ELG5255 Applied Machine Learning
Group :19
Assignment No:1

Team membares names:

1.Abdulrahman Ahmed

2.Amir Youssef

3.Mohamed Elesawy

Goal :

Our goal is to build models that can discriminate well between different data classes giving high performance and accuracy and comparing between more than one algorithm.

Dataset :

The data is clean and suitable to work directly with the model.
Dataset is already divided into the training and testing and that helps to make detection easier.

```
#  PREPARATION FUNCTION

def load_dataset(dada):
    with open(dada) as csv_file:
        data_reader = csv.reader(csv_file)
        feature_names = next(data_reader)[:-1]
        data = []
        target = []
        for row in data_reader:
            features = row[:-1]
            label = row[-1]
            data.append([float(num) for num in features])
            target.append((label))

        data = np.array(data)
        target = np.array(target)
    return Bunch(data=data, target=target, feature_names=feature_names)
```

Data Exploration: Exploration techniques (***EDA)*** were applied to the dataset discovering that the target label is slightly unbalanced.

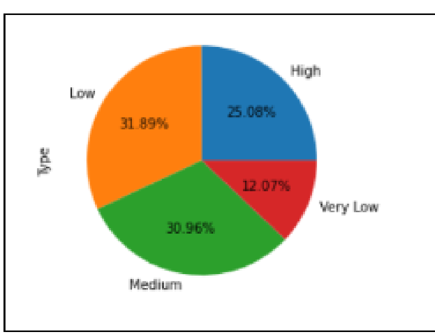


Figure a. Unbalanced train dataset

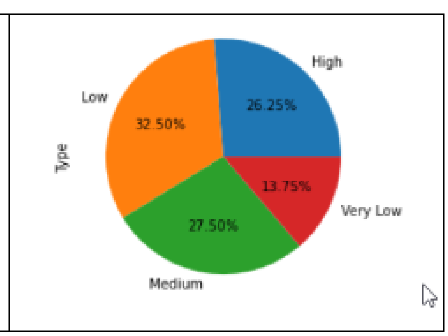


Figure b. Unbalanced test dataset

Data Pre-processing:

LabelEncoder was used to convert the categorical features into numerical features.

Feature engineering : Statistical F-anova test was applied to discover the two most important features in the dataset and "LPR" and "PEG" were found to be the most important ones

```
# EXTRACT THE MOST VALUABLE TWO FEATURES
# Score function anova tells the feature to be selected using anova test
test = SelectKBest(score_func=f_classif, k=2)
fit = test.fit(train_data.iloc[:,:-1], train_data.iloc[:,-1])
fit.scores_
```

```
array([  7.98709716,   5.92669075,   5.35222489,  15.02514254,
       633.67829829])
```

Figure:Feature Engineering.

Design SVM and Perceptron Comparison :

- **SVM MODEL**

  Code and output: SVM model was used with RBF kernel, having accuracy of 98.75% that means most of the values were predicted correctly by the model.

```
model = svm.SVC(kernel='rbf')
model.fit(X_tr, y_tr)
print('Accuracy of model: {:.2f}%'.format(getAccuracy(model, X_ts, y_ts)))
```

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

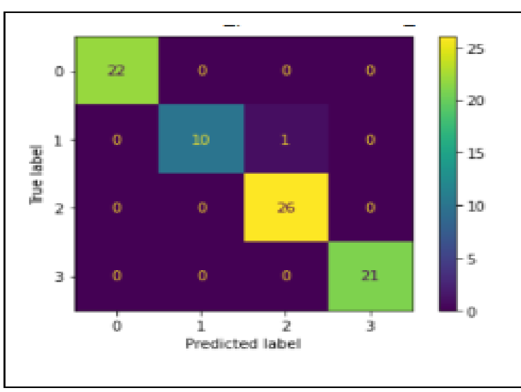


Figure a :confusion matrix of svm

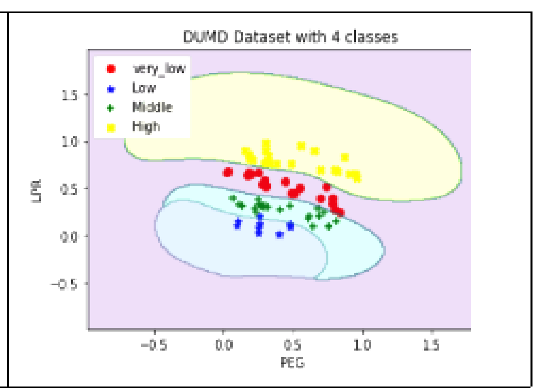


Figure b :Decision boundry of svm

- **Perceptron Model:**

  Code and output: Perceptron model was used to get a classification accuracy of `83.75`%.

```
from sklearn.linear_model import Perceptron
p = Perceptron(random_state=42)
p.fit(X_tr, y_tr)
print('Accuracy of model: {:.2f}%'.format(getAccuracy(p, X_ts, y_ts)))
```

```
Accuracy of model: 83.75%
```

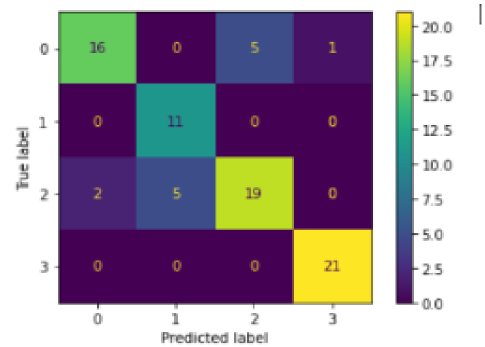


Figure a:confusion matrix of perceptron

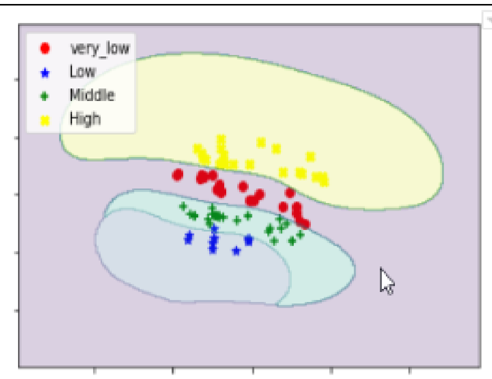


Figure b:Decision boundry of perceptron

Summary of comparison :

SVM performs much better than perceptron on multi-classification problems.

We have two types of svm

1. **SVM OVR :** One hot label encoder was used to encode each class and to know this label that represents this specific class, For ex:[1,0,0,0] that represent class High.

```
# EXCUTING THE ONE HOT LABELENCODER
# ['high' , 'middle' , 'low' , 'very low']
#>>> [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]

ytrr = y_tr.reshape((-1,1))
ytt = y_ts.reshape((-1,1))

mlb = MultiLabelBinarizer()

ytr2 = mlb.fit_transform(ytrr)
yt2 = mlb.fit_transform(ytt)
```

The data was separated into four classes for training and testing which helped us to put each class individually inside the model

```python
# ONE HOT ENCODER FUNCTION

def encode(arr):
    ylist = arr.tolist()
    yout = []
    i = 2
    for ele in ylist:

        if ele == 'Very Low':
            yout.append([0,0,0,1])
        elif ele == 'Low':
            yout.append([0,0,1,0])
        elif ele == 'Medium':
            yout.append([0,1,0,0])
        elif ele == 'High':
            yout.append([1,0,0,0])
        i=i+1
    youtf = np.array(yout)
    return youtf
```

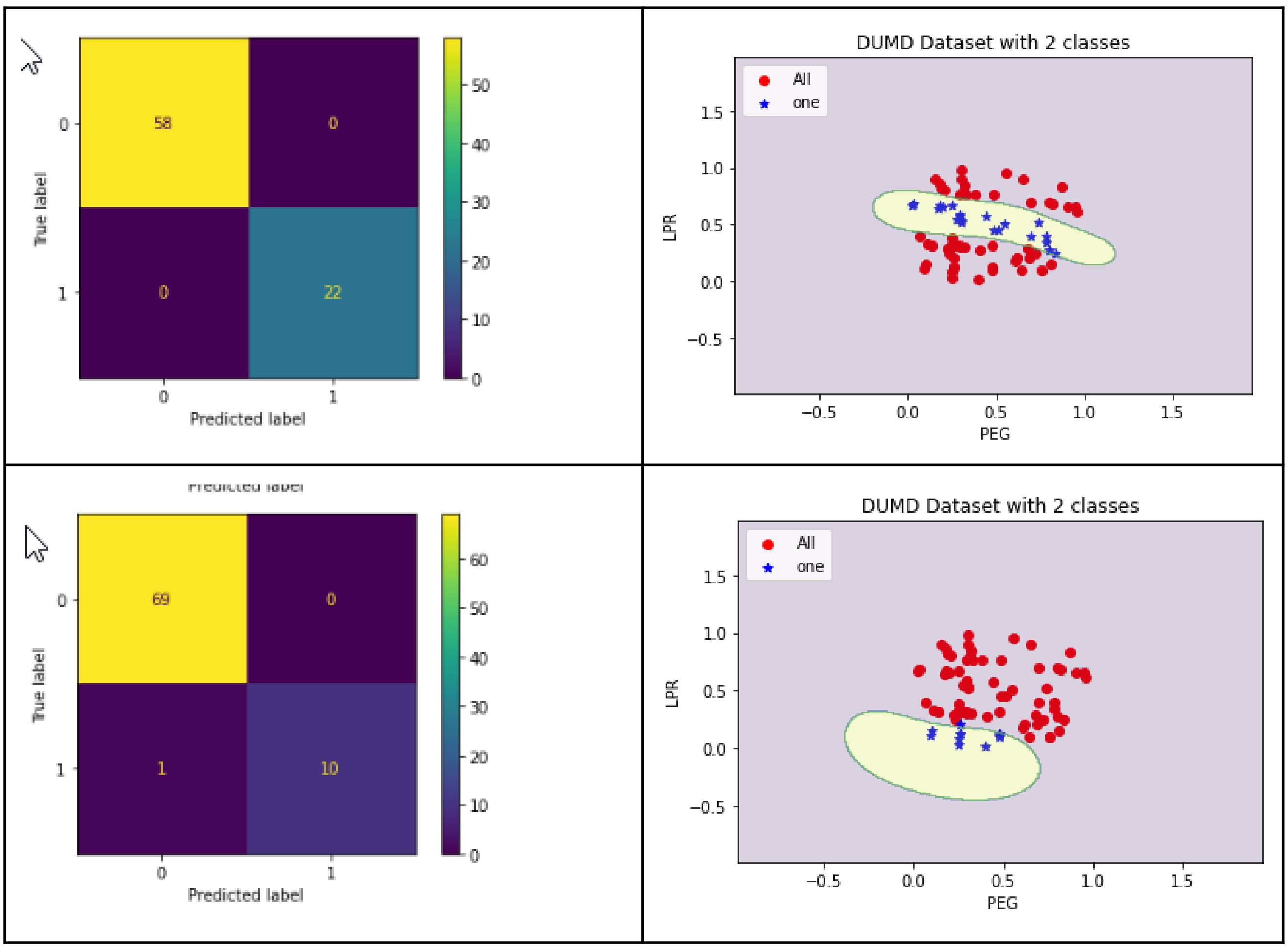
The classes were binarized so as to train each classifier on one class vs other classes where one class is encoded as one while the other were encoded as zeros

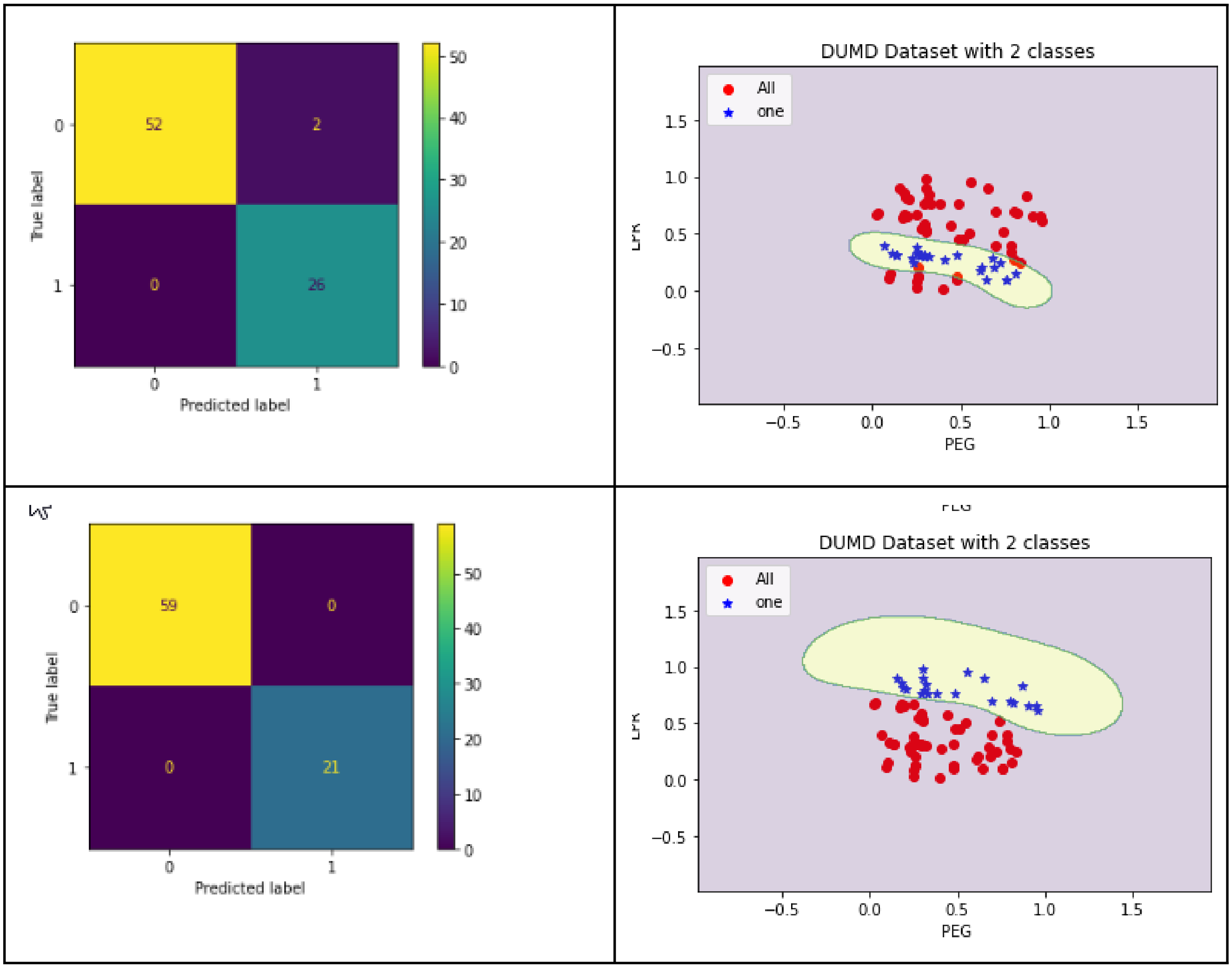
```python
# OVR CLASSIFIERS AND THEIR ACCURACY

# CLF_1
clf_1 = svm.SVC(kernel='rbf', probability=True)
clf_1.fit(X_tr, yb1)
yb1_pred = clf_1.predict_proba(X_ts)[:,1].reshape(-1,1)
```

```python
# MODELS ACCURACY

print('Accuracy of high classifier: {:.2f}%'.format(getAccuracy(clf_1, X_ts, ytb1)))
print('Accuracy of low classifier : {:.2f}%'.format(getAccuracy(clf_2, X_ts, ytb2)))
print('Accuracy of medium classifier : {:.2f}%'.format(getAccuracy(clf_3, X_ts, ytb3)))
print('Accuracy of very low classifier : {:.2f}%'.format(getAccuracy(clf_4, X_ts, ytb4)))
```

```
Accuracy of high classifier: 100.00%
Accuracy of low classifier : 98.75%
Accuracy of medium classifier : 97.50%
Accuracy of very low classifier : 100.00%
```

Our confusion matrix and decision boundry for each class invidually



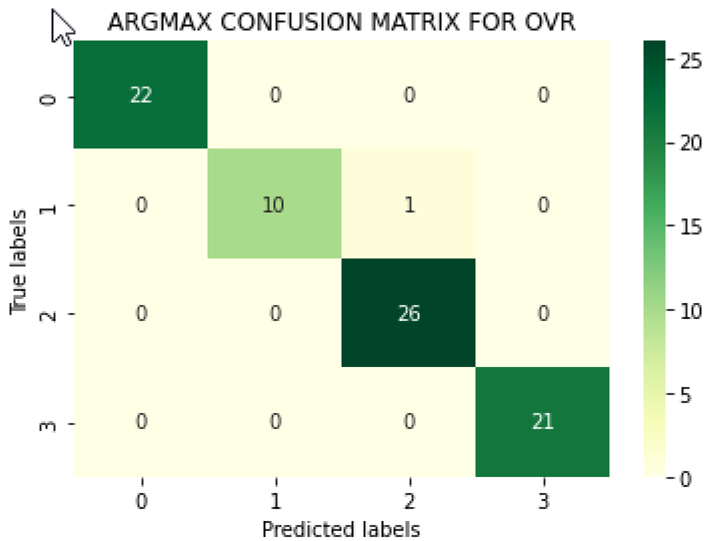


Our custom Aggregation function argmax: Our strategy is to focus on the value returned from each classifier and we aggregate it it using hstack then we get a total accuracy and confusion matrix

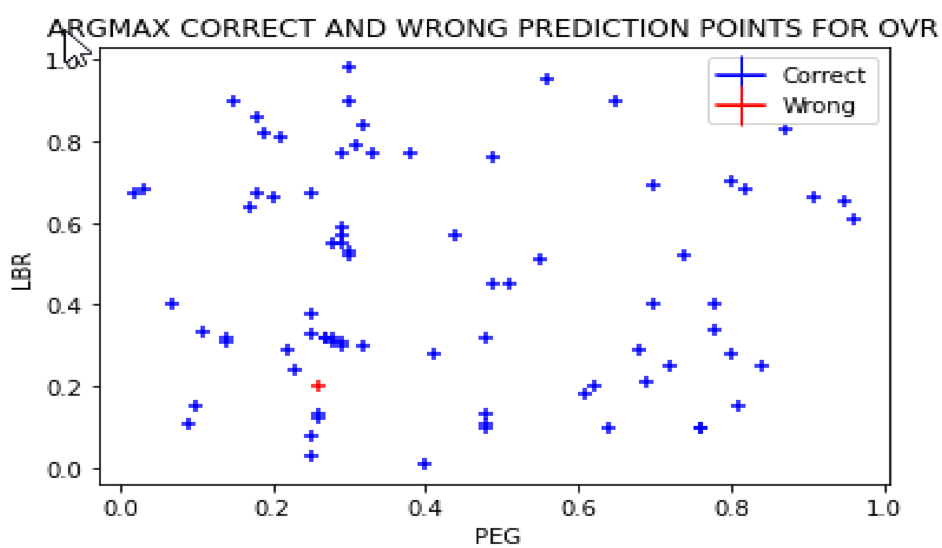
```
# ARGMAX

yb_all = np.hstack((yb1_pred, yb2_pred, yb3_pred, yb4_pred))

m = np.argmax(yb_all, axis=1)
```

```
# ARGMAX ACCURACY

print('Accuracy of Argmax: {:.2f}%'.format(accuracy_score(y_ts, m)*100))
```

```
Accuracy of Argmax: 98.75%
```



We implemented a function that help us to find the Correct and wrong prediction points plot as Shown :



Code of wrong function

```
# DETECTING THE WRONG CLASS PREDICTION DATA REGARDING POSITION IN TARGET DATA

def wrong(rong):
  mlist = rong.tolist()
  ytlist = y_ts.tolist()
  yout = []
  i = 0
  for ele in ytlist:

    if ele != mlist[i]:
      mlist[i]= 1
    else :
      mlist[i]= 0

    i=i+1
  mout = np.array(mlist)
  return mout
```

2-)SVM one VS one :

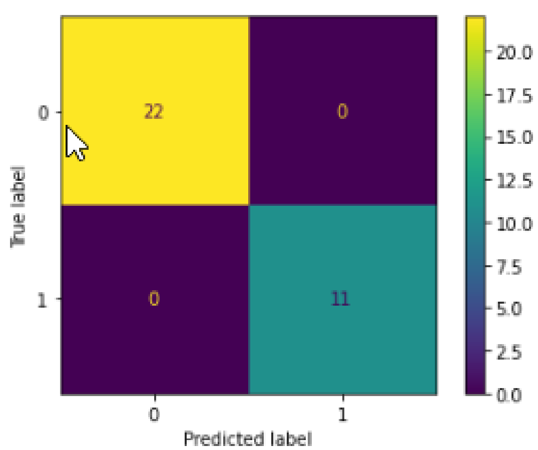
Data preprocessing in this part involves binarizing the labels of the dataset creating six different datasets, each dataset contains only two classes of the original dataset represented as 0 and 1When testing Each classifier on the portion of test data that contains the classes it was trained on, all the models preformed really well with accuracies above 90%

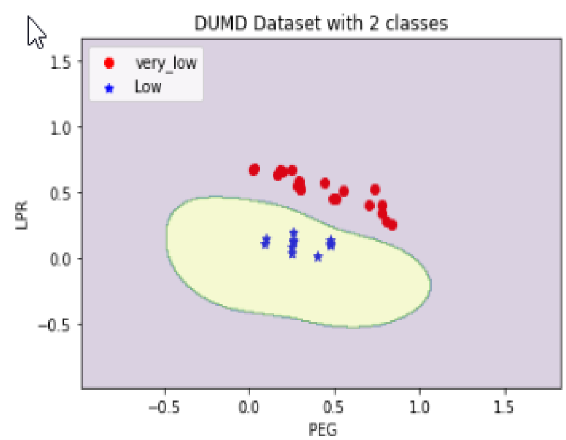
```
# OVO CLASSIFIERS AND THEIR ACCURACY

# CLF0_1
clf_0_1 = svm.SVC(kernel='rbf', probability=True)
clf_0_1.fit(X_tr0_1, y_tr0_1)

# CLF0_2
clf_0_2 = svm.SVC(kernel='rbf', probability=True)
clf_0_2.fit(X_tr0_2, y_tr0_2)

# CLF0_3
clf_0_3 = svm.SVC(kernel='rbf', probability=True)
clf_0_3.fit(X_tr0_3, y_tr0_3)
# CLF 1_2
clf_1_2 = svm.SVC(kernel='rbf', probability=True)
clf_1_2.fit(X_tr1_2, y_tr1_2)

# CLF1_3
clf_1_3 = svm.SVC(kernel='rbf', probability=True)
clf_1_3.fit(X_tr1_3, y_tr1_3)
# CLF2_3
clf_2_3 = svm.SVC(kernel='rbf', probability=True)
clf_2_3.fit(X_tr2_3, y_tr2_3)
yb1_pred = clf_2_3.predict_proba(X_ts2_3)
```

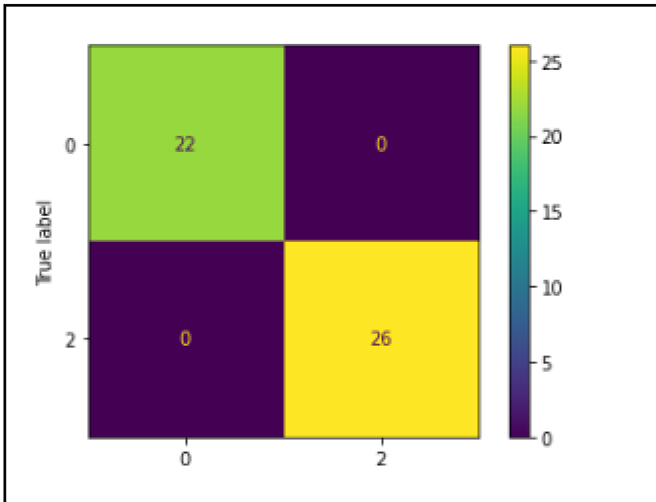
```
print('Accuracy of 0 and 1 classifier: {:.2f}%'.format(getAccuracy(clf_0_1, X_ts0_1, y_ts0_1)))
print('Accuracy of 0 and 1 classifier: {:.2f}%'.format(getAccuracy(clf_0_2, X_ts0_2, y_ts0_2)))
print('Accuracy of 0 and 1 classifier: {:.2f}%'.format(getAccuracy(clf_0_3, X_ts0_3, y_ts0_3)))
print('Accuracy of 1 and 2 classifier: {:.2f}%'.format(getAccuracy(clf_1_2, X_ts1_2, y_ts1_2)))
print('Accuracy of 1 and 3 classifier: {:.2f}%'.format(getAccuracy(clf_1_3, X_ts1_3, y_ts1_3)))
print('Accuracy of 2 and 3 classifier: {:.2f}%'.format(getAccuracy(clf_2_3, X_ts2_3, y_ts2_3)))
```

```
Accuracy of high classifier: 100.00%
Accuracy of high classifier: 100.00%
Accuracy of high classifier: 100.00%
Accuracy of high classifier: 97.30%
Accuracy of high classifier: 100.00%
Accuracy of high classifier: 100.00%
```

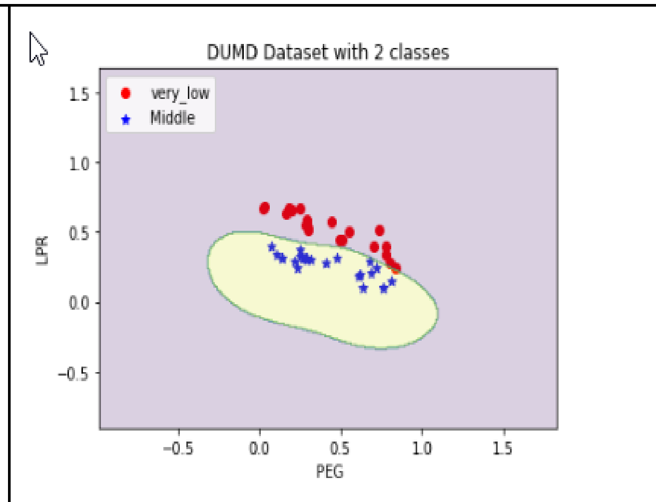
Plotting the decision boundary of each classifier showed that all the classifiers can discriminate well between the classes they were trained on and that the data itself is linearly separable.



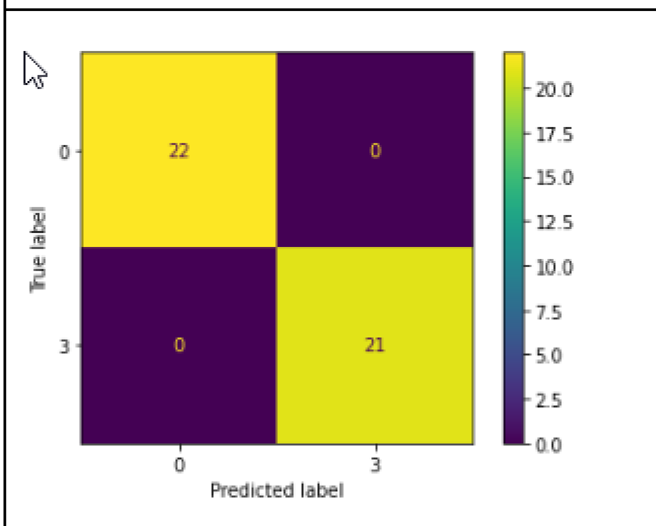




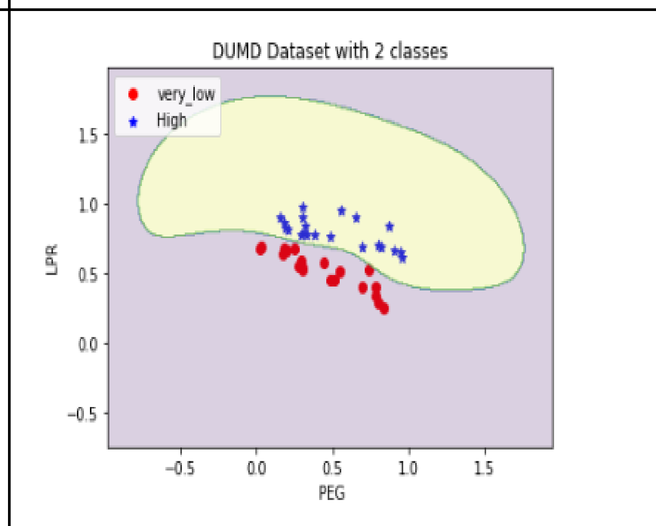
True label
0
2
22
0
0
26
0
2
25
20
15
10
5
0



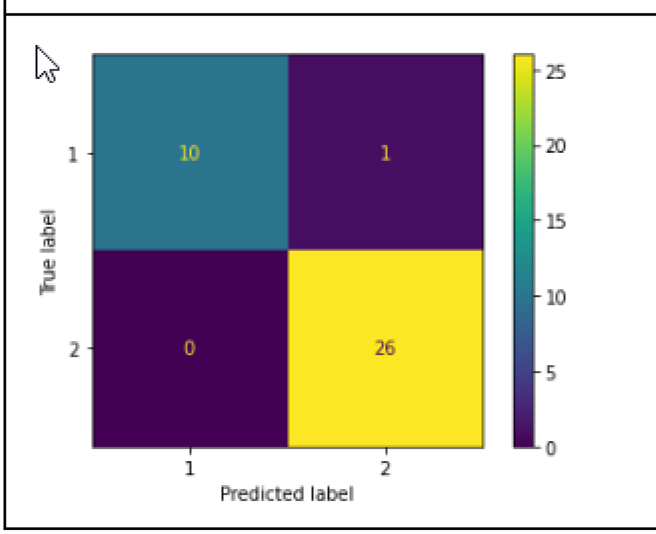
DUMD Dataset with 2 classes
very_low
Middle
LPR
PEG



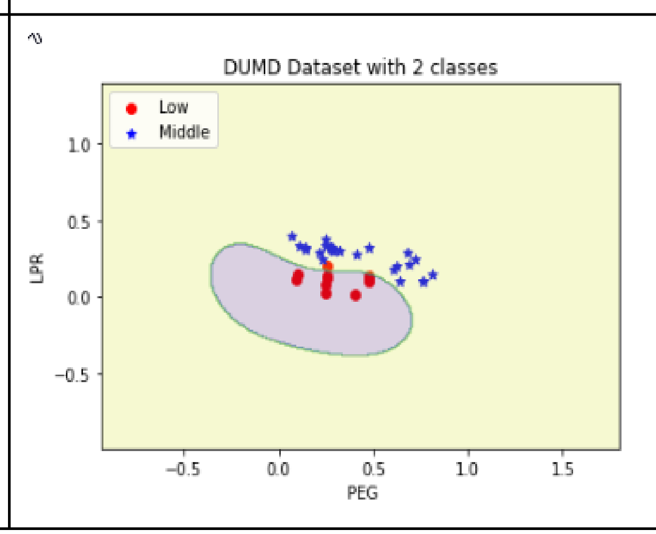
True label
0
3
22
0
0
21
Predicted label
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0



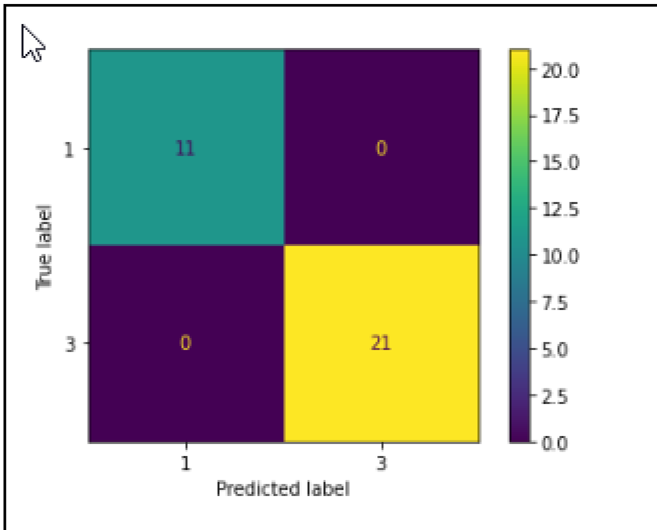
DUMD Dataset with 2 classes
very_low
High
LPR
PEG

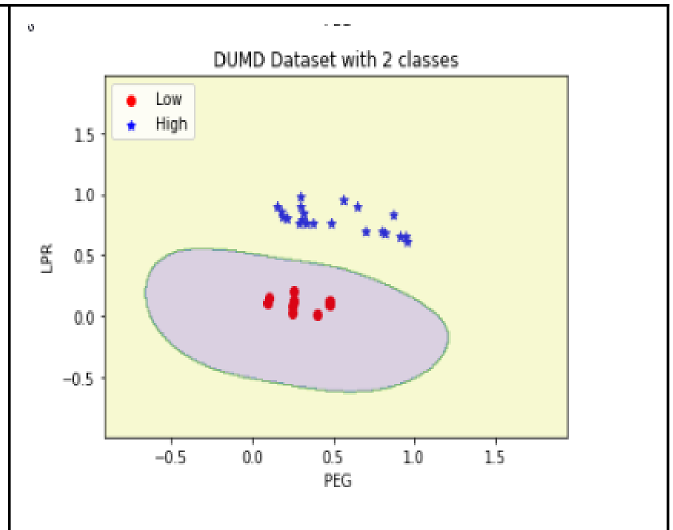


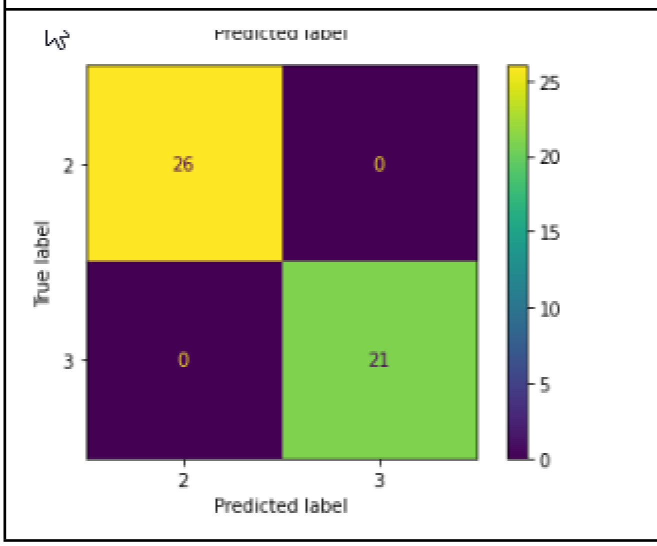
True label
1
2
10
1
0
26
Predicted label
25
20
15
10
5
0

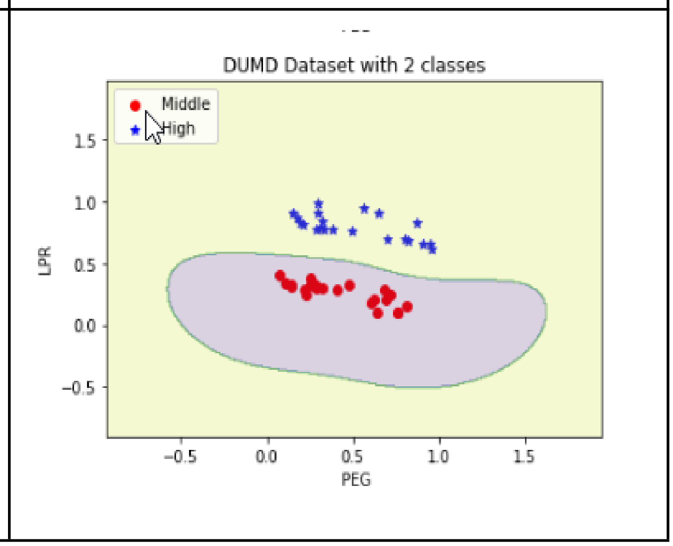


DUMD Dataset with 2 classes
Low
Middle
LPR
PEG









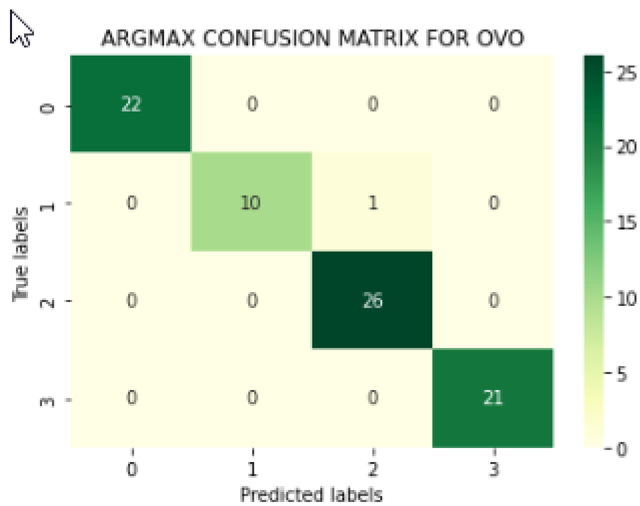
We aggregate the probabilities from each classifiertaging the classing in the consideration

Our code and confusion matrix

```python
# ARGMAX

yb_all = np.hstack(([totalP0, totalP1, totalP2, totalP3]))
len(yb_all)
m = np.argmax(yb_all, axis=1)
```

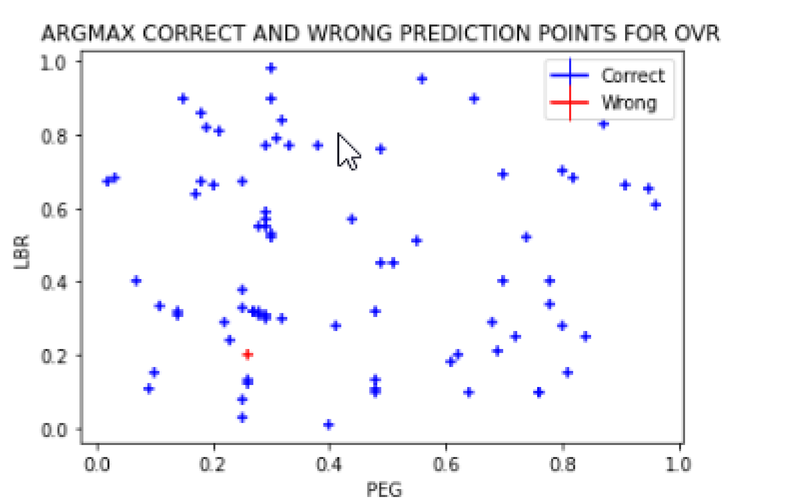
```python
# ARGMAX ACCURACY

print('Accuracy of Argmax: {:.2f}%'.format(accuracy_score(y_ts, m)*100))
```

```
Accuracy of Argmax: 98.75%
```



We implemented a function that help us to find the Correct and wrong prediction points plot as Shown :

```
# PLOTING CORRECT AND WRONG PREDICTION POINTS

fig, ax = plt.subplots()
plt.scatter(X_ts[:,0],X_ts[:,1],
            marker="+",c=mout, cmap='bwr')
blue_line = mlines.Line2D([], [], color='blue', marker='+',
                          markersize=20, label='Correct')
red_line = mlines.Line2D([], [], color='red', marker='+',
                          markersize=20, label='Wrong')
plt.title('ARGMAX CORRECT AND WRONG PREDICTION POINTS FOR OVR')
plt.xlabel(' PEG')
plt.ylabel('LBR')
ax.legend(handles=[blue_line ,red_line])
plt.show()
```



Conclusion:

• Models (Perceptron and SVM) :Some model can give good prediction in data and other model give less prediction and in our data the SVM give us good prediction ~ (100 %) than perceptron ~(80%)

• OvR strategy : We learn that how to make binary classifier on data to build a model to make classification to a class versus all classes and get different prediction .

• OvOstrategy :We learn that how to make binary classifier on data to build a model to make classification to a class versus class and we make this permutation with other class sepereable and get different prediction.

• Argmax : Help us to improve the high performance and improve the accuracy

• Our Aggregation Strategy : Our Strategy performs well with SVM , It needs more updates to be applied to Perceptron Model to achieve better accuracy.