Report Assignment 1

(Multiclass Classification)

Applied Machine Learning ELG5255[EG]

Group Number: G 11

Team Members:

Toka Mostafa,

Sawsan Awad,

Sondos Mohammed

Table of contents

1 Introduction

In this assignment, we had Data User Modeling Dataset (DUMD). Training and test splits are provided in csv file format. Data description:

- STG (The degree of study time for goal object materials), (input value)
- SCG (The degree of repetition number of user for goal object materials) (input value)
- STR (The degree of study time of user for related objects with goal object) (input value)
- LPR (The exam performance of user for related objects with goal object) (input value)
- PEG (The exam performance of user for goal objects) (input value)
- UNS (The knowledge level of user) (target value)
    - Very Low: 50
    - Low:129
    - Middle: 122
    - High 130

We made 3 different techniques to predict the output of DUMDs:

1. First: we used multi classes using 2 different models ("SVM" and "Perceptron").
2. Second: we used One versus Rest (OvR).
3. Third: we used One Versus One (OvO).

2 Problems

1.1 Load the DUMD dataset

1.2 convert categorical class labels under the "UNS" column to numerical values by using the Label Encoder.

1.3 Choose two features from DUMD dataset to apply SVM and Perceptron algorithms for classification.

1.4 Plot the data by showing classes separately.

1.5 Classify testing data by using SVM and Perceptron classifiers. Provide accuracies, confusion matrix and decision boundaries for both classifiers.

2.1 Build OvR-SVM, test on DUMD testing dataset with obtained features from Problem 1.

2.2 For each binary classifier:

- Obtain the binarized labels (OvR)
- Obtain the SVM's accuracy
- Plot SVM's decision boundary

2.3 Use argmax to aggregate confidence scores and obtain the final predicted labels and obtain the performance (i.e., confusion matrix, accuracy, plotting correct and wrong prediction points) of OvR-SVM. You can check MBC Simple Data example in lab 2 for aggregation of confidence scores.

3.1 Build OvO-SVM, test on DUMD testing dataset with obtained features from Problem 1.

3.2 For each binary classifier:

- Obtain the binarized labels (OvO)
- Obtain the SVM's accuracy
- Plot SVM's decision boundary

3.3 Use argmax to aggregate confidence scores and obtain the final label and obtain the performance (i.e., confusion matrix, accuracy, plotting correct and wrong prediction points) of OvO-SVM.

3 Methodology

3.1 Importing the libraries & datasets:

- ✓ We import all libraries we need in the first cell.
    - For Preprocessing:
        - Pandas
        - NumPy
    - For Visualization:
        - Seaborn
        - Matplotlib
- ✓ We put the id of our datasets from drive to make it easy to read it then we read it by pd.read_csv() function.

3.2 Preprocessing

3.2.1 Manipulating the datasets:

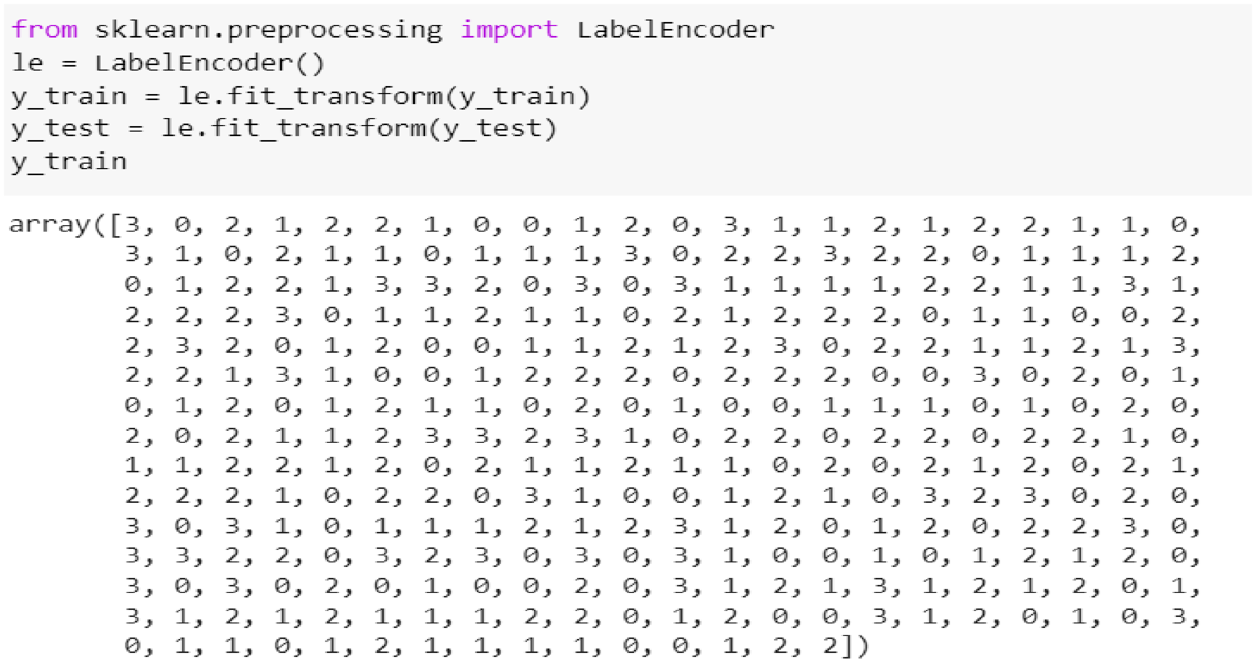
- ✓ We use the .describe() function to get a descriptive statistics summary of 2 datasets that we have. This function includes mean, count, std deviation, percentiles, and min-max values of all the features.
- ✓ We use .info() function to prints information about the 2 datasets. The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values).

3.2.2 Splitting features and labels

- ✓ We split the 2 datasets by a unique function called .iloc[], which is used to select a value that belongs to a particular row or column from a set of values of a data frame or dataset. For example from our code, we gave it [:,:-1] in x_train, which means that retrieve all rows and all columns except the last column.

3.2.3 Label Encoder:

- ✓ A very efficient tool for encoding the levels of categorical features into numeric values which encode labels with a value between 0 and n_classes-1 where n is the number of distinct labels.



```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.fit_transform(y_test)
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])
```

3.3 Feature Selection:

- ✓ A very efficient tool for encoding the levels of categorical features into numeric values which encode labels with a value between 0 and n_classes-1 where n is the number of distinct labels.

3.3.1 We use SelectKBest () method to select the features according to the k highest score to prepare data for training. And the highest two feature scores were (PEG, and LPR).

Sol (1.3)

```
from sklearn.feature_selection import SelectKBest,f_classif
select = SelectKBest(score_func=f_classif,k=2)
y_train = y_train.reshape(len(y_train),1)
x_train_score=select.fit(x_train,y_train)
np.set_printoptions(precision=2)
print(x_train_score.scores_)
x_train_new = x_train_score.transform(x_train)
x_train_new
```

The features scores are:

```
[  7.99   5.93   5.35  15.03 633.68]
```

4 Modeling for multiclass

4.1 SVM - SVC Model:

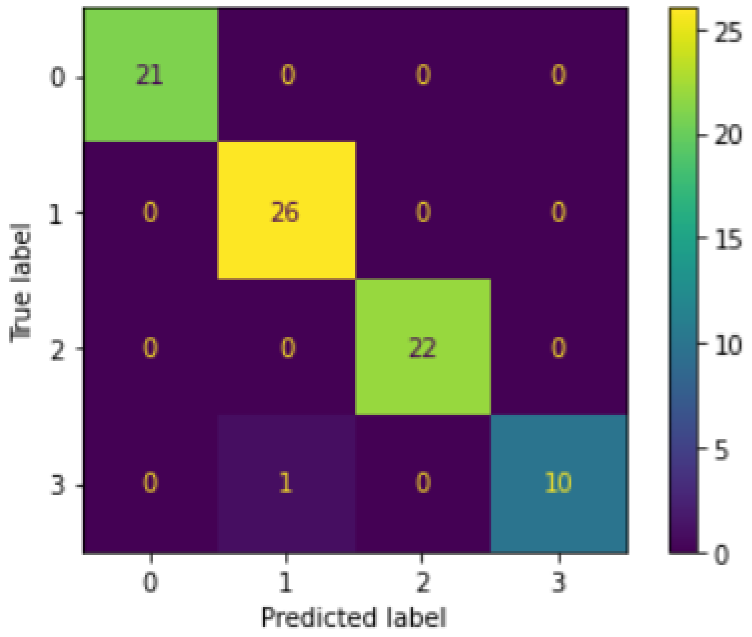
- ✓ We use the SVM library to enable us to call the SVC model to apply a rbf kernel function to perform classification and it performs well with many samples.
- ✓ Model accuracy: 99%

Sol (1.4)

```
from sklearn.svm import SVC
classifier_svc = SVC(kernel="rbf")
classifier_svc.fit(x_train_new,y_train)
y_pred_svc = classifier_svc.predict(x_test_new)
y_pred_svc
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: Dat
  y = column_or_1d(y, warn=True)
array([2, 2, 2, 3, 0, 0, 2, 0, 1, 2, 2, 1, 0, 3, 0, 1, 2, 0, 1, 2, 1, 1,
       2, 2, 2, 1, 2, 0, 2, 0, 0, 0, 2, 0, 1, 0, 1, 3, 1, 1, 2, 0, 1, 2,
       1, 0, 1, 0, 3, 1, 2, 0, 1, 0, 1, 3, 1, 1, 2, 0, 0, 1, 3, 2, 1, 3,
       3, 3, 0, 1, 1, 1, 1, 2, 2, 2, 1, 3, 0, 1])
```

```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report,ConfusionMatrixDisplay
cm_svc=confusion_matrix(y_test,y_pred_svc)
cr_svc = classification_report(y_test,y_pred_svc)
acc_svc=accuracy_score(y_test,y_pred_svc)
disp = ConfusionMatrixDisplay(confusion_matrix =cm_svc,display_labels=classifier_svc.classes_)
disp.plot()
print(cm_svc)
print(cr_svc)
print("Accuracy : {:.2f}%".format(acc_svc))
```



- ✓ The confusion matrix showed us the model is a fault in one sample only.

4.2 Perceptron :

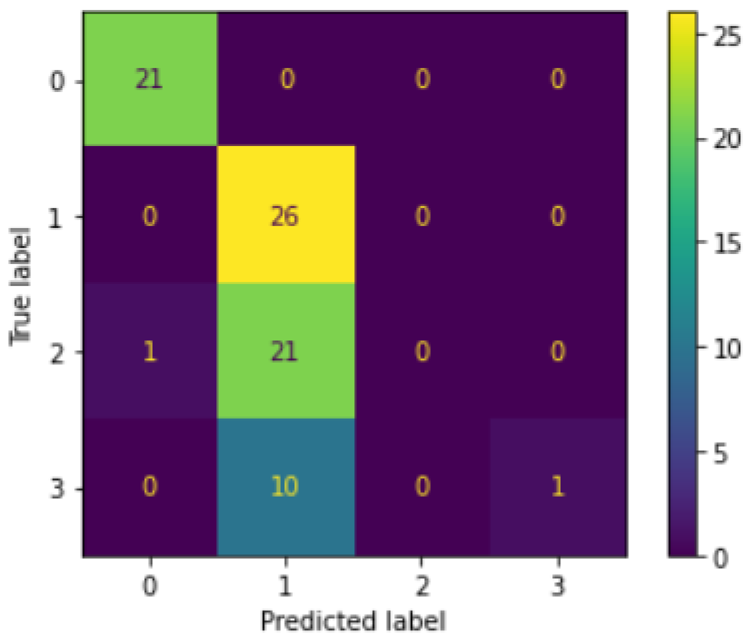
- ✓ We use a perceptron which is a linear machine learning algorithm for binary classification tasks. It may be considered one of the first and one of the simplest types of artificial neural networks
- ✓ Model accuracy: 60%

Sol (1.4)

```
from sklearn.linear_model import Perceptron
classifier_perc = Perceptron(tol=0.001,random_state=0 )
classifier_perc.fit(x_train_new,y_train)
y_pred_perc = classifier_perc.predict(x_test_new)
y_pred_perc
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: Dat
  y = column_or_1d(y, warn=True)
array([1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 0, 3, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1])
```

```
cm_perc = confusion_matrix(y_test,y_pred_perc)
cr_perc = classification_report(y_test,y_pred_perc)
acc_perc = accuracy_score(y_test,y_pred_perc)
disp = ConfusionMatrixDisplay(confusion_matrix = cm_perc,display_labels=classifier_perc.classes_)
disp.plot()
print(cm_perc)
print(cr_perc)
print("Accuracy : {:.2f}%".format(acc_perc))
```

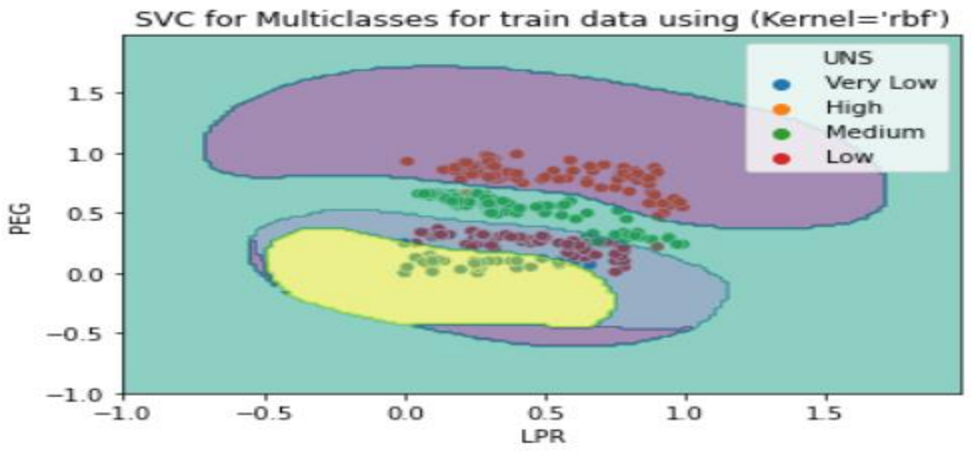


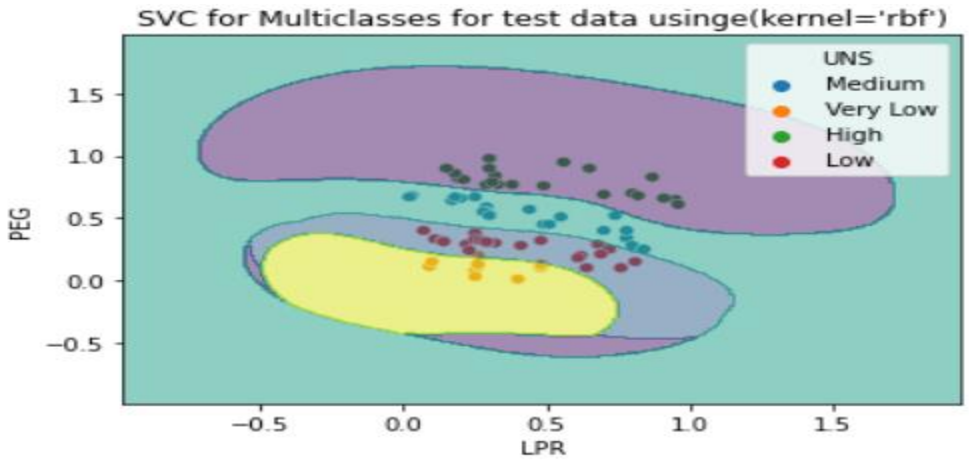
- ✓ The confusion matrix showed that the model is a fault in 10 samples. It considers these samples to belong to class 1, but they belong to class 3.

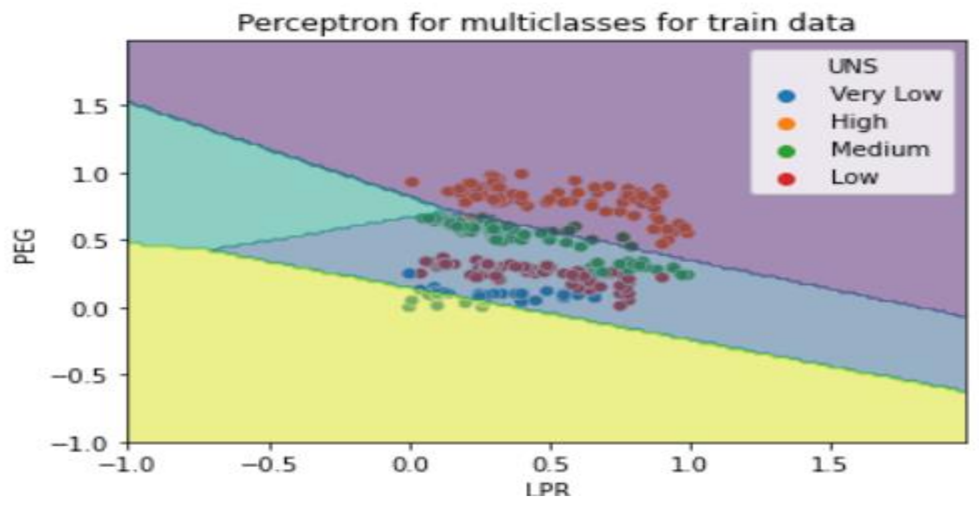
5 Visualization

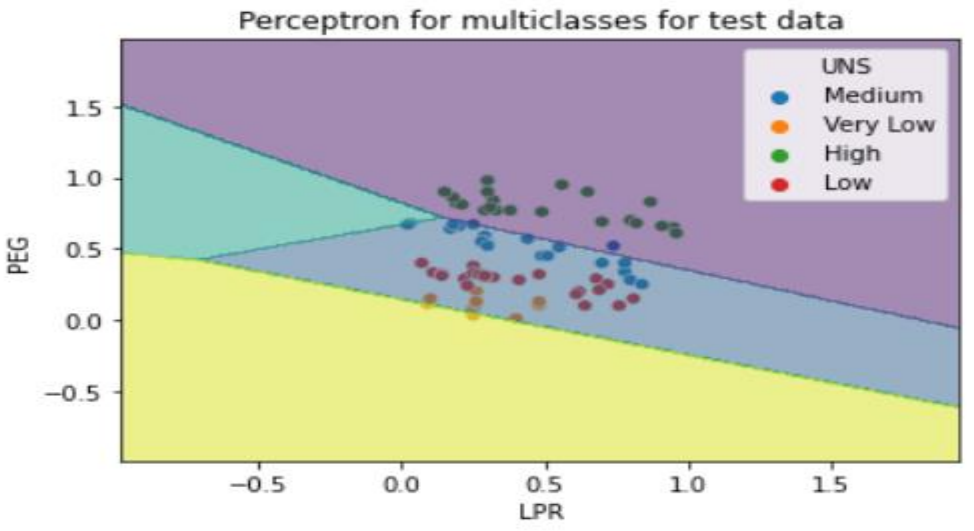
5.1 SVC Model:

Sol (1.5)









- ✓ These visualizations show how perfectly the SVM model separates classes very well than the perceptron model

5.2 OvR-SVM (One versus Rest):

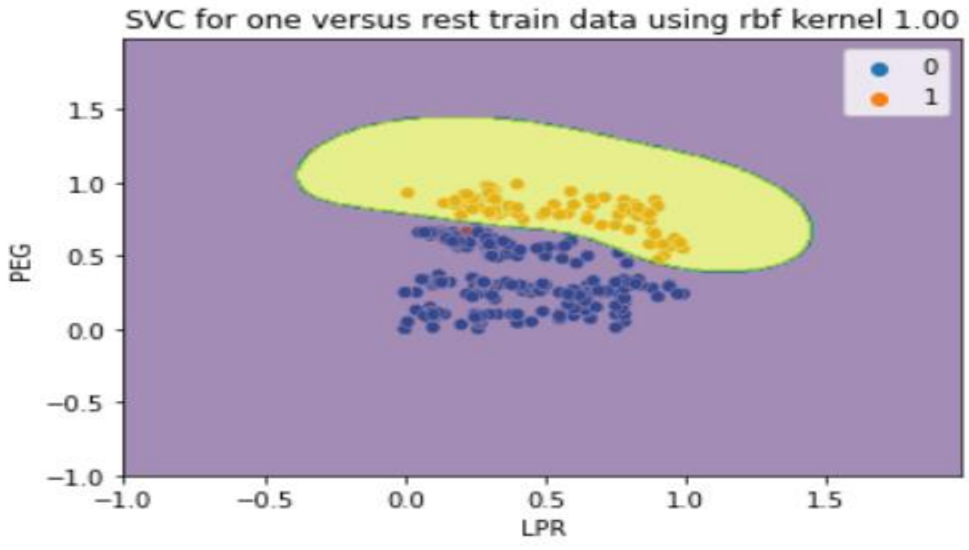
- ✓ This strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. In addition to its computational efficiency (only n_classes classifiers are needed)
- ✓ The label is binarized using MultiLabelBinarizer. After that, four SVM models are constructed to distinguish between each class and other classes using probability. After the model is learned the predict_proba function to test each class and bring the probability of each class.

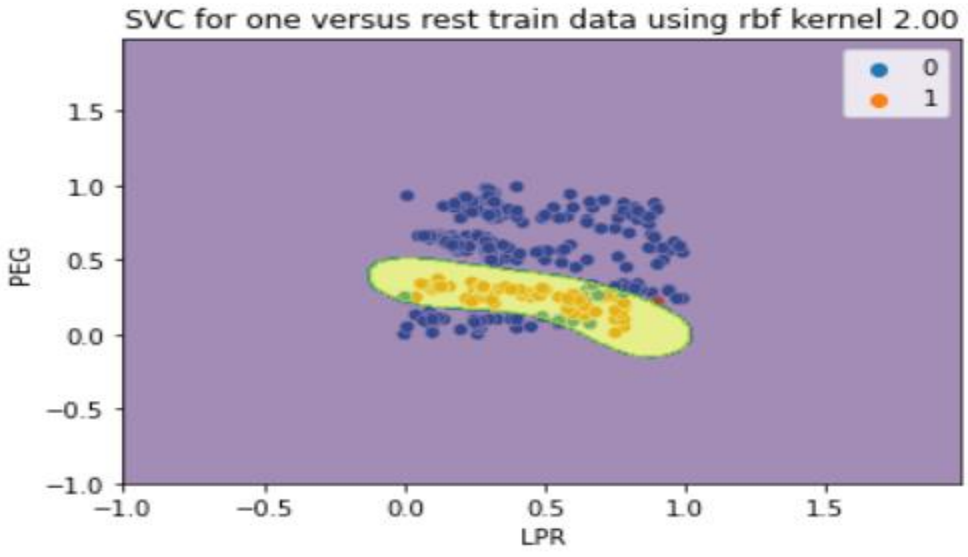
Sol (2.1)

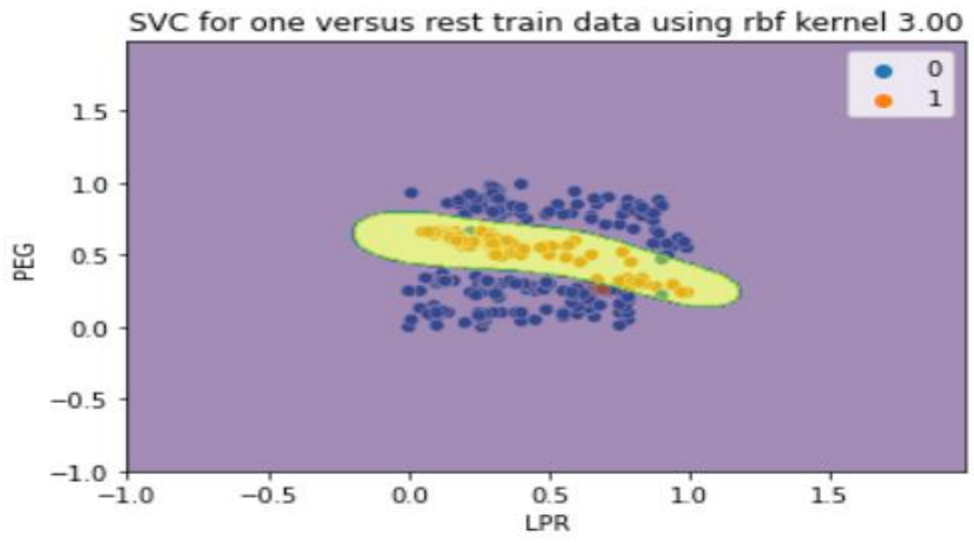
```
y_pred_svc_ovr = pd.DataFrame()
for i in range (4):
  classifier_svc_ovr = SVC(kernel='rbf', probability=True)
  classifier_svc_ovr.fit(x_train_new, y_train_new[:,i])
  y_pred_svc_ovr[i]=classifier_svc_ovr.predict_proba(x_test_new)[:,1]
  sns.scatterplot('LPR','PEG',hue=y_train_new[:,i],data=train)
  x_min, x_max = x_train_new[:, 0].min() - 1, x_train_new[:, 0].max() + 1
  y_min, y_max = x_train_new[:, 1].min() - 1, x_train_new[:, 1].max() + 1
  xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
  z = classifier_svc_ovr.predict(np.c_[xx.ravel(), yy.ravel()])
  Z= z.reshape(xx.shape)
  plt.contourf(xx, yy, Z, alpha=0.5)
  plt.title("SVC FOR one versus rest train data {:.2f}".format(i+1))
  plt.show();
```

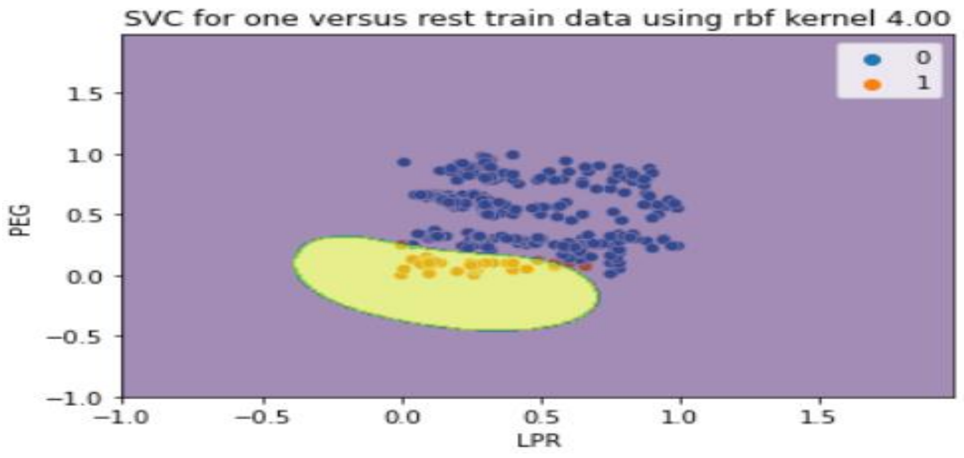
```
y_pred_svc_ovr = pd.DataFrame()
for i in range (4):
  classifier_svc_ovr = SVC(kernel='rbf', probability=True)
  classifier_svc_ovr.fit(x_train_new, y_train_new[:,i])
  score=classifier_svc_ovr.score(x_train_new,y_train_new[:,i])*100
  y_pred_svc_ovr[i]=classifier_svc_ovr.predict_proba(x_test_new)[:,1]
  sns.scatterplot('LPR','PEG',hue=y_test_new[:,i],data=test)
  x_min, x_max = x_test_new[:, 0].min() - 1, x_test_new[:, 0].max() + 1
  y_min, y_max = x_test_new[:, 1].min() - 1, x_test_new[:, 1].max() + 1
  xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
  z = classifier_svc_ovr.predict(np.c_[xx.ravel(), yy.ravel()])
  Z= z.reshape(xx.shape)
  plt.contourf(xx, yy, Z, alpha=0.5)
  plt.title("SVC For one versus rest test data {:.2f}".format(i+1))
  plt.show()
```

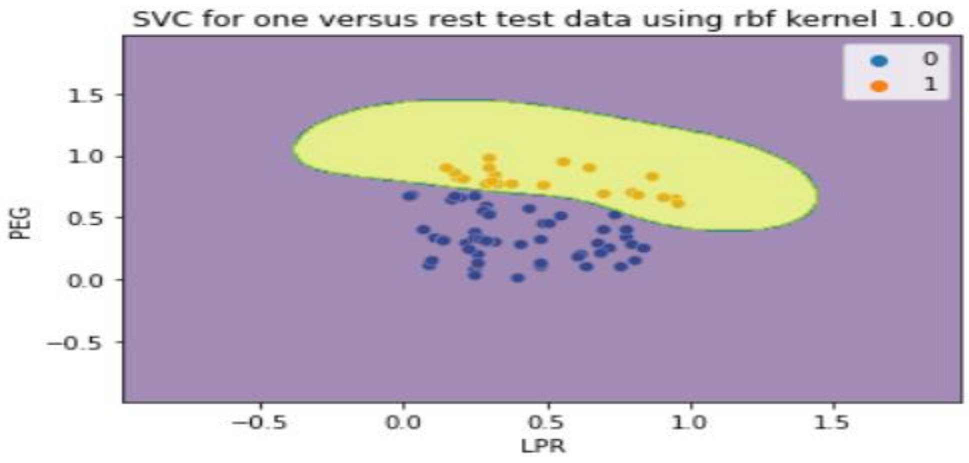
5.2.1 OvR-SVM Visualizations

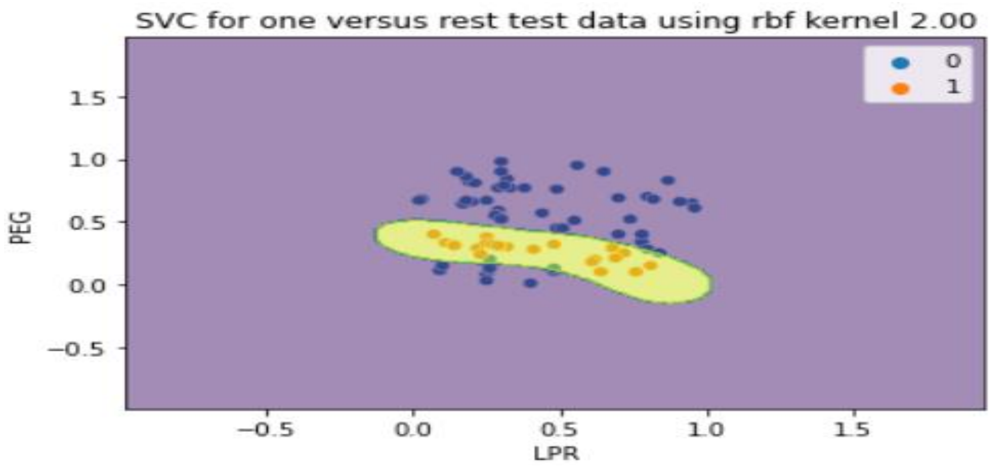


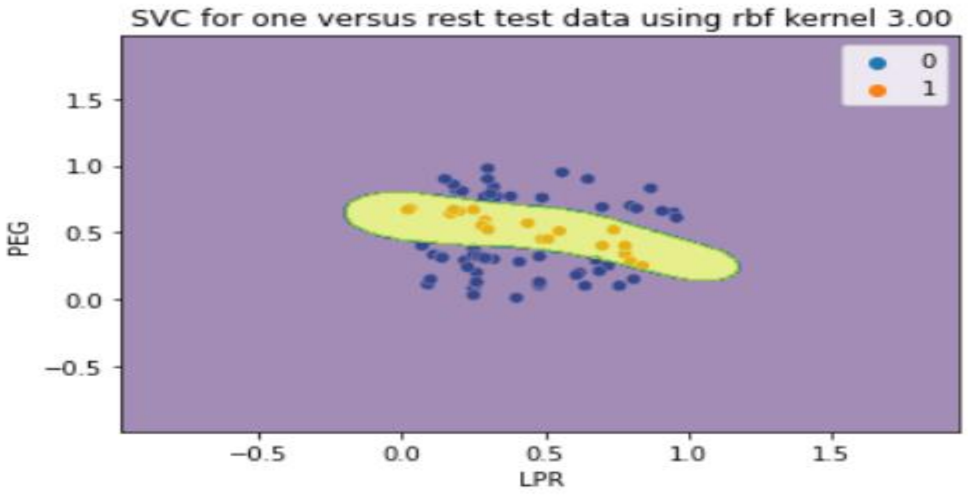


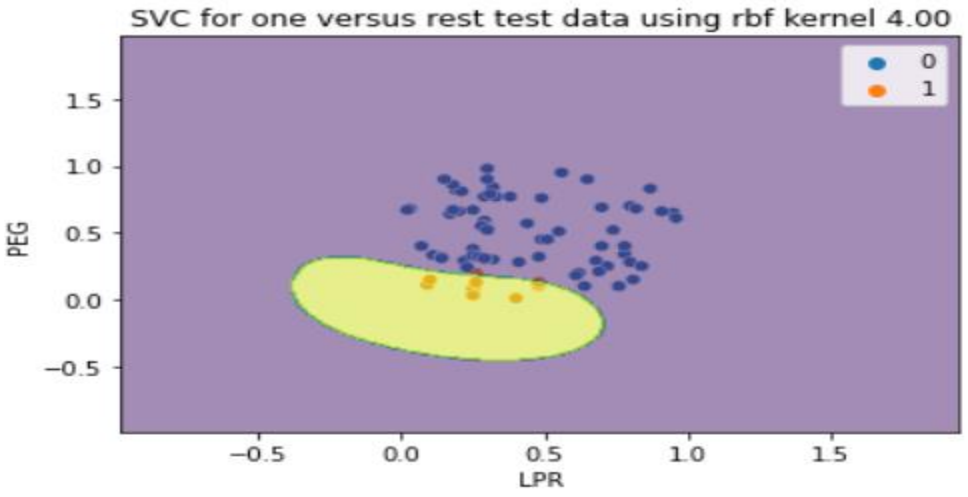

SVC for one versus rest train data using rbf kernel 3.00
0
1
PEG
LPR


SVC for one versus rest train data using rbf kernel 4.00
0
1
PEG
LPR

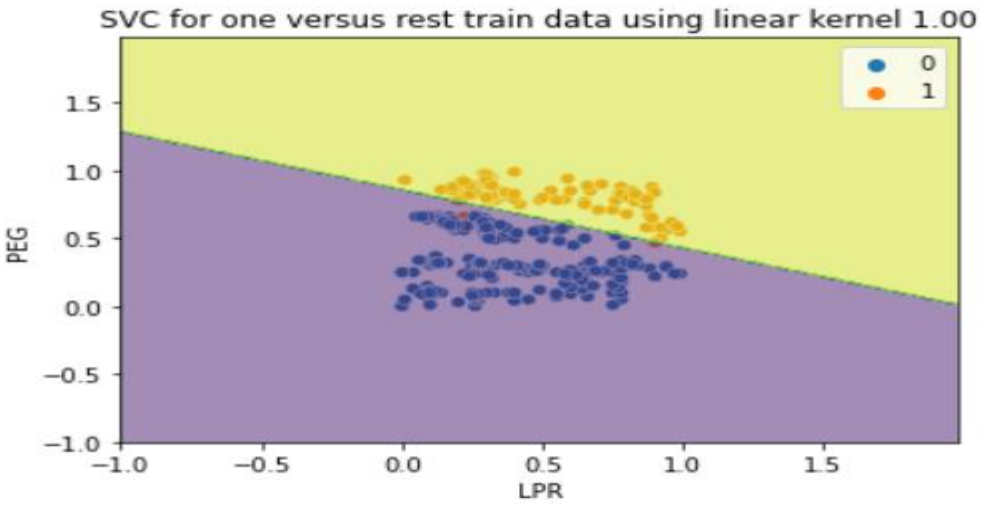

SVC for one versus rest test data using rbf kernel 1.00
0
1
PEG
LPR

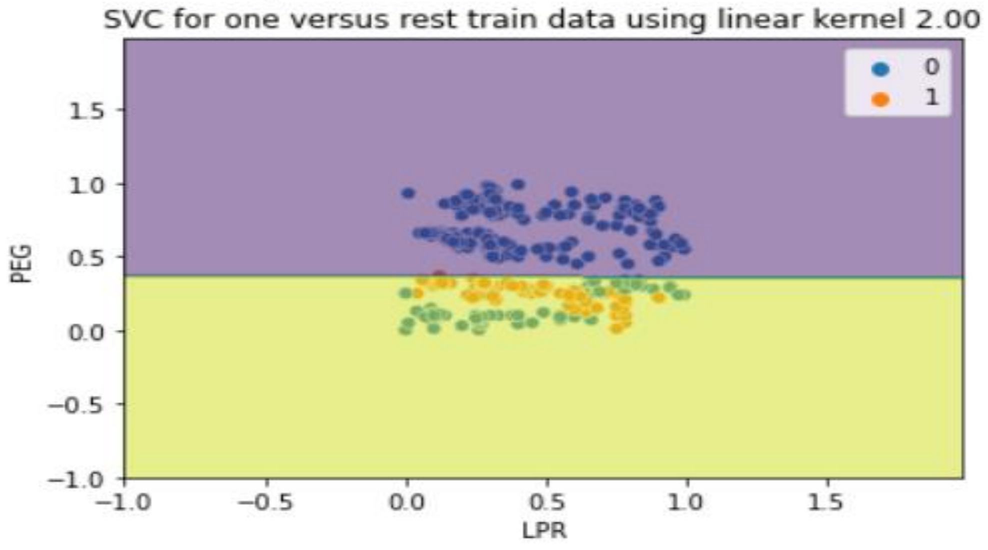


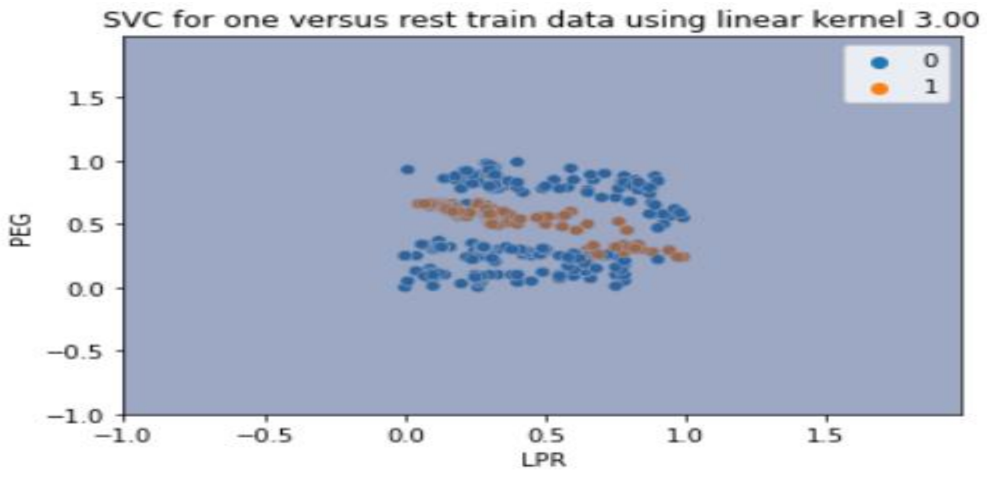


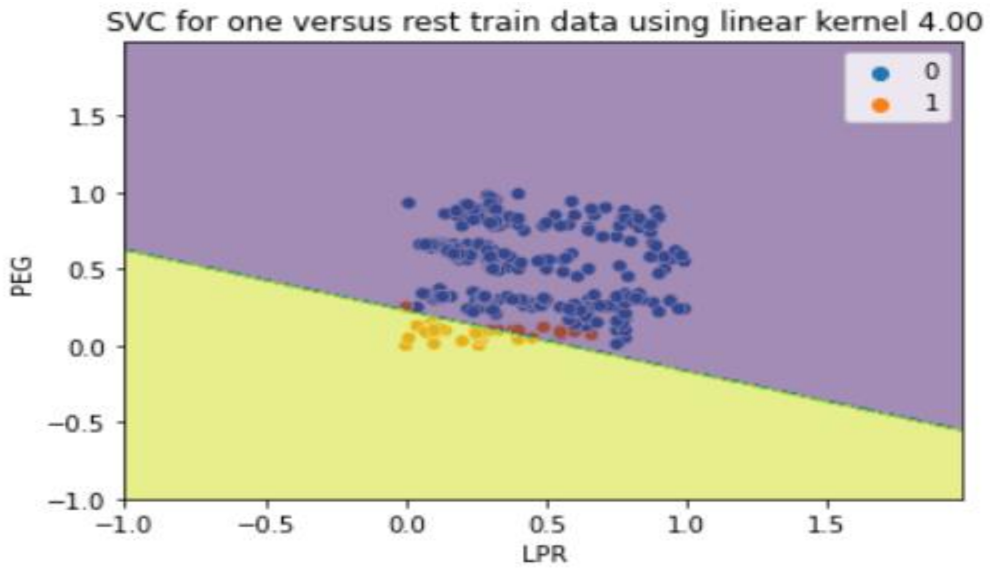


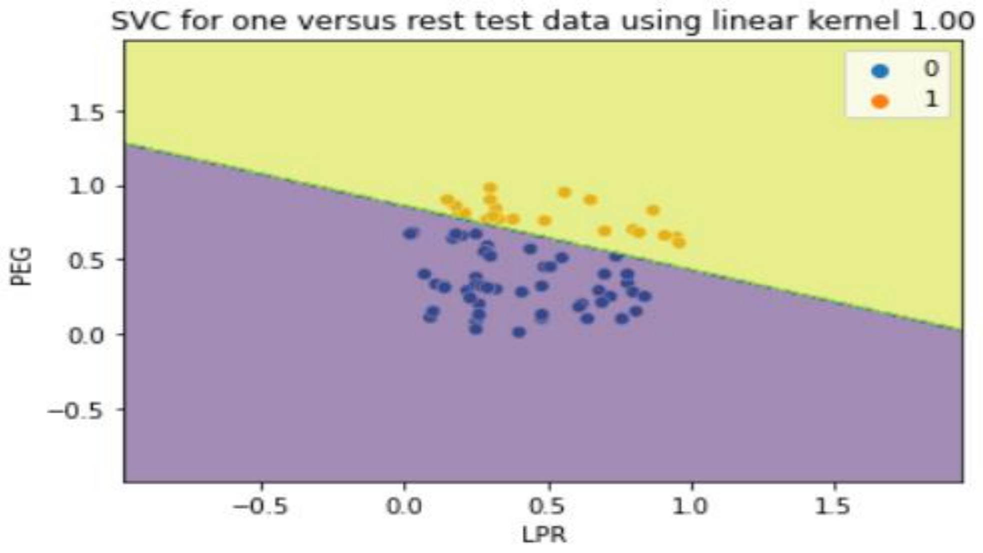
the visualization showed that the dataset is nonlinear so a nonlinear kernel should be used such as the rbf kernel. The performance in the SVM model is very well with small overfitting. Then we tried linear kernel to avoid overfitting the accuracy was down from 98% to 86%

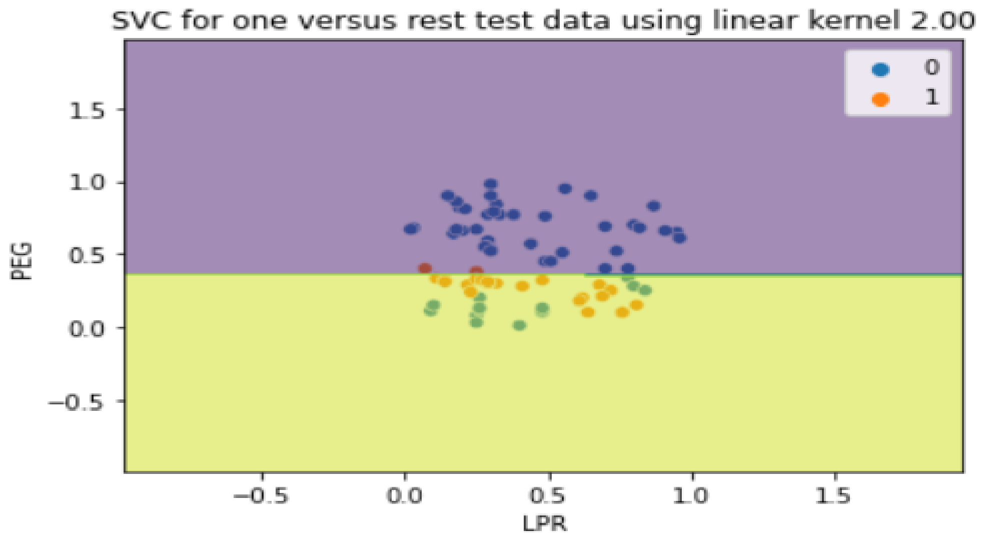


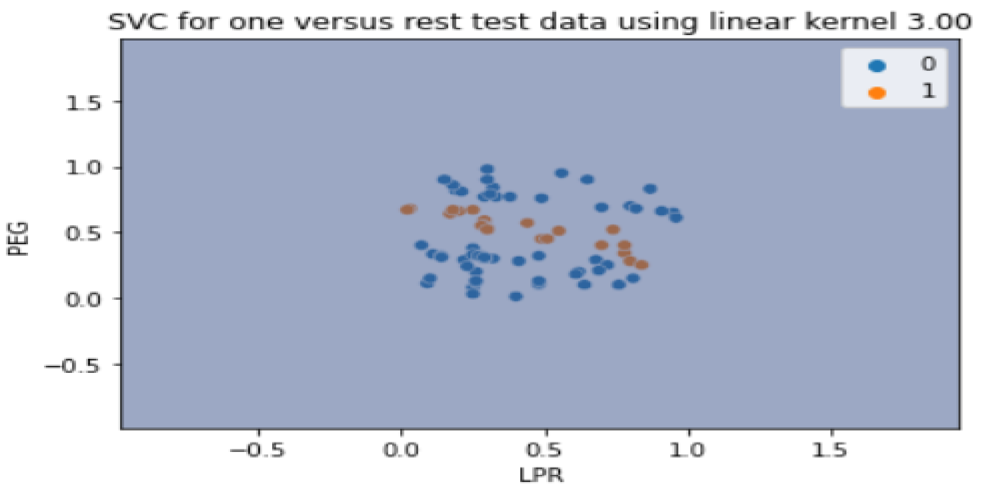


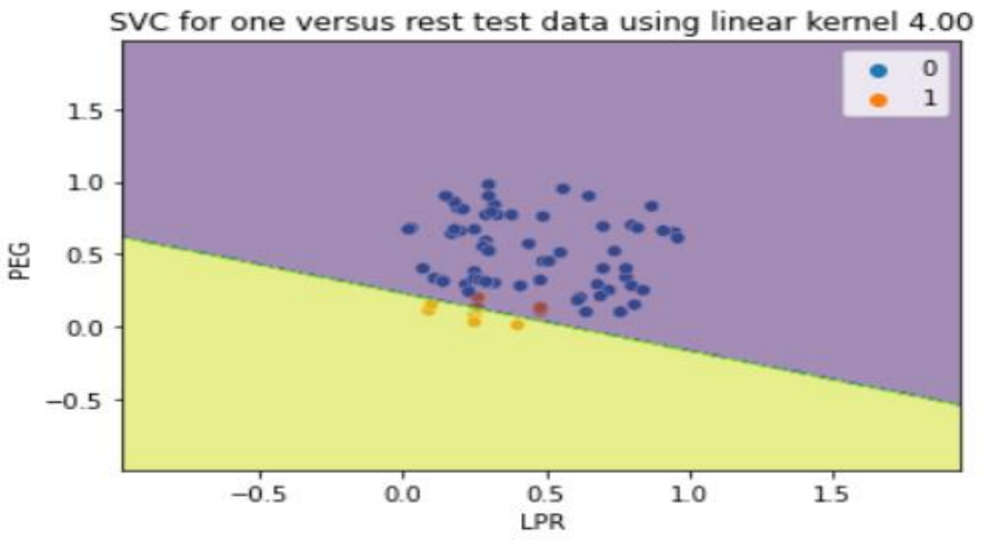



SVC for one versus rest train data using linear kernel 4.00
PEG
LPR
● 0
● 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


SVC for one versus rest test data using linear kernel 1.00
PEG
LPR
● 0
● 1
1.5
1.0
0.5
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5


SVC for one versus rest test data using linear kernel 2.00
PEG
LPR
● 0
● 1
1.5
1.0
0.5
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5

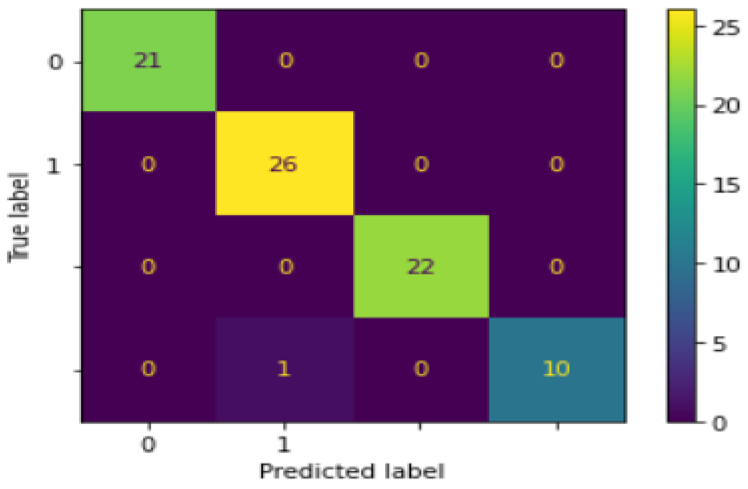




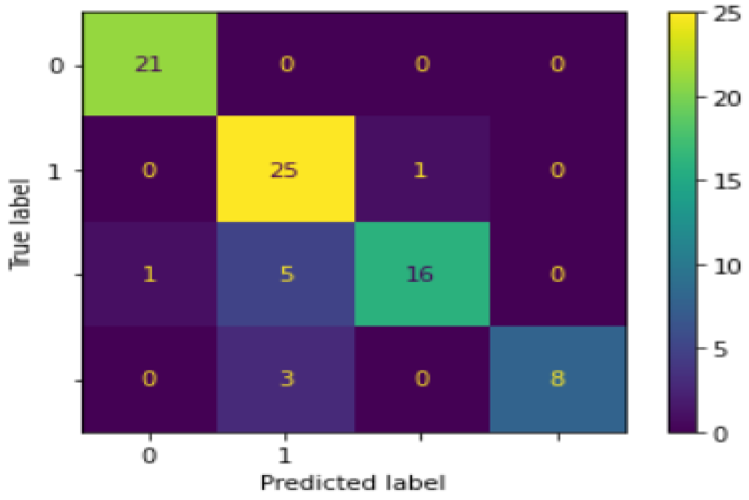
- ✓ After we predict our model using two different kernels we use the argmax function to figure out the maximum probability for each class then we construct a confusion matrix to figure out the performance of each technique. As we said above the rbf technique showed overfitting while linear model showed underfitting
- ✓ After prediction the argmax function is used to calculate the highest class probability

```
probabilties_ovr= y_pred_svc_ovr.to_numpy()
ar_ovr =np.argmax(probabilties_ovr, axis=1)
ar_ovr
```

- ✓ Confusion matrix for rbf kernel



✓ Confusion matrix for linear kernel



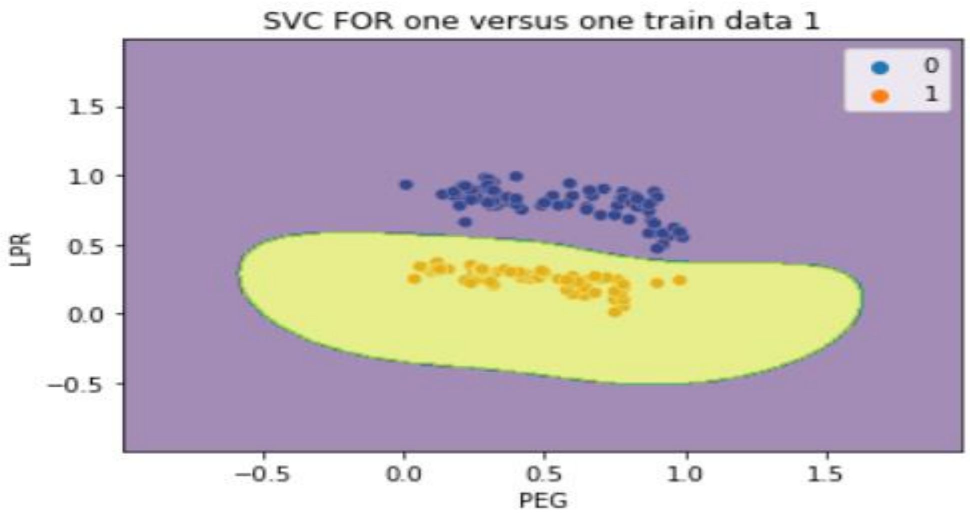
5.3 OvO-SVM (One versus One):

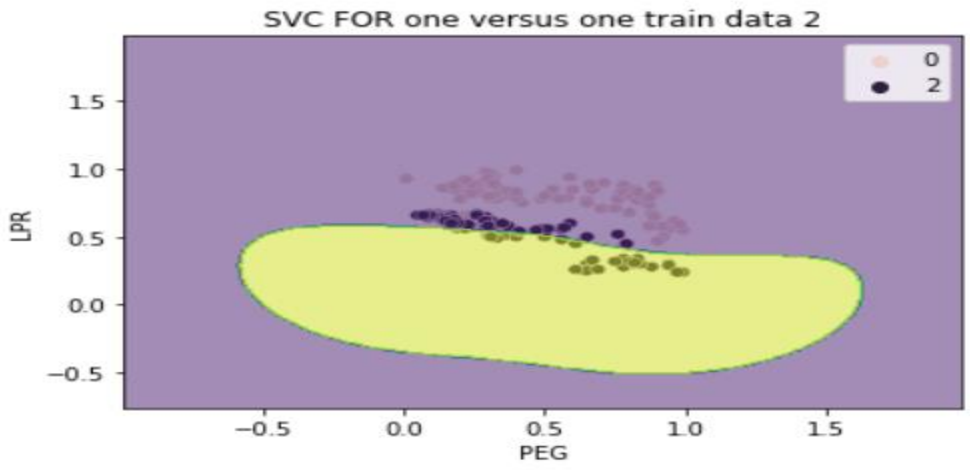
- ✓ This strategy consists in fitting two classes together. In addition to its computational efficiency (only (n_classes* (n_classes-1) /2) classifiers are needed)
- ✓ First, separate the two classes together. Then, six SVM models were performed to distinguish between two classes using probability. After the model is trained, predict_proba was applied to predict the probability of each class.

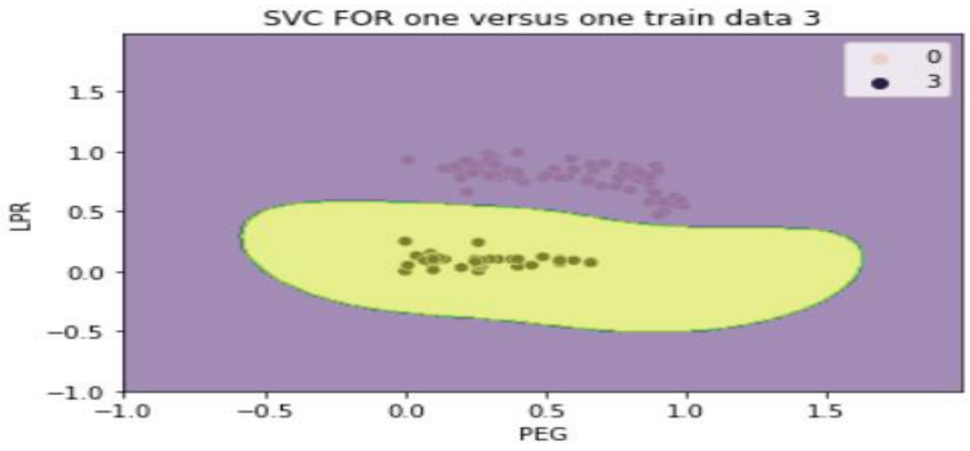
```
for i in range(len(train)):
  if train.UNS[i]==0 or train.UNS[i]==1:
    x_train_1[i] = train.iloc[i,[0,1]]
    y_train_1.append(train.UNS[i])
  if train.UNS[i]==0 or train.UNS[i] ==2:
    x_train_2[i] = train.iloc[i,[0,1]]
    y_train_2.append(train.UNS[i])
  if train.UNS[i]==0 or train.UNS[i] ==3:
    x_train_3[i] = train.iloc[i,[0,1]]
    y_train_3.append(train.UNS[i])
  if train.UNS[i] ==1 or train.UNS[i] ==2:
    x_train_4[i] = train.iloc[i,[0,1]]
    y_train_4.append(train.UNS[i])
  if train.UNS[i] ==1 or train.UNS[i] ==3:
    x_train_5[i] = train.iloc[i,[0,1]]
    y_train_5.append(train.UNS[i])
  if train.UNS[i] ==2 or train.UNS[i] ==3:
    x_train_6[i] = train.iloc[i,[0,1]]
    y_train_6.append(train.UNS[i])
x_train_1=x_train_1.T.to_numpy()
x_train_2=x_train_2.T.to_numpy()
x_train_3=x_train_3.T.to_numpy()
x_train_4=x_train_4.T.to_numpy()
x_train_5=x_train_5.T.to_numpy()
x_train_6=x_train_6.T.to_numpy()
```

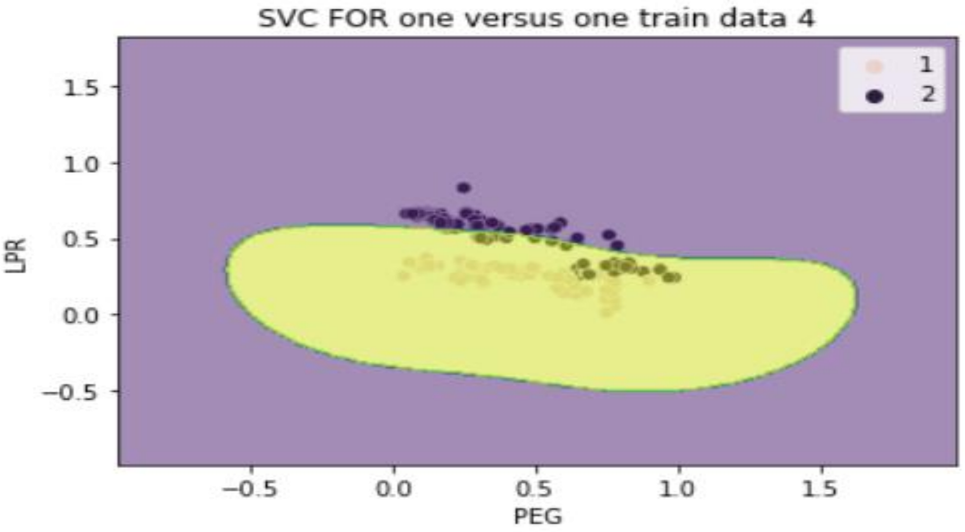
```
classifier_svc_ovo = SVC(kernel='rbf', probability=True)
classifier_svc_ovo.fit(x_train_1, y_train_1)
y_pred_svc_ovo_1=classifier_svc_ovo.predict_proba(x_test_new)
sns.scatterplot(x_train_1[:,0],x_train_1[:,1],hue=y_train_1,data=x_train_1)
x_min, x_max = x_train_1[:, 0].min() - 1, x_train_1[:, 0].max() + 1
y_min, y_max = x_train_1[:, 1].min() - 1, x_train_1[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
z = classifier_svc_ovo.predict(np.c_[xx.ravel(), yy.ravel()])
Z= z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.5)
plt.title("SVC FOR one versus rest train data 1")
plt.xlabel("PEG")
plt.ylabel("LPR")
plt.show();
```

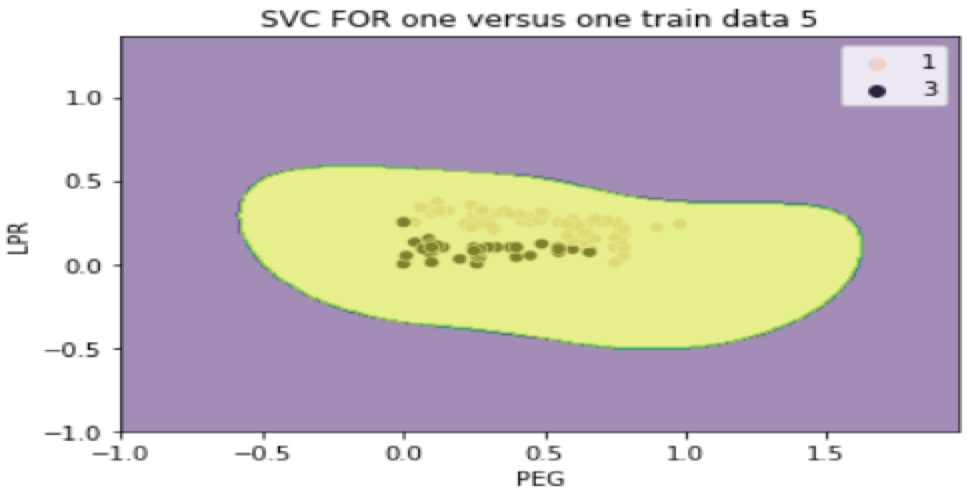
5.3.1 OvO-SVM Visualization:

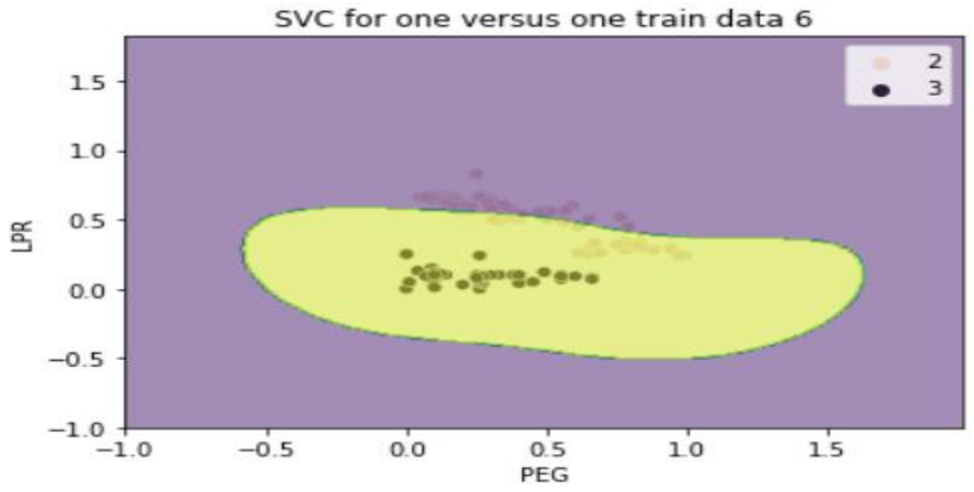


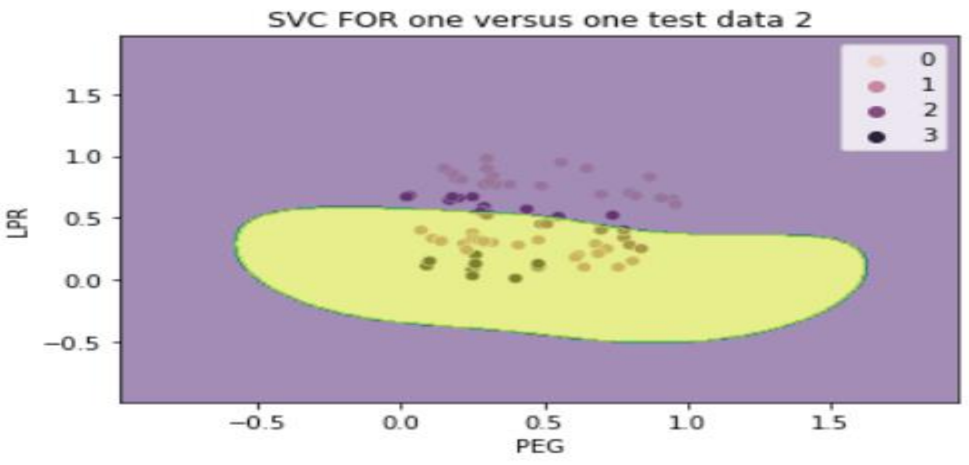


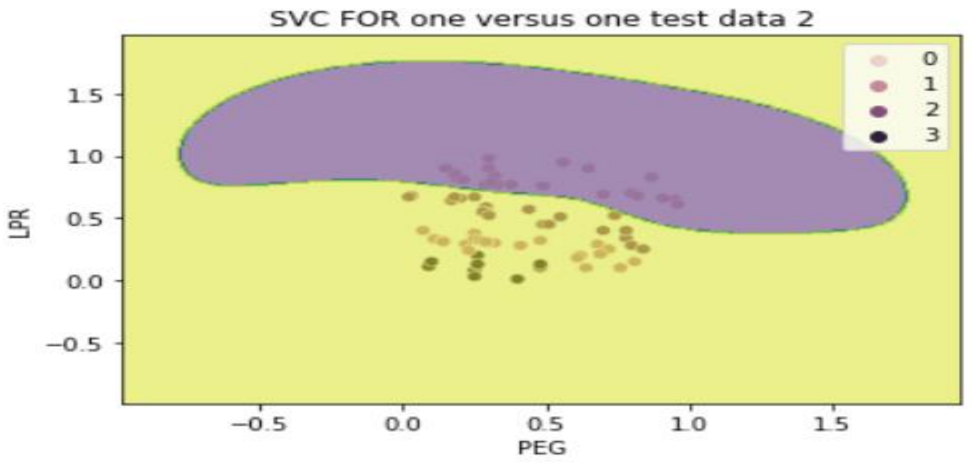

SVC FOR one versus one train data 3
0
3
LPR
PEG
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

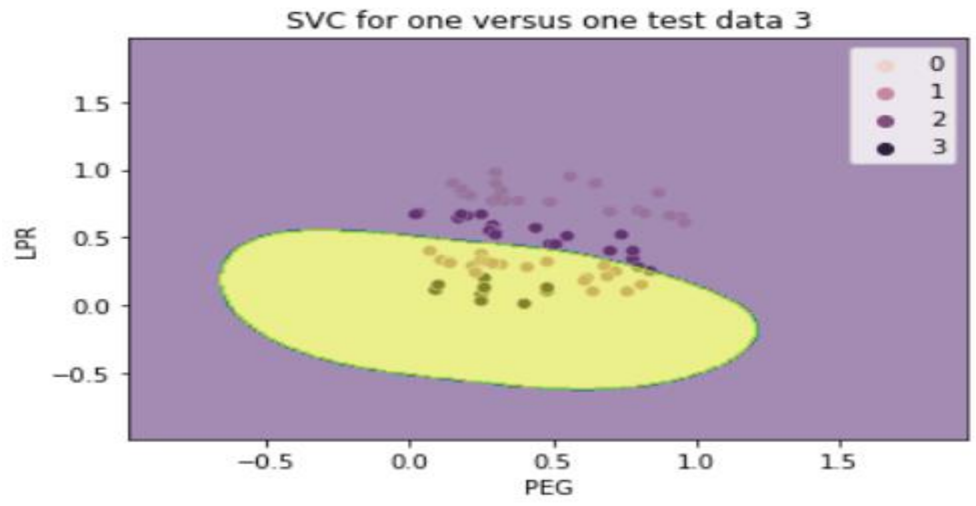

SVC FOR one versus one train data 4
1
2
LPR
PEG
1.5
1.0
0.5
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5

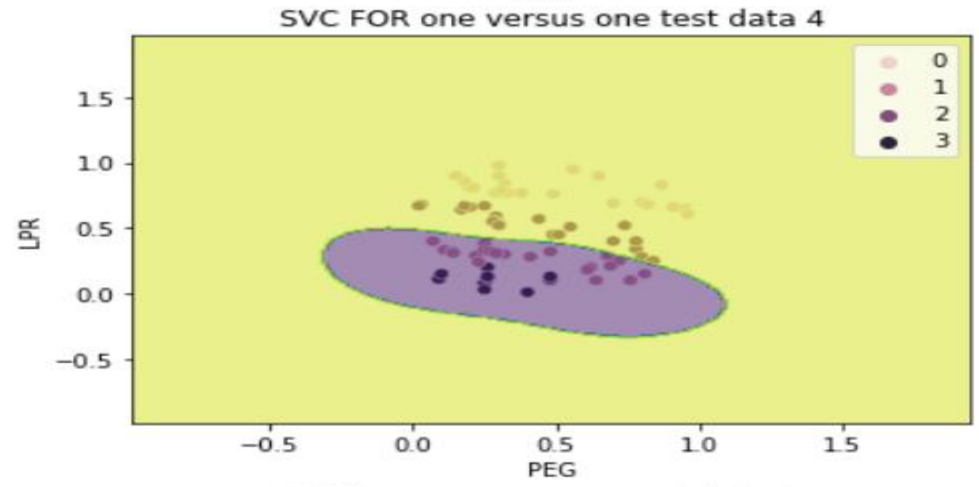

SVC FOR one versus one train data 5
1
3
LPR
PEG
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

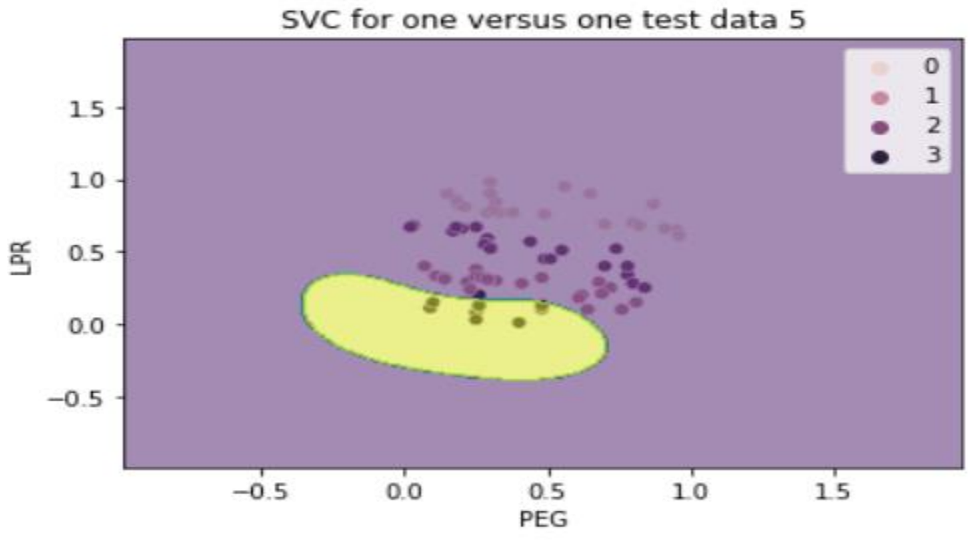

SVC for one versus one train data 6
2
3
LPR
PEG
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

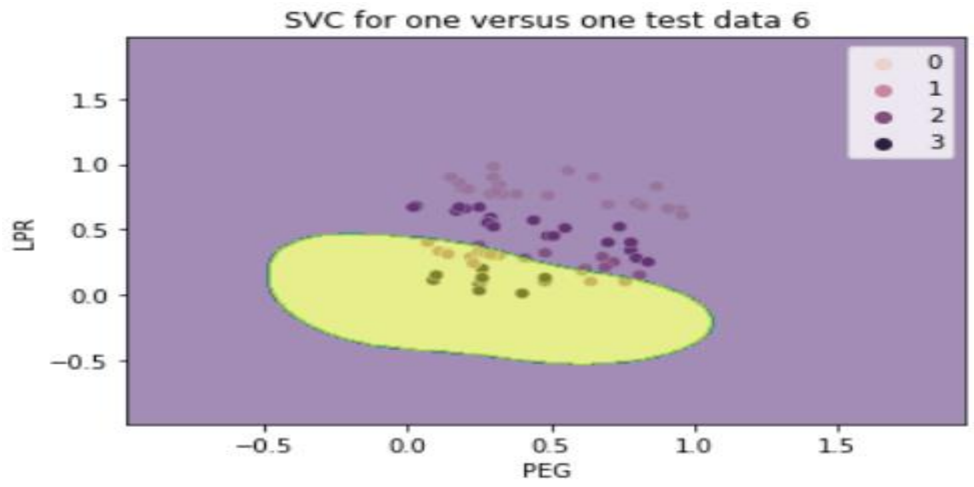

SVC FOR one versus one test data 2
0
1
2
3
LPR
PEG
1.5
1.0
0.5
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5


SVC FOR one versus one test data 2
0
1
2
3
LPR
PEG
1.5
1.0
0.5
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5


SVC for one versus one test data 3
LPR
PEG
0
1
2
3


SVC FOR one versus one test data 4
LPR
PEG
0
1
2
3


SVC for one versus one test data 5
LPR
PEG
0
1
2
3



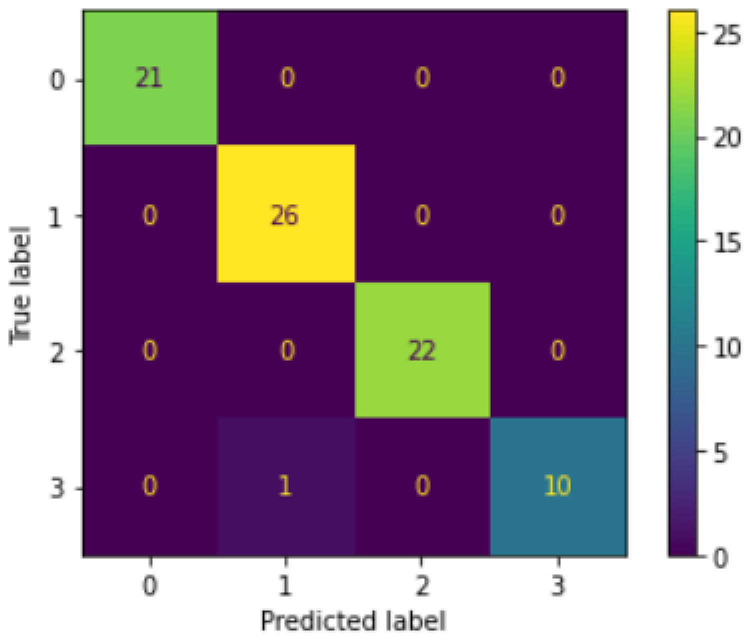
- ✓ The visualization performance showed us the model didn't fit very well because there are a lot of classes the models didn't train on them.
- ✓ After we get the probability form predict_proba. We compute the mean of each class. Then, hstack function is used to combine all classes together in one array to perform argmax probability
- ✓ The accuracy is 98.75%

```
sum_0 = y_pred_svc_ovo_1[:,0] + y_pred_svc_ovo_2[:,0] + y_pred_svc_ovo_3[:,0]
sum_1 = y_pred_svc_ovo_1[:,1] + y_pred_svc_ovo_4[:,0] + y_pred_svc_ovo_5[:,0]
sum_2 = y_pred_svc_ovo_2[:,1] + y_pred_svc_ovo_4[:,1] + y_pred_svc_ovo_6[:,0]
sum_3 = y_pred_svc_ovo_3[:,1] + y_pred_svc_ovo_5[:,1] + y_pred_svc_ovo_6[:,1]
classes_0 = (sum_0/3).reshape(-1,1)
classes_1 = (sum_1/3).reshape(-1,1)
classes_2 = (sum_2/3).reshape(-1,1)
classes_3 = (sum_3/3).reshape(-1,1)
classes_0
```

```
probabilties_ovo = np.hstack((classes_0,classes_1,classes_2,classes_3))
probabilties_ovo
```

```
ar_ovo =np.argmax(probabilties_ovo, axis=1)
ar_ovo
```

- ✓ Confusion matrix



6 Conclusion:

First, we learned we need to transform labels into numbers using a label encoder because the model doesn't work with characters. Second, in multiclass, we learned SVM model has the capability to distinguish between multi classes than perceptron. Third, we learned how to draw a decision boundary. After that, we learned how to construct one versus one and one versus rest from scratch, what the difference between them, and the performance of one versus rest is better than one versus one because in one versus one we test the model with data that it doesn't have any similar experience with it. We learned we need to binarize the model before we used it one versus rest and one versus one. We also learned we need to extract the probability of each class using predict_proba. We also learned about the hstack function that is used to combine the probability together, and the argmax that is used to extract the highest probability class to compute the accuracy. Lastly, we earned how to display the confusion matrix

7 References:

[1] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

[2]https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

[3]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html