Applied machine learning

assignment 1

Team members-Group 8:

- Abdelrhman Gaber Youssef Saad Rezkallah
- Eman Metwally Mohammed Abood
- Basma Reda Shaban Abd-Elsalam Abd-Elwahab

Problem:

The objective of this assignment is to solve a multiclassification problem using SVM model and perceptron model, using One Versus Rest (OVR) SVM, use argmax to aggregate the confidence score, obtain the final predicted labels, measure the performance of the model, and apply One Versus One (OVO) SVM.

Implementation steps:

- **Load the Data User Modeling Dataset (DUMP), define the target label, and perform label encoding**

Loading the DUMB dataset for training, and apply it for testing sheet also.

```
from google.colab import files
files.upload()
```

Choose Files Data_User...HRAMAN.xls

- **Data_User_Modeling_Dataset_Hamdi Tolga KAHRAMAN.xls**(application/vnd.ms-excel) - 57856 bytes, last modified: 5/26/2022 - 100% done

```
Saving Data_User_Modeling_Dataset_Hamdi Tolga KAHRAMAN.xls to Data_User_Modeling_Dataset_Hamdi Tolga KAHRAMAN.xls
```

{'Data_User_Modeling_Dataset_Hamdi Tolga KAHRAMAN.xls': b'\xd0\xcf\x11\xe0\xa1\xb1\x1a\xe1\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0

[5]

```
df = pd.read_excel('Data_User_Modeling_Dataset_Hamdi Tolga KAHRAMAN.xls',sheet_name='Training_Data')
```

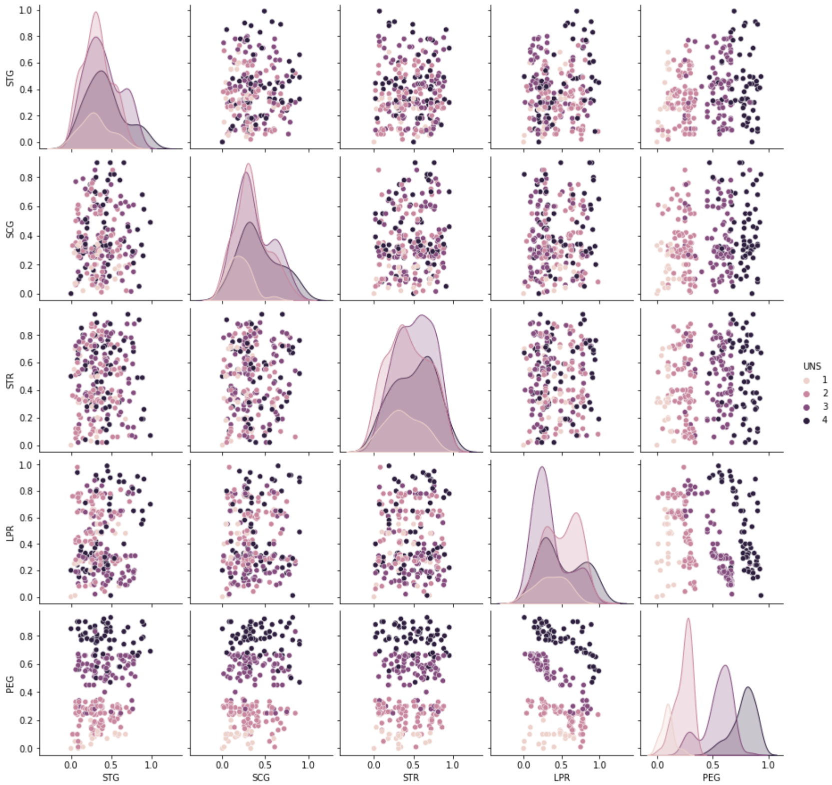
Perform label encoding for 'UNS' column

	STG	SCG	STR	LPR	PEG	UNS
0	0.00	0.00	0.00	0.00	0.00	1
1	0.08	0.08	0.10	0.24	0.90	4
2	0.06	0.06	0.05	0.25	0.33	2
3	0.10	0.10	0.15	0.65	0.30	3
4	0.08	0.08	0.08	0.98	0.24	2

➢ **Extracting the most two important features:**

- The pairplot diagram can show the most important features to select, A pairplot plot a pairwise relationships in a dataset.
- The pairplot function creates a grid of Axes such that each variable in data will by shared in the y-axis across a single row and in the x-axis across a single column.

Pairplot of DUMB dataset



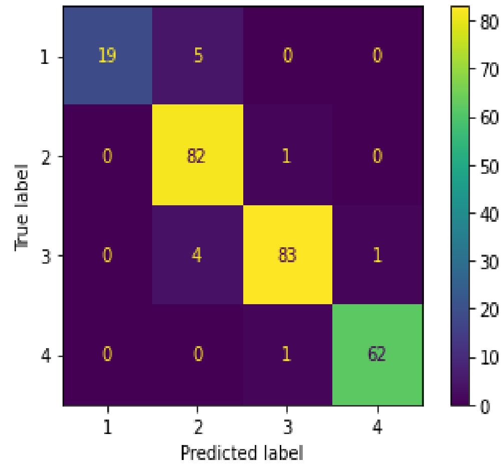
- The Two features selected are ‘PEG’ and ‘LPR,’ because the plot shows that we can separate between these classes.

- **Apply SVM and perceptron classifiers:**

```
from sklearn.metrics import classification_report,plot_confusion_matrix
print(classification_report(svm.predict(x_train),y_train))
```

	precision	recall	f1-score	support
1	0.79	1.00	0.88	19
2	0.99	0.90	0.94	91
3	0.94	0.98	0.96	85
4	0.98	0.98	0.98	63
accuracy			0.95	258
macro avg	0.93	0.97	0.94	258
weighted avg	0.96	0.95	0.95	258

Classification report for training SVM

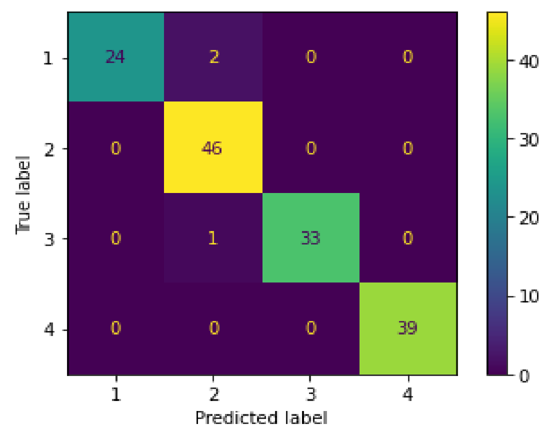


confusion matrix for training SVM

```
#to print the classification report of the prediction label for SVM
print(classification_report(y_test,y_pred))
```

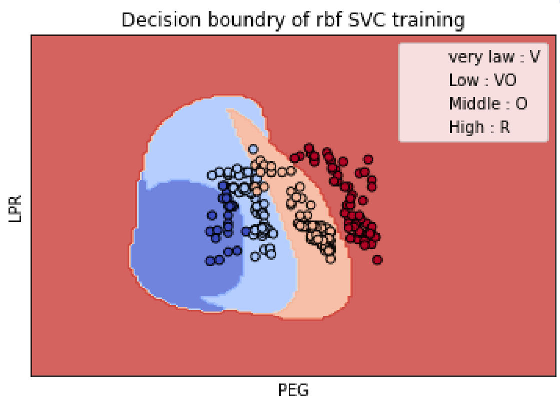
	precision	recall	f1-score	support
1	1.00	0.92	0.96	26
2	0.94	1.00	0.97	46
3	1.00	0.97	0.99	34
4	1.00	1.00	1.00	39
accuracy			0.98	145
macro avg	0.98	0.97	0.98	145
weighted avg	0.98	0.98	0.98	145

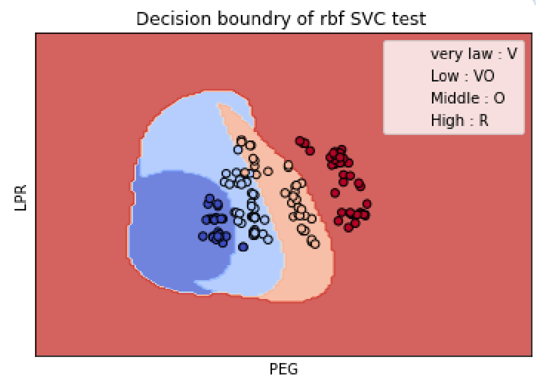
Classification report for testing SVM



confusion matrix for training SVM

- After evaluating the SVM model, we found that the accuracy of the model for predicting the target label is particularly good.



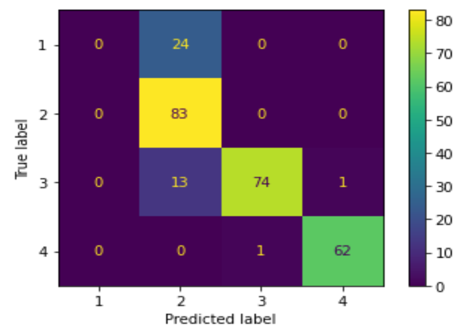


- **Implementing perceptron model:**

Classification report for training

	precision	recall	f1-score	support
1	0.00	0.00	0.00	0
2	1.00	0.69	0.82	120
3	0.84	0.99	0.91	75
4	0.98	0.98	0.98	63
accuracy			0.85	258
macro avg	0.71	0.67	0.68	258
weighted avg	0.95	0.85	0.88	258

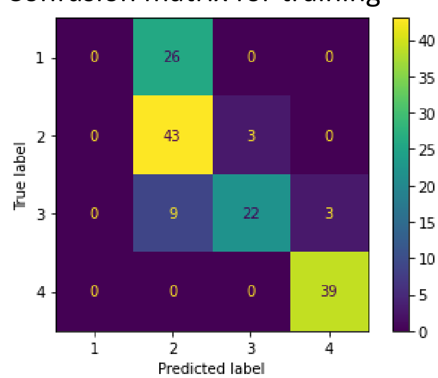
Confusion matrix for training

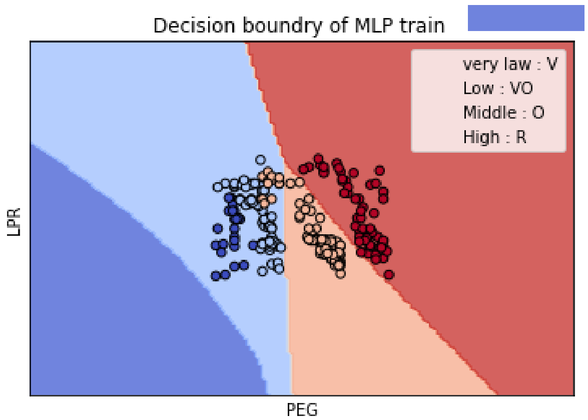


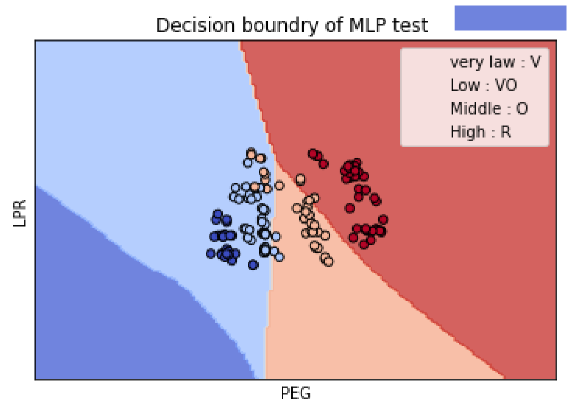
Classification report for training

	precision	recall	f1-score	support
1	0.00	0.00	0.00	0
2	0.93	0.55	0.69	78
3	0.65	0.88	0.75	25
4	1.00	0.93	0.96	42
accuracy			0.72	145
macro avg	0.65	0.59	0.60	145
weighted avg	0.90	0.72	0.78	145

Confusion matrix for training







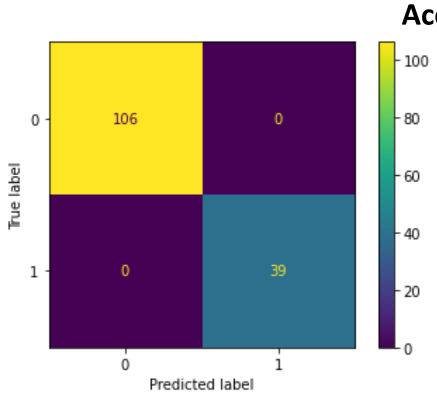
- The accuracy of SVM model is better than perceptron model, so the SVM model is the best model for this problem.

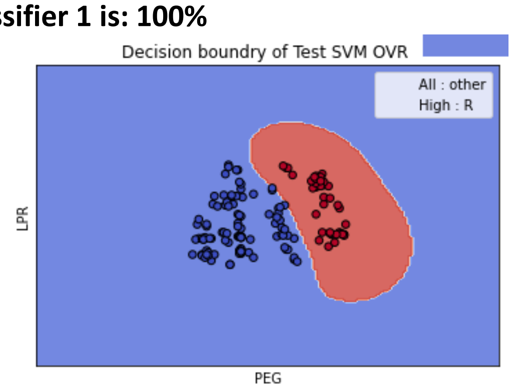
➢ **Apply OVR on SVM model:**

1- Binarize the data:

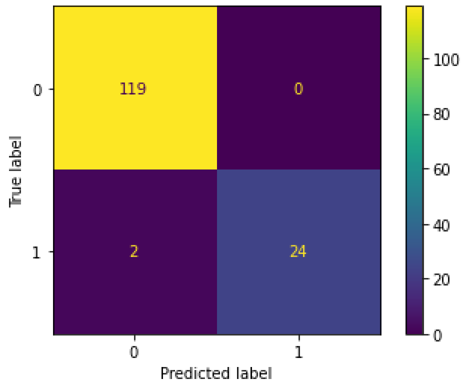
```
[49] #training data for ovr 1
     x_train_ovr_1 = df_1[['PEG','LPR']]

[50] y_train_ovr_1 = df_1['UNS'].map({4 : 1, 3 : 0, 2 : 0, 1 : 0})# High = 1 All = 0
     print(y_train_ovr_1)

     0      0
     1      1
     2      0
     3      0
     4      0
           ..
     253    1
     254    0
     255    1
     256    0
     257    1
     Name: UNS, Length: 258, dtype: int64
```

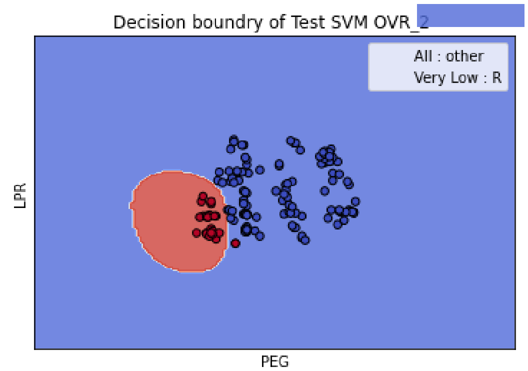
Accuracy of classifier 1 is: 100%



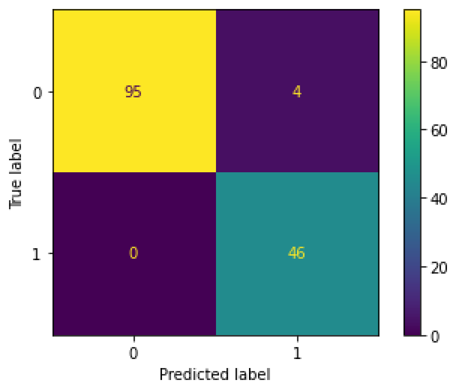


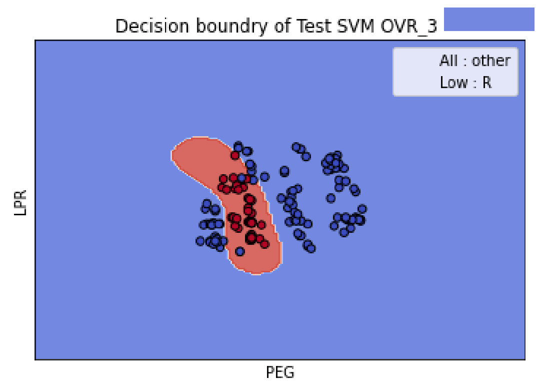
Accuracy of classifier 2 is: 99%



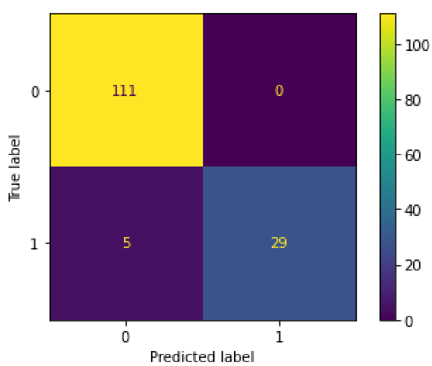


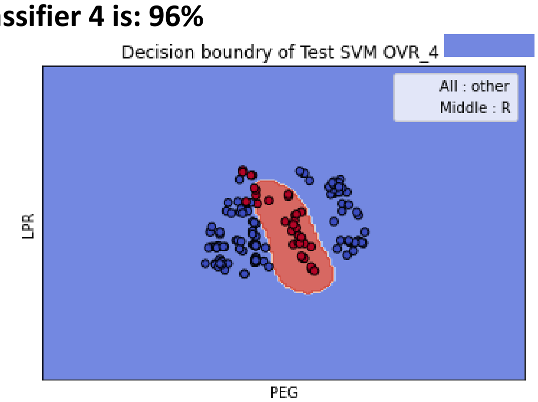
Accuracy of classifier 3 is: 97%





Accuracy of classifier 4 is: 96%



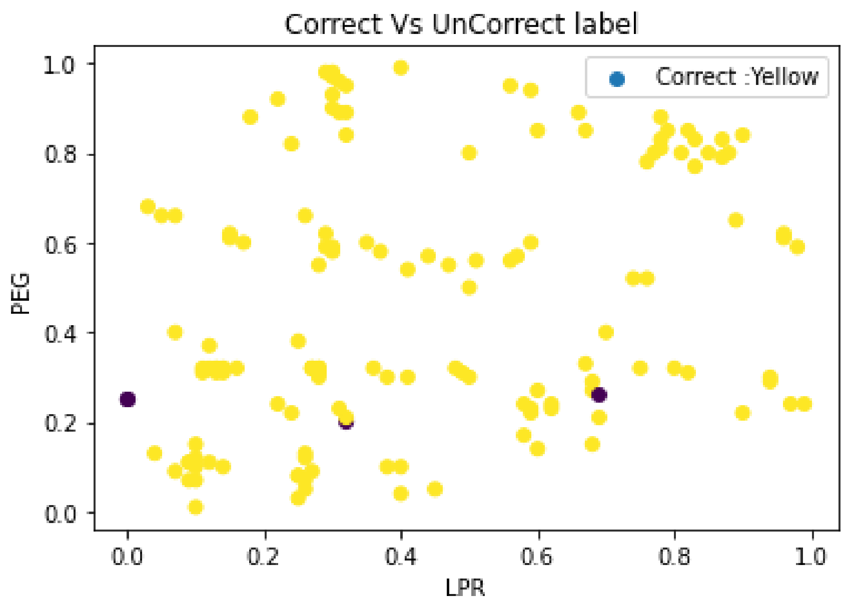


- **Apply argmax to aggregate confidence score and model performance:**

The accuracy of SVM model in OVR is particularly good, after applying the probability the model can classify the classes successfully, the actual class is the yellow one in the figure.

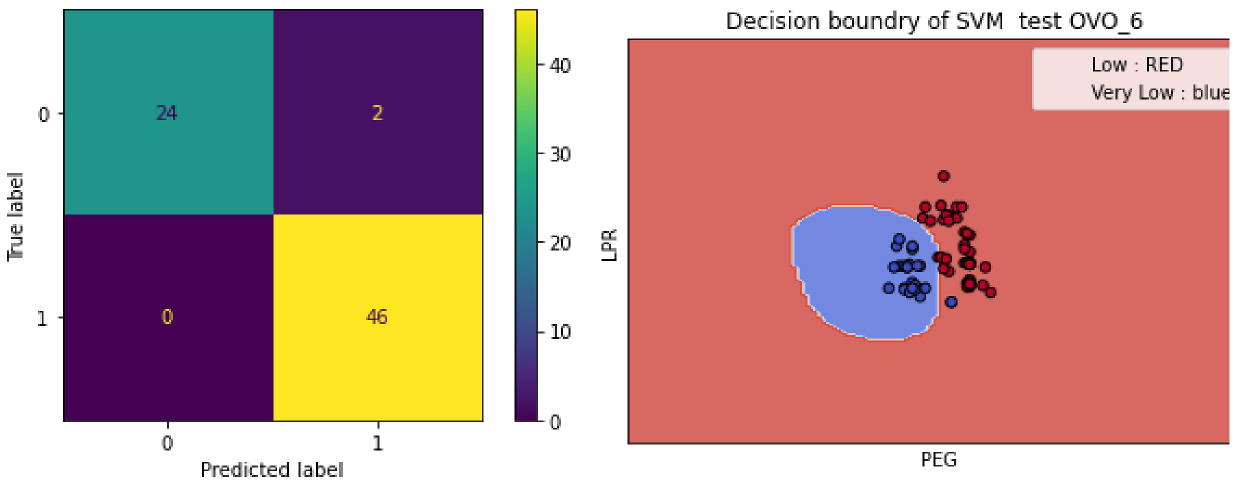
```
[ ]  svm_ovr_proba_1 = SVC(probability=True)
     svm_ovr_proba_1.fit(x_train_ovr_1,y_train_ovr_1)
     svm_ovr_proba_2 = SVC(probability=True)
     svm_ovr_proba_2.fit(x_train_ovr_1,y_train_ovr_2)
     svm_ovr_proba_3 = SVC(probability=True)
     svm_ovr_proba_3.fit(x_train_ovr_1,y_train_ovr_3)
     svm_ovr_proba_4 = SVC(probability=True)
     svm_ovr_proba_4.fit(x_train_ovr_4,y_train_ovr_4)

     SVC(probability=True)
```

```
def actualclass_prob(model_1,model_2,model_3,model_4,x_test):
  prob_1 = model_1.predict_proba(x_test)[:,1].reshape(-1,1)
  prob_2 = model_2.predict_proba(x_test)[:,1].reshape(-1,1)
  prob_3 = model_3.predict_proba(x_test)[:,1].reshape(-1,1)
  prob_4 = model_4.predict_proba(x_test)[:,1].reshape(-1,1)

  y = np.hstack((prob_1,prob_2,prob_3,prob_4))

  return np.argmax(y,axis = 1)
```

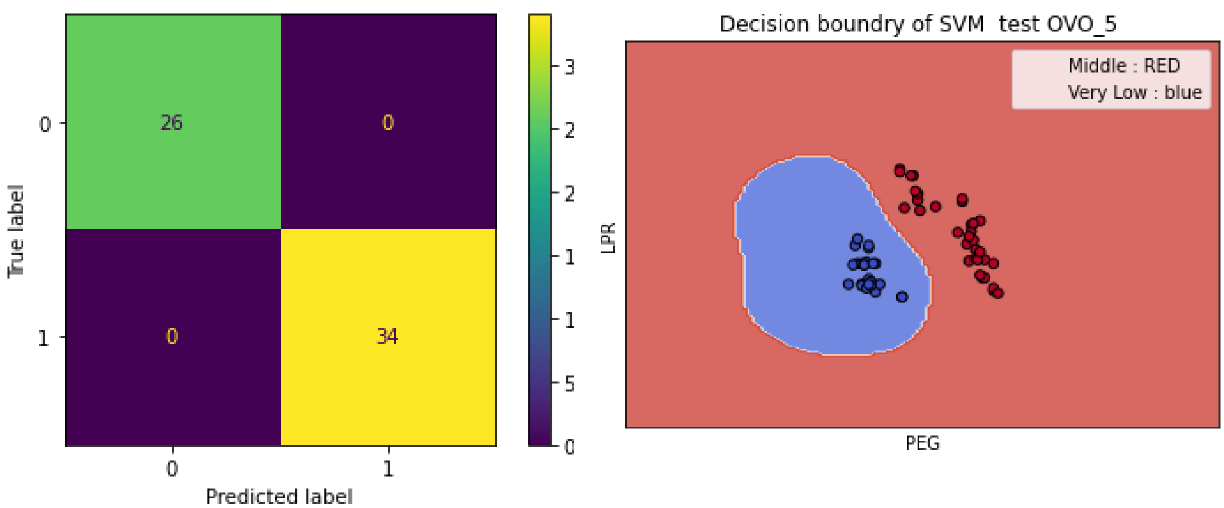


- **Apply OVO on SVM model:**

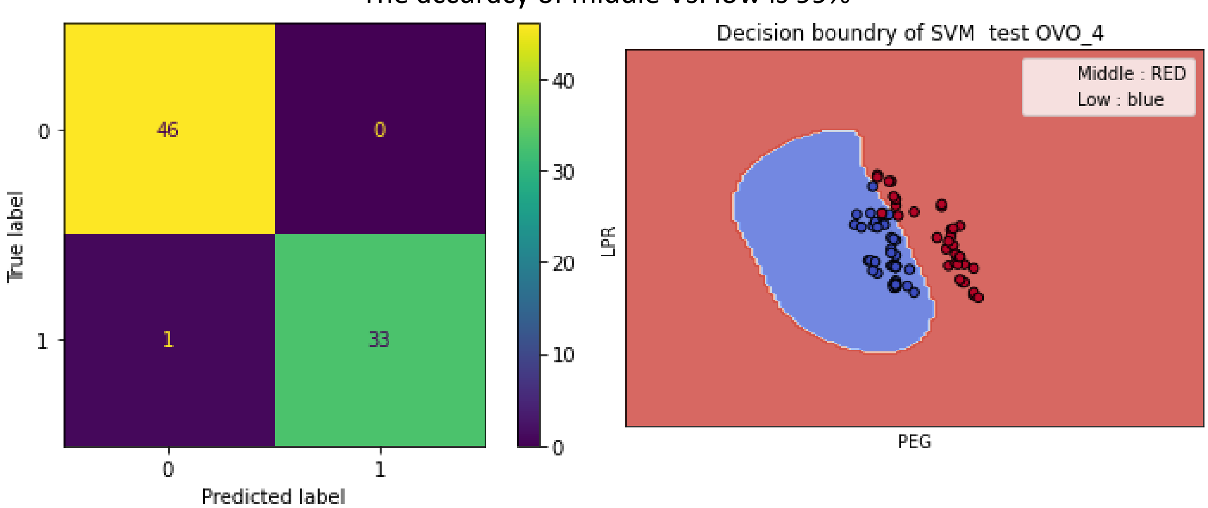
The accuracy of low Vs. very low is 97%



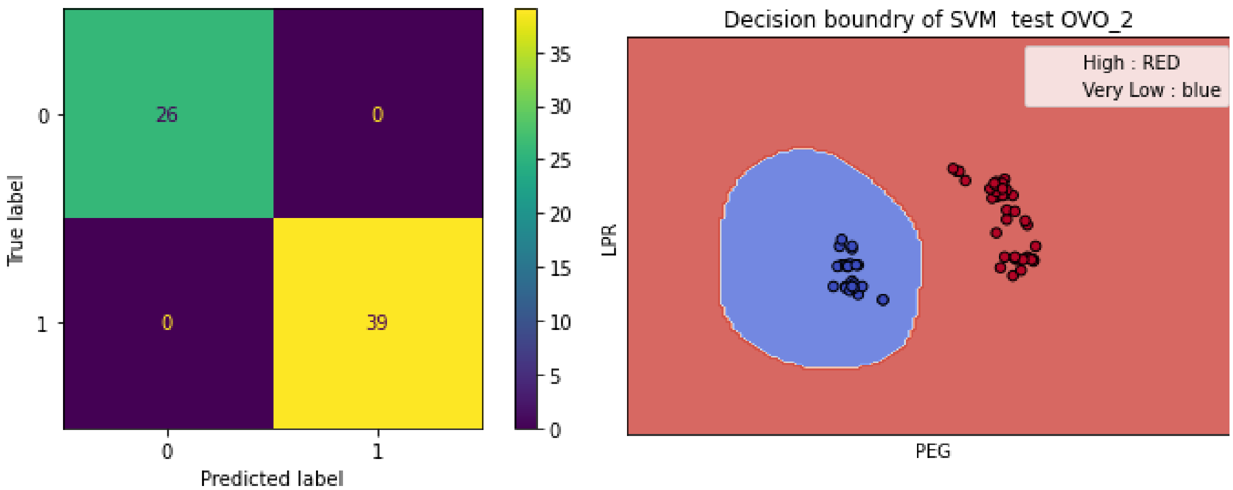
The accuracy of middle Vs. very low is 97%



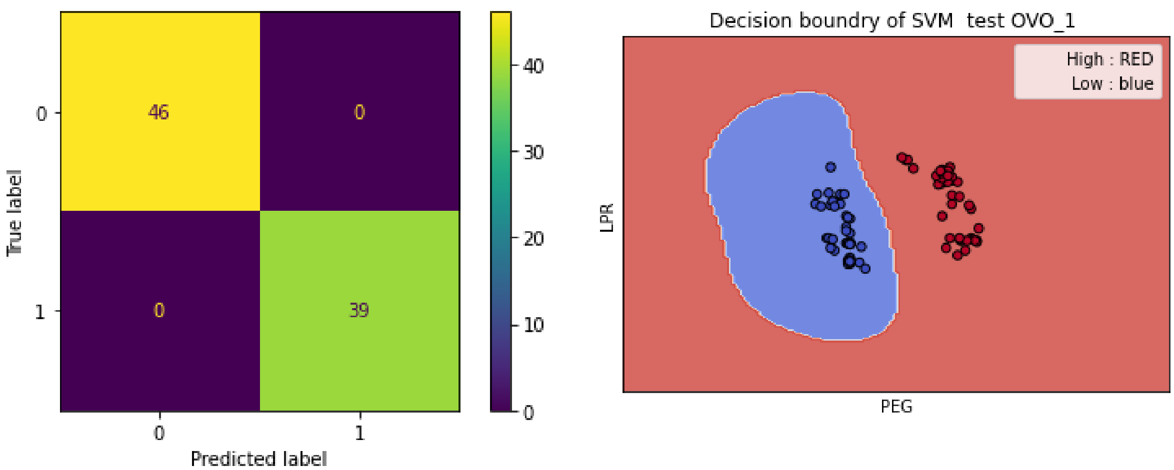
The accuracy of middle Vs. low is 99%



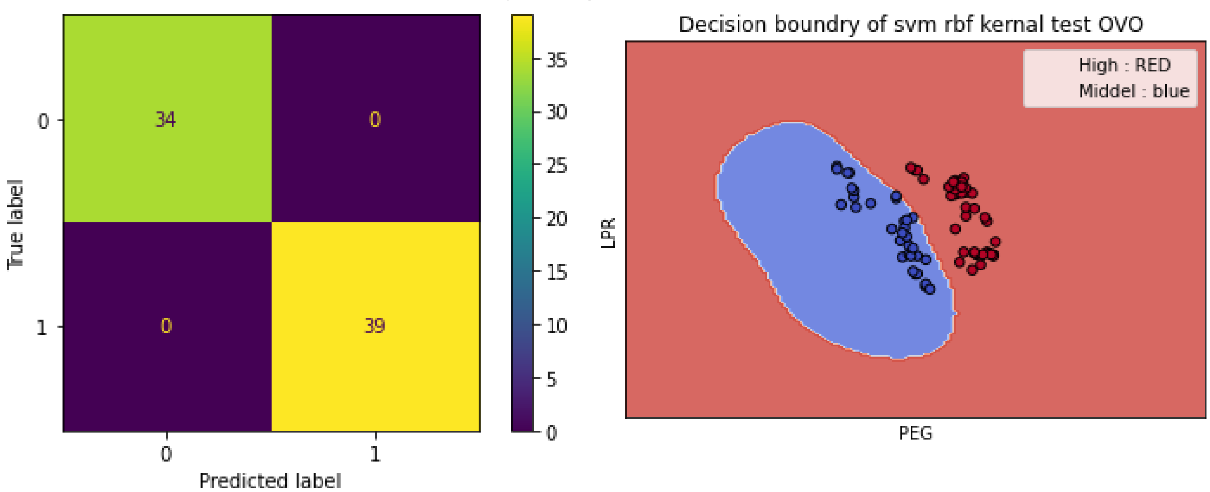
The accuracy of high Vs. very low is 100%



The accuracy of high Vs. low is 100%



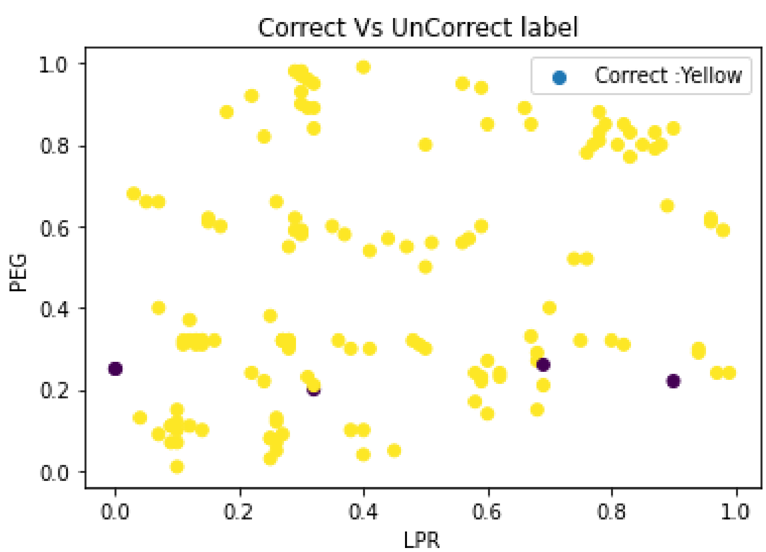
The accuracy of high Vs. middle is 100%



- **Apply argmax for OVO SVM model:**

```python
#function for predicting the probabilities of OVO classes
def actualclass_prob(model_1,model_2,model_3,model_4,model_5,model_6,x_test):
  prob_1 = model_1.predict_proba(x_test) #1-h  0- m
  prob_2 = model_2.predict_proba(x_test) #1-h  0 -l
  prob_3 = model_3.predict_proba(x_test) #1-h  0-vl
  prob_4 = model_4.predict_proba(x_test) #1-m  0- l
  prob_5 = model_5.predict_proba(x_test) #1-m  0-vl
  prob_6 = model_6.predict_proba(x_test) #1-l  0-vl

  high = np.hstack((prob_1[:,1].reshape(-1,1),prob_2[:,1].reshape(-1,1),prob_3[:,1].reshape(-1,1)))
  middle = np.hstack((prob_1[:,0].reshape(-1,1),prob_4[:,1].reshape(-1,1),prob_5[:,1].reshape(-1,1)))
  low = np.hstack((prob_2[:,0].reshape(-1,1),prob_4[:,0].reshape(-1,1),prob_6[:,1].reshape(-1,1)))
  very_low = np.hstack((prob_3[:,0].reshape(-1,1),prob_5[:,0].reshape(-1,1),prob_6[:,0].reshape(-1,1)))
  result = [4,3,2,1]
  sum_high = np.sum(high,axis = 1).reshape(-1,1)
  sum_middle = np.sum(middle,axis = 1).reshape(-1,1)
  sum_low = np.sum(low,axis = 1).reshape(-1,1)
  sum_very_low = np.sum(very_low,axis = 1).reshape(-1,1)
  pred = np.hstack((sum_high,sum_middle,sum_low,sum_very_low))

  final = np.argmax(pred,axis= 1)
  y = []
  for i in final:
    if i==1:
      y.append(3)
    elif i ==3:
      y.append(1)
    elif i==0:
      y.append(4)
    else:
      y.append(i)
  final = np.array(y)
  return final
```

- The accuracy is 0.96% and it is exceptionally good for the model.
- The model can predict the classes successfully.



- **Conclusion:**

During this assignment we made an exceptionally good practice about classification and applied different binary classifiers with each other to achieve multiple classifications and compare the results of them to make analysis and learn the causes of different performances Therefore, we found out that the performance of support vector machine is better than the perceptron as - Perceptron model try to find the hyperplane that separates two sets but does not try to optimize the separation "distance".

On the other hand, SVM tries to maximize the "support vector", the distance between the two closest opposite sample points. -The SVM typically tries to use a (kernel function) to project the sample points into high dimension space to make them linearly separable, while the perceptron assumes the sample points are already linearly separable.