ELG 5255: Applied Machine Learning

Assignment 1

Group23_HW1

1. Overview

The main objective of this assignment is to build some ML models and check their performance using many different packages and tools of python and learning new Techniques (OVR and OVO) to deal with muti-classification problems as it is binary classification problems.

2. Methodology

We followed some defined steps to obtain the aimed results:

1. **(a) Load the DUMD dataset and convert UNS column into numerical data**

```python
# Function to read the Dataset
@staticmethod
def readDataSet(DataSet_name, Sheet_Name):
    Extension = re.findall('((.csv)|(.xls)|(.xlsx))', DataSet_name)
    Extension = str(Extension)
    if '.csv' in Extension:
        Extension = '.csv'
    elif '.xls' in Extension:
        Extension = '.xls'
    elif '.xlsx' in Extension:
        Extension = '.xlsx'

    if Extension == ('.xls' or '.xlsx'):
        Assignment1.DataFrame = pd.read_excel(DataSet_name, sheet_name = Sheet_Name)
    elif Extension == '.csv':
        Assignment1.DataFrame = pd.read_csv(DataSet_name)
    return Assignment1.DataFrame
```

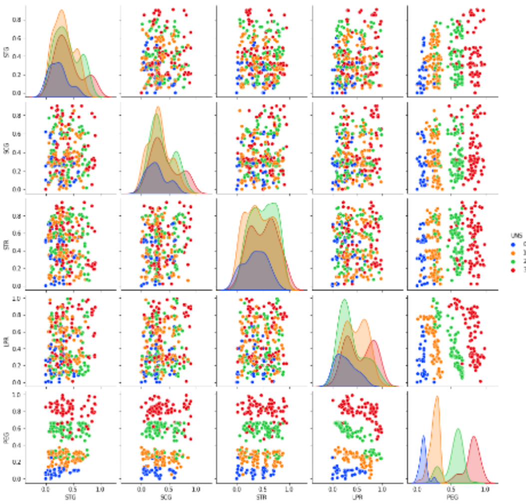
```python
# Function to Encode Labels of Categorical data
@staticmethod
def labelEncoder(DataFrame , Categorical_Column, ListOfClassesNames):
    for i in range(len(ListOfClassesNames)):
        DataFrame.loc[DataFrame[str(Categorical_Column)] == ListOfClassesNames[i], str(Categorical_Column)] = i
    return DataFrame
```

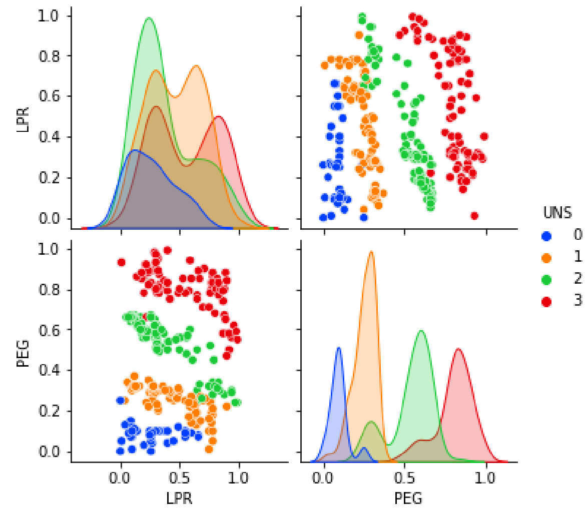
Before and after.

Index	STG	SCG	STR	LPR	PEG	UNS
0	0.000	0.000	0.000	0.000	0.000	Very Low
1	0.080	0.080	0.100	0.240	0.900	High
2	0.100	0.100	0.150	0.650	0.300	Medium
3	0.080	0.080	0.080	0.980	0.240	Low
4	0.090	0.150	0.400	0.100	0.660	Medium
5	0.100	0.100	0.430	0.290	0.560	Medium
6	0.200	0.140	0.350	0.720	0.250	Low
7	0.000	0.000	0.500	0.200	0.850	High
8	0.180	0.180	0.550	0.300	0.810	High
9	0.060	0.060	0.510	0.410	0.300	Low
10	0.200	0.200	0.700	0.300	0.600	Medium
11	0.120	0.120	0.750	0.350	0.800	High
12	0.050	0.070	0.700	0.010	0.050	Very Low

Index	STG	SCG	STR	LPR	PEG	UNS
0	0.000	0.000	0.000	0.000	0.000	0
1	0.080	0.080	0.100	0.240	0.900	3
2	0.100	0.100	0.150	0.650	0.300	2
3	0.080	0.080	0.080	0.980	0.240	1
4	0.090	0.150	0.400	0.100	0.660	2
5	0.100	0.100	0.430	0.290	0.560	2
6	0.200	0.140	0.350	0.720	0.250	1
7	0.000	0.000	0.500	0.200	0.850	3
8	0.180	0.180	0.550	0.300	0.810	3
9	0.060	0.060	0.510	0.410	0.300	1
10	0.200	0.200	0.700	0.300	0.600	2
11	0.120	0.120	0.750	0.350	0.800	3
12	0.050	0.070	0.700	0.010	0.050	0

1. **(b) Choose two features based on PairPlot (the two most separable features that classification model can classify the different classes from each other easily) and "ExtraTreesClassifier"**






```python
from sklearn.ensemble import ExtraTreesClassifier
```

```python
    # Function to detemine which features will be eliminated
    @staticmethod
    def FeatureSelection(X,Y):
        Selector = ExtraTreesClassifier(n_estimators=2)
        Selector = Selector.fit(X, Y)
        return Selector.feature_importances_
```

Output : feature importance score for each feature.

[0.05059389, 0.02684576, 0.06839334, 0.15636901, 0.697798]

After choosing two features.

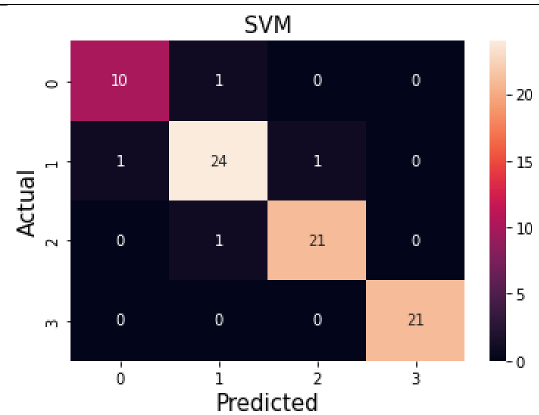
DataSetTrain - DataFrame

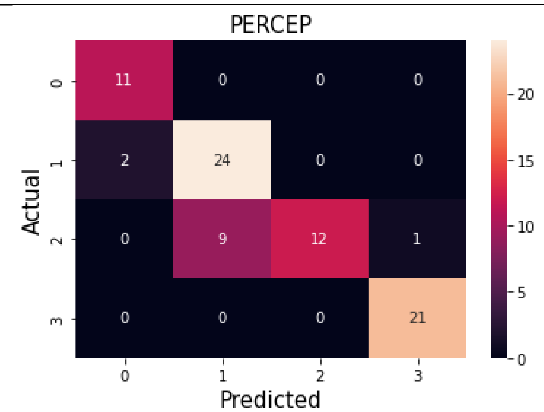
Index	LPR	PEG	UNS
0	0.000	0.000	0
1	0.240	0.900	3
2	0.650	0.300	2
3	0.980	0.240	1
4	0.100	0.660	2
5	0.290	0.560	2
6	0.720	0.250	1
7	0.200	0.850	3
8	0.300	0.810	3
9	0.410	0.300	1
10	0.300	0.600	2
11	0.350	0.800	3

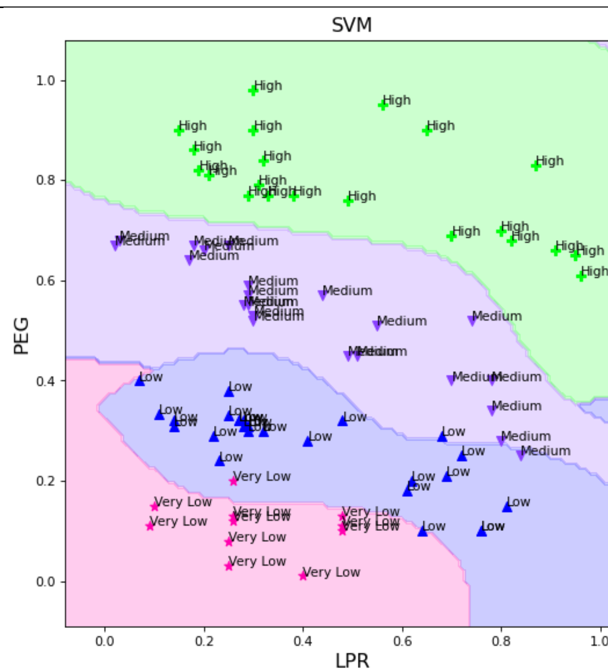
1. **(c) Apply SVM (rbf) and Perceptron**

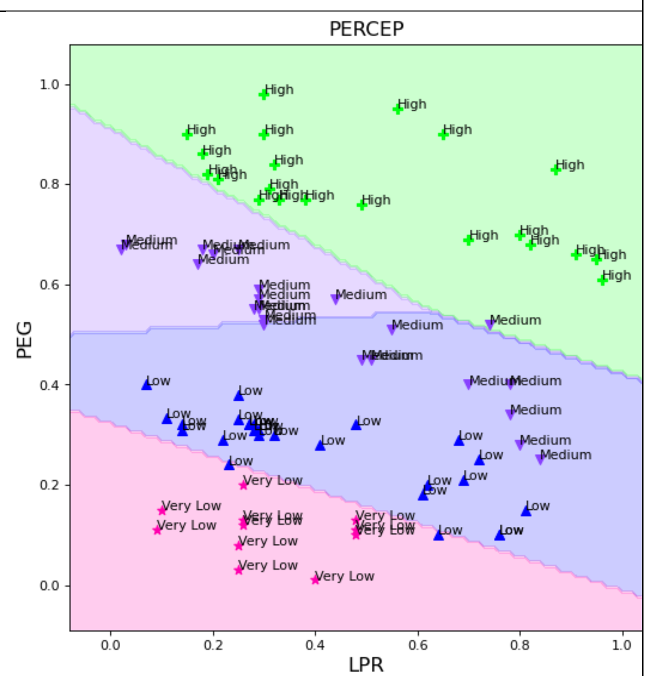
```
# SVM Classifier
@staticmethod
def SVM(X ,Y ,GeneralizationTerm):
    ClassifierSVM = SVC(kernel="rbf", C = GeneralizationTerm, probability=True)
    ClassifierSVM.fit(X,Y)
    return ClassifierSVM

# PERCEP Classifier
@staticmethod
def PERCEP(X,Y,LearningRate,Epoch):
    ClassifierPERCEP = Perceptron(eta0=LearningRate, max_iter=Epoch)
    ClassifierPERCEP.fit(X,Y)
    return ClassifierPERCEP
```

1. **(c) Classify testing data by using SVM and Perceptron classifiers. Provide accuracies, confusion matrix and decision boundaries for both classifiers.**

<table>
<tr><th></th><th>SVM</th><th>Perceptron</th></tr>
<tr><td>Accuracies on test</td><td>95.0</td><td>85.0</td></tr>
<tr><td>Confusion matrix</td><td>


</td><td>


</td></tr>
<tr><td>Decision boundaries</td><td>


</td><td>


</td></tr>
</table>

2. (a) Obtain the binarized labels (OvR)

```python
    # One Vs Rest Spliting
    @staticmethod
    def OvR(DataSet,RequiredClass, Categorical_Column):
        BinaryDF = DataSet.copy()
        if RequiredClass == 0:
            BinaryDF.loc[BinaryDF[str(Categorical_Column)] != RequiredClass, str(Categorical_Column)] = 2
            BinaryDF.loc[BinaryDF[str(Categorical_Column)] == RequiredClass, str(Categorical_Column)] = 1
            BinaryDF.loc[BinaryDF[str(Categorical_Column)] == 2, str(Categorical_Column)] = 0
        else:
            BinaryDF.loc[BinaryDF[str(Categorical_Column)] != RequiredClass, str(Categorical_Column)] = 0
            BinaryDF.loc[BinaryDF[str(Categorical_Column)] == RequiredClass, str(Categorical_Column)] = 1
        return BinaryDF
```

The required class = 1, and the rest = 0.

Y_train for each of 4 models.

OvR_DF_0_TrainY - Se

Index	UNS
0	1
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	1

OvR_DF_1_TrainY - Se

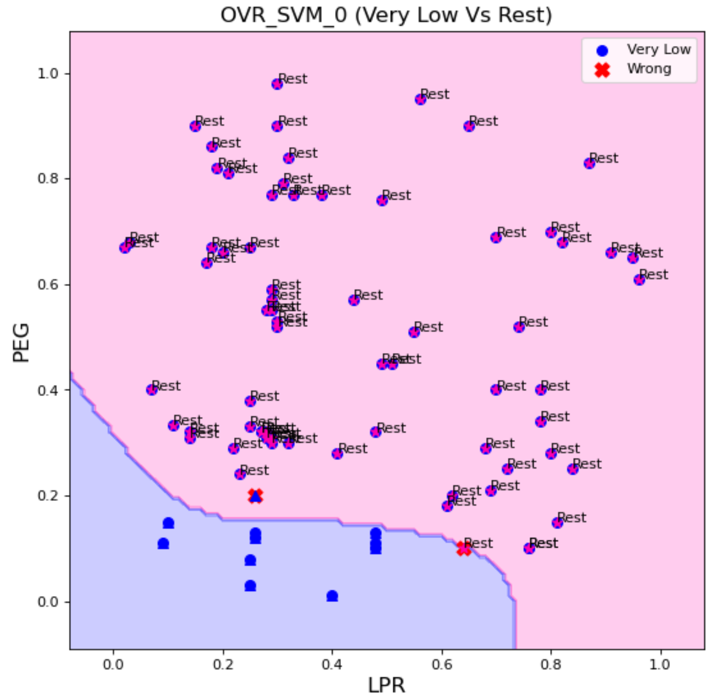
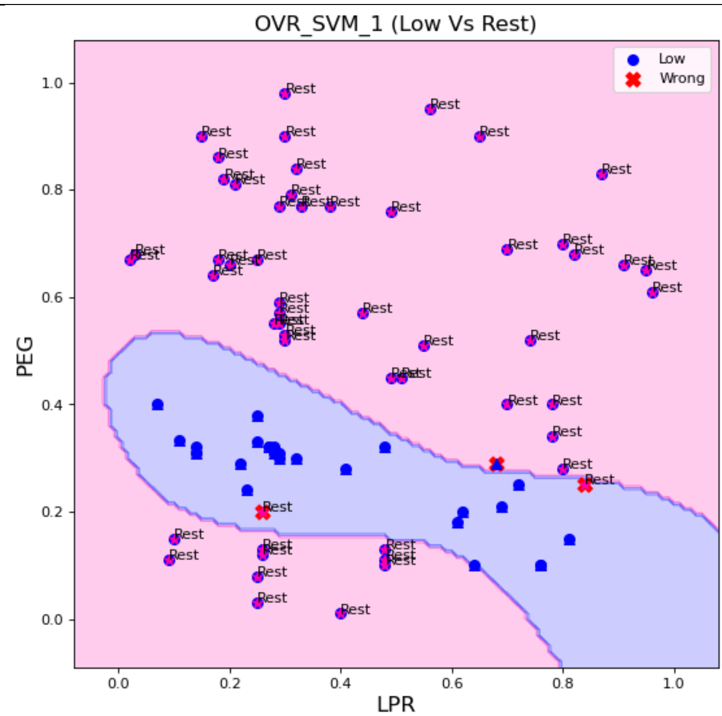
Index	UNS
0	0
1	0
2	0
3	1
4	0
5	0
6	1
7	0
8	0
9	1
10	0
11	0
12	0

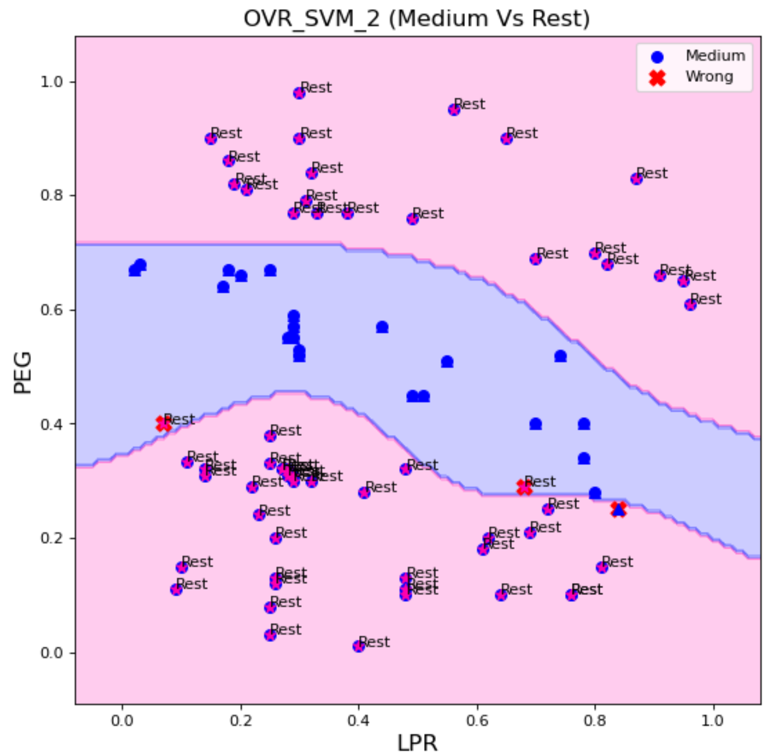
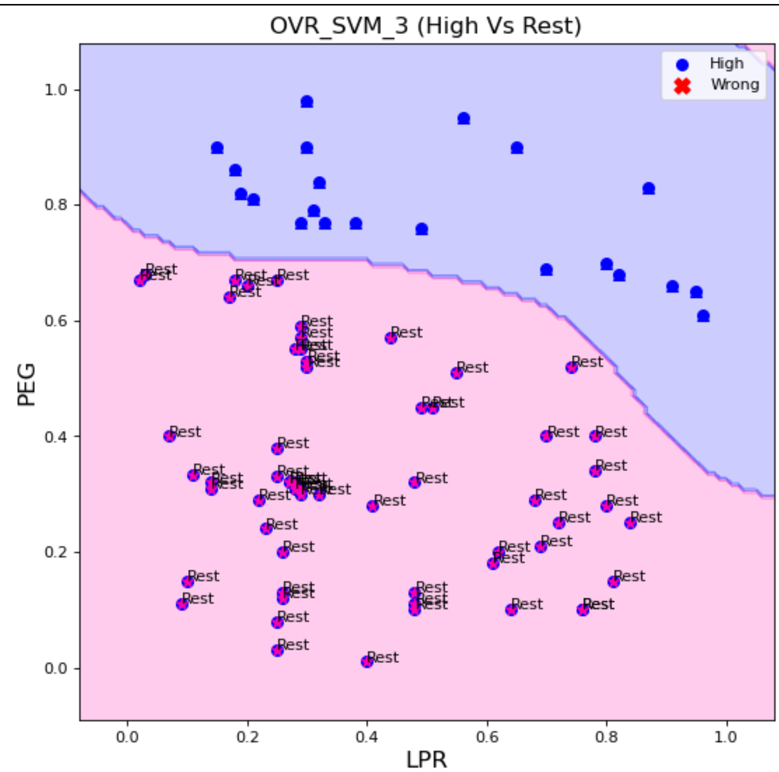
OvR_DF_2_TrainY - Se

Index	UNS
0	0
1	0
2	1
3	0
4	1
5	1
6	0
7	0
8	0
9	0
10	1
11	0
12	0

OvR_DF_3_TrainY - Series

Index	UNS
0	0
1	1
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	0
10	0
11	1
12	0

Classes	SVM's accuracies on test	SVM's decision boundary	Comments
Very low vs Rest	**97.5**		**This model correctly classifies very low class from the rest.** **And there are only 2 wrong points.**
low vs Rest	**96.25**		**This model correctly classifies low class from the rest.** **And there are only 3 wrong points.**

Medium vs Rest	**96.25**		**This model correctly classifies medium class from the rest.** **And there are only 3 wrong points.**
High VS Rest	**100.0**		**This model correctly classifies very low class from the rest.** **And there are no wrong points detected.**

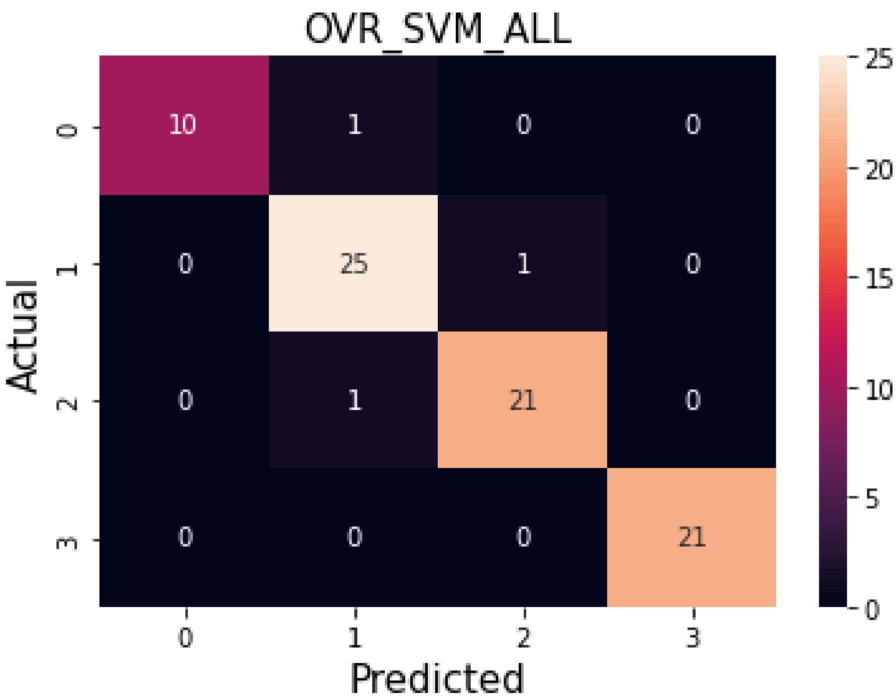
2. (b) **OvR Overall accuracy (after aggregation) is = 96.25**
   We applied numpy.argmax() to these probabilities.

	Y_PredSVM_0_Prop - N	Y_PredSVM_1_Prop - I	Y_PredSVM_2_Prop - I	Y_PredSVM_3_Prop - NumPy object array
	0	0	0	0
0	0.022	0.009	0.8660	0.046
1	0.000	0.000	0.9864	0.001
2	0.053	0.013	0.9490	0.036
3	0.866	0.019	0.0000	0.003
4	0.061	0.000	0.2165	0.880
5	0.002	0.000	0.0000	1.000
6	0.000	0.002	0.9905	0.000
7	0.050	0.000	0.0780	0.952
8	0.001	0.965	0.0018	0.000
9	0.003	0.000	0.9882	0.001
10	0.000	0.001	0.9949	0.000

To obtain this (OvR Y_Predict) and compare it with Y_Actual.

YProbALL_OvR - NumPy object array

	0
0	2
1	2
2	2
3	0
4	3
5	3
6	2
7	3
8	1
9	2
10	2



There are 3 wrong points predicted.

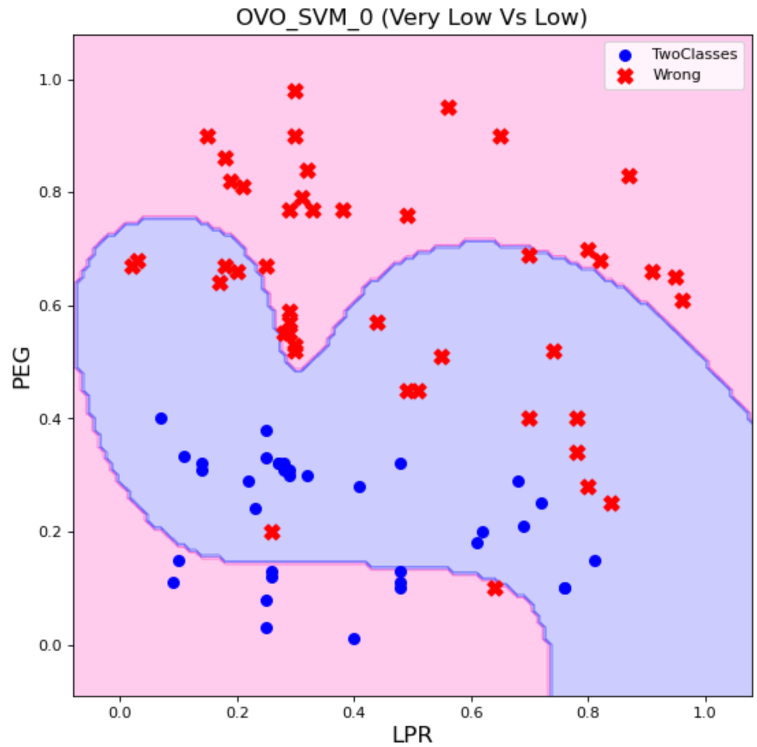
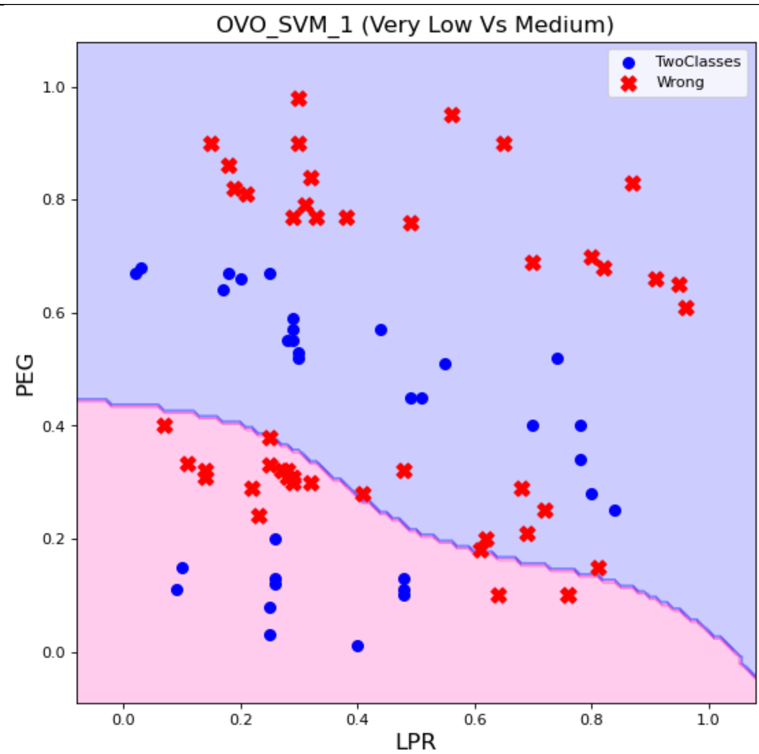
3. (a) Obtain the binarized labels (OvR)

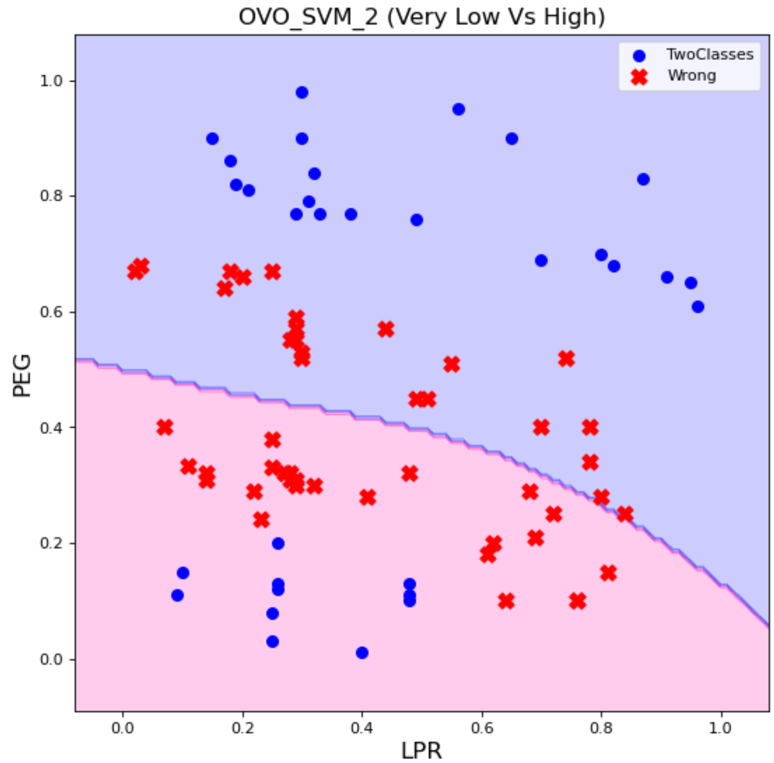
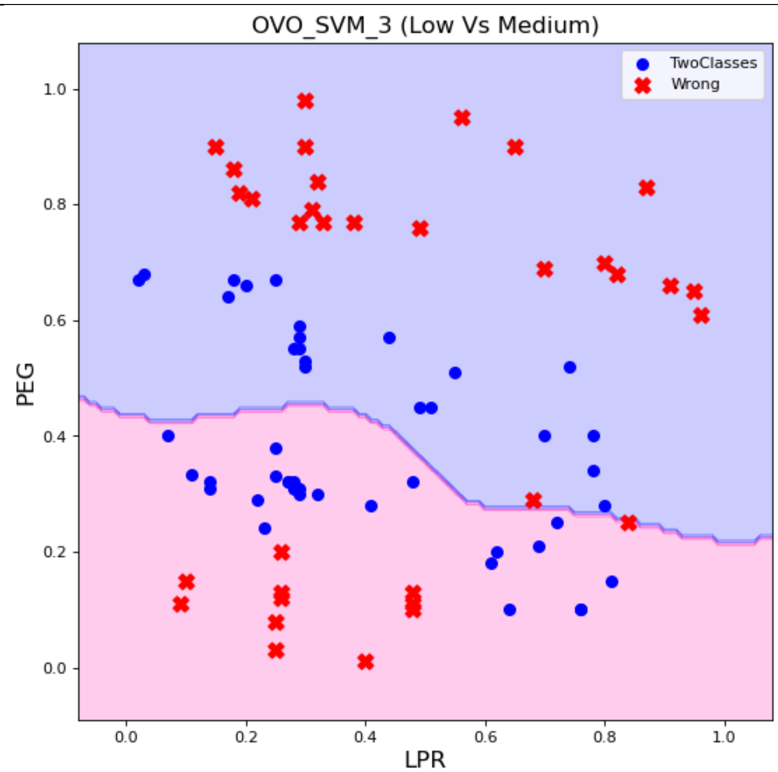
```python
    # One Vs Rest Spliting
    @staticmethod
    def OvO(FristClass, SecondClass, DataSet, Categorical_Column):
        TwoClassesDF = DataSet.copy()
        TwoClassesDF = TwoClassesDF[(TwoClassesDF[str(Categorical_Column)] == FristClass) | (TwoClassesDF[str(Categorical_
        TwoClassesDF = TwoClassesDF.reset_index(drop=True)
        TwoClassesDF.loc[TwoClassesDF[str(Categorical_Column)] == FristClass, str(Categorical_Column)] = FristClass
        TwoClassesDF.loc[TwoClassesDF[str(Categorical_Column)] == SecondClass, str(Categorical_Column)] = SecondClass
        return TwoClassesDF
```

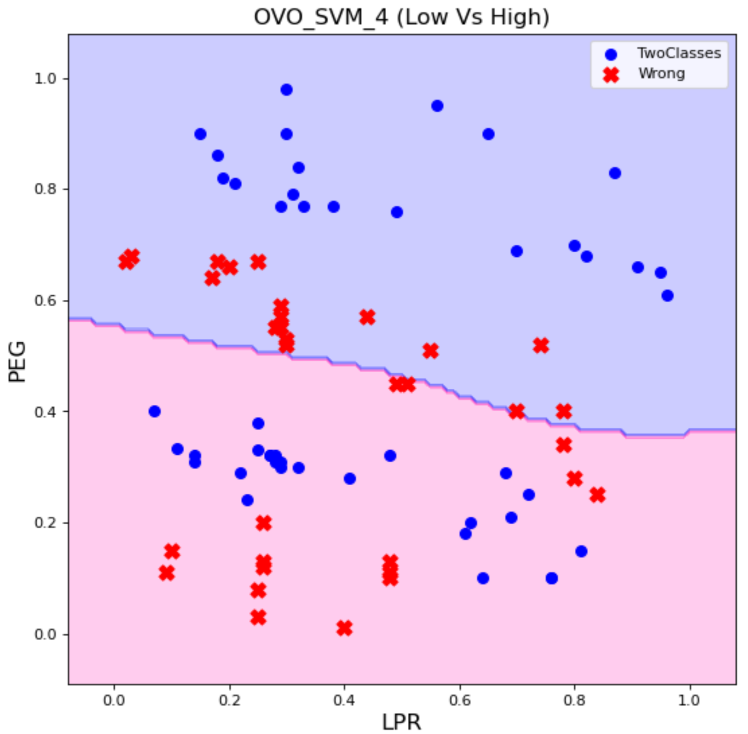
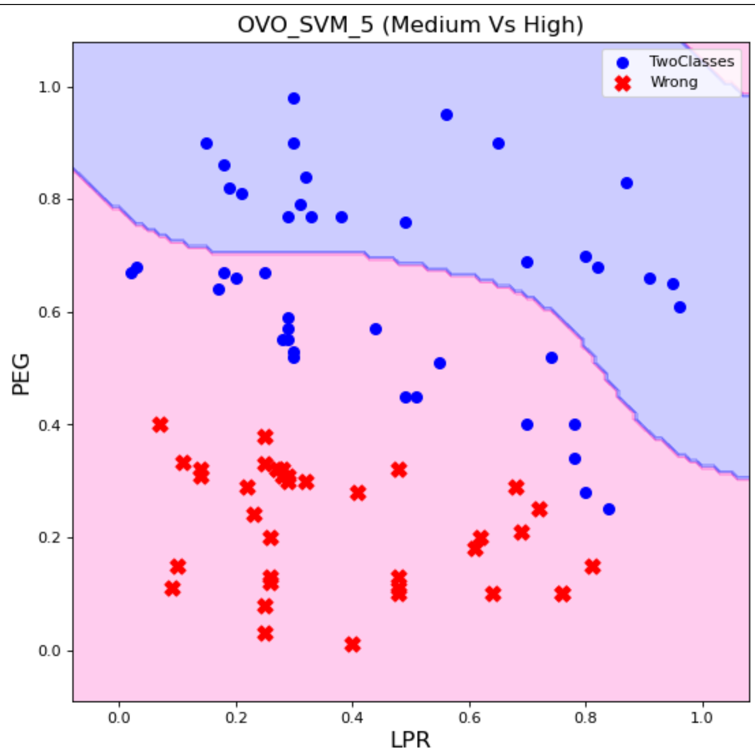
Very Low -> 0, Low -> 1, Medium -> 2, High -> 3

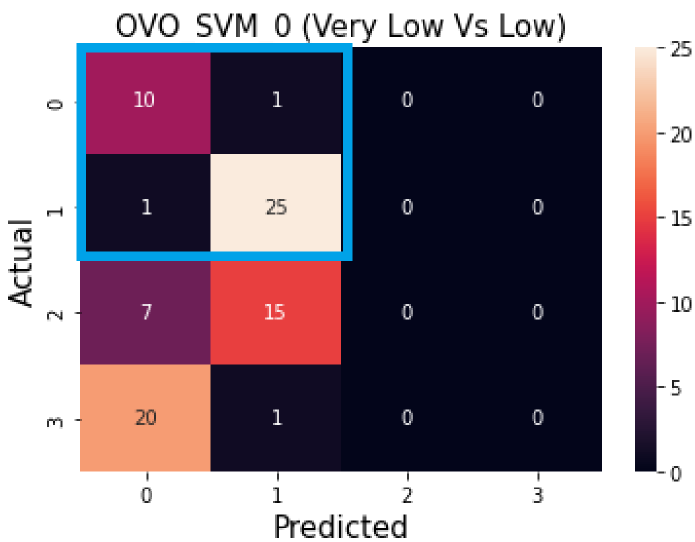
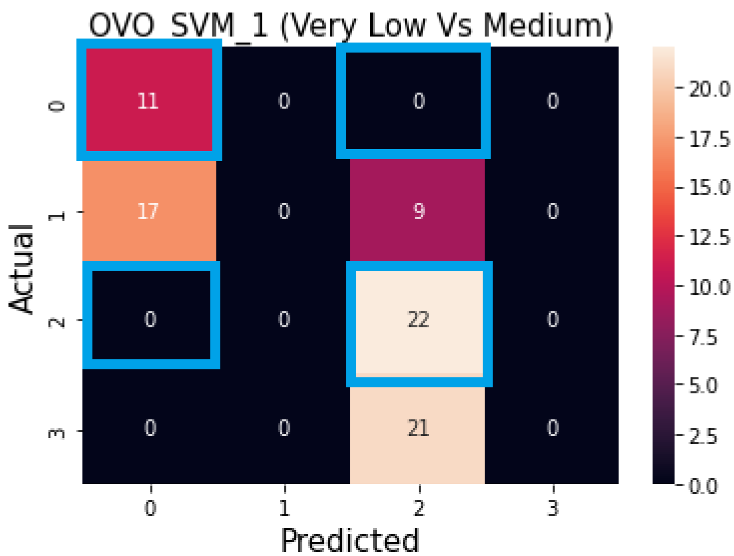
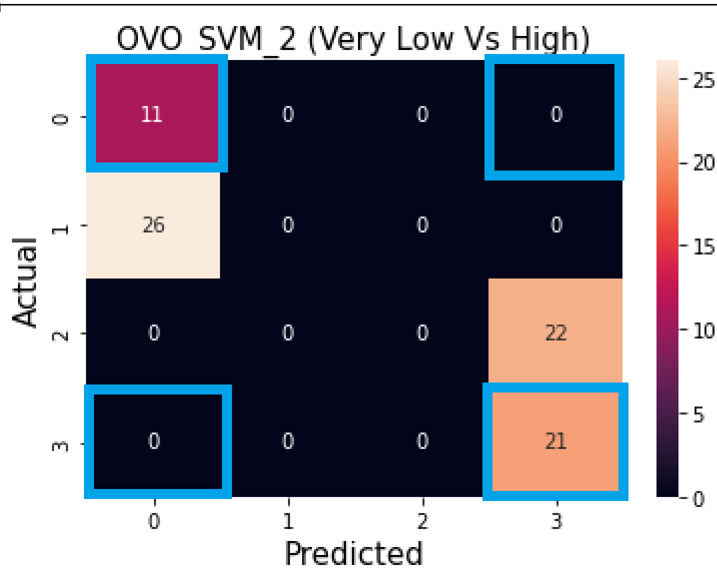
Y_train for each of 6 models.

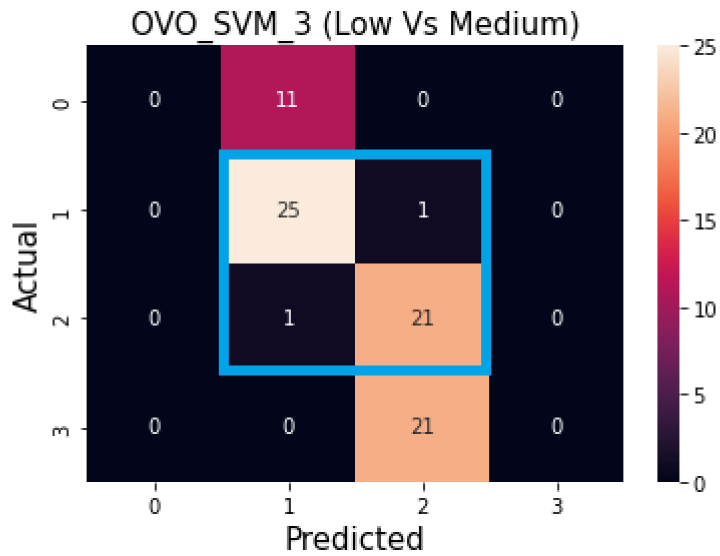
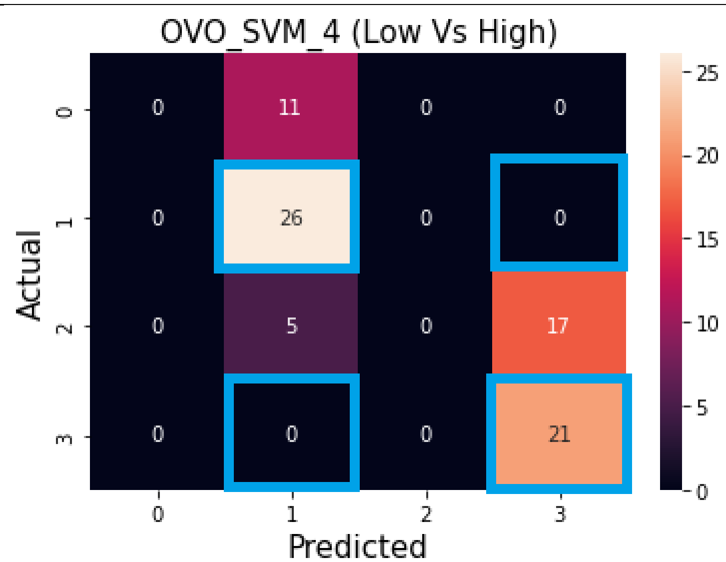
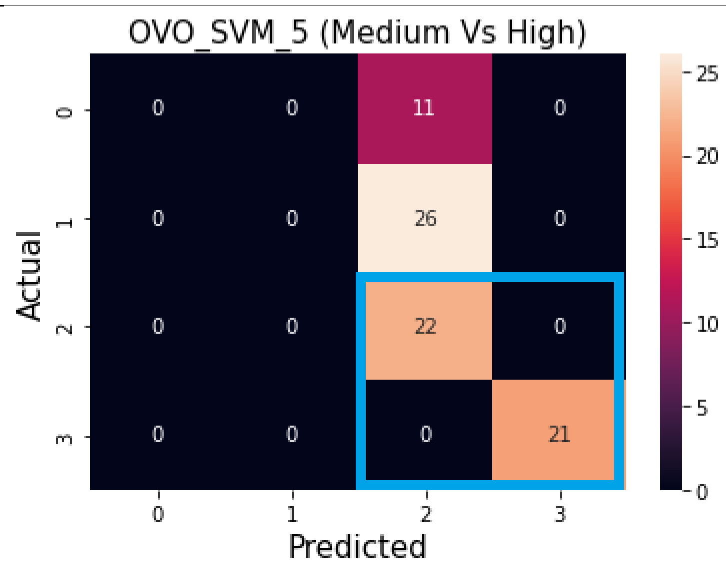
Index	OvO_DF_0_TrainY - Se UNS	OvO_DF_1_TrainY - UNS	OvO_DF_2_TrainY - UNS	OvO_DF_3_TrainY - UNS	OvO_DF_4_TrainY - UNS	OvO_DF_5_TrainY - Series UNS
0	0	0	0	2	3	3
1	1	2	3	1	1	2
2	1	2	3	2	1	2
3	1	2	3	2	3	2
4	0	2	3	1	3	3
5	1	0	0	1	1	3
6	1	2	3	2	3	2
7	1	2	0	1	1	3
8	1	2	3	1	1	2
9	1	0	3	2	1	2
10	0	2	0	1	1	2
11	1	0	3	2	1	3

Classes	SVM's accuracies on test	SVM's decision boundary
Very Low Vs Low	94.5 for the two classes only And 43.75 (Compared to all classes)	
Very Low Vs Medium	100 for the two classes only And 41.25 (Compared to all classes)	

Very low Vs High	**100 for the two classes only** **And** **40.0 (Compared to all classes)**	
Low Vs Medium	**95.8 for the two classes only** **And** **57.49 (Compared to all classes)**	

Low Vs High	**100 for the two classes only** **And** **58.75 (Compared to all classes)**	
Medium Vs High	**100 for the two classes only** **And** **53.75 (Compared to all classes)**	

<table>
<tr><th>Classes</th><th>SVM’s Confusion matrix and comments</th><th></th></tr>
<tr><td>Very Low Vs Low</td><td>
</td><td>This model correctly classifies Very low class from low class correctly.
And there are 2 wrong points.</td></tr>
<tr><td>Very Low Vs Medium</td><td>
</td><td>This model correctly classifies Very low class from medium class correctly.
And there are no wrong points detected.</td></tr>
<tr><td>Very low Vs High</td><td>
</td><td>This model correctly classifies Very low class from high class correctly.
And there are no wrong points detected.</td></tr>
</table>

<table>
<tr><td>Low Vs Medium</td><td>
</td><td>This model correctly classifies low class from medium class correctly.
And there are 2 wrong points.</td></tr>
<tr><td>Low Vs High</td><td>
</td><td>This model correctly classifies low class from high class correctly.
And there are no wrong points detected.</td></tr>
<tr><td>Medium Vs High</td><td>
</td><td>This model correctly classifies medium class from high class correctly.
And there are no wrong points detected.</td></tr>
</table>

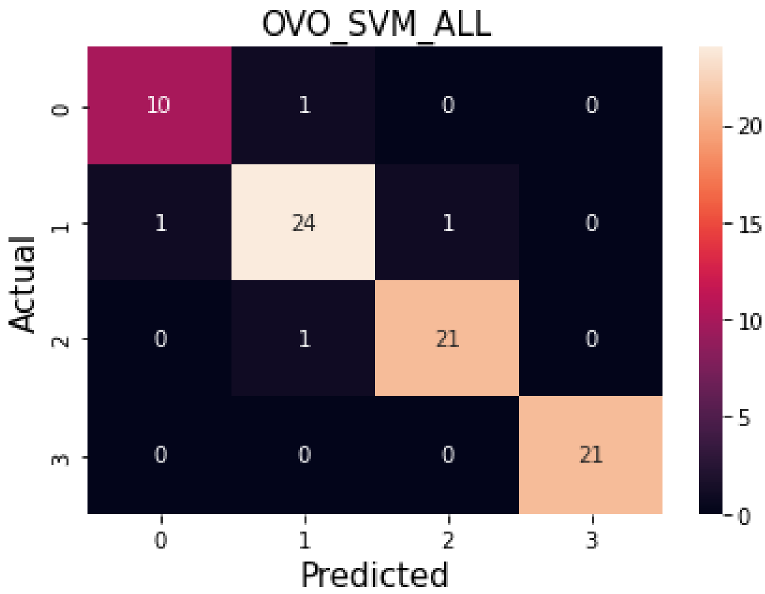
1. (b) **OvO Overall accuracy (after aggregation) is = 95.0**
   We applied numpy.argmax() to these probabilities.

	VeryLow_Aggregation -	Low_Aggregation - Nu	Medium_Aggregation -	High_Aggregation - NumPy object array
	0	0	0	0
0	0.107213	0.309074	0.936275	0.647438
1	0.0652639	0.592218	0.998943	0.343575
2	0.09403	0.273229	0.975206	0.657535
3	0.937252	0.716295	0.333967	0.012486
4	0.203998	0.14569	0.693825	0.956487
5	0.216133	0.144854	0.643092	0.995921
6	0.0699496	0.550776	0.985792	0.393482
7	0.216478	0.153331	0.650732	0.97946
8	0.410991	0.993057	0.559245	0.0367066
9	0.0896021	0.288878	0.995859	0.625661
10	0.0627806	0.533418	0.993407	0.410395

To obtain this (OvO Y_Predict) and compare it with Y_Actual.

YProbALL_OvO - NumPy object array

	0
0	2
1	2
2	2
3	0
4	3
5	3
6	2
7	3
8	1
9	2
10	2



There are 4 wrong points predicted.

4. (a) Conclusion

We have learned many new things during this assignment, and we have discovered some useful techniques like OvR and OvO. We have gotten familiar with new libraries. We have learnt how to select features based on PairPlot and discover which features is more important.

Now, we can say that we are capable of dealing with different types of SVM (We have tried rbf and linear) and we discover that the Perceptron algorithm is based on neural network.

We learnt how to divide a multi-classification problem into small Binary classification problems by using OVO and OVR techniques.

Those approaches divide a big classification problem into small ones by changing the labels of the target variables in the train dataset for each of binary classification problem (for OvR) and train one model on each dataset, for instance **->** 1 for Very low class and 0 for others and so on….

Or by splitting our train dataset into different datasets for each model to train on it (for OvO) the number of binary datasets will be equal to (NumClasses * (NumClasses – 1)) / 2.

After that, we aggregate our results to obtain final accuracy for OvR and OvO and, we have obtained an accuracy of 96.25 from OvR which is higher then the normal SVM (rbf) approach that we tried in point number 1 (c).