 20, 20, dtype=np.int32)
tuned_parameters = [{"n_neighbors": n_neighbors}]
knn = KNeighborsClassifier(n_neighbors=1, metric = "euclidean")
n_folds = 10
clf = GridSearchCV(knn, tuned_parameters, cv=n_folds, refit=False)
clf.fit(X_train, y_train)
scores = clf.cv_results_["mean_test_score"]
scores_std = clf.cv_results_["std_test_score"]

plt.figure().set_size_inches(8, 6)
plt.plot(n_neighbors, scores, marker="o")
std_error = scores_std / np.sqrt(n_folds)
plt.plot(n_neighbors, scores + std_error, "b--")
plt.plot(n_neighbors, scores - std_error, "b--")

# alpha=0.2 controls the translucency of the fill color
plt.fill_between(n_neighbors, scores + std_error, scores - std_error, alpha=0.2)
# Show the plot
plt.ylabel("CV score +/- std accuracy")
plt.xlabel("number of neighbors (K-Value)")
plt.axhline(np.max(scores), linestyle="--", color=".5")
plt.xticks(n_neighbors)
plt.show()



cli = KNeighborsClassifier(n_neighbors=5)  ###
cli.fit(X_train, y_train)
y_pred = cli.predict(X_test)
print("Test set score: {:.2f}".format(cli.score(X_test, y_test)))
print("Final result of the model \n {}".format(classification_report(y_test, y_pred)))
```

the output:



We can see that the accuracy on the validation set is highest when K=5, and decreases as K increases beyond that. This demonstrates the importance of choosing an appropriate value of K for a KNN classifier to achieve good performance.

The results of



```
Test set score: 0.92
Final result of the model
              precision    recall  f1-score   support

           0       0.84      0.80      0.82        92
           1       0.91      0.59      0.71        17
           2       0.94      0.99      0.96       303
           3       1.00      0.60      0.75        15

    accuracy                           0.92       427
   macro avg       0.92      0.74      0.81       427
weighted avg       0.92      0.92      0.91       427
```

e) **conclusions from the experiments of question (c) and (d)**
Based on the experiments conducted in questions (c) and (d), we can draw the following conclusions:

- The performance of the KNN classifier generally improves as the size of the training set increases. This is because a larger training set provides more information for the classifier to learn from and can help to reduce overfitting.

- The optimal value of K for the KNN classifier will depend on the specific dataset and problem you are working with. In our experiments in question (d), we found that the performance of the KNN classifier generally improved as K increased from 1 to around 5, and then decreased slightly as K continued to increase. However, the optimal value of K =5

It is important to carefully choose the size of the training set and the value of K for the KNN classifier.