ELG5255 Applied Machine Learning

REPORT of: Group Assignment 1 (Group-18)

Problem 1:

1. load and understand the dataset

1. DUMD It is the real dataset about the students' knowledge status about the subject of Electrical DC Machines. The data is cleaned and divided into training and testing. the data belong to classes (0,1,2,3) ((Very Low, High, Medium, and Low)).
2. We loaded the DUMD dataset.

```
[53] #'DUMD_train.csv'
     link_train = "https://drive.google.com/uc?id=1pbnWqF-Mo61q2Kpp1CF_rV-58_9mmOVs"
     #'DUMD_test.csv'
     link_test = "https://drive.google.com/uc?id=1JAut9ED1IKO95w1LKvCCOxMuERKqoKCW"

     X_train,y_train,X_test,y_test,training_dataset,testing_dataset=load_traing_testing_sets(link_train, link_test)
```

	STG	SCG	STR	LPR	PEG	UNS
0	0.00	0.00	0.00	0.00	0.00	Very Low
1	0.08	0.08	0.10	0.24	0.90	High
2	0.10	0.10	0.15	0.65	0.30	Medium
3	0.08	0.08	0.08	0.98	0.24	Low
4	0.09	0.15	0.40	0.10	0.66	Medium
5	0.10	0.10	0.43	0.29	0.56	Medium
6	0.20	0.14	0.35	0.72	0.25	Low
7	0.00	0.00	0.50	0.20	0.85	High
8	0.18	0.18	0.55	0.30	0.81	High
9	0.06	0.06	0.51	0.41	0.30	Low

2. Label Encoding

3. We converted categorical class labels (UNS) to numerical values by using the LabelEncoder.

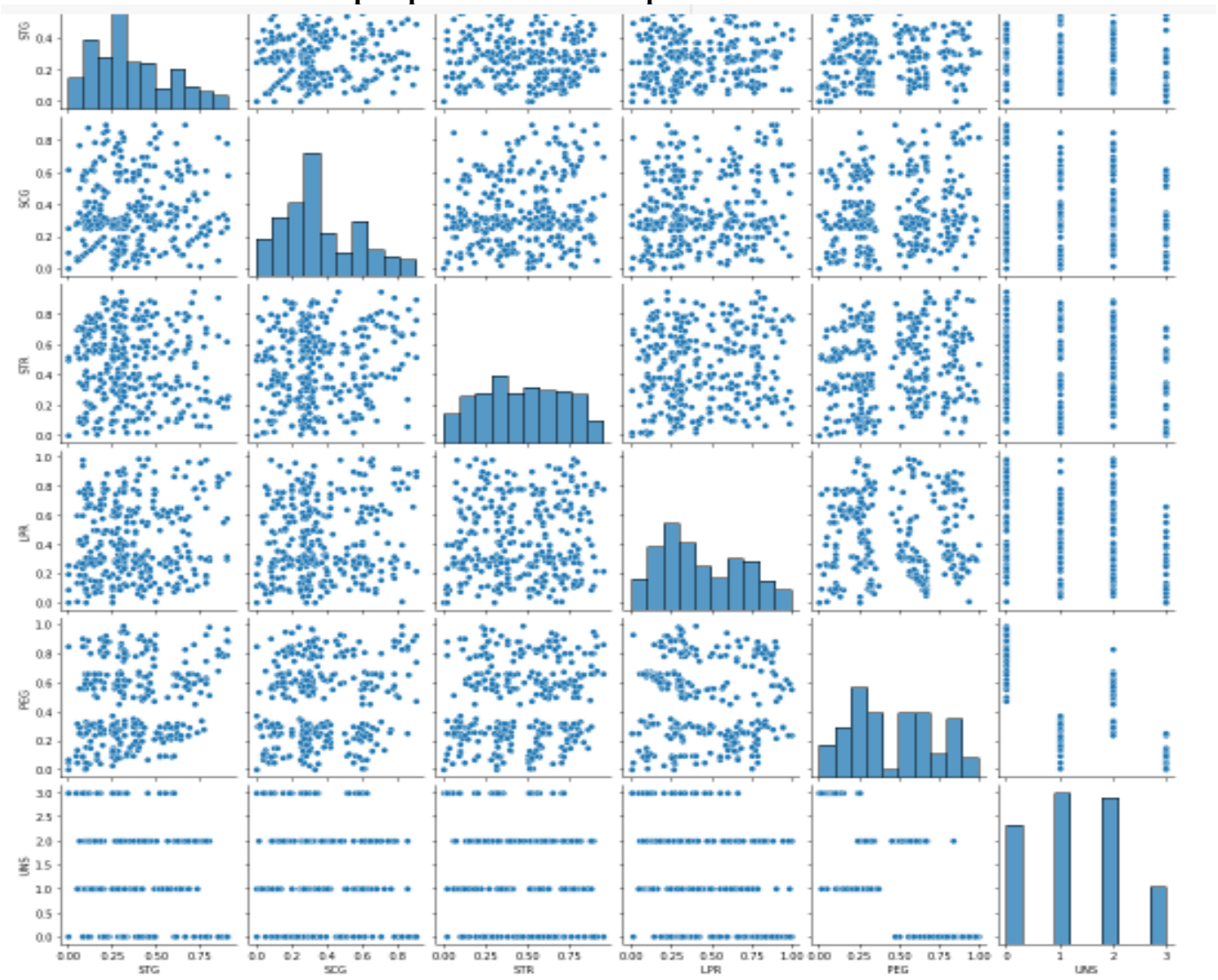
Very Low	3
High	0
Medium	2
Low	1

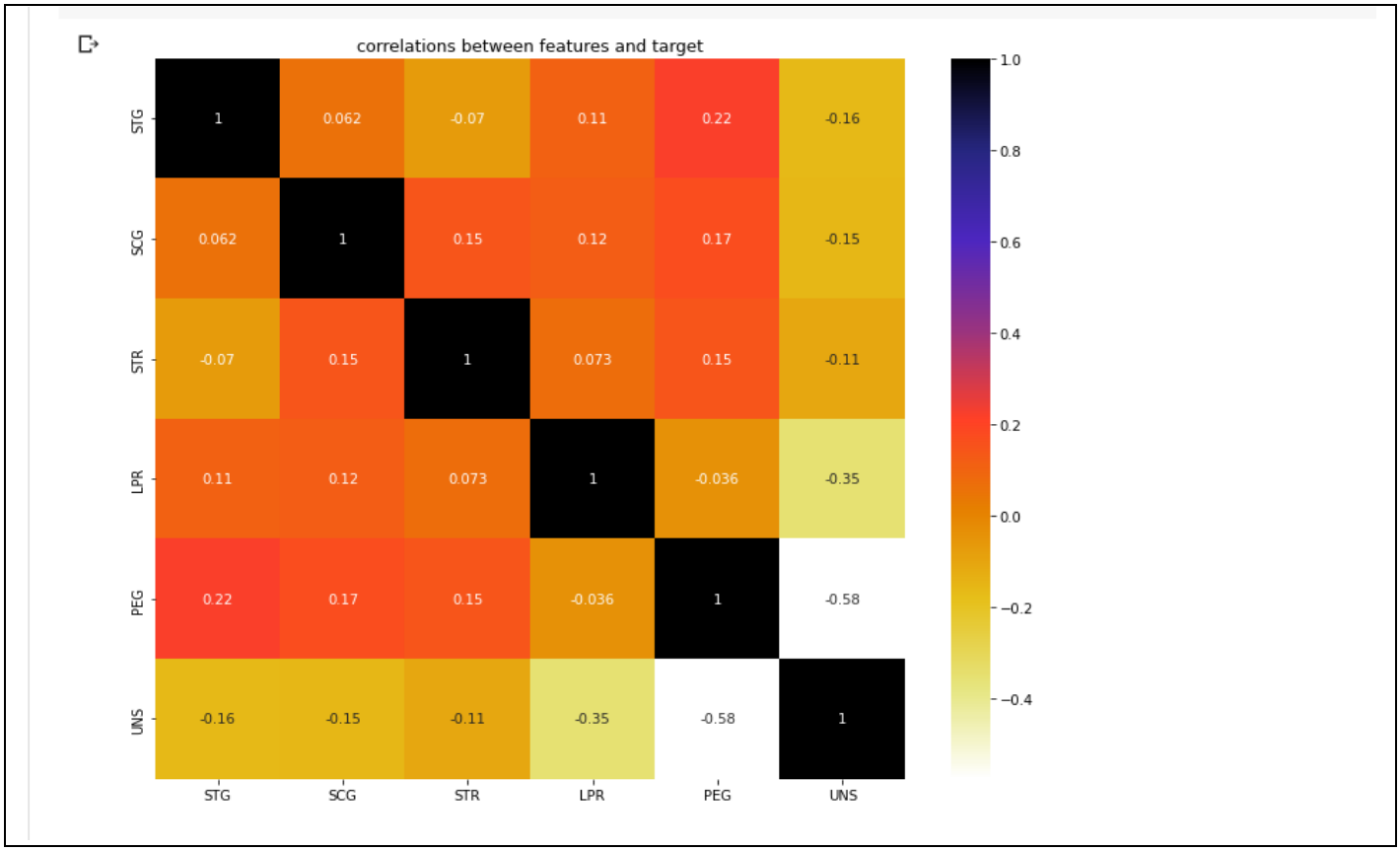
3. Features selection

4. we used a sklearn function to choose the best 2 features using the correlation between features and the target and the result was

[LPR, PEG] features.

plot pairwise relationships in a dataset





4. build OVO-SVM, OVR-SVM classifiers, and perceptron classifiers

5. we used python libraries to build three classifiers:

1. OVO-SVM 2. OVR-SVM 3. perceptron classifier

```
model_OVO = svm.SVC(kernel='rbf', decision_function_shape='ovo', C=1)
model_OVO.fit(X_train, y_train)
print('Accuracy of model: {:.2f}%'.format(getAccuracy(model_OVO, X_test,y_test)))
preSVMOVOtwofeatures_data = model_OVO.predict(X_test)
```

```
Accuracy of model: 98.75%
```

```
model = svm.SVC(kernel='rbf', decision_function_shape='ovr', C=1)
model.fit(X_train, y_train)
print('Accuracy of model: {:.2f}%'.format(getAccuracy(model, X_test,y_test)))
preSVMOVRtwofeatures_data = model.predict(X_test)
```

```
Accuracy of model: 98.75%
```

```
from sklearn.linear_model import Perceptron
clf = Perceptron(tol=1e-3)
clf.fit(X_train, y_train)
print('Accuracy of model: {:.2f}%'.format(getAccuracy(clf, X_test,y_test)))
prePertwofeatures_data = clf.predict(X_test)
```

```
Accuracy of model: 60.00%
```

6. We built functions to get the best accuracy, confusion matrix, decision boundaries, and plot correct and wrong prediction points.

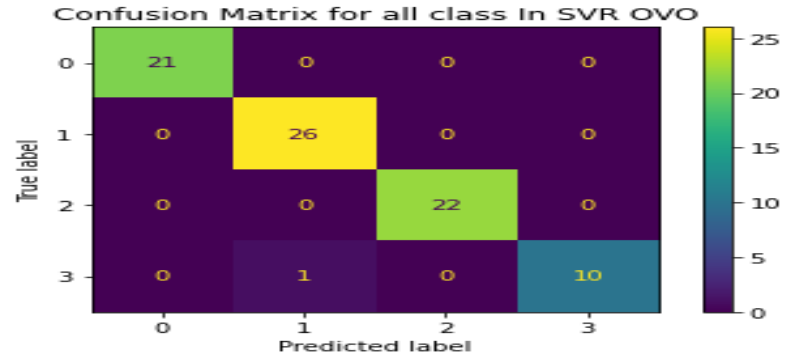
OVO-SVM using sklearn Confusion Matrix and decision boundary:

The accuracy of the model: 98.75%

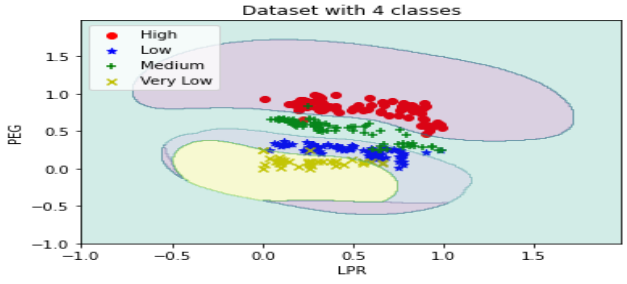
Classification Report for all class In SVR OVO

	precision	recall	f1-score	support
3	1.00	0.91	0.95	11
0	1.00	1.00	1.00	21
2	1.00	1.00	1.00	22
1	0.96	1.00	0.98	26
accuracy			0.99	80
macro avg	0.99	0.98	0.98	80
weighted avg	0.99	0.99	0.99	80

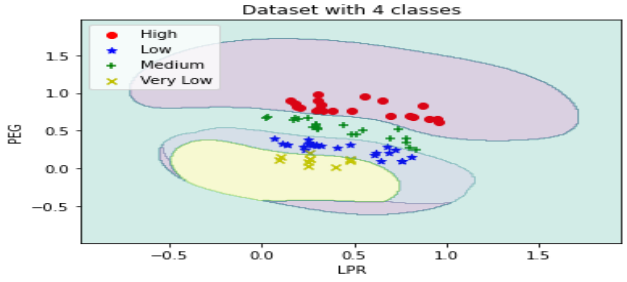
```
[[21  0  0  0]
 [ 0 26  0  0]
 [ 0  0 22  0]
 [ 0  1  0 10]]
```



Training for all classes High , Low , Medium , Very Low In SVM OVO



Testing for all classes High , Low , Medium , Very Low In SVM OVO



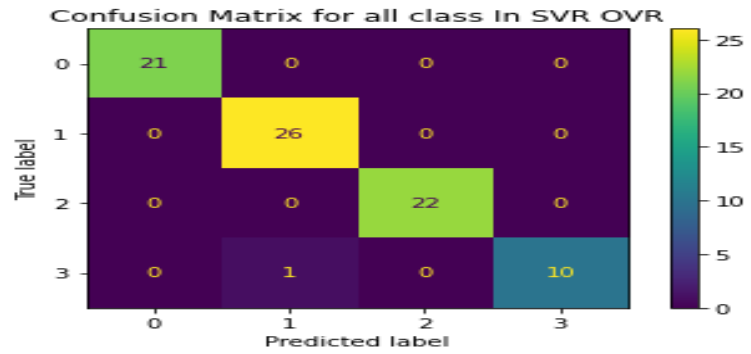
OVR-SVM using sklearn Confusion Matrix and decision boundary:

The accuracy of the model: 98.75%

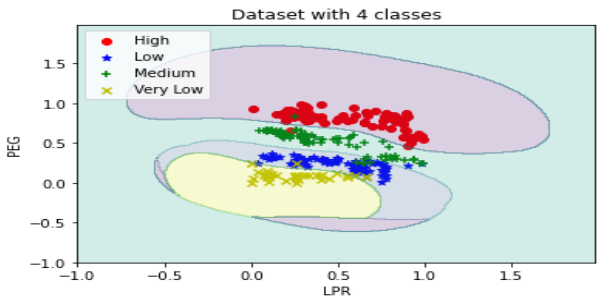
Classification Report for all class In SVM OVR

	precision	recall	f1-score	support
3	1.00	0.91	0.95	11
0	1.00	1.00	1.00	21
2	1.00	1.00	1.00	22
1	0.96	1.00	0.98	26
accuracy			0.99	80
macro avg	0.99	0.98	0.98	80
weighted avg	0.99	0.99	0.99	80

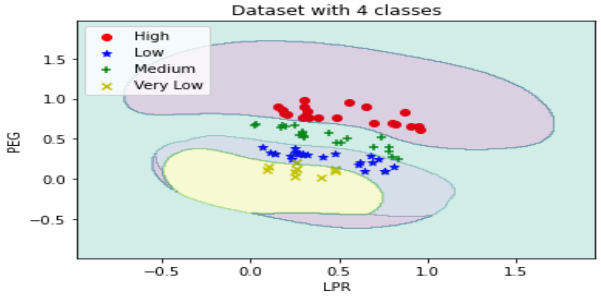
```
[[21  0  0  0]
 [ 0 26  0  0]
 [ 0  0 22  0]
 [ 0  1  0 10]]
```



Training for all classes High , Low , Medium , Very Low In SVM OVR



Testing for all classes High , Low , Medium , Very Low In SVM OVR

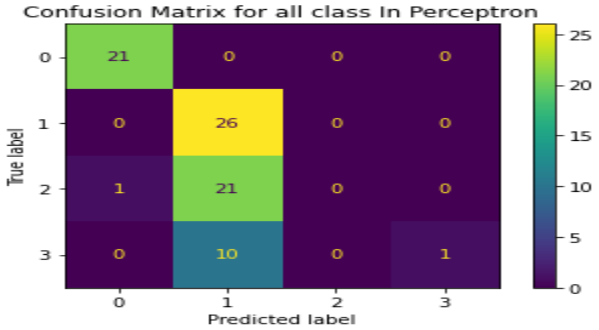


Perceptron using sklearn Confusion Matrix and decision boundary:

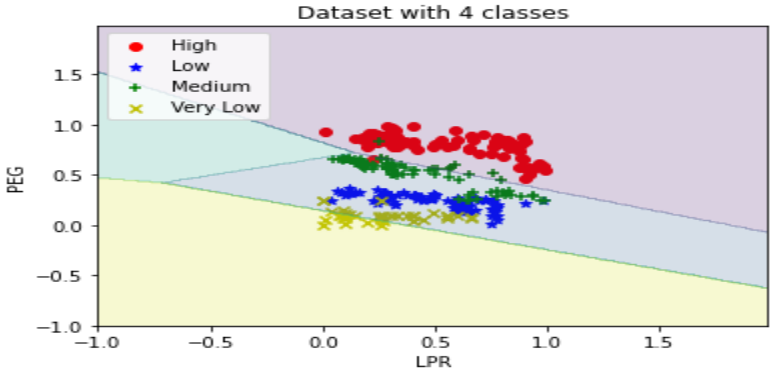
The accuracy of the model: 60%

```
Classification Report for all class In Perceptron
              precision    recall  f1-score   support

           3       1.00      0.09      0.17        11
           0       0.95      1.00      0.98        21
           2       0.00      0.00      0.00        22
           1       0.46      1.00      0.63        26

    accuracy                           0.60        80
   macro avg       0.60      0.52      0.44        80
weighted avg       0.54      0.60      0.48        80
```

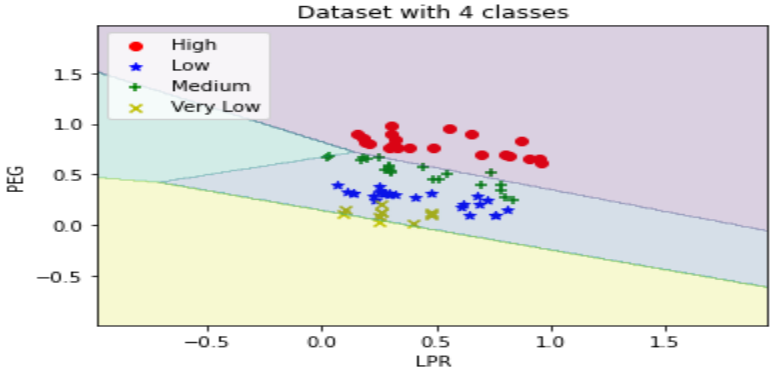
```
[[21  0  0  0]
 [ 0 26  0  0]
 [ 1 21  0  0]
 [ 0 10  0  1]]
```



Training for all classes High , Low , Medium , Very Low In Perceptron



Testing for all classes High , Low , Medium , Very Low In Perceptron



Problem 2: Build OVR-SVM

2.1 obtain the binarized labels

Before binarization

```
y_train
```

```
array([3, 0, 2, 1, 2, 2, 1, 0, 0, 1, 2, 0, 3, 1, 1, 2, 1, 2, 2, 1, 1, 0,
       3, 1, 0, 2, 1, 1, 0, 1, 1, 1, 3, 0, 2, 2, 3, 2, 2, 0, 1, 1, 1, 2,
       0, 1, 2, 2, 1, 3, 3, 2, 0, 3, 0, 3, 1, 1, 1, 1, 2, 2, 1, 1, 3, 1,
       2, 2, 2, 3, 0, 1, 1, 2, 1, 1, 0, 2, 1, 2, 2, 2, 0, 1, 1, 0, 0, 2,
       2, 3, 2, 0, 1, 2, 0, 0, 1, 1, 2, 1, 2, 3, 0, 2, 2, 1, 1, 2, 1, 3,
       2, 2, 1, 3, 1, 0, 0, 1, 2, 2, 2, 0, 2, 2, 2, 0, 0, 3, 0, 2, 0, 1,
       0, 1, 2, 0, 1, 2, 1, 1, 0, 2, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 2, 0,
       2, 0, 2, 1, 1, 2, 3, 3, 2, 3, 1, 0, 2, 2, 0, 2, 2, 0, 2, 2, 1, 0,
       1, 1, 2, 2, 1, 2, 0, 2, 1, 1, 2, 1, 1, 0, 2, 0, 2, 1, 2, 0, 2, 1,
       2, 2, 2, 1, 0, 2, 2, 0, 3, 1, 0, 0, 1, 2, 1, 0, 3, 2, 3, 0, 2, 0,
       3, 0, 3, 1, 0, 1, 1, 1, 2, 1, 2, 3, 1, 2, 0, 1, 2, 0, 2, 2, 3, 0,
       3, 3, 2, 2, 0, 3, 2, 3, 0, 3, 0, 3, 1, 0, 0, 1, 0, 1, 2, 1, 2, 0,
       3, 0, 3, 0, 2, 0, 1, 0, 0, 2, 0, 3, 1, 2, 1, 3, 1, 2, 1, 2, 0, 1,
       3, 1, 2, 1, 2, 1, 1, 1, 2, 2, 0, 1, 2, 0, 0, 3, 1, 2, 0, 1, 0, 3,
       0, 1, 1, 0, 1, 2, 1, 1, 1, 1, 0, 0, 1, 2, 2])
```

After Binarization

```
[1010] y_train
```

```
array([[0, 0, 0, 1],
       [1, 0, 0, 0],
       [0, 0, 1, 0],
       ...,
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 1, 0]])
```

2.2 build the OVR (OVA) and obtain the Accuracy

```
[38] def handMadeOVA(features,Label,x_test,y_test):
        """
        used to make OVA model redicition
        Input : features,Label,x_test,y_test
        Output : returns the probabilities of each model (y_pred) and the model itself (oVoClf)
        """
        oVoClf = svm.SVC(kernel='rbf', probability=True)
        oVoClf.fit(features, Label)
        print('Accuracy of the classifier is : {:.2f}%'.format(getAccuracy(oVoClf, x_test,y_test)))
        getConfusionMatrix(oVoClf,x_test,y_test)
        y_pred = oVoClf.predict_proba(x_test)[:,1].reshape(-1,1)

        return y_pred ,oVoClf
```

The accuracy

(All, High) (100.00%), **(All, Low)** (97.50%), **(All, Median)** (100.00%), **(All, Very Low)** (98.75%).

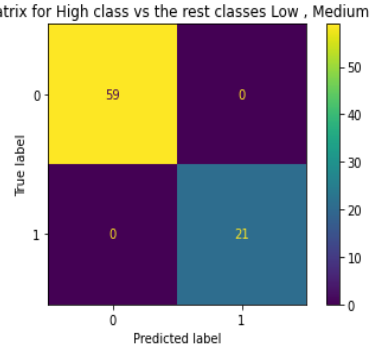
Classification Report for High class vs the rest classes Low , Medium , Very Low

	precision	recall	f1-score	support
0	1.00	1.00	1.00	59
1	1.00	1.00	1.00	21
accuracy			1.00	80
macro avg	1.00	1.00	1.00	80
weighted avg	1.00	1.00	1.00	80

Accuracy of the classifier is : 100.00%

```
[[59  0]
 [ 0 21]]
```

Confusion Matrix for High class vs the rest classes Low , Medium , Very Low



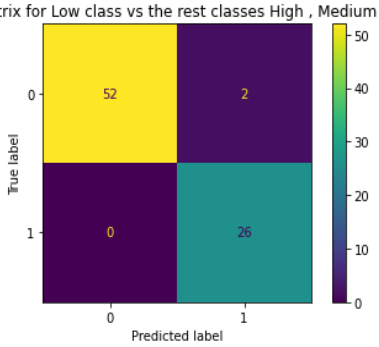
Classification Report for Low class vs the rest classes High , Medium , Very Low

	precision	recall	f1-score	support
0	1.00	0.96	0.98	54
1	0.93	1.00	0.96	26
accuracy			0.97	80
macro avg	0.96	0.98	0.97	80
weighted avg	0.98	0.97	0.98	80

Accuracy of the classifier is : 97.50%

```
[[52  2]
 [ 0 26]]
```

Confusion Matrix for Low class vs the rest classes High , Medium , Very Low



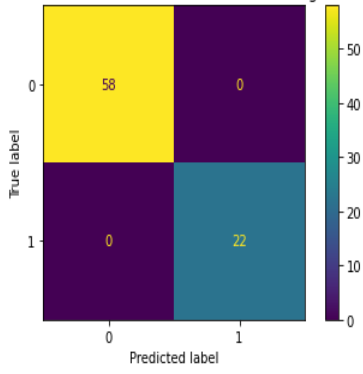
Classification Report for Medium class vs the rest classes High , Low , Very Low

	precision	recall	f1-score	support
0	1.00	1.00	1.00	58
1	1.00	1.00	1.00	22
accuracy			1.00	80
macro avg	1.00	1.00	1.00	80
weighted avg	1.00	1.00	1.00	80

Accuracy of the classifier is : 100.00%

```
[[58  0]
 [ 0 22]]
```

Confusion Matrix for Medium class vs the rest classes High , Low , Very Low



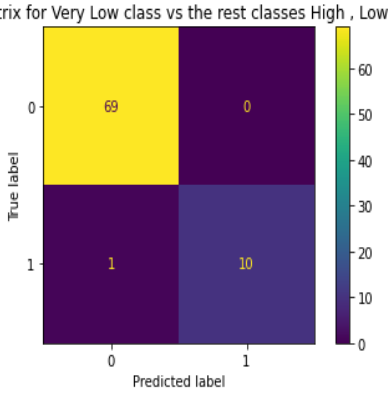
Classification Report for Very Low class vs the rest classes High , Low , Medium

	precision	recall	f1-score	support
0	0.99	1.00	0.99	69
1	1.00	0.91	0.95	11
accuracy			0.99	80
macro avg	0.99	0.95	0.97	80
weighted avg	0.99	0.99	0.99	80

Accuracy of the classifier is : 98.75%

```
[[69  0]
 [ 1 10]]
```

Confusion Matrix for Very Low class vs the rest classes High , Low , Medium

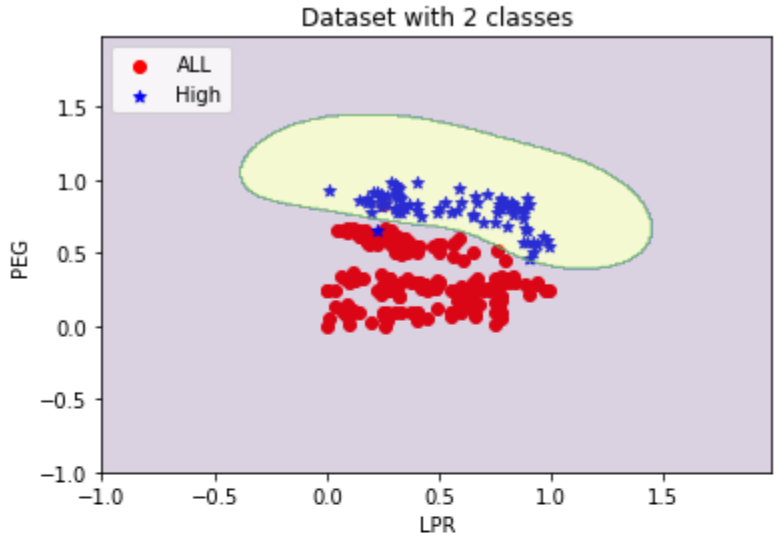


2.3 plots and Boundaries for both training and testing data

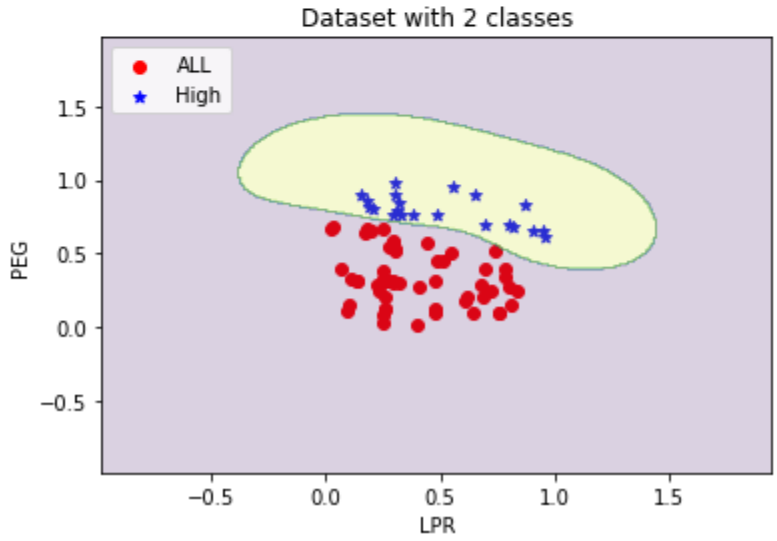
Decision Boundary: (All, High), (All, Low), (All, Median), (All, Very Low)

(All, High)

Training for High class vs the rest classes Low , Medium , Very Low

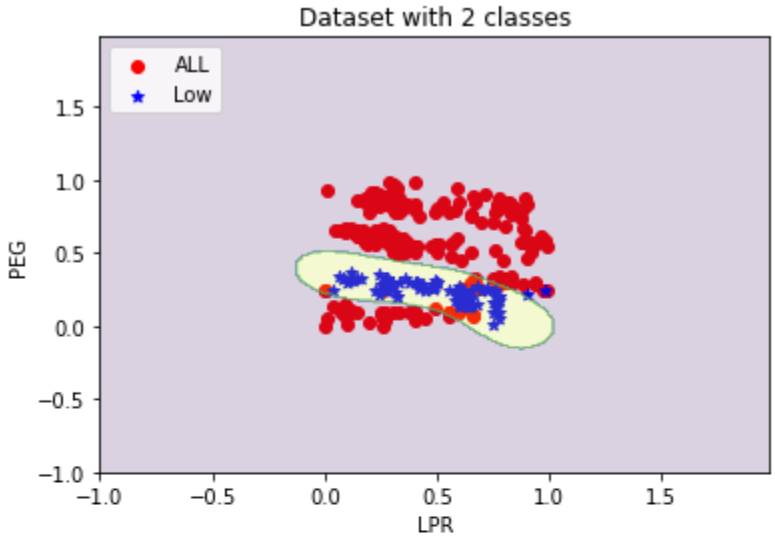


Testing for High class vs the rest classes Low , Medium , Very Low

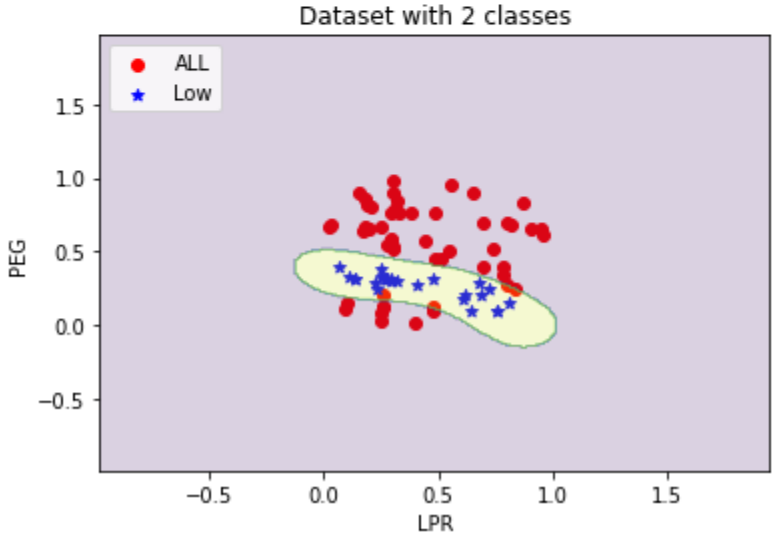


(All, Low)

```
Training for Low class vs the rest classes High , Medium , Very Low
```

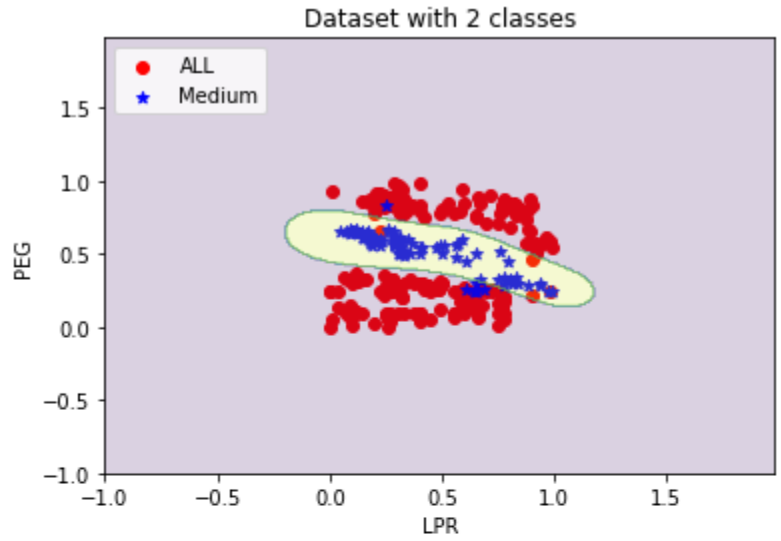



```
Testing for Low class vs the rest classes High , Medium , Very Low
```

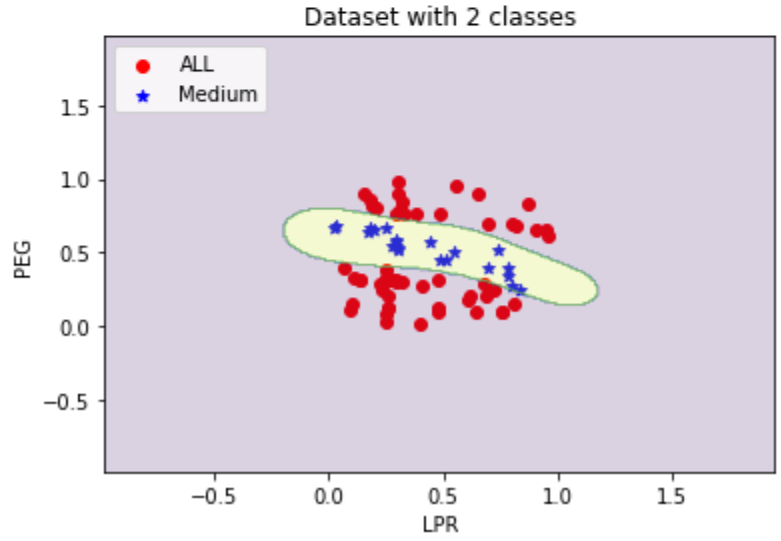


(All, Median)

```
Training for Medium class vs the rest classes High , Low , Very Low
```

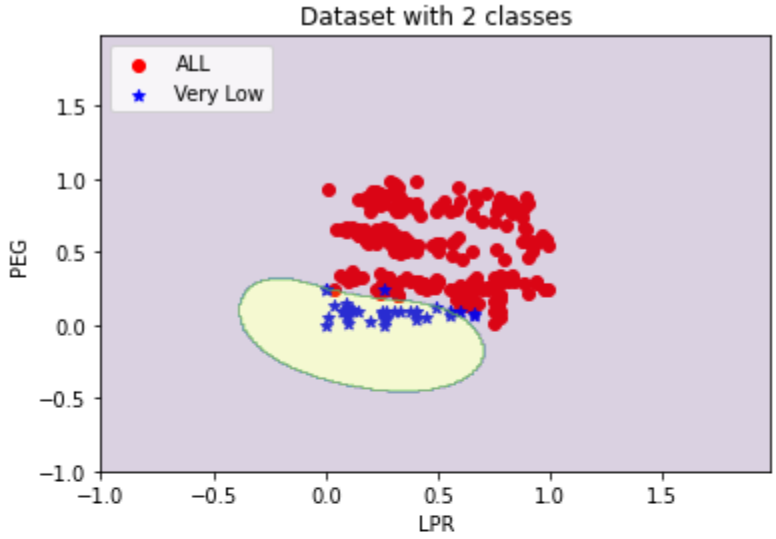



```
Testing for Medium class vs the rest classes High , Low , Very Low
```

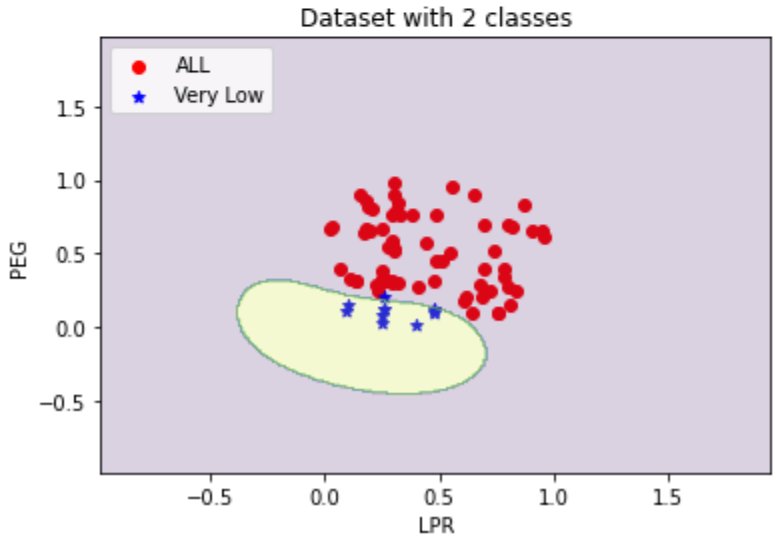


(All, Very Low)

```
Training for Very Low class vs the rest classes High , Low , Medium
```




```
Testing for Very Low class vs the rest classes High , Low , Medium
```



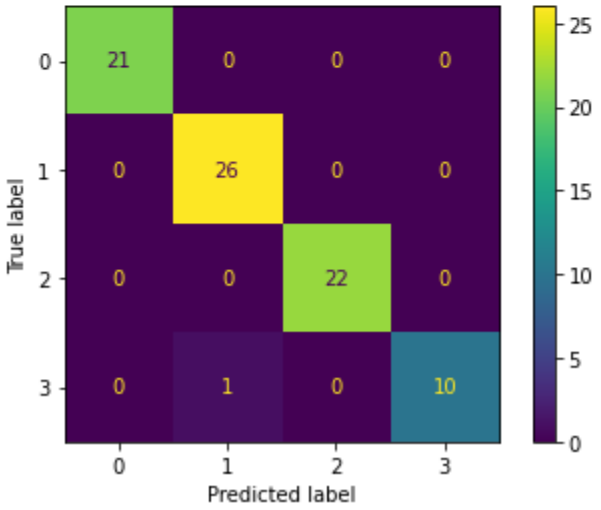
(2.B) using argmax to obtain the final predicted labels

```
m1 = mlb.classes_[np.argmax(y_all, axis=1)]
```

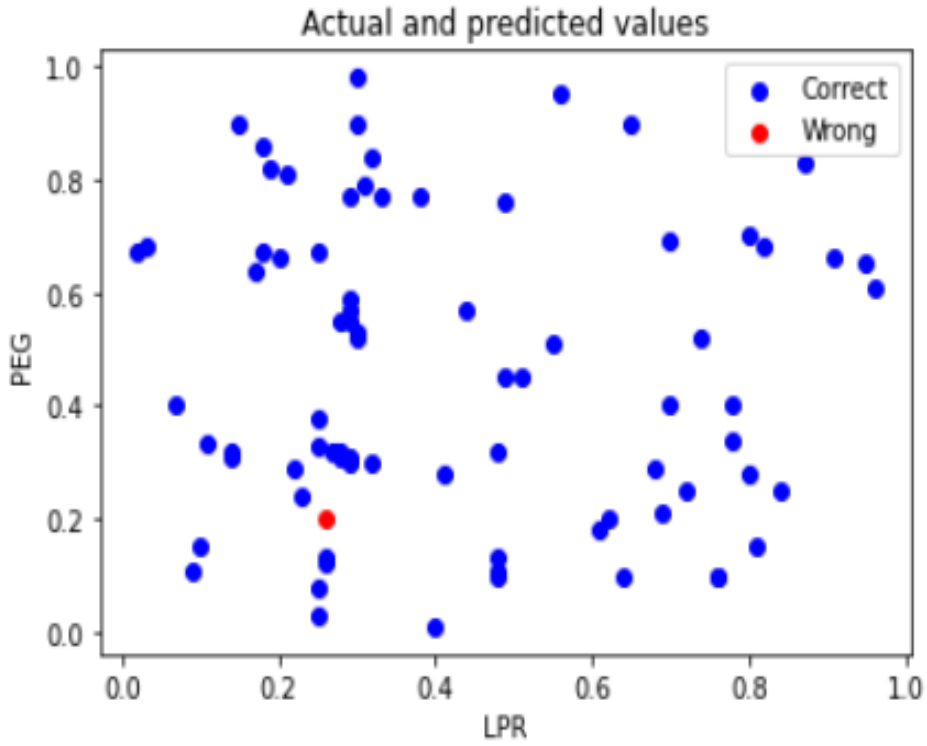
```
#print the accuracy of the aggregation
from sklearn.metrics import accuracy_score
print("Accuarcy {} %".format(accuracy_score(list(check_testSet),list(m1))*100))
```

```
Accuarcy 98.75 %
```

```
Confusion Matrix for OVR argmax list
Class names = Low , Very Low
Updated class index: [1, 3]
Number of samples: 323
Number of features: 2

Classification Report for first class Low:

              precision    recall  f1-score   support

           0       0.00      0.00      0.00        21
           1       0.37      1.00      0.54        26
           2       0.00      0.00      0.00        22
           3       1.00      0.91      0.95        11

    accuracy                           0.45        80
   macro avg       0.34      0.48      0.37        80
weighted avg       0.26      0.45      0.31        80
```



```
#plotting correct and wrong prediction points
_,ax = plt.subplots()

ax.scatter(x=X_test_DF[check_testSet==m1][0],
           y=X_test_DF[check_testSet==m1][1],
           c = 'blue', label = 'Correct', alpha = 1)
ax.scatter(x=X_test_DF[check_testSet!=m1][0],
           y=X_test_DF[check_testSet!=m1][1],
           c = 'red', label = 'Wrong', alpha = 1)

plt.title('Actual and predicted values')
plt.xlabel(axis_labels[0])
plt.ylabel(axis_labels[1])
plt.legend()
plt.show()
```



Accuracy comment: **the accuracy of the model is nice, maybe because we have a small number of data.**

Problem 3: Build OVO-SVM:

3.1 obtain the binarized labels: already explained in problem 2

3.2 build the OVR (OVA) and obtain the Accuracy

We built 6 model because the target contains 4 classes so **(n*(n-1)/2 = 6)**

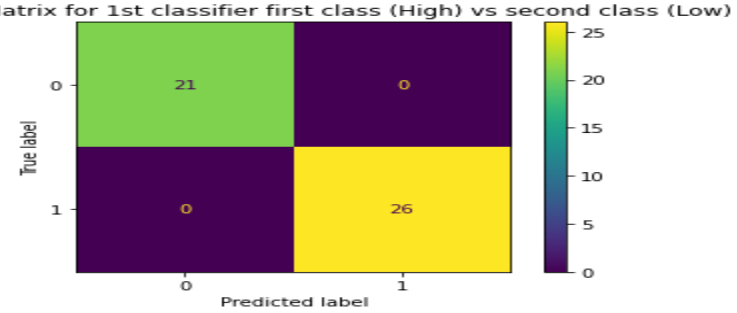
```
[56] def handMadeOVO(features,Label,x_test,y_test):
         """
         used to make OVO model redicition
         Input : features,Label,x_test,y_test
         Output : returns the probabilities of each model (y_pred0, y_pred1) and the model itself (oVoClf)
         """
         oVoClf = svm.SVC(kernel='rbf', decision_function_shape='ovo',probability=True)
         oVoClf.fit(features, Label)
         print('Accuracy of the classifier is : {:.2f}%'.format(getAccuracy(oVoClf, x_test,y_test)))
         getConfusionMatrix(oVoClf,x_test,y_test)
         y_pred_0 = oVoClf.predict_proba(features)[:,0]
         y_pred_1 = oVoClf.predict_proba(features)[:,1]
         return y_pred_0,y_pred_1,oVoClf
```

The accuracy: **(0, 1)** is (100%), **(0, 2)** is (100%) , **(0,3)** is (100%), **(1, 2)** is (100%), **(1, 3)** is (97.30%) and **(2,3)** is (100%)

Confusion Matrix: (0, 1), (0, 2), (0,3), (1, 2), (1, 3) and (2, 3)

(HIGH, LOW)

```
Accuracy of the classifier is : 100.00%
[[21  0]
 [ 0 26]]
```

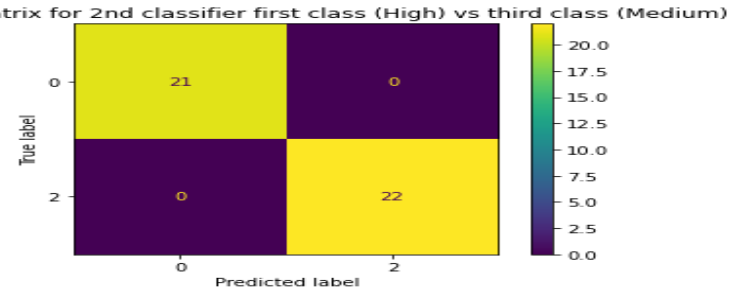



```
Class names = High , Low
Updated class index: [0, 1]
Number of samples: 323
Number of features: 2

Classification Report for first class High:
```

	precision	recall	f1-score	support
0	0.62	1.00	0.76	21
1	0.57	1.00	0.72	26
2	0.00	0.00	0.00	22
3	0.00	0.00	0.00	11
accuracy			0.59	80
macro avg	0.30	0.50	0.37	80
weighted avg	0.35	0.59	0.44	80

(HIGH, MEDIUM)

```
Accuracy of the classifier is : 100.00%
[[21  0]
 [ 0 22]]
```

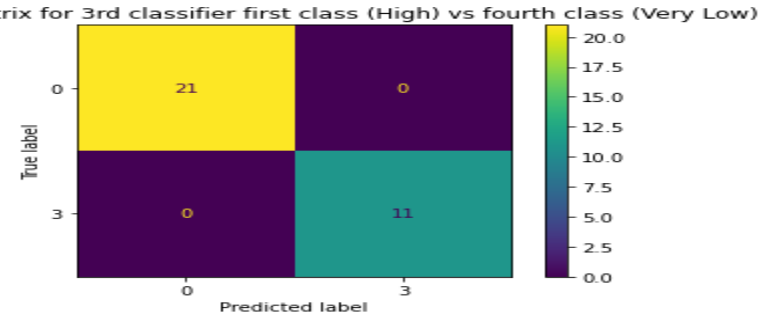



```
Class names = High , Medium
Updated class index: [0, 2]
Number of samples: 323
Number of features: 2

Classification Report for first class High:
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	21
1	0.00	0.00	0.00	26
2	0.37	1.00	0.54	22
3	0.00	0.00	0.00	11
accuracy			0.54	80
macro avg	0.34	0.50	0.39	80
weighted avg	0.37	0.54	0.41	80

(HIGH, VERY LOW)

```
Accuracy of the classifier is : 100.00%
[[21  0]
 [ 0 11]]
```

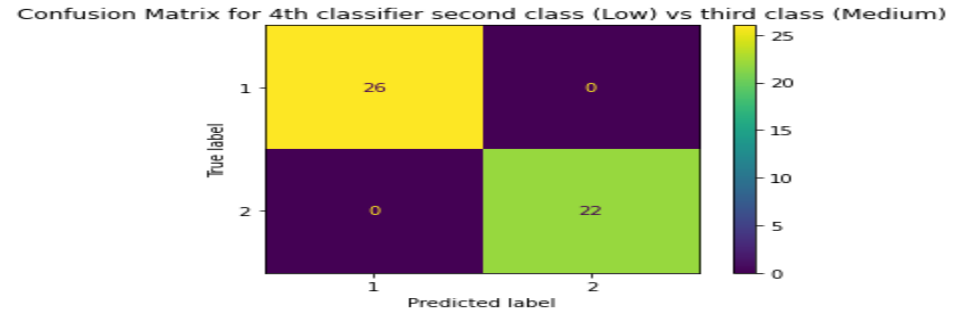



```
Class names = High , Very Low
Updated class index: [0, 3]
Number of samples: 323
Number of features: 2

Classification Report for first class High:
```

	precision	recall	f1-score	support
0	0.50	1.00	0.67	21
1	0.00	0.00	0.00	26
2	0.00	0.00	0.00	22
3	0.29	1.00	0.45	11
accuracy			0.40	80
macro avg	0.20	0.50	0.28	80
weighted avg	0.17	0.40	0.24	80

(LOW, MEDIUM)

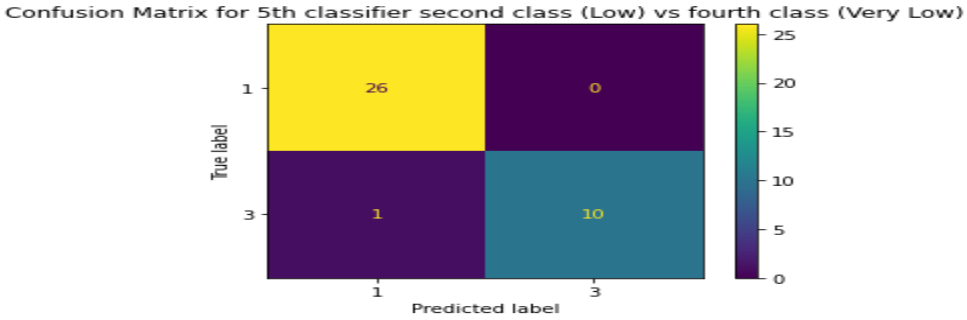
```
Accuracy of the classifier is : 100.00%
[[26  0]
 [ 0 22]]
```




```
Class names = Low , Medium
Updated class index: [1, 2]
Number of samples: 323
Number of features: 2

Classification Report for first class Low:
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	21
1	0.70	1.00	0.83	26
2	0.51	1.00	0.68	22
3	0.00	0.00	0.00	11
accuracy			0.60	80
macro avg	0.30	0.50	0.38	80
weighted avg	0.37	0.60	0.45	80

(LOW, VERY LOW)

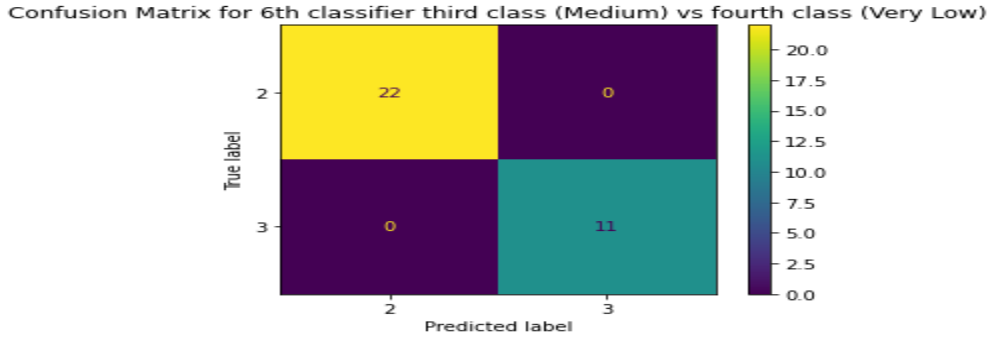



```
Class names = Low , Very Low
Updated class index: [1, 3]
Number of samples: 323
Number of features: 2

Classification Report for first class Low:
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	21
1	0.37	1.00	0.54	26
2	0.00	0.00	0.00	22
3	1.00	0.91	0.95	11
accuracy			0.45	80
macro avg	0.34	0.48	0.37	80
weighted avg	0.26	0.45	0.31	80

(MEDIUM, VERY LOW)

```
Accuracy of the classifier is : 100.00%
[[22  0]
 [ 0 11]]
```

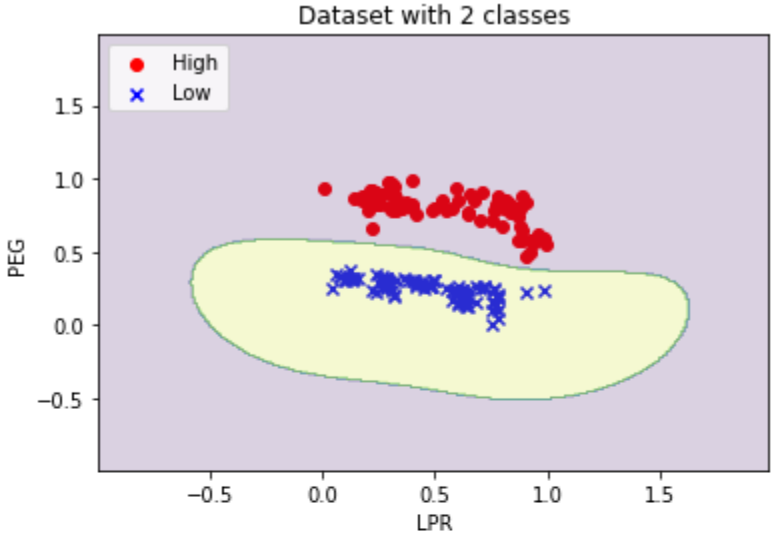



```
Class names = Medium , Very Low
Updated class index: [2, 3]
Number of samples: 323
Number of features: 2

Classification Report for first class Medium:

              precision    recall  f1-score   support

           0       0.00      0.00      0.00        21
           1       0.00      0.00      0.00        26
           2       0.43      1.00      0.60        22
           3       0.38      1.00      0.55        11

    accuracy                           0.41        80
   macro avg       0.20      0.50      0.29        80
weighted avg       0.17      0.41      0.24        80
```

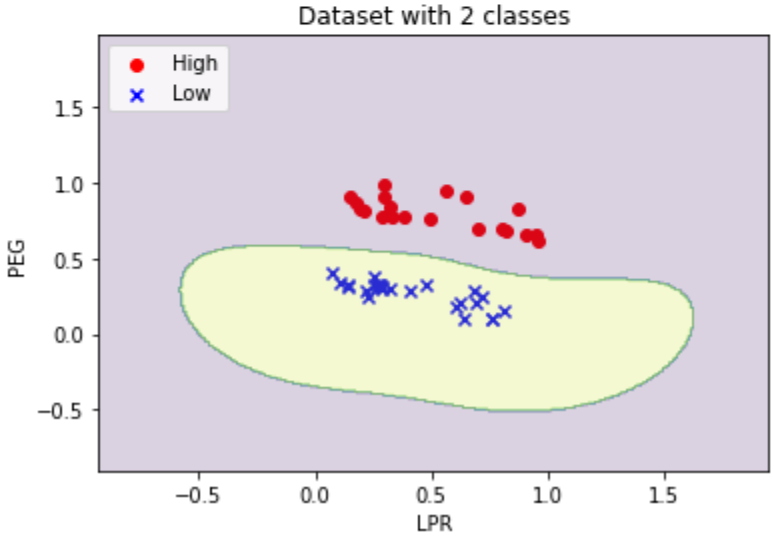
3.3 plots and Boundaries for both training and testing data

(HIGH, LOW)

Training for 1st classifier first class (High) vs second class (Low)

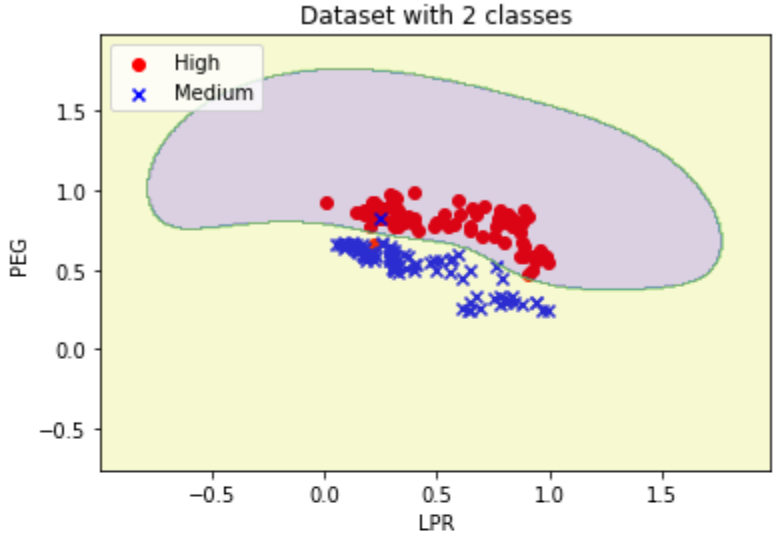


Testing for 1st classifier first class (High) vs second class (Low)

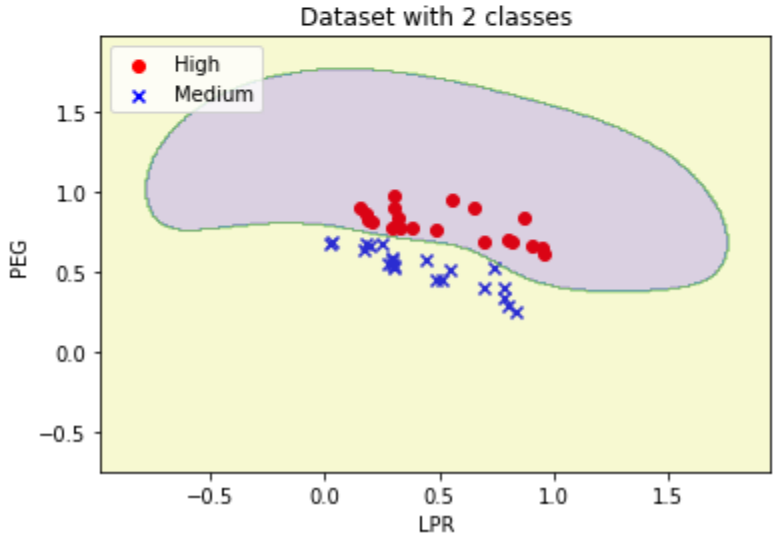


(HIGH, MEDIUM)

```
Training for 2nd classifier first class (High) vs third class (Medium)
```

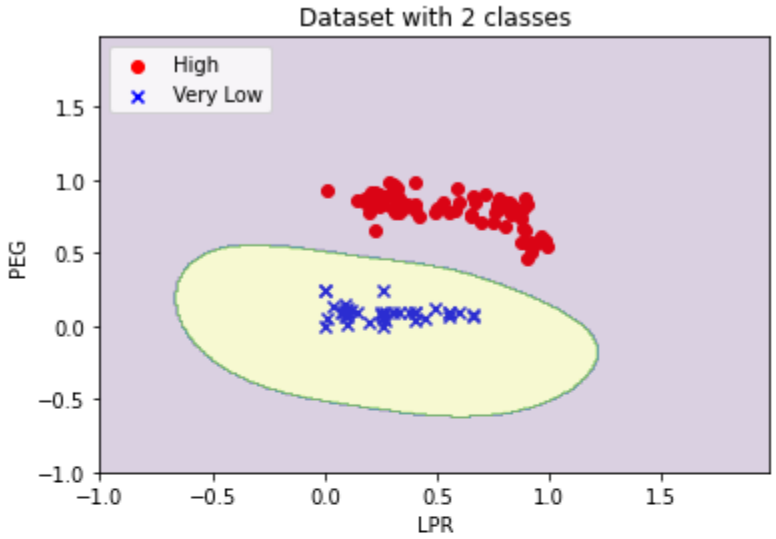



```
Testing for 2nd classifier first class (High) vs third class (Medium)
```

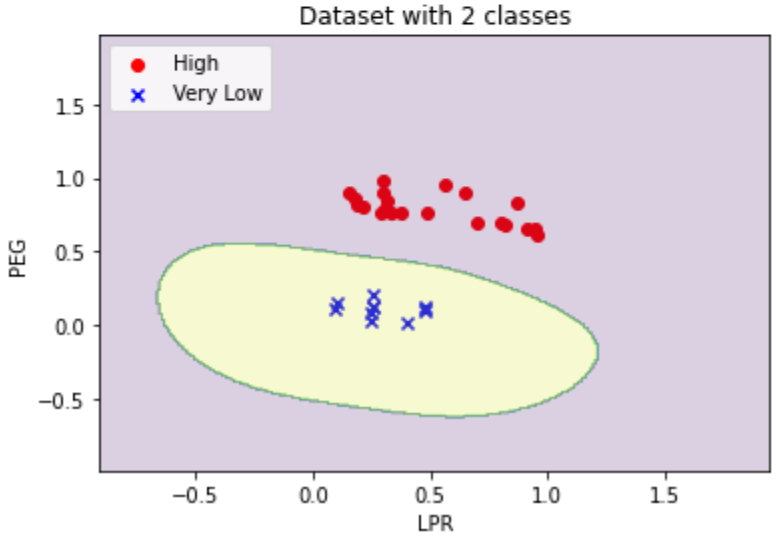


(HIGH, VERY LOW)

```
Training for 3rd classifier first class (High) vs fourth class (Very Low)
```

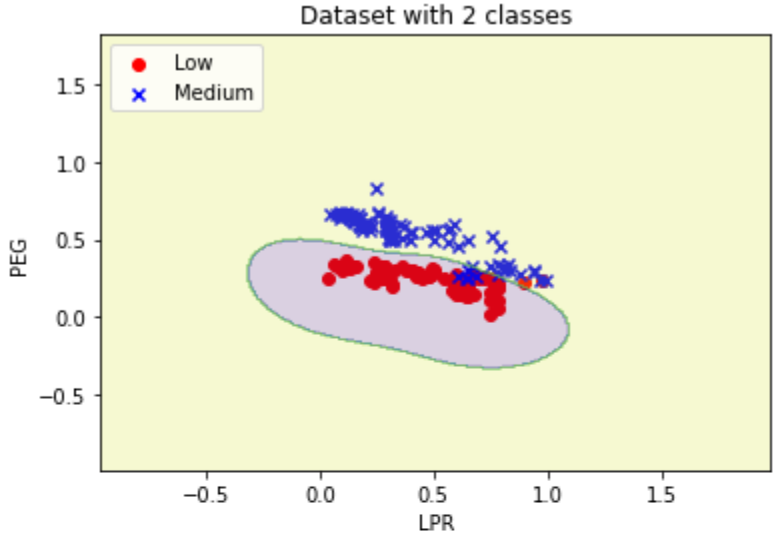



```
Testing for 3rd classifier first class (High) vs fourth class (Very Low)
```

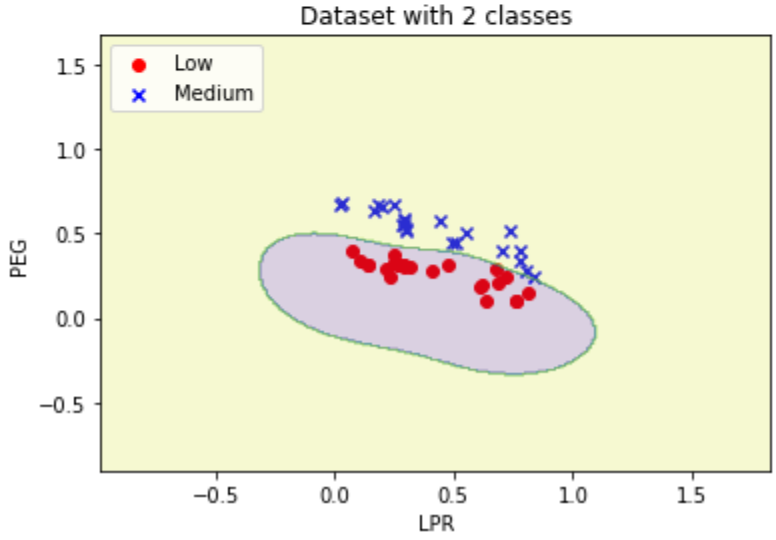


(LOW, MEDIUM)

Training for 4th classifier second class (Low) vs third class (Medium)

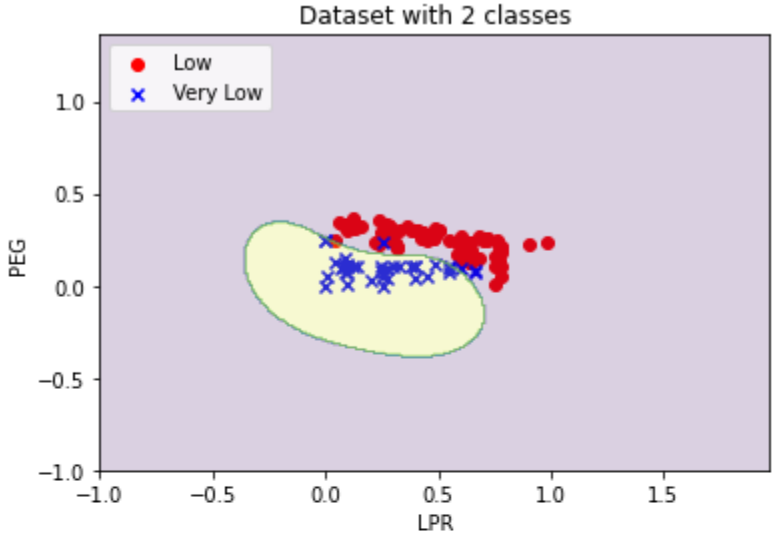


Testing for 4th classifier second class (Low) vs third class (Medium)

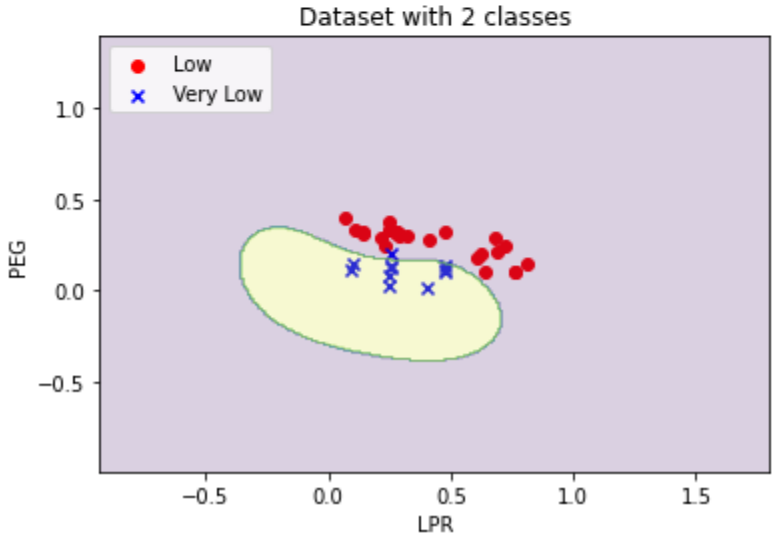


(LOW, VERY LOW)

```
Training for 5th classifier second class (Low) vs fourth class (Very Low)
```

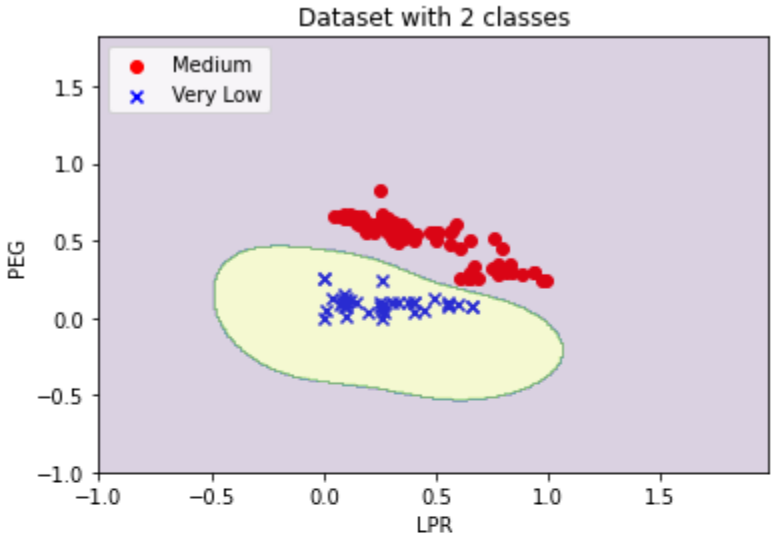



```
Testing for 5th classifier second class (Low) vs fourth class (Very Low)
```

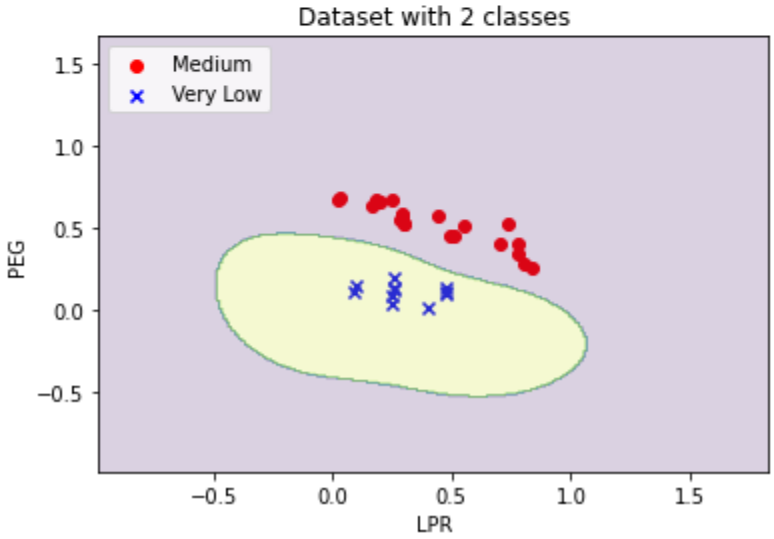


(MEDIUM, VERY LOW)

```
Training for 6th classifier third class (Medium) vs fourth class (Very Low)
```




```
Testing for 6th classifier third class (Medium) vs fourth class (Very Low)
```



3.3 Use argmax to aggregate confidence scores and obtain the final label and obtain the performance

comment: because we are using one versus one technique that made us face a problem which is how we obtain the opinion of each model in a specific sample. that is because each model will say for example it's 0 or 1, but each time 0 and 1 have a different meaning depending on the model itself, so we took two approaches to solve this problem and get all the probability of a sample to belong to specific class from all classifiers:

The first approach: we built a weight matrix, and the mission was to fill this matrix with the summation of the probability of each class, that's why we called it a weight matrix because the values not located between 0 and 1 like probabilities

The second approach: we built a voting matrix, and the mission was to fill this matrix with the summation of the raised hands "how many classifiers said this sample belongs to that class"

the accuracy of the first approach was so bad, so we used the second approach.

the first approach

```
# depending on the length of the X_test set
#I'll create a list of list of zeros each item is a 4X1 list
weightMatrix = []
for i in range(len(X_test)):
    weightMatrix.append(list(np.zeros(4,int)))
```

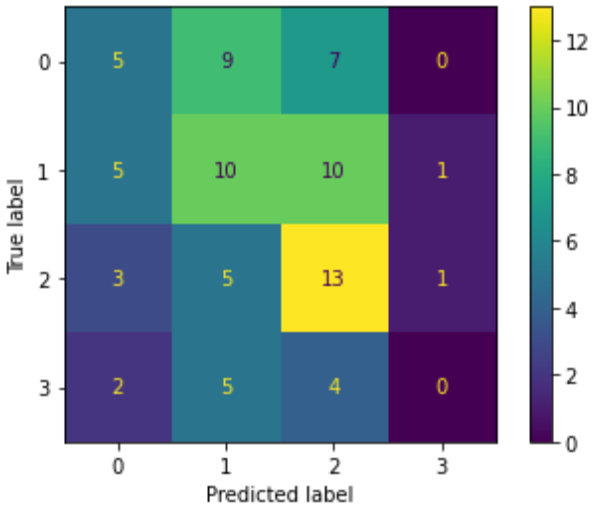
```
# depending on the length of the X_test set
#I'll create a list of list of zeros each item is a 4X1 list
votingMatrix = []
for i in range(len(X_test)):
    votingMatrix.append(list(np.zeros(4,int)))
```

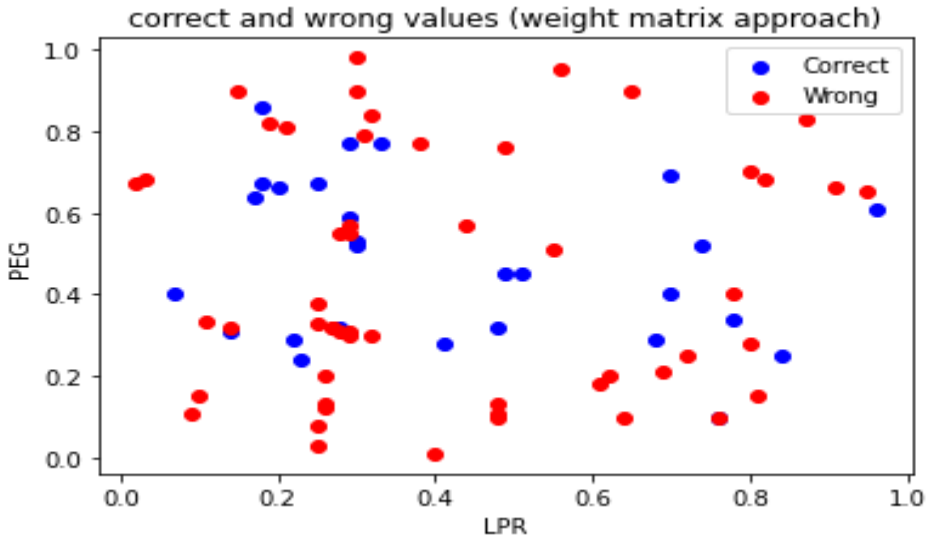
```
def fillWeightMatrix(weightMatrix,ypred_0,ypred_1,class0,class1):
    for i in range(len(X_test)):
        weightMatrix[i][class0] += ypred_0[i]
        weightMatrix[i][class1] += ypred_1[i]
```

```
def fillvotingMatrix(votingMatrix,prediction):
    for i in range(len(X_test)):
        votingMatrix[i][prediction[i]] += 1
```

```
fillWeightMatrix(weightMatrix,ypred1_0,ypred1_1,cls_newtrian1[0],cls_newtrian1[1])
fillWeightMatrix(weightMatrix,ypred2_0,ypred2_1,cls_newtrian2[0],cls_newtrian2[1])
fillWeightMatrix(weightMatrix,ypred3_0,ypred3_1,cls_newtrian3[0],cls_newtrian3[1])
fillWeightMatrix(weightMatrix,ypred4_0,ypred4_1,cls_newtrian4[0],cls_newtrian4[1])
fillWeightMatrix(weightMatrix,ypred5_0,ypred5_1,cls_newtrian5[0],cls_newtrian5[1])
fillWeightMatrix(weightMatrix,ypred6_0,ypred6_1,cls_newtrian6[0],cls_newtrian6[1])
```

```
Confusion Matrix for the first approach (weight matrix)
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f9118ce08d0>
```






```
print(f"Accuracy depending on the maximum of( summation of probabilities ) : {accuracy_score(list(check_testSet),list(votingResults))*100}%")
```

```
Accuracy depending on the maximum of( summation of probabilities ) : 33.75%
```

---

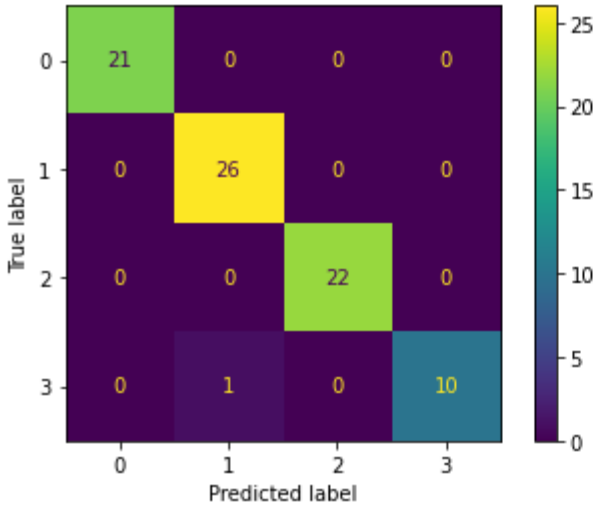
the second approach

```
handUpResults = mlb.classes_[np.argmax(votingMatrix, axis=1)]
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(list(check_testSet),list(handUpResults))
print(cm)
```

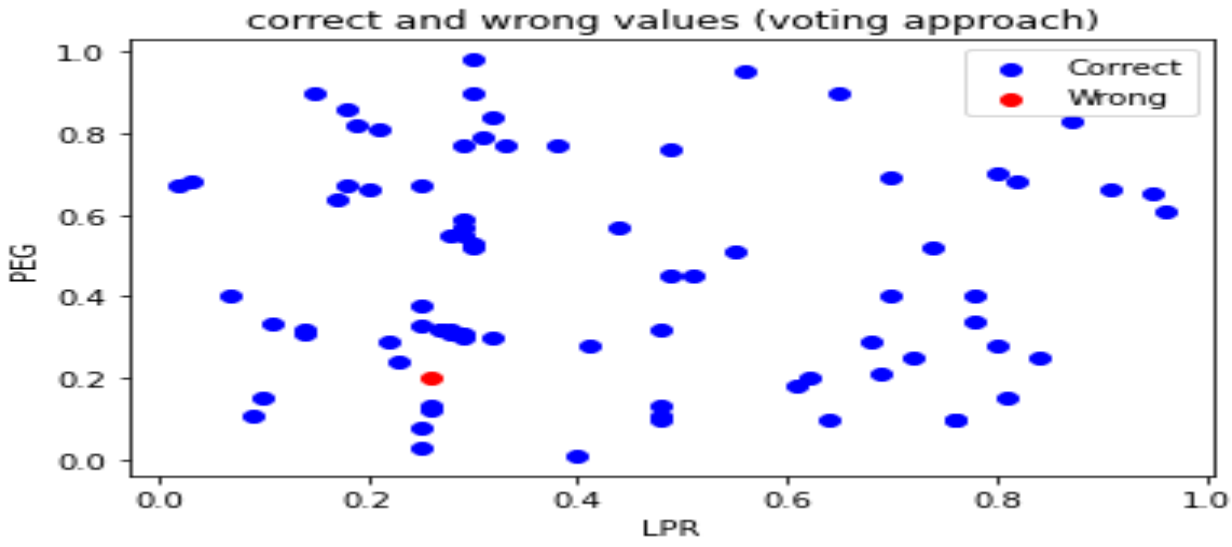
```
[[21  0  0  0]
 [ 0 26  0  0]
 [ 0  0 22  0]
 [ 0  1  0 10]]
```

```
Confusion Matrix for the first approach (voting)
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f9118bda750>
```




```
print(f"Accuracy depending on the voting  :
{accuracy_score(list(check_testSet),list(handUpResults))*100}%")
```

```
Accuracy depending on the voting  : 98.75%
```



Conclusion:

we loaded the dataset and understood it carefully, we checked the correlation between the features and the label and we selected two features with the highest correlation ratio with the target.

in problem 1 we built three models and trained them with the selected features, some models gave good predictions such as SVM and others gave fewer predictions such as Perceptron. then we used the MultiLabelBinarizer library to binarize the encoded target then we built an OVR-SVM and OVO-SVM (in problems 2 and 3).

we learned how to use OVO-SVM and OVR-SVM, and we also learned the difference between both of them and the mechanism that each one of them uses to predict the class of a sample.

some of the problems that we faced while working on this assignment:

1. in problem 2 some of the decision boundaries were a bit weird but when we used the kernel trick

("rbf" kernel) it got much better. So, we tried to use the kernel trick also in problem 1 and we got a higher accuracy than the accuracy with the linear kernel

2. in problem 3 the weight method to collect the probabilities of each sample to be in a specific class the method didn't give us a good accuracy so we used the second approach which was to get the voting results.