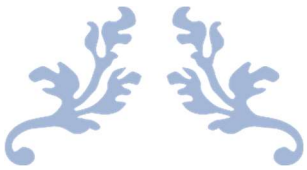
uOttawa



ELG 5255:
Applied Machine Learning
Assignment 1

Dr. Murat Simsek

GROUP 14

Table of Contents

Table of Figures

Part 1:

A. Label Encoder

Encoding class labels and visualizing data frames after encoding.

```
le=LabelEncoder()
le.fit(df_train.iloc[:,-1])
print(le.classes_)
df_train.iloc[:,-1]=le.transform(df_train.iloc[:,-1])
df_test.iloc[:,-1]=le.transform(df_test.iloc[:,-1])
df_train.head()
```

```
df_test.head()
```

	STG	SCG	STR	LPR	PEG	UNS
0	0.00	0.00	0.00	0.00	0.00	3
1	0.08	0.08	0.10	0.24	0.90	0
2	0.10	0.10	0.15	0.65	0.30	2
3	0.08	0.08	0.08	0.98	0.24	1
4	0.09	0.15	0.40	0.10	0.66	2

Train data

	STG	SCG	STR	LPR	PEG	UNS
0	0.420	0.290	0.140	0.03	0.68	2
1	0.100	0.100	0.520	0.78	0.34	2
2	0.510	0.255	0.550	0.17	0.64	2
3	0.250	0.540	0.310	0.25	0.08	3
4	0.258	0.250	0.295	0.33	0.77	0

Test data

B. Choosing Best Features

Choosing the best two features based on 3 different techniques which are heat map, pair plot and ANOVA.

```
corrmat = df_train.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(10,10))
#plot heat map
g=sns.heatmap(corrmat,annot=True,cmap="Blues")
```

Showing correlation between all features and the target and with themselves:

It's clear that "LPR" and "PEG" features are the two most correlated features to the target feature "UNS".

Figure 1 Heat Map

Using pair plot to show data between features with respect to target feature "UNS":

```
sns.pairplot(df_train, hue = 'UNS', palette="bright");
```

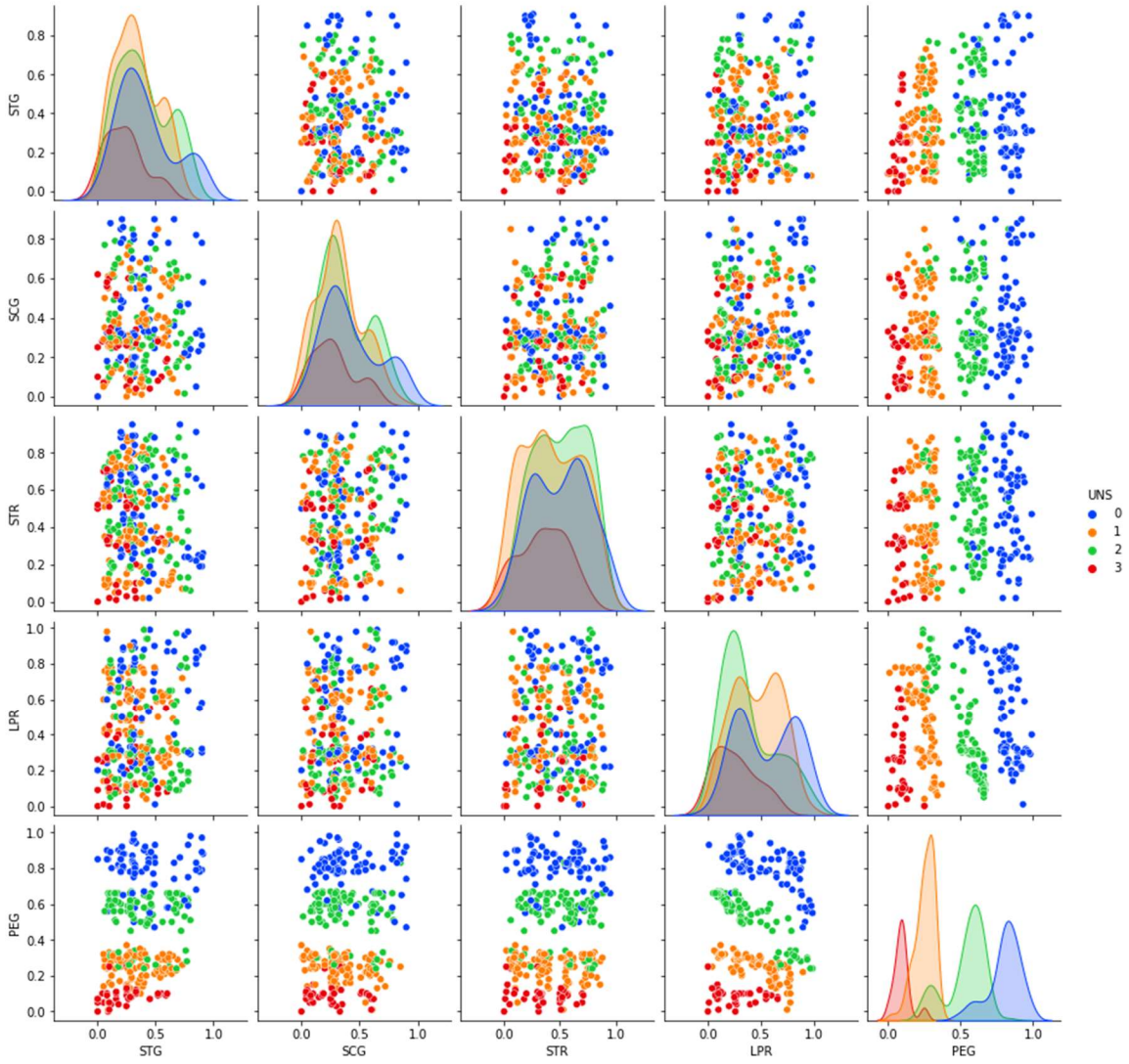


Figure 2 Pair Plot

As per shown in the previous plot the PEG is a clear feature to work with as the UNS feature's attributes are obviously almost separated.

Using ANOVA to double check and pick best two features:

```
# Select Features With Best ANOVA F-Values
# Create an SelectKBest object to select features with two best ANOVA F-Values
fvalue_selector = SelectKBest(f_classif, k=2)

# Apply the SelectKBest object to the features and target
x=df_train.iloc[:,:-1]
y=df_train.iloc[:,-1]
X_kbest = fvalue_selector.fit_transform(x, y)

# View results
print('Original number of features:', x.shape[1])
print('Reduced number of features:', X_kbest.shape[1])
print('The selected features are: ',
fvalue_selector.get_feature_names_out())

# Training data features after selecting the two features
X_train = np.array(df_train[['LPR','PEG']])
X_test = np.array(df_test[['LPR','PEG']])
Y_train = np.array(y)
Y_test = np.array(df_test.iloc[:,-1])
```

The output:

```
Original number of features: 5
Reduced number of features: 2
The selected features are:  ['LPR' 'PEG']
```

ANOVA system has selected “LPR” and “PEG” as the best two features.

Plotting the data between the two selected features:

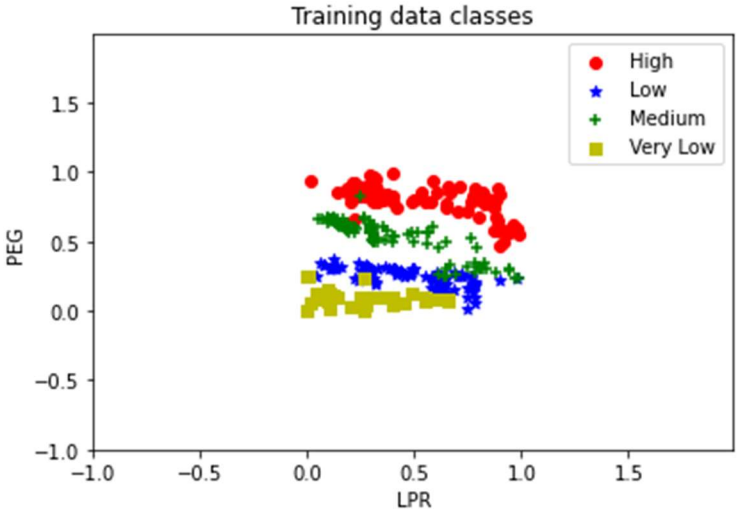


Figure 3 The training data

C. SVM and Perceptron classifications

Classifying testing data by using SVM classifier:

```
SVM_Model= SVC(kernel='linear', C=200)
SVM_Model.fit(X_train,Y_train)

print('>>>Accuracy of training data: \n')
print(SVM_Model.score(X_train,Y_train))
print('>>>Accuracy of testing data: \n')
print(SVM_Model.score(X_test,Y_test))
ConfusionMatrixDisplay.from_estimator(SVM_Model, X_test, Y_test)
print('\n >>>Confusion Matrix: \n')
print(plt.show())
print('\n >>>Decision Boundaries: \n')
cls=[0,1,2,3]
cls_names =le.inverse_transform(cls)
plotData(X_test, Y_test, cls, cls_names, "SVM Classifier")
plotRegions(SVM_Model, X_test)
```

SVM model accuracy on training data: 95.98%

SVM model accuracy on testing data: 98.75%

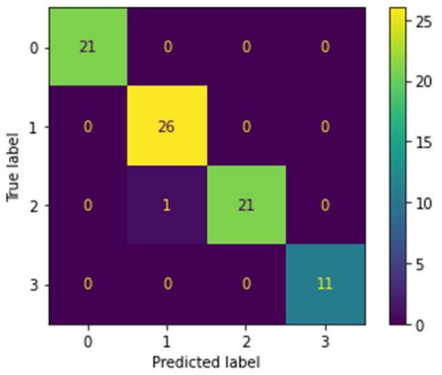


Figure 4 Confusion matrix of testing data using SVM model

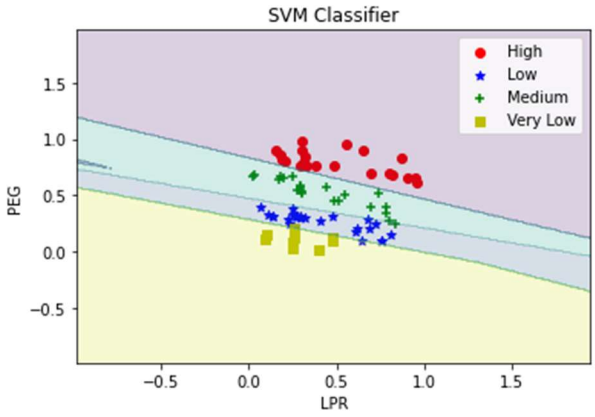


Figure 5 Decision boundary of the testing data using SVM model

Perceptron Classifier:

```
percep_model = Perceptron(eta0=0.1)
percep_model.fit(X_train,Y_train)

print('>>>Accuracy of training data: \n')
print(percep_model.score(X_train,Y_train))
print('>>>Accuracy of testing data: \n')
print(percep_model.score(X_test,Y_test))
ConfusionMatrixDisplay.from_estimator(percep_model, X_test, Y_test)
print('\n >>>Confusion Matrix: \n')
print(plt.show())
print('\n >>>Decision Boundaries: \n')
cls=[0,1,2,3]
cls_names =le.inverse_transform(cls)
plotData(X_test, Y_test, cls, cls_names, "Perceptron Classifier")
plotRegions(percep_model, X_test)
```

perceptron model accuracy on training data: 78.02%

perceptron model accuracy on testing data: 75%

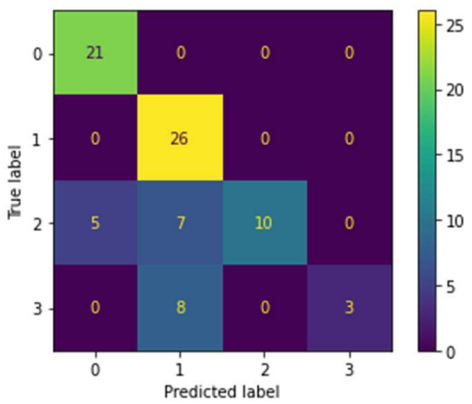


Figure 6 Confusion matrix of testing data using perceptron model

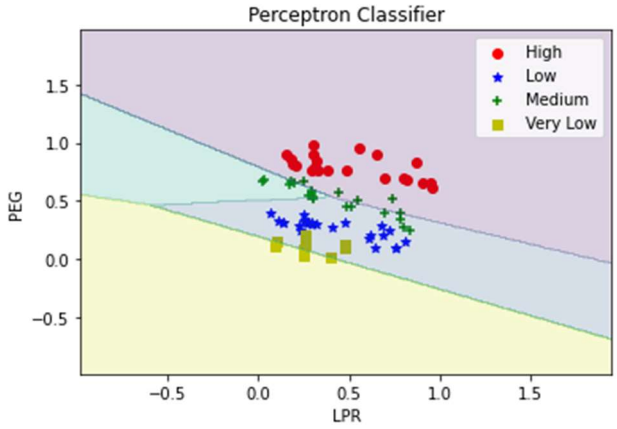


Figure 7 Decision boundary of the perceptron model

Part 2:

A. One versus Rest

Obtaining the binarized labels of the data for OvR:

```
b_train=binarized(X_train,Y_train,le)
b_test=binarized(X_test,Y_test,le)
data_train=b_train.OVR_binarized()
data_test=b_test.OVR_binarized()
for key, value in data_train.items():
  print(key)
  print(value)
for key, value in data_test.items():
  visualAndTest(classifers[count],value[0],value[1],key[:-2])
```

High vs Rest classifier:

Accuracy of model classify between High and rest: 100%

The model accuracy is 100% and this is what we can see from the confusion matrix and the decision boundary plot as the model managed to classify all points from class "high" correctly.

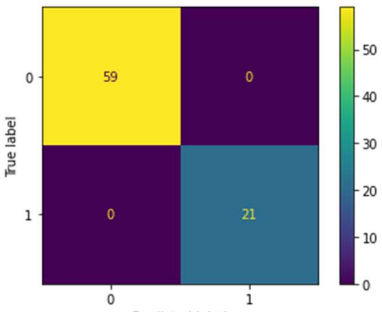


Figure 8 Confusion matrix of testing data using High vs rest classifier

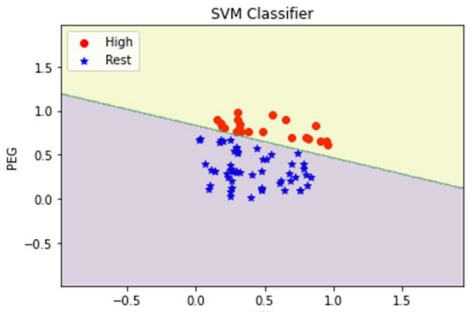


Figure 9 Decision boundary of the High vs rest classifier

Low vs Rest classifier:

Accuracy of model classify between Low and rest: 41%

The model accuracy is very low and this that we can conclude from the confusion matrix and the decision boundary plot as the model failed to classify any point from class "low" correctly

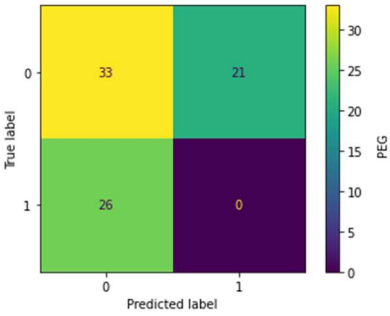


Figure 10 Confusion matrix of testing data using low vs rest classifier

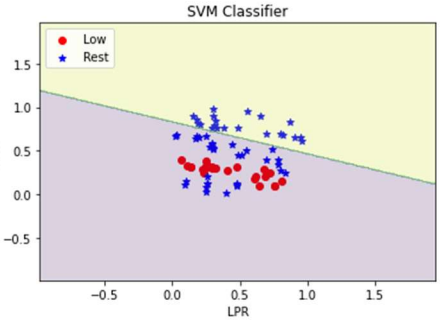


Figure 11 Decision boundary of the low vs rest classifier

Medium vs Rest classifier:

Accuracy of model classify between Medium and rest: 46%

The model accuracy is very low and this that we can conclude from the confusion matrix and the decision boundary plot as the model failed to classify any point from class "medium" correctly

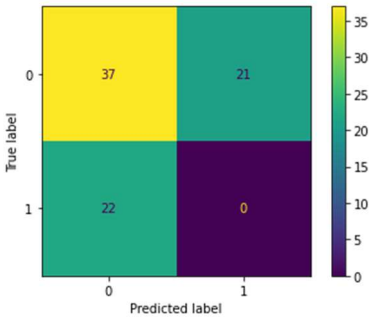


Figure 12 Confusion matrix of testing data using medium vs rest classifier

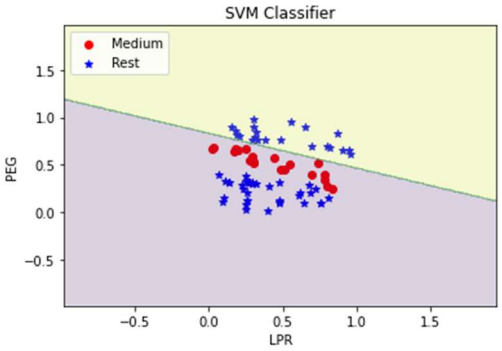


Figure 13 Decision boundary of the medium vs rest classifier

Very low vs Rest classifier:

Accuracy of model classify between Very Low and rest: 60%

The model accuracy is very low and this that we can conclude from the confusion matrix and the decision boundary plot as the model failed to classify any point from class "very low" correctly

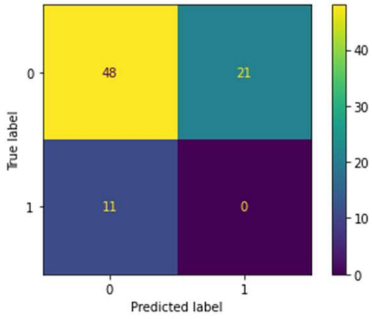


Figure 14 Confusion matrix of testing data using very low vs rest classifier

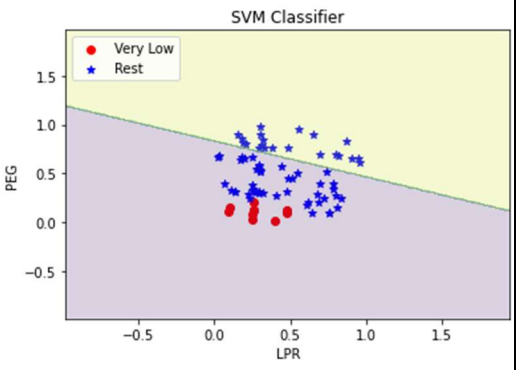


Figure 15 Decision boundary of the very low vs rest classifier

B. Argmax of OvR

Using argmax to aggregate confidence scores:

```
pred_ls=[]
for classifer in classifers:
    pred_ls.append(classifer.predict_proba(X_test)[:,1].reshape(-1,1))
pred_ls=np.array(pred_ls)
pred_ls= np.hstack(pred_ls)
print(pred_ls)

y_pred=np.argmax(pred_ls, axis=1)
print(y_pred)
print(Y_test)

print(classification_report(Y_test, y_pred, target_names=cls_names))
print('>>>Accuracy of testing data: ',accuracy_score(Y_test, y_pred),'\n')
ConfusionMatrixDisplay.from_predictions(Y_test,y_pred)

label=np.array(y_pred==Y_test)*1
plotData(X_test,label,[0,1],['False','True'],'prediction ')
x_min, x_max = X_test[:, 0].min() - 1, X_test[:, 0].max() + 1
y_min, y_max = X_test[:, 1].min() - 1, X_test[:, 1].max() + 1
plt.xlim(x_min,x_max)
plt.ylim(y_min,y_max)
```

Accuracy of testing data: 93.75%

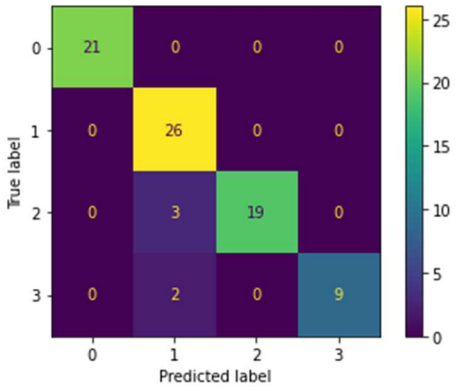


Figure 16 Confusion matrix of testing data using OVR model

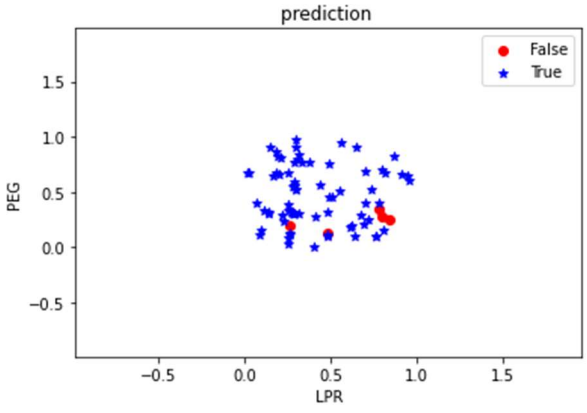


Figure 17 Plotting correct and wrong prediction points

Part 3:

A. One versus One

Obtaining the binarized labels of the data for OvO:

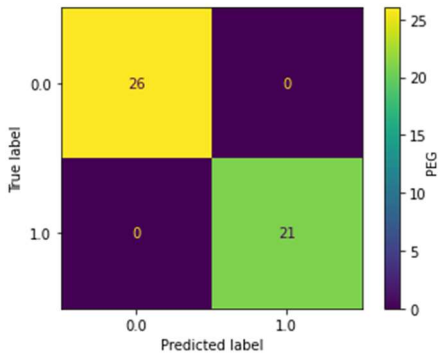
```
data_train=b_train.OVO_binarized()
data_test=b_test.OVO_binarized()
for key, value in data_train.items():
  print(key, value)

for key, value in data_train.items():
    keys=key.split('V',1)
    visualAndTest(classifers2[key],value[0],value[1],keys[0],keys[1])
```

High vs low classifier:

Accuracy of model classify between High and Low: 100%

The model accuracy is 100% and this is what we can conclude from the confusion matrix and the decision boundary plot as the model managed to distinguish between class "high" and class "low" accurately.



ɟure 18 Confusion matrix of testing data using High vs Low classifier

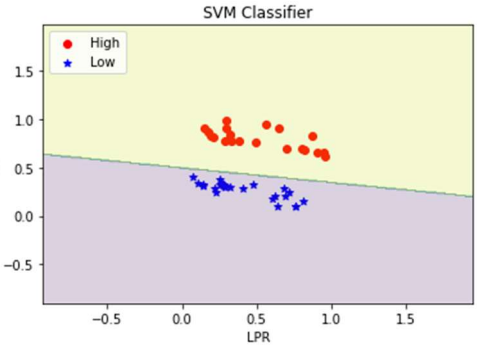


Figure 19 Decision boundary of the High vs Low classifier

High vs medium classifier.

Accuracy of model classify between High and Medium: 100%

The model accuracy is 100% and this is what we can conclude from the confusion matrix and the decision boundary plot as the model managed to distinguish between class "high" and class "medium" accurately.

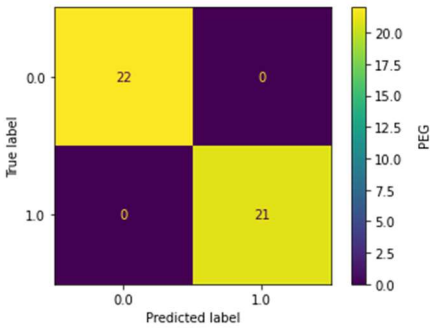


Figure 20 Confusion matrix of testing data using High vs medium classifier

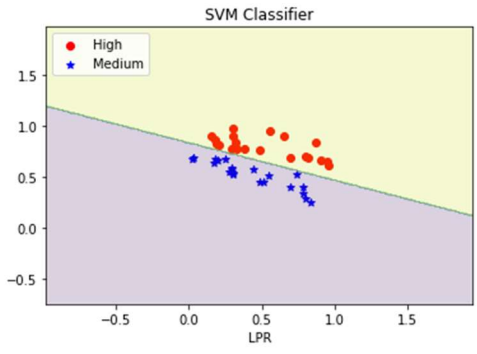


Figure 21 Decision boundary of the High vs medium classifier

High vs very low classifier:

Accuracy of model classify between High and Very Low: 100%

The model accuracy is 100% and this is what we can conclude from the confusion matrix and the decision boundary plot as the model managed to distinguish between class "high" and class "very low" accurately.

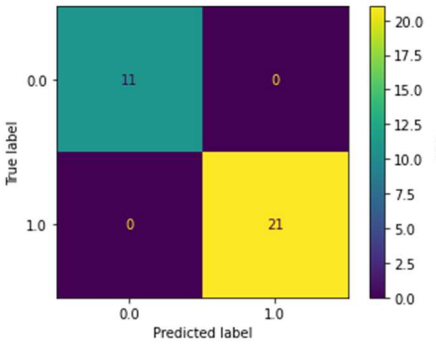


Figure 23 Confusion matrix of testing data using High vs very low classifier

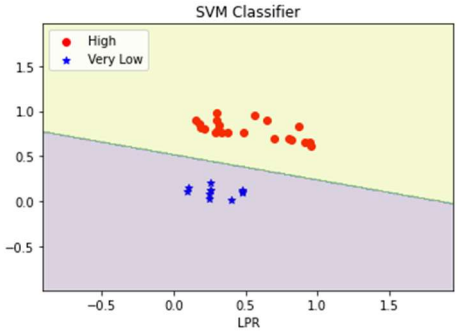


Figure 22 Decision boundary of the High vs very low classifier

Low vs medium classifier:

Accuracy of model classify between Low and Medium: 98%

The model accuracy is 98% and this is what we can conclude from the confusion matrix and the decision boundary plot as the model managed to distinguish between class "low" and class "medium" accurately except one point that the model misclassified.

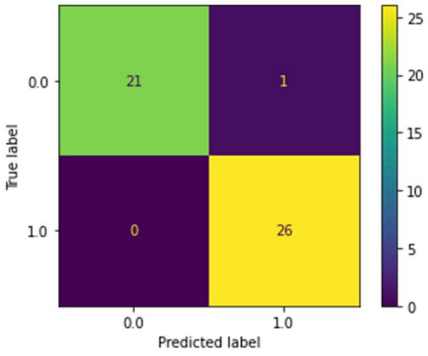


Figure 24 Confusion matrix of testing data using Low vs medium classifier

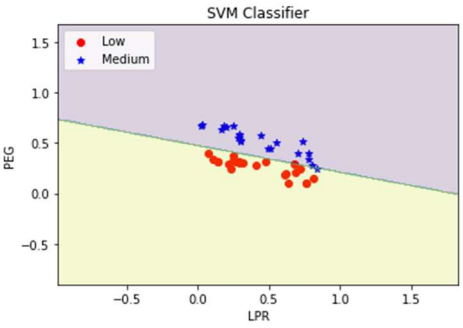


Figure 25 Decision boundary of the Low vs medium classifier

Low vs very low classifier:

Accuracy of model classify between Low and Very Low: 100%

The model accuracy is 100% and this is what we can conclude from the confusion matrix and the decision boundary plot as the model managed to distinguish between class "low" and class "very low" accurately.

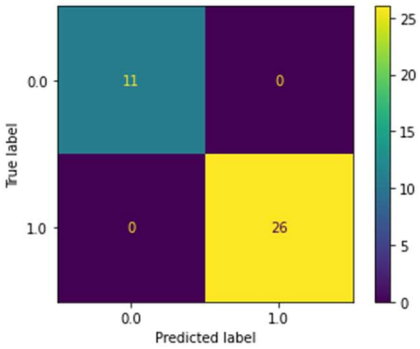


Figure 26 Confusion matrix of testing data using Low vs very low classifier

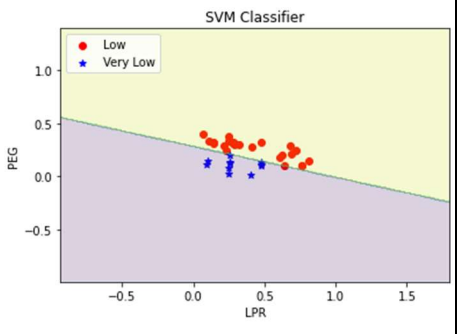


Figure 27 Decision boundary of the Low vs very low classifier

Medium vs very low classifier:

Accuracy of model classify between Medium and Very Low: 100%

The model accuracy is 100% and this is what we can conclude from the confusion matrix and the decision boundary plot as the model managed to distinguish between class "medium" and class "very low" accurately.

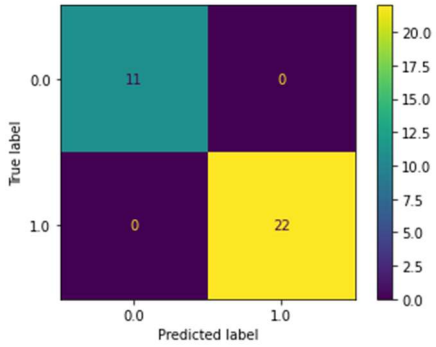


Figure 28 Confusion matrix of testing data using medium vs very low classifier

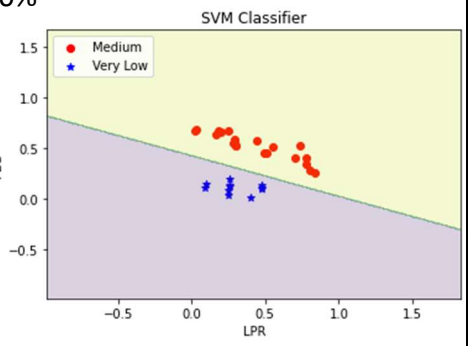


Figure 29 Decision boundary of the medium vs very low classifier

B. Argmax of OvO

Using argmax to aggregate confidence scores:

```
pred_ls2={}
for key,classifer in classifers2.items():
    keys=key.split('V',1)
    if keys[0] in pred_ls2.keys():
      pred_ls2[keys[0]]+=(classifer.predict_proba(X_test)[:,1].reshape(-1,1))
    else:
      pred_ls2[keys[0]]=(classifer.predict_proba(X_test)[:,1].reshape(-1,1))
    if keys[1] in pred_ls2.keys():
      pred_ls2[keys[1]]+=(classifer.predict_proba(X_test)[:,0].reshape(-1,1))
    else:
      pred_ls2[keys[1]]=(classifer.predict_proba(X_test)[:,0].reshape(-1,1))

pred=[]
for key, value in pred_ls2.items():
    pred.append(value)
pred=np.array(pred)
pred= np.hstack(pred)
print(pred)

y_pred2=np.argmax(pred, axis=1)
print(y_pred2)

print(classification_report(Y_test, y_pred2, target_names=cls_names))
print(confusion_matrix(Y_test, y_pred2))
ConfusionMatrixDisplay.from_predictions(Y_test,y_pred2)
print(accuracy_score(Y_test, y_pred2))

label=np.array(y_pred2==Y_test)*1
plotData(X_test,label,[0,1],['False','True'],'prediction ')
x_min, x_max = X_test[:, 0].min() - 1, X_test[:, 0].max() + 1
y_min, y_max = X_test[:, 1].min() - 1, X_test[:, 1].max() + 1
plt.xlim(x_min,x_max)
plt.ylim(y_min,y_max)
```

Accuracy of testing data: 96.25%

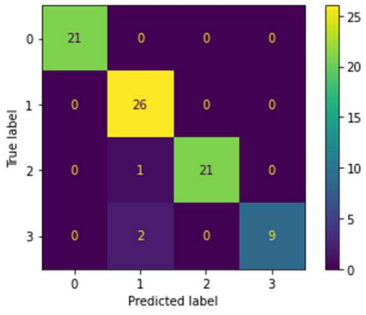


Figure 31 Confusion matrix of testing data using OvO model

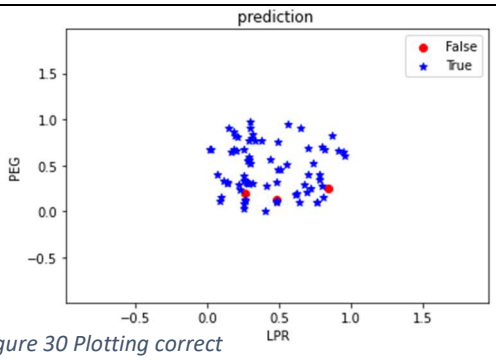


Figure 30 Plotting correct and wrong prediction points

Part 4:

A. Conclusion

Finally, in classifying testing data SVM model showed higher accuracy of 98.75% than that of perceptron model of 75% accuracy. In using One versus rest technique, most of the models resulted low accuracies between 41% and 60% instead in high versus rest which was 100%. In One versus One technique all of the models showed marvelously high accuracies of 100% except low versus medium that resulted an accuracy of 98%. Secondly, using argmax in both techniques resulted accuracies of 93.75% and 96.25 for OvR and OvO respectively. Generally, OvO technique has better results than OvR. What has been learnt is that OvO is practically better than OvR, how to use SVM and perceptron classifiers and how to binarize the data and use argmax.

Appendix

A. Class created and used for binarizing data in both OvR and OvO:

```
class binarized:
    def __init__(self,X,Y,le):
        self.X=np.array(X)
        self.Y=np.array(Y)
        self.le=le

    def OVR_binarized(self):
        data={}
        classes=set(self.Y)
        for cl in classes:
            name= self.le.inverse_transform([cl])[0]+'VR'
            y=np.array(self.Y)
            y[self.Y != cl]=0
            y[self.Y == cl]=1
            data[name]=[self.X,y]
        return data

    def OVO_binarized(self):
        data={}
        classes=set(self.Y)
        for cl in classes:
            for cl2 in classes:
                if cl2<=cl:
                    continue
                else:
                    name=self.le.inverse_transform([cl])[0]+'V'+le
.inverse_transform([cl2])[0]
                    y= np.ones(self.Y.shape)*-1
                    y[self.Y == cl]=1
                    y[self.Y == cl2]=0
                    mask=np.where((y==0) | (y==1))
                    data[name]=[self.X[mask],y[mask]]

        return data
```

B. Some of the functions created and used:

```
def visualAndTest(model,x,y,cl1,cl2=None):
    class_names=[]
    if(cl2==None):
        print("Class {} versus rest".format(cl1))
        class_names=['Rest',cl1]
    else:
        print("Class {} versus {}".format(cl1,cl2))
        class_names=[cl2,cl1]

    print("Number of samples:", x.shape[0])

    if(cl2==None):
        print('Accuracy of model classify between {} and rest : {:.2f}'.
format(cl1,model.score(x, y)))
    else:
        print('Accuracy of model classify between {} and {} : {:.2f}'.fo
rmat(cl1,cl2,model.score(x, y)))

    cls_new=[1,0]
    plotData(x, y, cls_new, class_names)
    plotRegions(model, x)
    plt.legend(loc="upper left")
    plt.show()

    print('\nConfusion Matrix:\n')

    ConfusionMatrixDisplay.from_estimator(model, x, y)
    plt.show()
```

```
def plotData(features, labels, cls, class_names, title="SVM Classifier")
:
  colors = ['r', 'b', 'g', 'y']
  markers = ['o', '*', '+', 's']
  for class_index in range(len(cls)):
    plt.scatter(features[labels == cls[class_index],0], features[labels
== cls[class_index],1], c=colors[class_index], marker=markers[class_inde
x], label=class_names[cls[class_index]])
  plt.title(title)
  plt.xlabel('LPR')
  plt.ylabel('PEG')
  plt.legend()
```

```
def plotRegions(model, X):

  x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
  y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
  XX, YY = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y
_max, 0.01))

  z = model.predict(np.c_[XX.ravel(), YY.ravel()])
  ZZ = z.reshape(XX.shape)
  plt.contourf(XX, YY, ZZ, alpha=0.2)
```