Applied Machine Learning

ASSIGNMENT1

Prepared by:
Ahmed Ahmed
Hadir Hassan
Donia Abdelreheem

Supervised by:

DR/Murat

Q1) Task 1

a)

First, we loaded the data at the figure (1) below.

▾ Load DataSet

```
[18] #Define the training and test datasets
     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])
```

Figure 1 load the data set.

We created default SVC from class SVM and Train the Data and get the result.

▾ Defult SVC Model

```
SVM=svm.SVC() # Create an instance of the SVM classifier
SVM.fit(X_Train,Y_Train) # Train the SVM classifier using the training data
Y_Pred=SVM.predict(X_Test) # test the model
print(Y_Pred)
```

Figure 2 building a default SVM Model

We display the Confusion Matrix to ensure the performance is done in the right way or not.

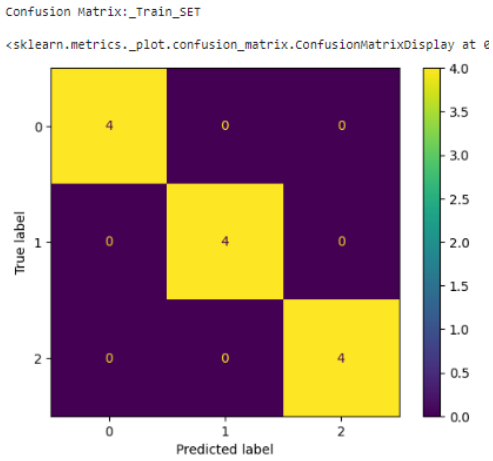

Figure 4 Confusion matrices for SVM default Training set

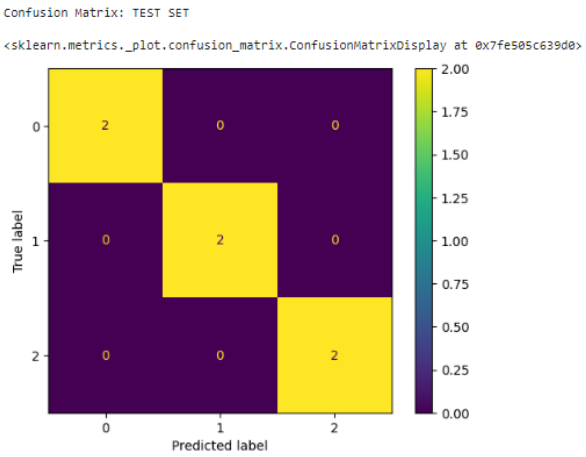

Figure 3 Confusion matrices for SVM default test set

We display the Decision boundary

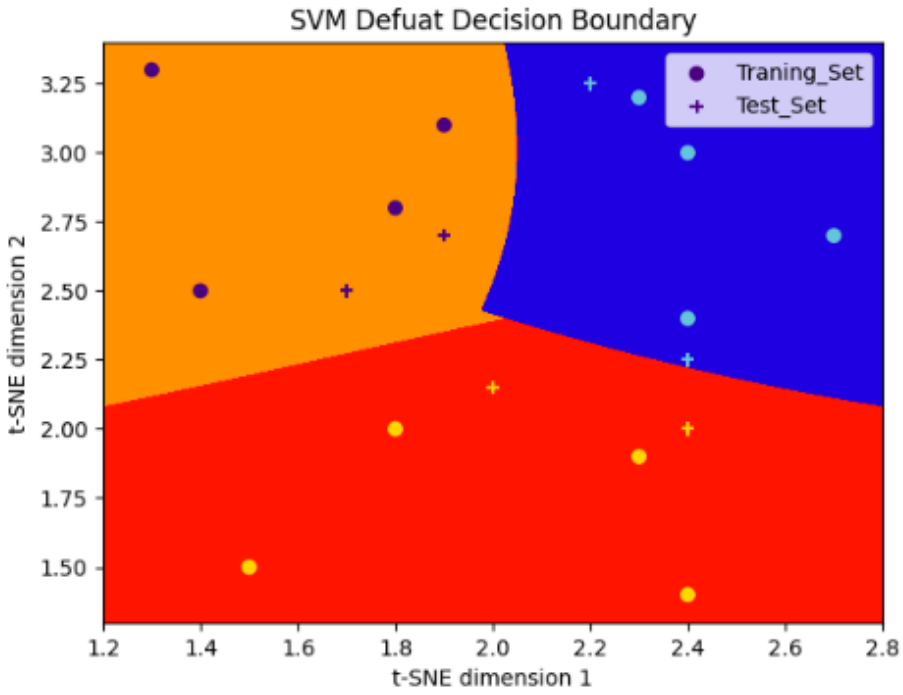

Figure 5

b)

We get instance from the class SVC and now we prepared it to differ between two classes 0 and 1,2

```
[3]  # This function preprocesses the labels of the training and test sets by converting labels that match the specified number_class to 1, and all other labels to 0.
def preprocceing(number_class,Y_Train,Y_Test):
  for i in range(len(Y_Train)):
    if Y_Train[i]!=number_class:
       Y_Train[i]=0
    else:
      Y_Train[i]=1
  for i in range(len(Y_Test)):
    if Y_Test[i]!=number_class:
         Y_Test[i]=0
    else:
        Y_Test[i]=1
```

Figure 6 the function to prepare the data for every class.

Here in figure 6 the function gets the number of class that will be trained in model to be detected and the test data that will be tested on it. Our function renames the target column to be 1 for the class that we train the model on it and 0 for other's classes.

```
SVM_0=svm.SVC(kernel='linear',probability=True,random_state=14)
SVM_0.fit(X_Train,Y_Train_0)  # Train the SVM model on the training data with the modified labels for class 0
print('accuracy_training ',SVM_0.score(X_Train,Y_Train_0))  # Evaluate the accuracy of the trained model on the training data with the modified labels for class 0
```

```
accuracy_training  0.8333333333333334
```

```
[46] Y_pro_0=SVM_0.predict_proba(X_Test) # Generate the predicted class probabilities for the test set using the trained SVM model with modified labels for class 0
     Y_pryd=SVM_0.predict(X_Test)
     SVM_0.score(X_Test,Y_Test_0) # Evaluate the accuracy of the trained model on the test data with the modified labels for class 0
     print(Y_pro_0)
```

```
[[0.5246027  0.4753973 ]
 [0.56950992 0.43049008]
 [0.80448775 0.19551225]
 [0.93469685 0.06530315]
 [0.53797872 0.46202128]
 [0.90550849 0.09449151]]
```

Figure 7 load the SVM model.

In figure 7 we defined the model and made the kernel linear and turned on the probability to get the probability of every element that the model convinces the element in class 0.

Confusion matrices for class 0 vs Class 1 ,2

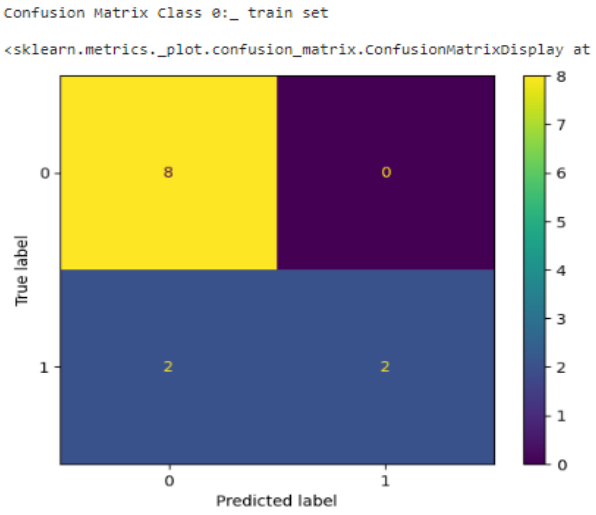

Figure 9 the Confusion matrices for SVM class 0 Training set.

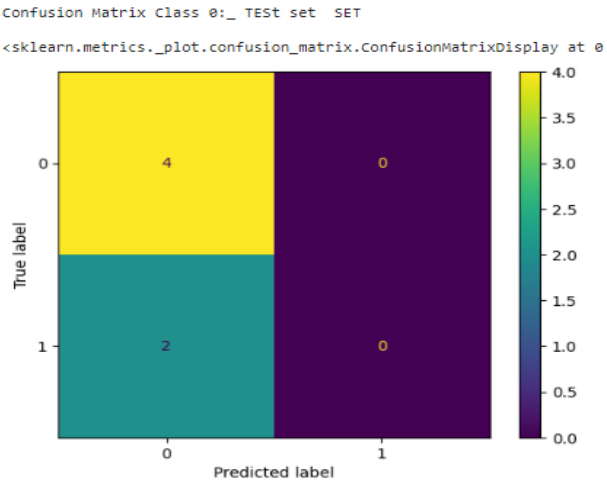

Figure 8 Confusion matrices for SVM class 0 test set.

Here in figure 8 and 9 we ensure the model performed in right way or not

And this is the decision boundary:

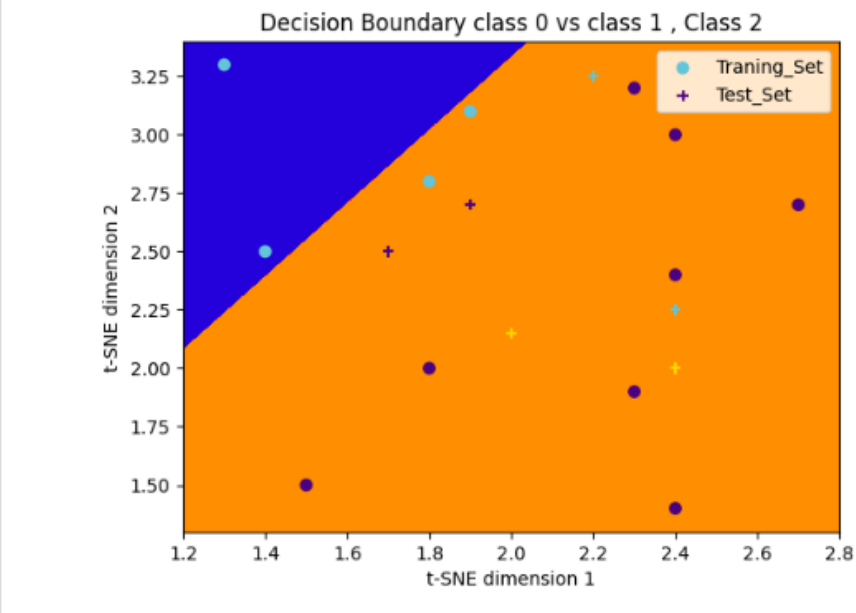

Figure 10 the decision boundary for SVM Class 0.

In figure 10 the decision boundary shows us the model not good in all situations

Class 1 VS Class 0, Class 2

```
[26] Y_Train_1,Y_Test_1=preprocceing(1,Y_Train.copy(),Y_Test.copy())
```

```
[27] SVM_1=svm.SVC(kernel='linear',probability=True,random_state=14)
     SVM_1.fit(X_Train,Y_Train_1)
     SVM_1.score(X_Train,Y_Train_1)

     1.0
```

Figure 11 prepare data set and load SVM for class 1

In figure 11 we used preprocessing to prepare data set and load it to fit the SVM model

Confusion matrices for class 1vs Class 0 ,2

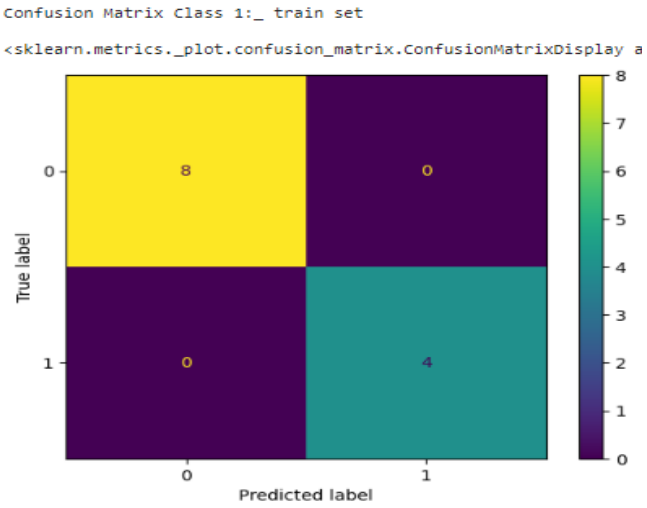

Figure 13 Confusion matrices for SVM class 1 Training set

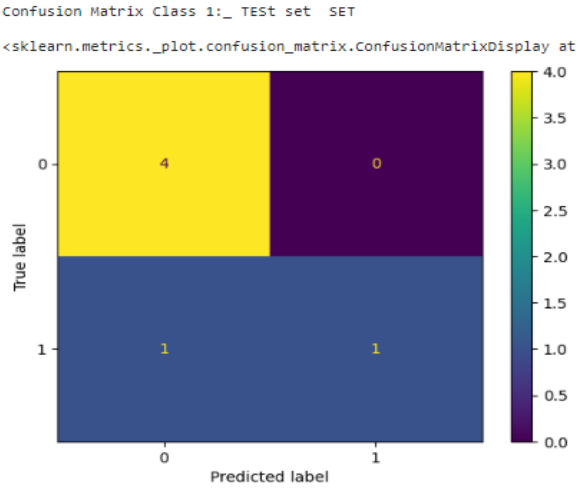

Figure 12 Confusion matrices for SVM class 1 test set

Here in figure 12 and 13 we ensure the model performed in right way or not

And this is the decision boundary:

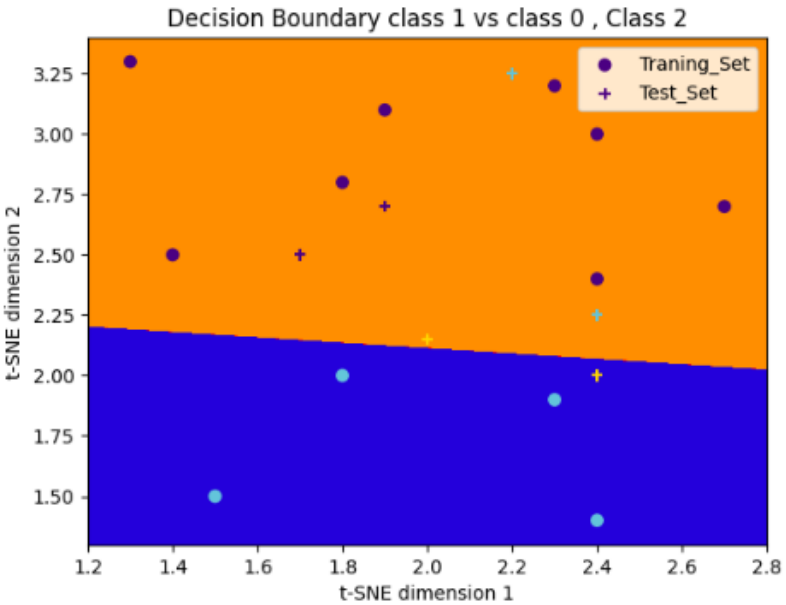

Figure 14 the decision boundary for SVM Class 1.

In figure 14 the decision boundary shows us the model not good in all situations

Class 2 VS Class 0, Class 1

```
[33] Y_Train_2,Y_Test_2=preprocceing(2,Y_Train.copy(),Y_Test.copy())
```

```
[34] SVM_2=svm.SVC(kernel='linear',probability=True,random_state=14)
     SVM_2.fit(X_Train,Y_Train_2)
     SVM_2.score(X_Train,Y_Train_2)

     1.0
```

Figure 15 prepare the data set and load model.

in figure 15 prepare The dataset and load the SVM for Class 2 and fit with this dataset

Confusion matrices for class 2 vs Class 0 ,1:

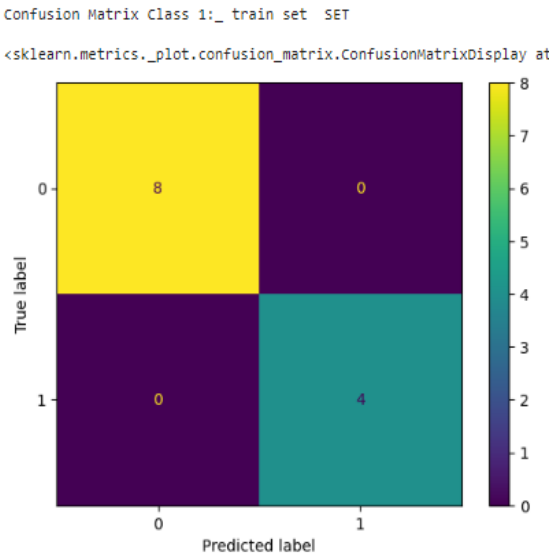

Figure 17 Confusion matrices for SVM class 2 Training set

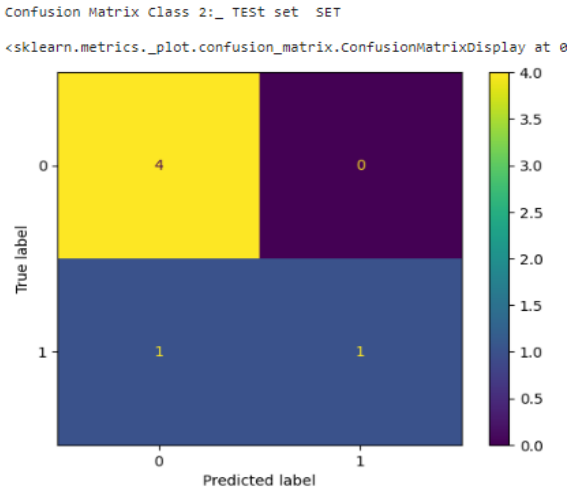

Figure 16 Confusion matrices for SVM class 2 Test set

Here in figures 16 and 17 we ensure the model performed in the right way or not.

And this is the decision boundary:

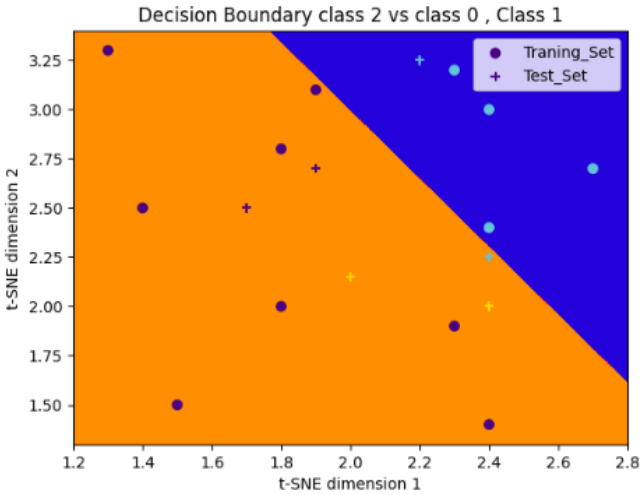

*Figure 18 the decision boundary for SVM Class **2***

In figure 18 the decision boundary shows us the model good in all situations

Perceptron default:

```
[45] per = Perceptron(random_state=1)
     per.fit(X_Train,Y_Train) # Train the Perceptron classifier using the training data
     print(" accurcy trianing :",per.score(X_Train,Y_Train)) # Compute the classification accuracy of the Perceptron classifier on the training data
     print(" accurcy Test :",per.score(X_Test,Y_Test))
```

```
 accurcy trianing : 0.9166666666666666
 accurcy Test : 0.8333333333333334
```

```
[46] print("reslut form Perceptron \n",per.decision_function(X_Test) )
```

```
reslut form Perceptron
 [[-0.07  -2.73  -1.95 ]
 [-0.21  -3.23  -0.89 ]
 [-1.19   1.785 -1.205]
 [-2.24   5.24   0.2  ]
 [-0.07  -6.005  1.025]
 [-1.89   3.215  0.525]]
```

Figure 19 prepare the data set and load model.

in figure 19 load model with random state 1 and fit the model and get the accuracy

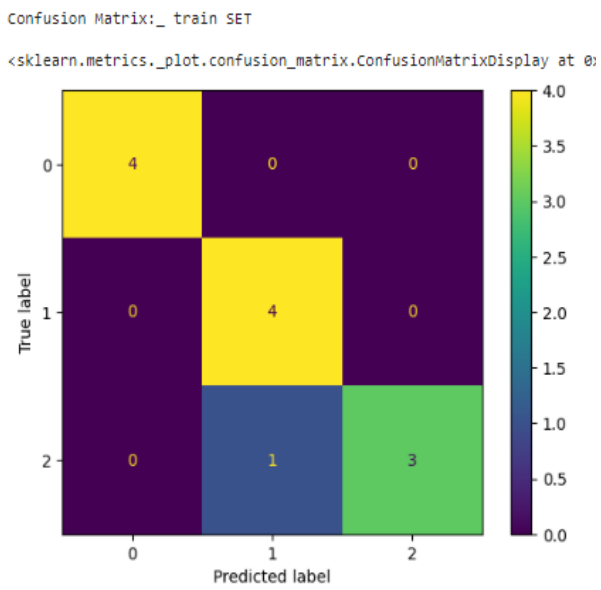

Figure 21 Confusion matrices for Perceptron default Training set

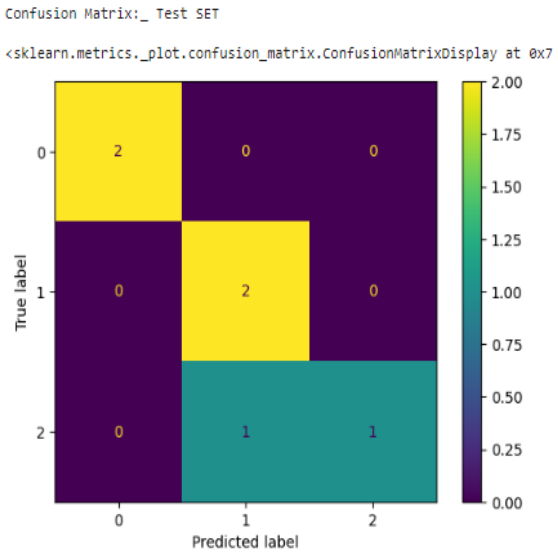

Figure 20 Confusion matrices for Perceptron default Test set

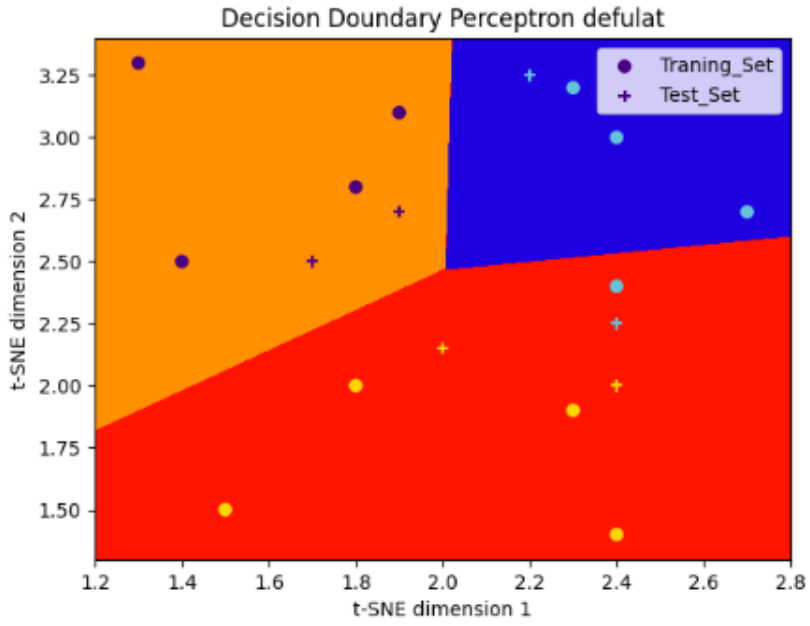

Figure 22 the decision boundary for Perceptron default

In model Perceptron default after training and testing the Perceptron default does not perform well in this situation

Perceptron Vs Rest

Perceptron class *0*

```
[72] Y_Train_0,Y_Test_0=preprocceing(0,Y_train.copy(),Y_Test.copy())
```

```
[73] perceptron_0 = Perceptron(random_state=10)  # Create a Perceptron object with a fixed random state
     perceptron_0.fit(X_Train,Y_Train_0)  # Train the Perceptron classifier on the training data for class 0
     print("accuracy Traning  : " , perceptron_0.score(X_Train,Y_Train_0))  # Compute the classification accuracy of the Perceptron classifier on the training data for class 0
     Y_pred=perceptron_0.predict(X_Test) # to predict the class labels for the test set (X_Test).
     print("accuracy Testing  : " ,perceptron_0.score(X_Test,Y_Test_0)) #to calculate the accuracy of the predicted labels compared to the true labels for the test set (Y_Test_0).

     accuracy Traning  :  1.0
     accuracy Testing  :  0.5
```

Figure 23 prepare the data set and load model.

in figure 23 load model for class 0 with random state 10 and fit the model and get the accuracy

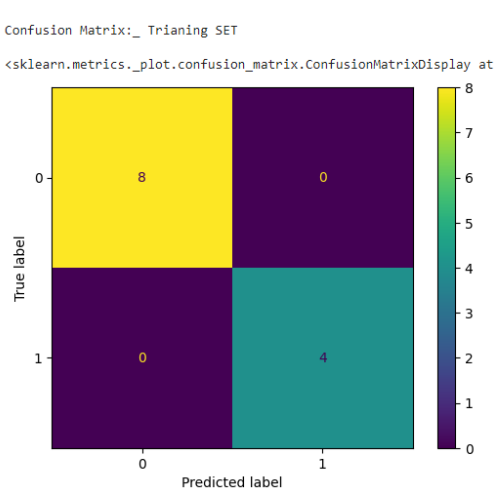

Figure 24 Confusion matrices for Perceptron class 0 Training set

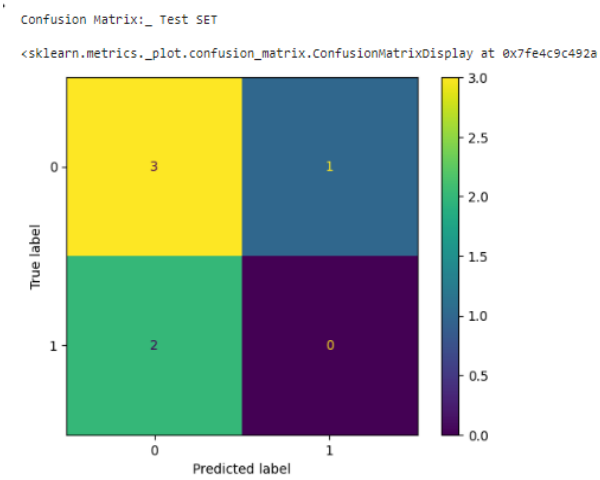

Figure 25 Confusion matrices for Perceptron class 0 Test set

In figures 22 ,23 Perceptron class 0 in training set is was very good but in test set not perform well

And this is the decision boundary:

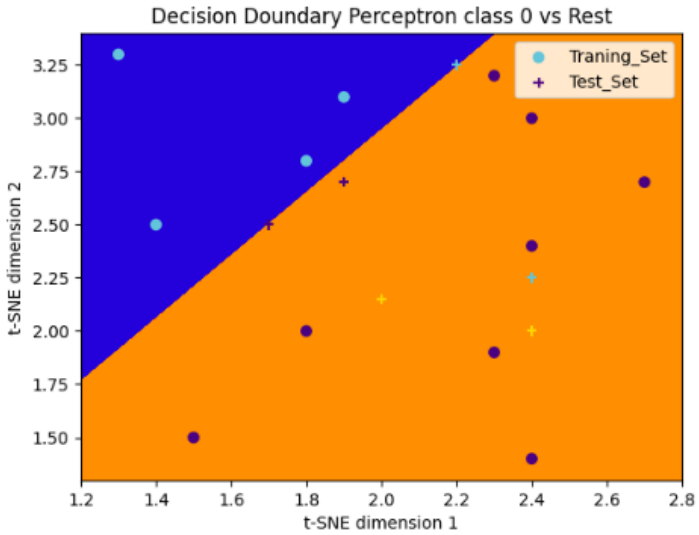

Figure 26 the decision boundary for Perceptron class 0

As we said before the model Perceptron class 0 is preform well in training set and not preform well in test set

Perceptron class 1:

```
perceptron_1= Perceptron(random_state=10)
perceptron_1.fit(X_Train,Y_Train_1)
print(" accuracy  traing  : ",perceptron_1.score(X_Train,Y_Train_1))
Y_pred=perceptron_1.predict(X_Test)
print(" accuracy  testing   : ",perceptron_1.score(X_Test,Y_Test_1))
```

```
 accuracy  traing  :  0.9166666666666666
 accuracy  testing   :  0.6666666666666666
```

Figure 27 the decision boundary for Perceptron class 1

in figure 27 load model for class 1 with random state 10 and fit the model and get the accuracy

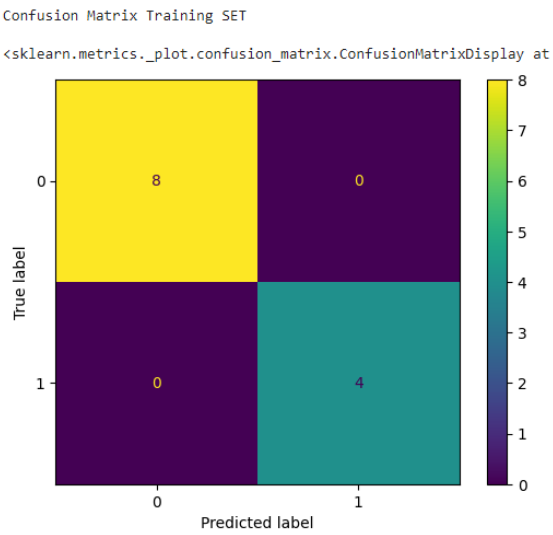

Figure 29 Confusion matrices for Perceptron class 1 Training set

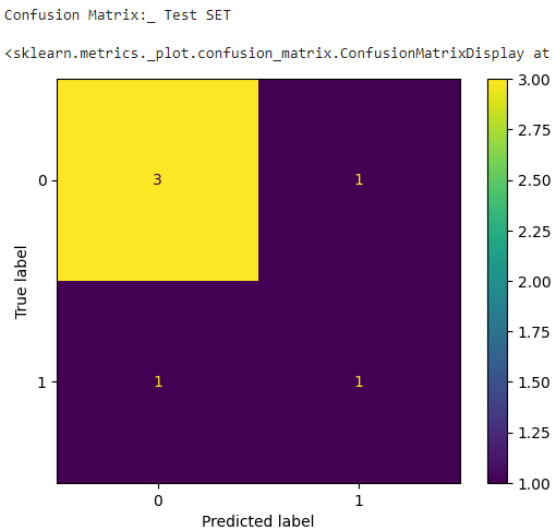

Figure 28 Confusion matrices for Perceptron class 1 Training set

And this is the decision boundary:

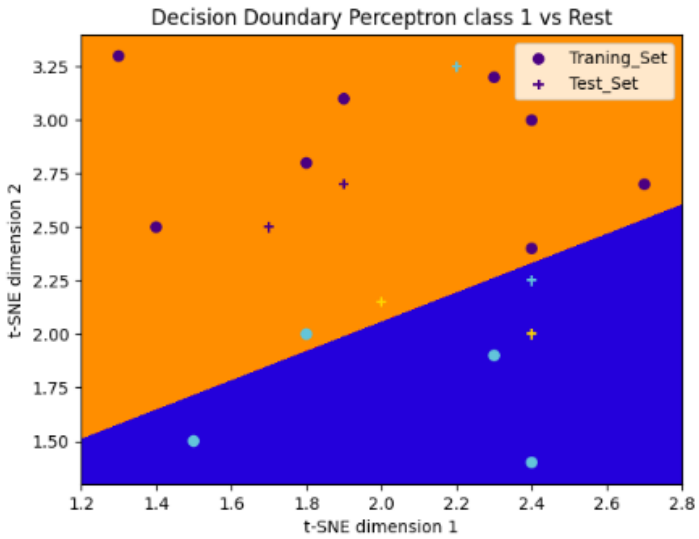

Figure 30 the decision boundary for Perceptron class 1

Perceptron class 2:

```
perceptron_2= Perceptron(random_state=42)
perceptron_2.fit(X_Train,Y_Train_2)
print(" Accuracy  tarning :",perceptron_2.score(X_Train,Y_Train_2))
Y_pred=perceptron_2.predict(X_Test)
print(" Accuracy  testing :",perceptron_2.score(X_Test,Y_Test_2))
```

```
 Accuracy  tarning : 0.75
 Accuracy  testing : 0.5
```

Figure 31 the decision boundary for Perceptron class 2

in figure 31 load model for class 2 with random state 42 and fit the model and get the accuracy

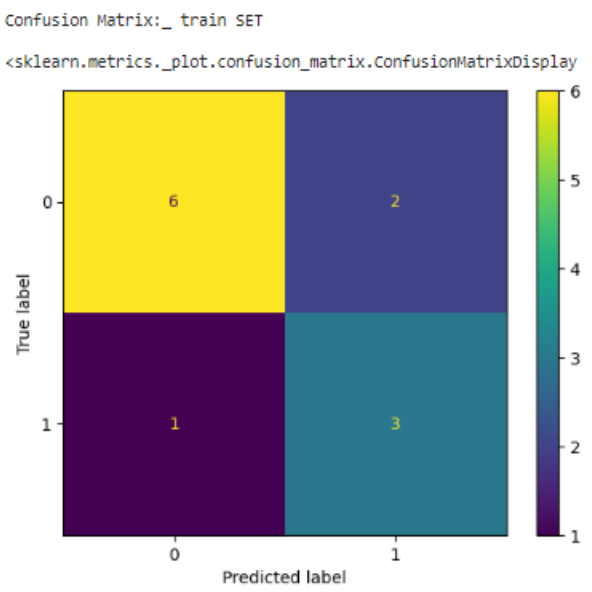

Figure 33 Confusion matrices for Perceptron class 2 Training set

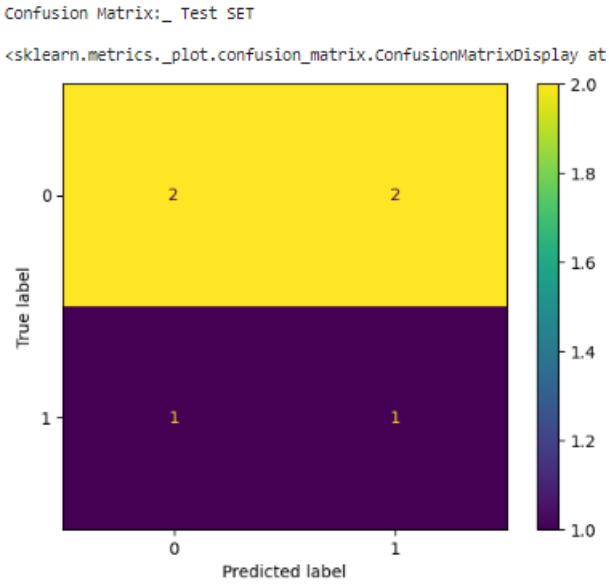

Figure 32 Confusion matrices for Perceptron class 2 Test set

And this is the decision boundary:

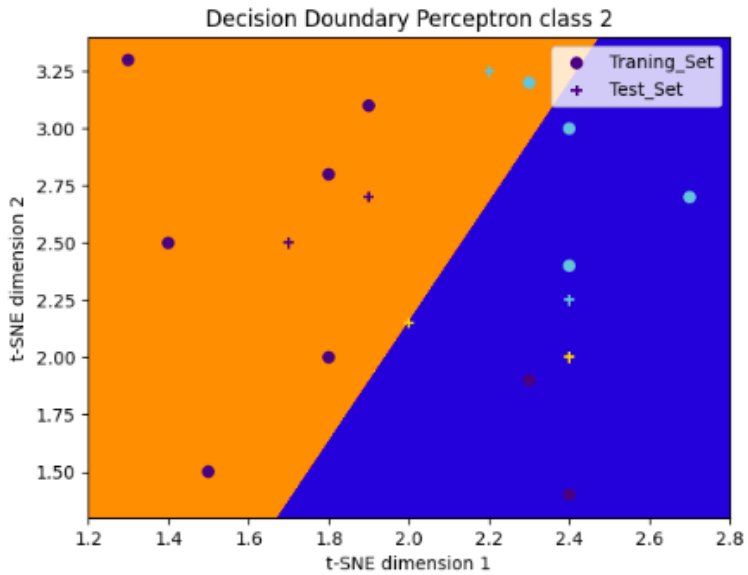

Figure 34 the decision boundary for Perceptron class 2

In figure 34 model Perceptron class 2 , it can detect class 2 not well but acceptable accuracy.

In Table 1 blew we make comparison in all models that we used before and get all result form it

Table 1

Model name	Accuracy training	Accuracy Test
SVM default	1.0	1.0
SVM class 0	0.8333333333333334	0.6666666666666666
SVM class 1	1.0	0.8333333333333334
SVM class 2	1.0	0.8333333333333334
Perceptron default	0.9166666666666666	0.8333333333333334
Perceptron class 0	1.0	0.5
Perceptron class 1	0.9166666666666666	0.6666666666666666
Perceptron class 2	0.75	0.5

After seeing this table, the result SVM with this data set is preforming well but if we enrich data set the Perceptron will be suitable and get results more accurate.

c)

SVM

Aggregate data form 3 models:

This is the function of Aggregate the result of class 0, class 1 and class 2

```
# This function takes the predicted class probabilities for each of the three classes and returns the predicted class labels for each sample in the test set.
def Svm_predict(Y_pro_0,Y_pro_1,Y_pro_2):
  y_pred=[]  # Create an empty list to store the predicted class labels
  print(Y_pro_0,"\n",Y_pro_1,'\n',Y_pro_2)
  for i in range (len(Y_pro_0)):
    # For each sample in the test set, determine the predicted class label based on the highest predicted probability for that sample
    if (Y_pro_0[i]>=Y_pro_1[i] and Y_pro_0[i]>=Y_pro_2[i]):
       y_pred.append(0)
    elif  Y_pro_1[i]>=Y_pro_2[i]:
      y_pred.append(1)
    else :
      y_pred.append(2)


  return  y_pred
```

Figure 35 Aggregate function to aggregate SVM model class 0,1 and 2

This is the result of the Aggregate function.

```
Agragate_data=Svm_predict(Y_pro_0,Y_pro_1,Y_pro_2)

print(Agragate_data)
```

```
[0.4753973  0.43049008 0.19551225 0.06530315 0.46202128 0.09449151]
 [0.29112136 0.18123838 0.5        0.57065014 0.04333263 0.39916305]
 [0.20961922 0.38903503 0.25768262 0.4519599  0.77782389 0.55156643]
[0, 0, 1, 1, 2, 2]
```

Figure 36 result of the Aggregate function

Confusion matrix for aggregate data:

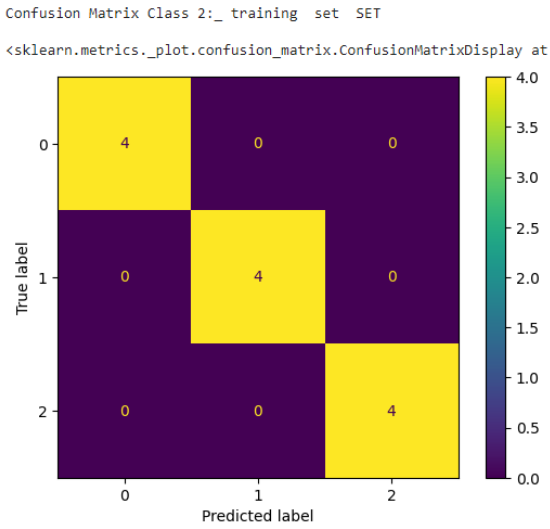

Figure 37 Confusion matrices for SVM model aggregated for Training set

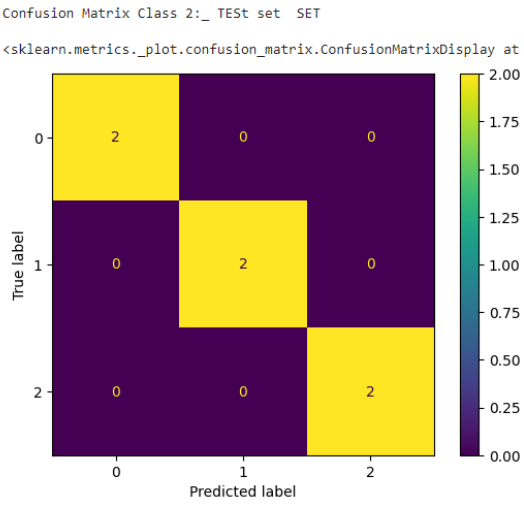

Figure 38 Confusion matrices for SVM model aggregated for Test set

In figure 37 ,38 we aggregate results from models and display the Confusion matrices.

And the result not different form the SVM default

And this is the decision boundary:

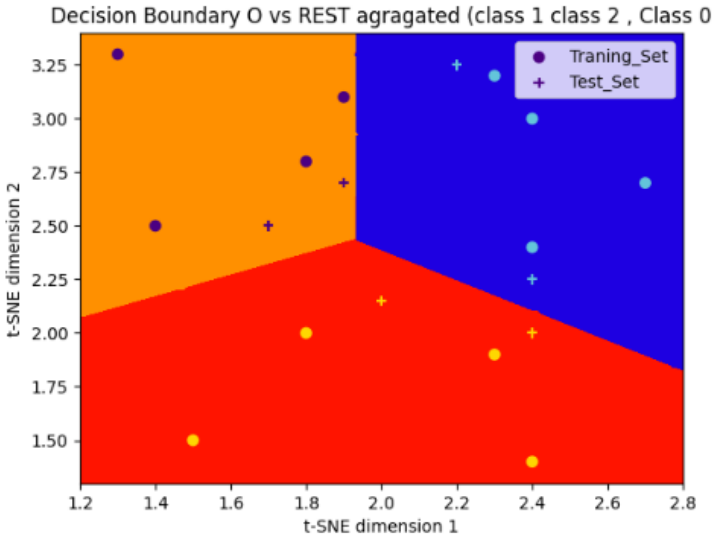

Figure 39 the decision boundary for SVM aggregated model

Here in figure 39 decision boundary for SVM aggregated model shows us the big difference between this model and SVM default. In SVM aggregated model we used Kernel linear but in default SVM the kernel was rbf

Perceptron

Confusion matrix for aggregate data in Perceptron:

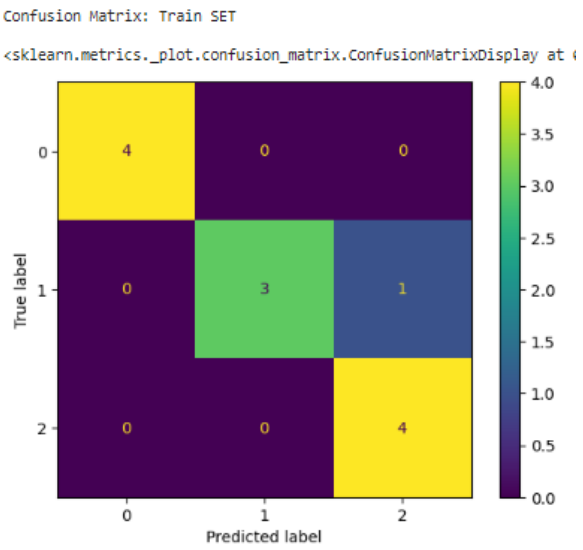

Figure 41 Confusion matrices for Perceptron model aggregated for Training set.

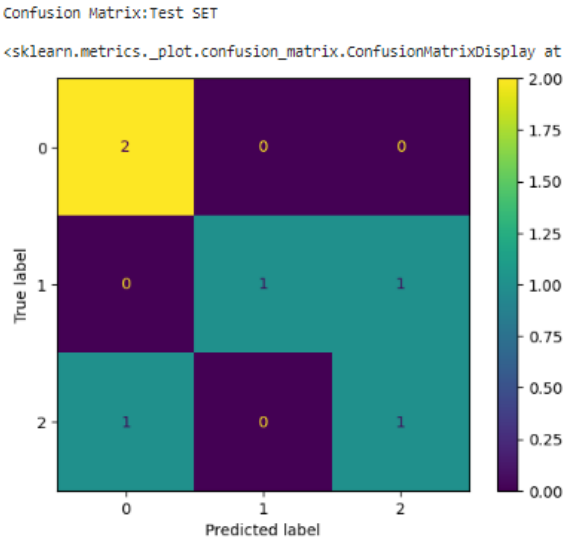

Figure 40 Confusion matrices for Perceptron model aggregated for test set.

And this is the decision boundary:

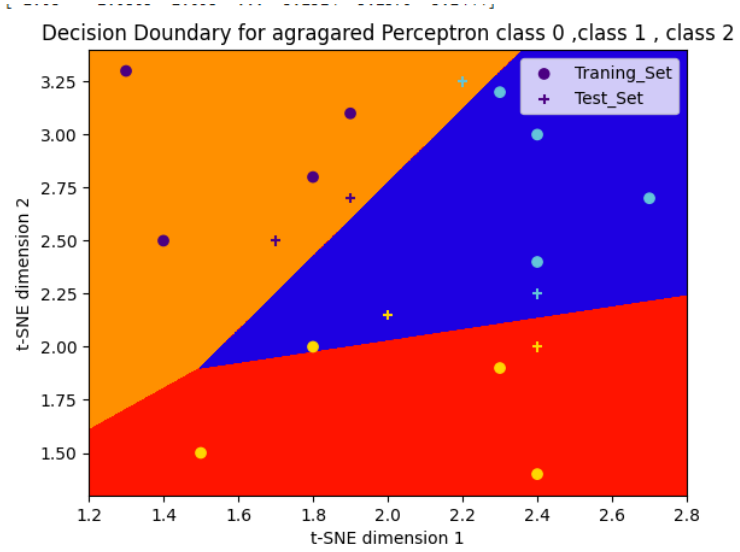

Figure 42 the decision boundary for Perceptron aggregated model

Compare and analyze SVM and SVM O VS REST results:

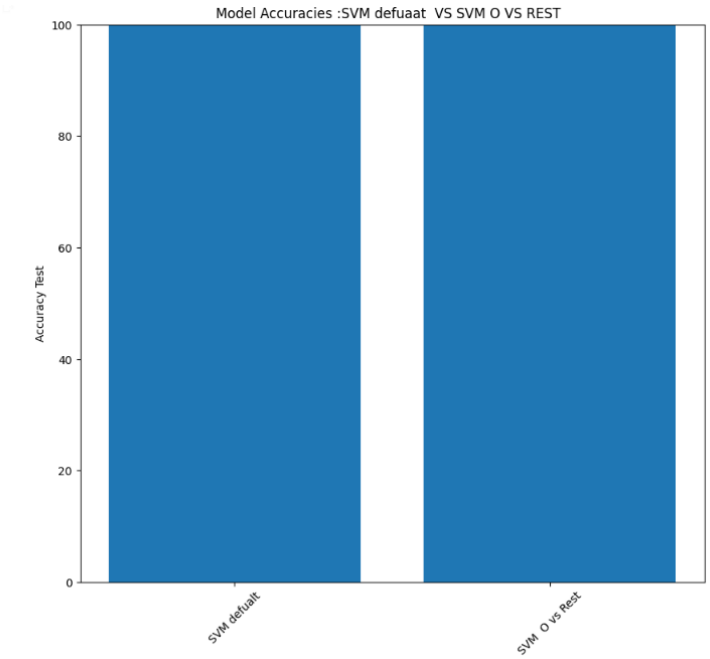

Figure 43 compare SVM default with SVM O VS REST accuracy.

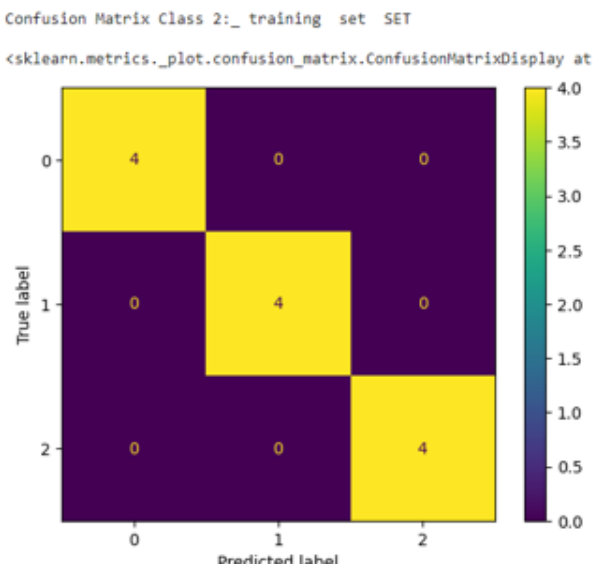

Figure 44 SVM default Confusion matrices

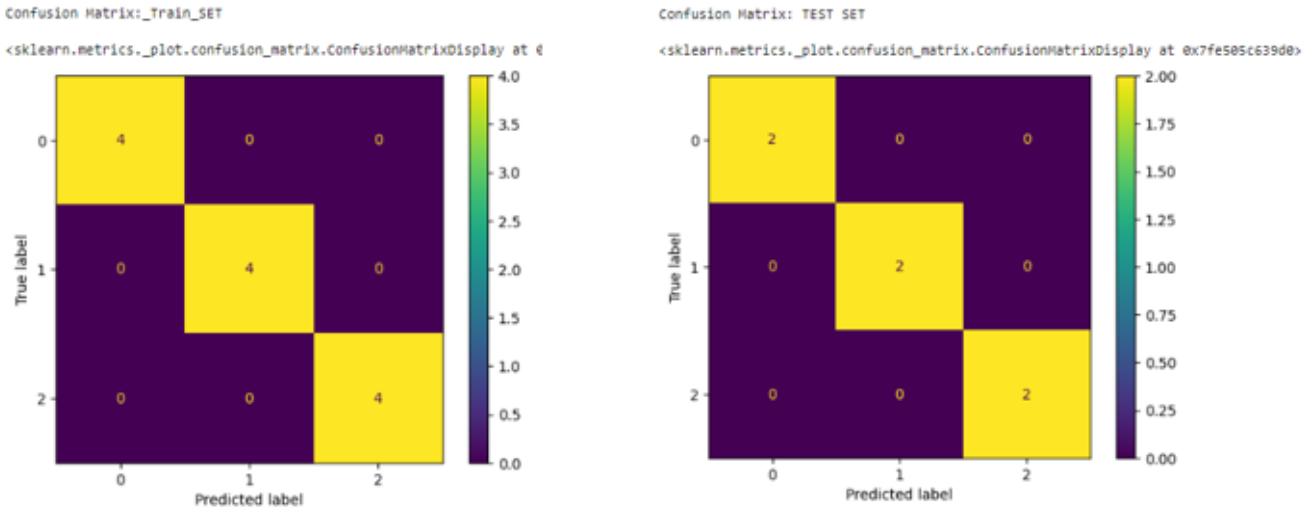

Figure 45 SVM O VS RSET Confusion matrices

Here in figures 44, 45 we can't see any different but the decision boundary we will ensure.

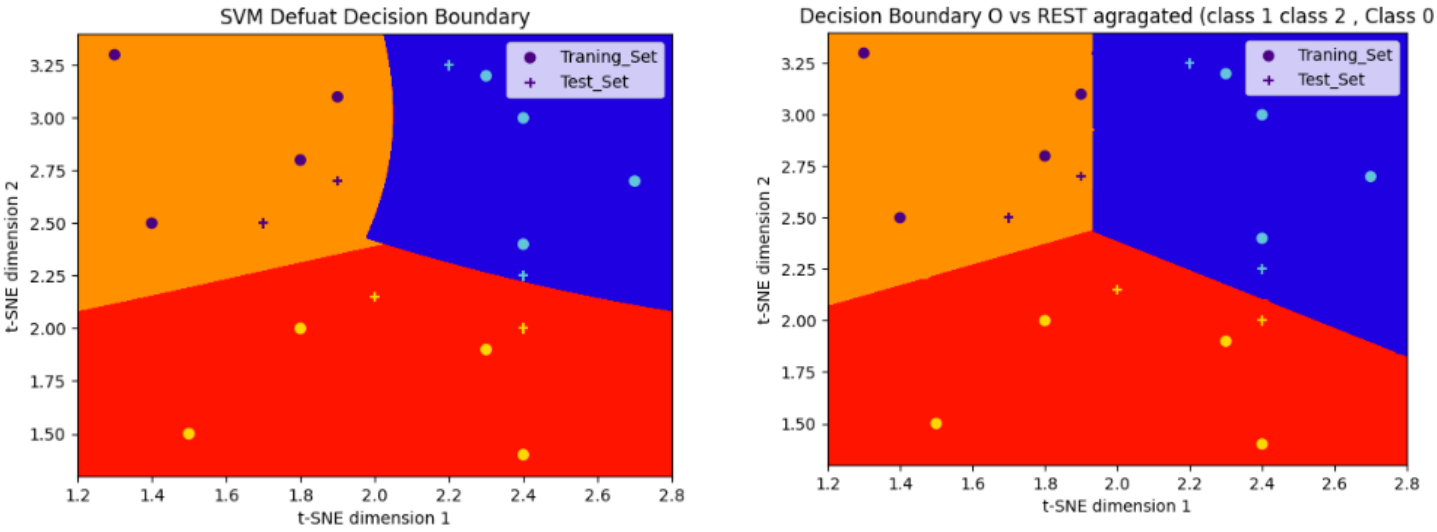

Figure 46 decision boundary for SVM Default and SVM O VS REST

As we mention before the SVM more flexible because the kernel is rbf can detect the curves in data to categories every class but in O VS REST the kernel linear that make edges is so sharped and no well in our data.

compare SVM VS *perceptron:*

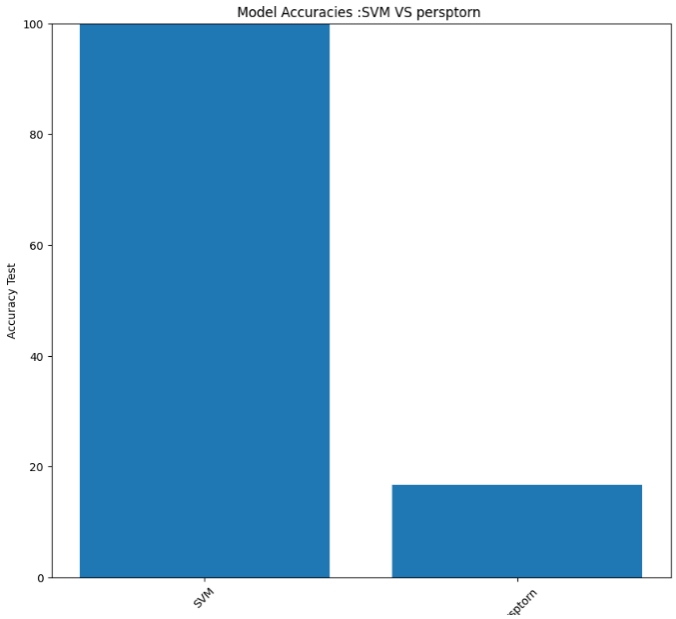

Figure 47 compare Perceptron with SVM O VS REST accuracy.

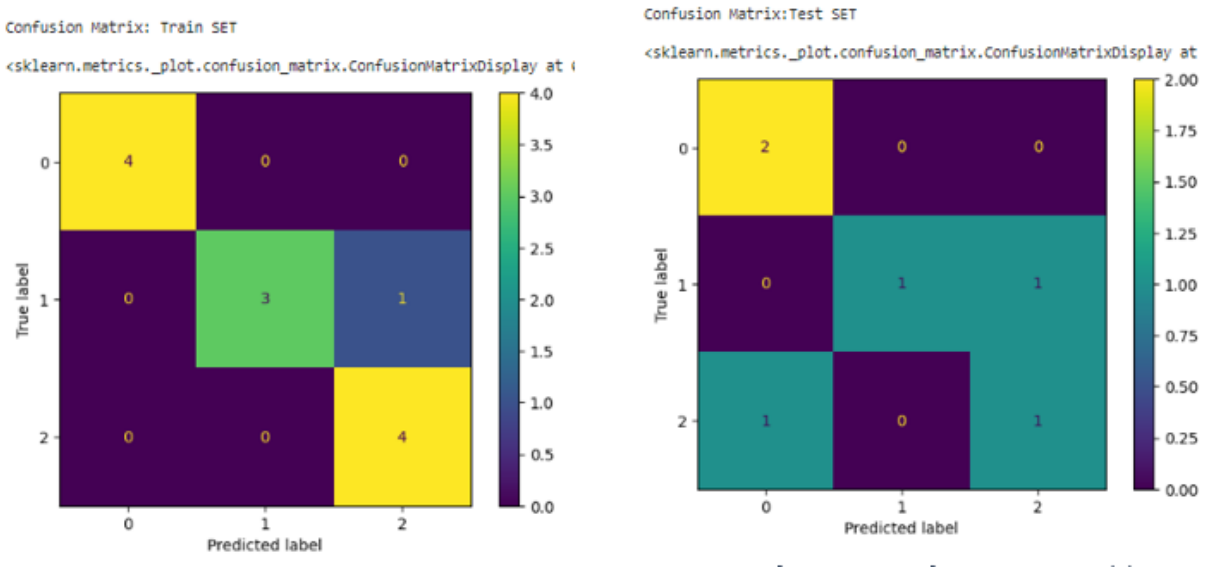

Figure 48 Confusion matrices for Perceptron model aggregated.

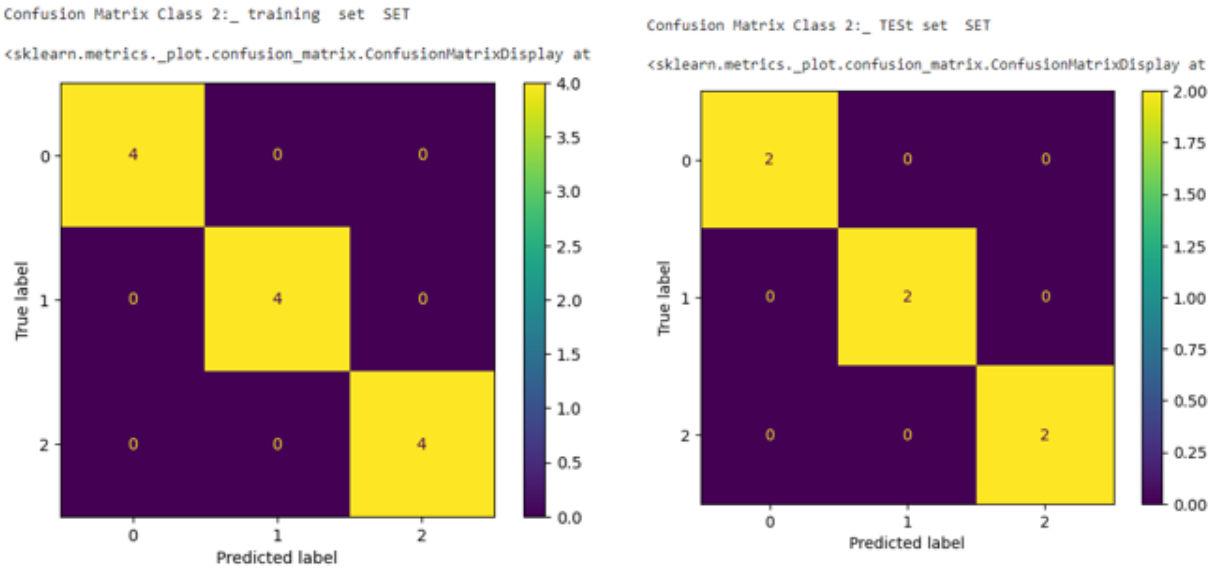

Figure 49 Confusion matrices for SVM model aggregated.

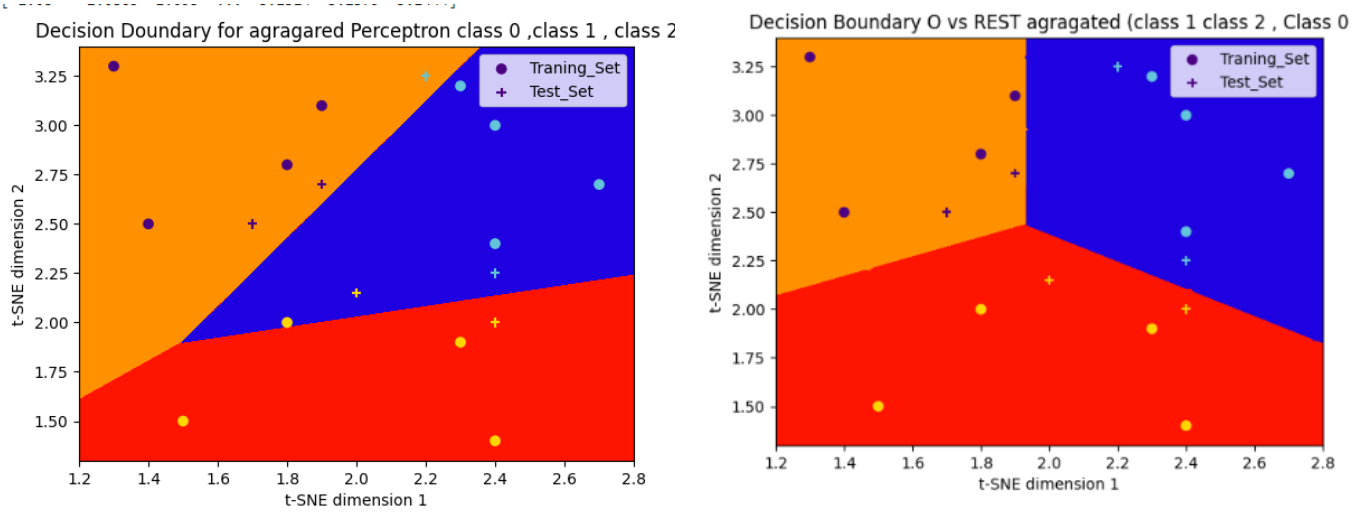

Figure 50 decision boundary for Perceptron and SVM O VS REST

D)

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

We used in section (a) default SVM kernel is rbf and in section (c) aggregated SVM for class 0 , class 1 and class 2 kernel is linear , kernel rbf made model more flexible and get curved boundary not sharper like linear kernel. SVM class 0 did not get appropriate accuracy in training and testing but class 1 and class 2 get appropriate accuracy in testing but in testing was not good although the default class gave 100% in testing and training you can review our results in the table (1) .

Selecting the appropriate parameter for SVM default

```
[104] SVM_best=svm.SVC(C=3,degree=4,kernel='rbf',verbose=True) # Create an instance of the SVM classifier
      SVM_best.fit(X_Train,Y_Train) # Train the SVM classifier using the training data
      Y_Pred=SVM_best.predict(X_Test) # test the model
      print('accuracy_training ',SVM_best.score(X_Train,Y_Train))  # Evaluate the accuracy of the trained model on the training data with the modified labels for class 0
      print('accuracy_test ',SVM_best.score(X_Test,Y_Test)) # Evaluate the accuracy of the trained model on the test data with the modified labels for class 0

      print(Y_Pred)

      [LibSVM]accuracy_training  1.0
      accuracy_test  1.0
      [0 0 1 1 2 2]
```

Figure 51 load SM model with new parameter

in figure 51 we select the Regularization parameter (C) with 3 and kernel with rbf and make the verbose TURE and see what the result.

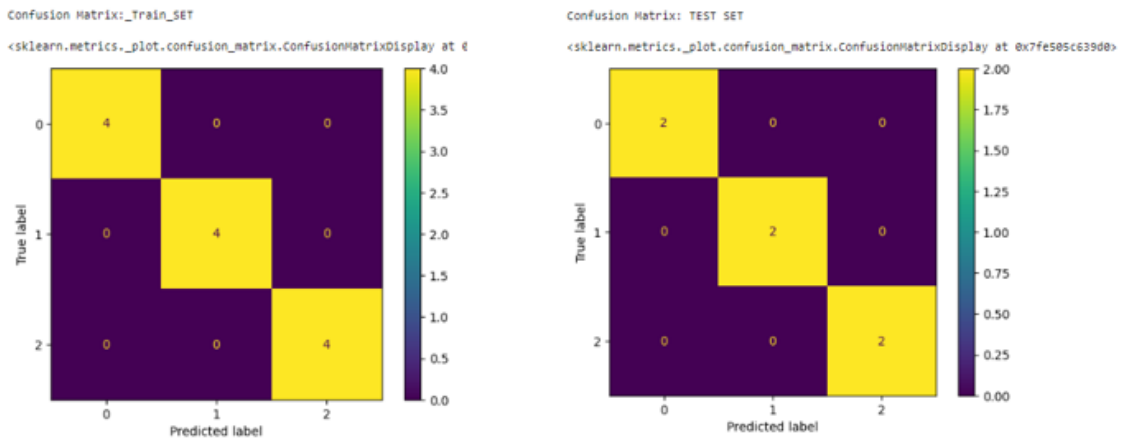

Figure 52 Confusion matrices for NEW SVM model

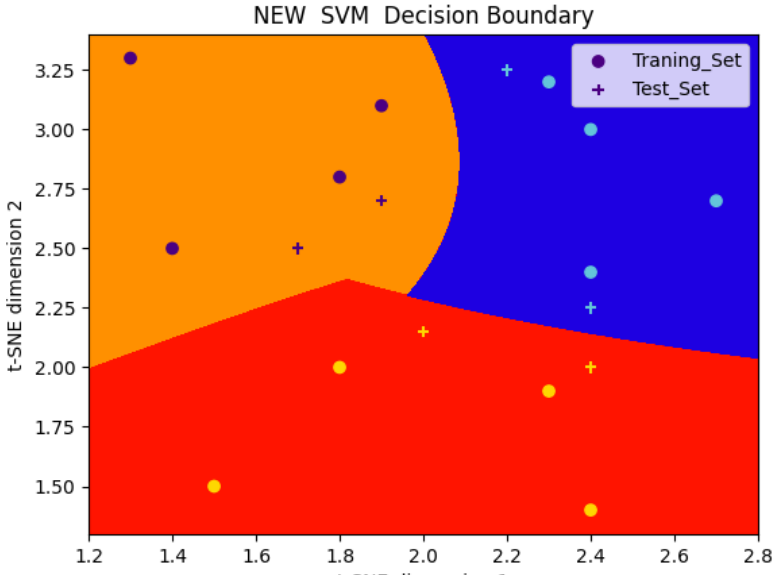

Figure 53 decision boundary for New SVM model

as we see the result form New SVM is not different in Confusion matrices form other models but the bigger difference in decision boundary.

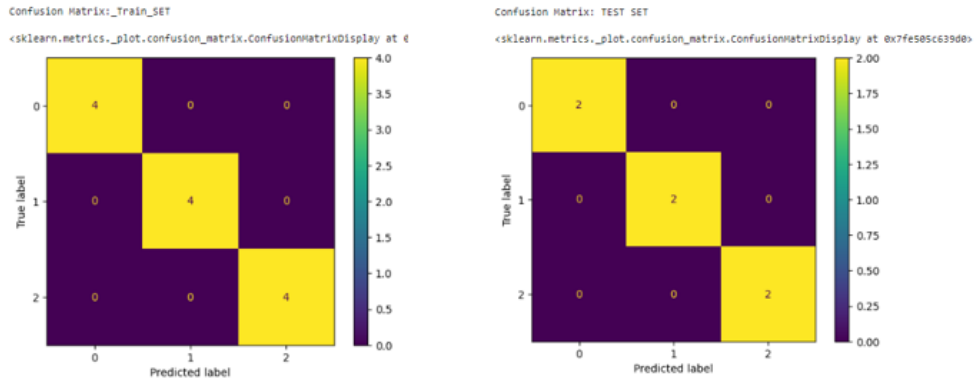

Figure 54 in Confusion matrices form SVM Default

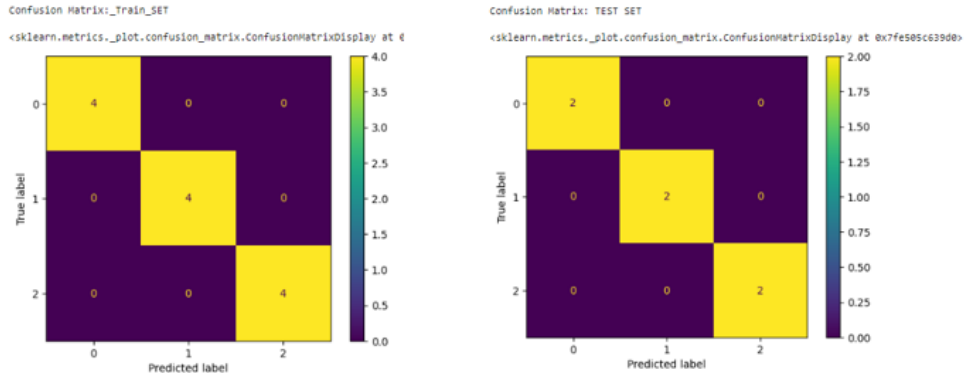

Figure 55 Confusion matrices form NEW SVM

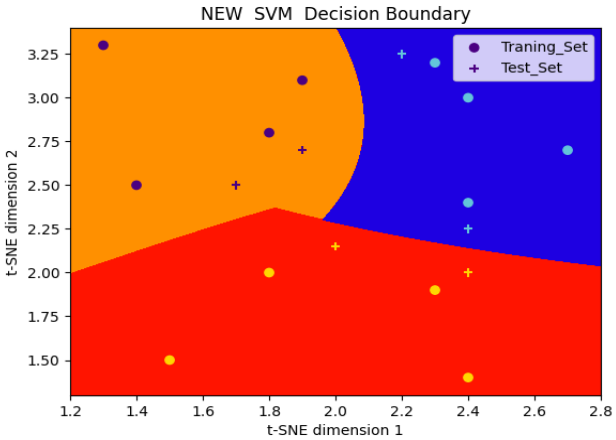

Figure 57 decision boundary for New SVM model

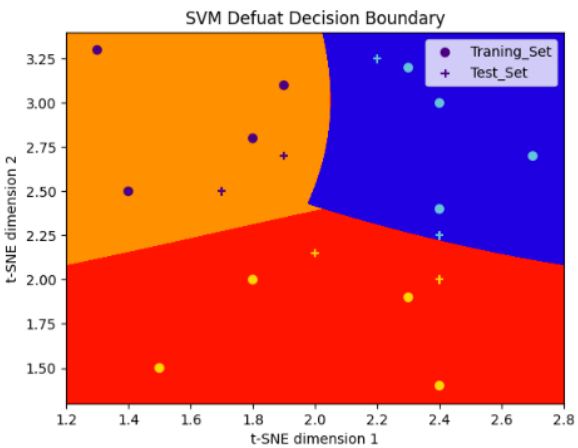

Figure 56 decision boundary for SVM default model

We can convince the result form New SVM is better than the default because we Regularization parameter (C) with 3 this make the learning process is more suitable to our situation in our problem.

The first different in O VS REST the kernel is linear and this make the boundary be sharped and in SVM default more flexible in detect classes like in red class is more SVM default but in O VS REST blue class and red class get the same area

Q2)Task 2

KNN

A)

```
#for error  in URL  used  the CSV file  because data in URL has been deleted
try :
  df=pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data")
except Exception as e:
  print("we can't found the Data Set in Your URL and Try to import from  a file CSV.\n and this is the Error .",e)
  try:
    df=pd.read_csv("./car_evaluation.csv")
    print("the file iS imported succsfully  .")
  except Exception as e:
     print("please ensure  the path file of Data set is right \n  and this is the Error .",e )
```

```
we can't found the Data Set in Your URL and Try to import from  a file CSV.
 and this is the Error . HTTP Error 404: Not Found
the file iS imported succsfully  .
```

Figure 58 load the data set form web site or local CSV file.

Because the data Set in web site are not found and almost deleted we made a try except to load it without any Error and if you faced error, you should ensure the path of file CSV is right.

```
df
```

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc
...	...	...	...	...	...	...	...
1722	low	low	5more	more	med	med	good
1723	low	low	5more	more	med	high	vgood
1724	low	low	5more	more	big	low	unacc
1725	low	low	5more	more	big	med	good
1726	low	low	5more	more	big	high	vgood

1727 rows × 7 columns

Figure 60 the data set after being imported.

```
[6] col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
    df.columns=col_names
```

```
df
```

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc
...	...	...	...	...	...	...	...
1722	low	low	5more	more	med	med	good
1723	low	low	5more	more	med	high	vgood
1724	low	low	5more	more	big	low	unacc
1725	low	low	5more	more	big	med	good
1726	low	low	5more	more	big	high	vgood

1727 rows × 7 columns

Figure 59 data set after renaming all columns name.

-...

```
[ ] # separate the target column
    X=df.drop('class',axis=1)
    Y=df['class']
```

```
[ ] # Split into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=428,shuffle=True,random_state=45)
    print(X_train.shape,X_test.shape)
```

```
    (1299, 6) (428, 6)
```

```
[ ] ## splitting the data into 1000 training ,428 testing and 300 for validation
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=300,shuffle=True,random_state=45)
    print(X_train.shape,X_val.shape)
```

Figure 61 prepare data to be fit to our model.

B)

encodeing data to numbers

```
ordinal_encoder = OrdinalEncoder()
y_train=np.array(y_train).reshape(-1, 1)
y_val=np.array(y_val).reshape(-1, 1)
y_test=np.array(y_test).reshape(-1, 1)
## Encode the training ,validation and testing data
X_train_new = ordinal_encoder.fit_transform(X_train)
y_train_new = ordinal_encoder.fit_transform(y_train)
y_val_new = ordinal_encoder.fit_transform(y_val)
X_val_new =ordinal_encoder.fit_transform(X_val)
X_test_new = ordinal_encoder.fit_transform(X_test)
y_test_new = ordinal_encoder.fit_transform(y_test)
```

Figure 62 encoding the data set.

C)

then we did encode to the training, validation, and testing data.

as we used instance from ordinal encoder and apply it to X_train, y_train, x_test, y_test ,x_val and y_val

and storing the result in new variables called X_train_new, y_train_new,

X_val_new, y_val_new, X_test_new, y_test_new.

```
# Define the range of proportions of the training set to use
train_proportions = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
# Initialize lists to store the accuracy scores
val_scores = []
test_scores = []

# Loop over the range of proportions and train KNN models
for proportion in train_proportions:
    num_train_samples = int(proportion * len(X_train))
    # Create a KNN classifier with K=2
    knn = KNeighborsClassifier(weights='distance',n_neighbors=2)
    # Fit the classifier on the training set
    knn.fit(X_train_new[:num_train_samples], y_train_new[:num_train_samples])
    # Evaluate the classifier on the validation set
    val_pred = knn.predict(X_val_new)
    val_acc = accuracy_score(y_val_new, val_pred)
    val_scores.append(val_acc)
    # Evaluate the classifier on the testing set
    test_pred = knn.predict(X_test_new)
    test_acc = accuracy_score(y_test_new, test_pred)
    test_scores.append(test_acc)
```

Figure 63 modeling and get result.

We define a range of proportions of the training set to use, from 10% to 100%. It then initializes empty lists to store the accuracy scores for both the validation set and the testing set.

The code then loops over the range of proportions and trains a KNN model with K=2 on each proportion of the training set. The KNeighborsClassifier function from the scikit-learn library is used to create the KNN model, and the 'distance' weight function is used, which assigns weights to the neighbors based on their distance from the input data point.

The X_train_new, y_train_new, X_val_new, y_val_new, X_test_new, and y_test_new variables are assumed to contain the training, validation, and testing data in a format suitable for the KNeighborsClassifier function.

The code evaluates the accuracy of the model on the validation set and appends it to the val_scores list. It then evaluates the accuracy of the model on the testing set and appends it to the test_scores list.

Finally, the code uses Matplotlib to plot the accuracy scores against the proportion of the training set used. The plot shows the relationship between the size of the training set and the accuracy of the KNN model on both the validation and testing sets. This can help identify the optimal size of the training set that maximizes the model's performance and helps prevent

overfitting or underfitting. The legend indicates which line represents the accuracy scores for the validation set and which represents the accuracy scores for the testing set.

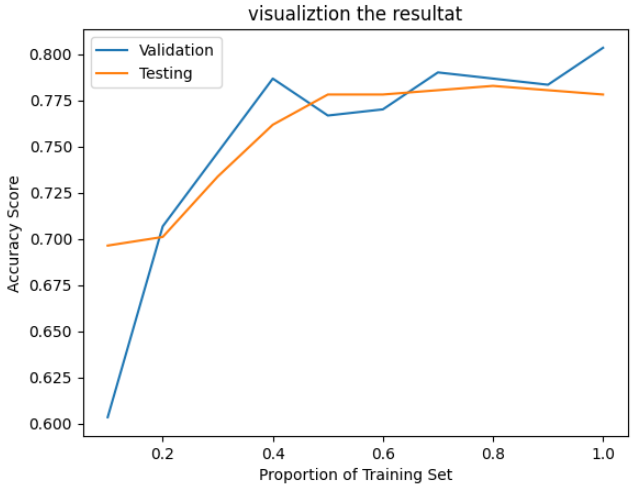

Figure 64 visualization the accuracy in all situations

We visualization the result as you in figure 64.

D)

```
# Define the range of K values to try
K_values = range(1, 11)

# Initialize a list to store the accuracy scores
val_scores = []

# Loop over the range of K values and train KNN models
for K in K_values:

    knn = KNeighborsClassifier(weights='distance',n_neighbors=K)


    knn.fit(X_train_new, y_train_new)

    # Evaluate the classifier on the validation set
    val_pred = knn.predict(X_val_new)
    val_acc = accuracy_score(y_val_new, val_pred)
    val_scores.append(val_acc)
```

Figure 65 prepare the layout of the KNN to make comparison.

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

We defined a range of K values from 1 to 10, and initializes an empty list to store the accuracy scores for each value of K. It then loops over the range of K values, trains a KNN model with the given value of K, and evaluates its accuracy on a validation set.

The KNeighborsClassifier function from the scikit-learn library is used to create the KNN model. The 'distance' weight function is used, which assigns weights to the neighbors based on their distance from the input data point. This means that closer neighbors have a greater influence on the classification decision than farther neighbors.

The X_train_new, y_train_new, X_val_new, and y_val_new variables are assumed to contain the training and validation data in a format suitable for the KNeighborsClassifier function.

Finally, the code uses Matplotlib to plot the accuracy scores against the corresponding values of K. The plot shows the relationship between the value of K and the accuracy of the KNN model on the validation set. This can help identify the optimal value of K that maximizes the model's performance.

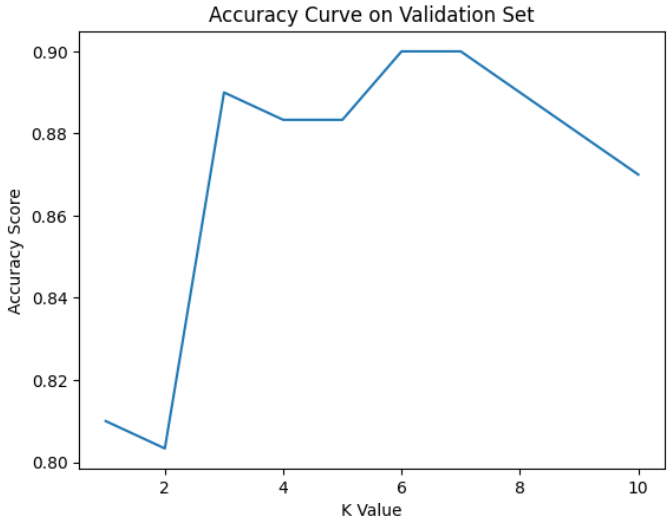

Figure 66 visualization the OUTPUT

E)

In the output graph, we can see that the accuracy scores on both the validation and testing sets start low for a small proportion of the training set (i.e., 0.1), but increase rapidly as more training data is added. The validation accuracy score continue to increase with increasing proportion of the training set, while the

testing accuracy score begins to plateau and even slightly decrease as the proportion of the training set used increases beyond a certain point.

Therefore, the optimal proportion of the training set appears to be around 0.6 to 0.7, as this provides a good balance between learning the patterns in the data and avoiding overfitting. Beyond this point, the performance of the classifier on the testing set begins to decrease, suggesting that the classifier is overfitting the training data.

2- In the output graph, we can see that the accuracy score on the validation set initially increases as the value of K increases from 1 to 2-3, which suggests that the classifier is becoming less sensitive to noise and outliers and is capturing the underlying patterns in the data more accurately. However, as the value of K continues to increase beyond 2-3, the accuracy score on the validation set starts to decrease, suggesting that the classifier is becoming too generalized and is losing its ability to accurately classify new, unseen data.

Reaching to the value k=6,7 we found that accuracy score starts to increase and remains stable and after that reaching to the value beyond the value k=7 the accuracy score starts to decrease again

Therefore, the optimal value of K appears to be around 6-7, as this provides a good balance between reducing the impact of noise and outliers and capturing the underlying patterns in the data. Beyond this point, the performance of the classifier on the validation set begins to decrease, suggesting that the classifier is becoming too generalized.