

ELG5255 Applied Machine Learning

Group Assignment 1

Group 4

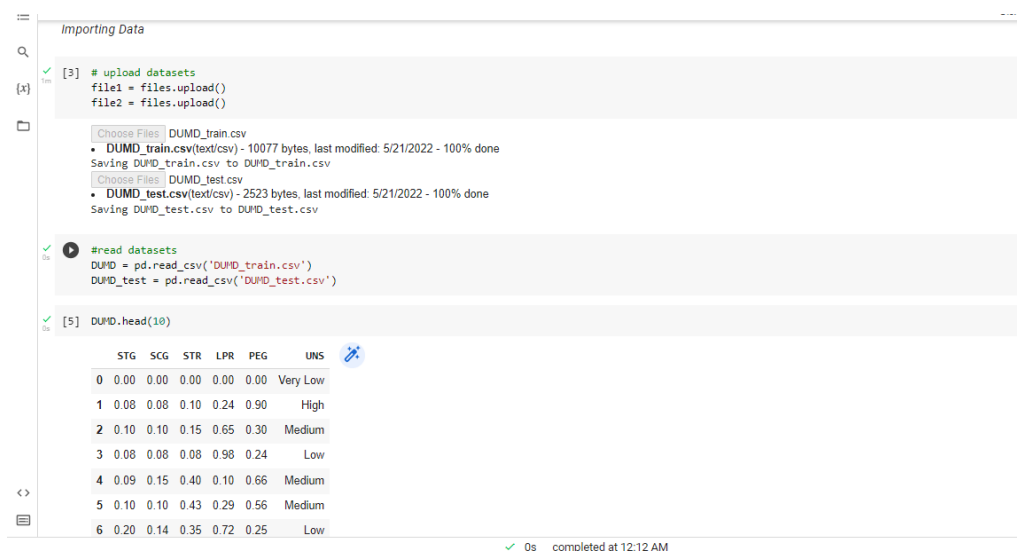
- | | |
|--|-----------|
| 1. Yomna Mohamed Sayed Ahmed ---- | 300327217 |
| 2. Khaled Mohamed Mohamed Mahmoud ---- | 300327242 |
| 3. Marwa Hamdi Boraie Mahmoud ---- | 300327267 |

Goal :

→ Our goal is to build model give me high performance and accuracy and make enhance to the model by using more than 1 algorism and make aggregation to the prediction and to improve the model.

Dataset :

1. The data is coming cleaned and was made a process on it and we chose the best features from the data to made with it directly with model.



The screenshot shows a Jupyter Notebook interface with three code cells. The first cell uploads two CSV files: 'DUMD_train.csv' (10077 bytes) and 'DUMD_test.csv' (2523 bytes). The second cell reads these files into pandas DataFrames named 'DUMD' and 'DUMD_test'. The third cell displays the first 10 rows of the 'DUMD' dataset using 'DUMD.head(10)'. The output is a table with 7 columns: STG, SCG, STR, LPR, PEG, UNS, and a categorical label. The labels are 'Very Low', 'High', 'Medium', 'Low', 'Medium', 'Medium', and 'Low' for rows 0 through 6 respectively.

```
[3] # upload datasets
file1 = files.upload()
file2 = files.upload()

Choose Files  DUMD_train.csv
• DUMD_train.csv(text/csv) - 10077 bytes, last modified: 5/21/2022 - 100% done
Saving DUMD_train.csv to DUMD_train.csv
Choose Files  DUMD_test.csv
• DUMD_test.csv(text/csv) - 2523 bytes, last modified: 5/21/2022 - 100% done
Saving DUMD_test.csv to DUMD_test.csv

[4] #read datasets
DUMD = pd.read_csv('DUMD_train.csv')
DUMD_test = pd.read_csv('DUMD_test.csv')

[5] DUMD.head(10)
```

	STG	SCG	STR	LPR	PEG	UNS	
0	0.00	0.00	0.00	0.00	0.00	Very Low	
1	0.08	0.08	0.10	0.24	0.90	High	
2	0.10	0.10	0.15	0.65	0.30	Medium	
3	0.08	0.08	0.08	0.98	0.24	Low	
4	0.09	0.15	0.40	0.10	0.66	Medium	
5	0.10	0.10	0.43	0.29	0.56	Medium	
6	0.20	0.14	0.35	0.72	0.25	Low	

0s completed at 12:12 AM

2. Dataset is already divided into the training and testing and that helps to could make detection the data belong to class (0,1,2,3) (High, Low, Medium and Very Low).

→ Represent the data to see the features and labels.

```
X_train0 = X_train0.loc[X_train0['UNS'] == 0]
X_train1 = X_train1.loc[X_train1['UNS'] == 1]
X_train2 = X_train2.loc[X_train2['UNS'] == 2]
X_train3 = X_train3.loc[X_train3['UNS'] == 3]

X_test0 = X_test0.loc[X_test0['UNS'] == 0]
X_test1 = X_test1.loc[X_test1['UNS'] == 1]
X_test2 = X_test2.loc[X_test2['UNS'] == 2]
X_test3 = X_test3.loc[X_test3['UNS'] == 3]

Train_Set
```

	LPR	PEG	UNS
0	0.00	0.00	3
1	0.24	0.90	0
2	0.65	0.30	2
3	0.98	0.24	1
4	0.10	0.66	2
...
318	0.32	0.89	0
319	0.83	0.83	0
320	0.13	0.32	1
321	0.57	0.57	2
322	0.97	0.24	2

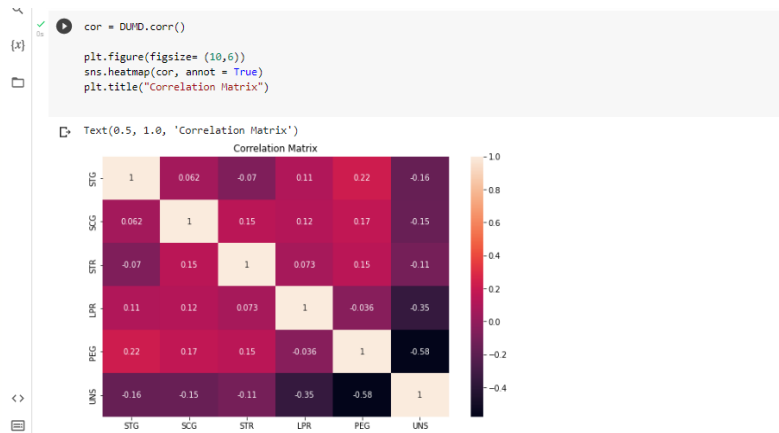
323 rows x 3 columns

0s completed at 12:12 AM

Using Correlation Coefficient to choose the best features:

Correlation is a measure of the linear relationship of 2 or more variables. Through correlation, we can predict one variable from the other. The logic behind using correlation for feature selection is that the good variables are highly correlated with the target. Furthermore, variables should be correlated with the target but should be uncorrelated among them. If two variables are correlated, we can predict one from the other. Therefore, if two features are correlated, the model only really needs one of them, as the second one does not add additional information. We will use the Pearson Correlation here. We need to set an absolute value, say 0.5 as the threshold for selecting the variables. If we find that the predictor variables are correlated among themselves, we can drop the variable which has a lower correlation coefficient value with the target variable. We can also compute multiple correlation coefficients to check whether more than two variables are correlated to each other. This phenomenon is known as multicollinearity.

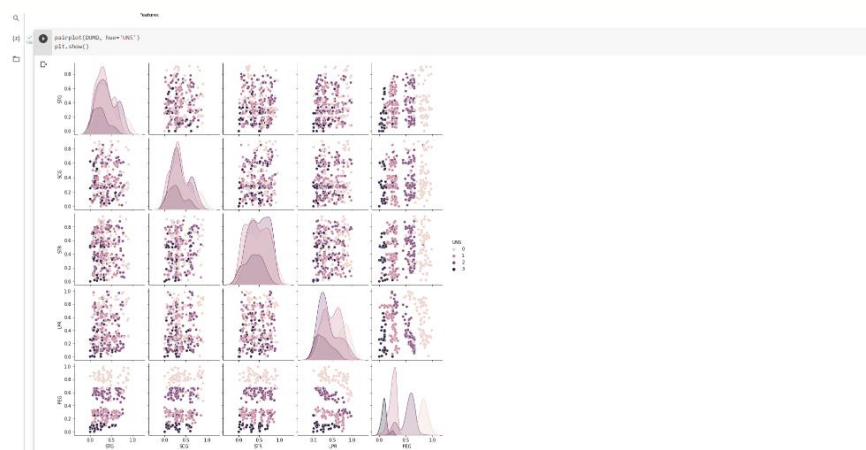
Finally we chose the last 2 features (PEG, LPR) as shown below.



Using f_classif : ANOVA F_value between label/feature for Classification tasks



→ We used Pairplot visualizes for given data to find the relationship between them where the variables can be continuous or categorical.



→ We make some function that will use it to make visualization to our model.

→ First model work on data binary after remove the first 3 features, compare performance of Perceptron and SVM on testing set and get the confusion matrix and classification report.

SVM Model

Output is: the accuracy of the model is 91.25% that means the model is predicted the most of values correctly.

Code and output:

```
Q
(x)
SVM
data_train=pd.DataFrame({'LPR':DUMD['LPR'],'PEG':DUMD['PEG']})
target_train=pd.DataFrame({'UNS':DUMD['UNS']})
X_train = data_train
Y_train= target_train

[ ] data_test=pd.DataFrame({'LPR':DUMD_test['LPR'],'PEG':DUMD_test['PEG']})
target_test=pd.DataFrame({'UNS':DUMD_test['UNS']})
X_test = data_test
Y_test= target_test

[ ] SVM_model = svm.SVC(kernel='linear', probability=True)
SVM_model.fit(X_train, Y_train)
y_predict_svm = SVM_model.predict(X_test)
evaluation_svm = accuracy_score(Y_test, y_predict_svm)

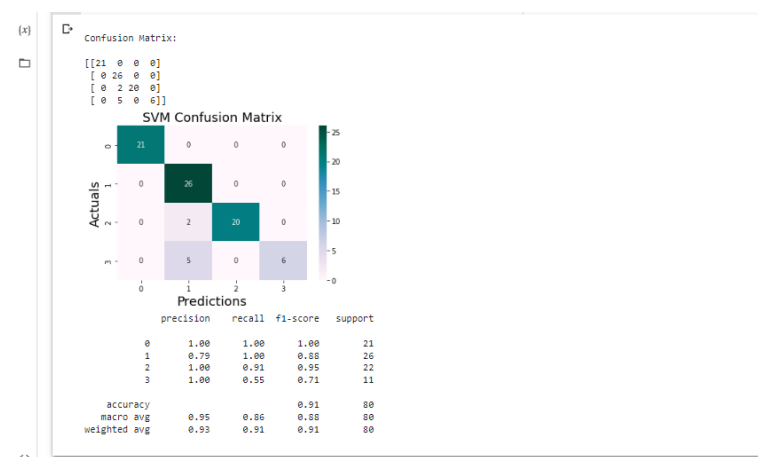
evaluation_svm

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for
y = column_or_1d(y, warn=True)
0.9125

[ ] print('\nConfusion Matrix:\n')
print(confusion_matrix(Y_test, y_predict_svm))
ax = sns.heatmap(confusion_matrix(Y_test,y_predict_svm ), annot=True, cmap='PuBuGn')
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('SVM Confusion Matrix', fontsize=18)
plt.show()

print(classification_report(Y_test,y_predict_svm))

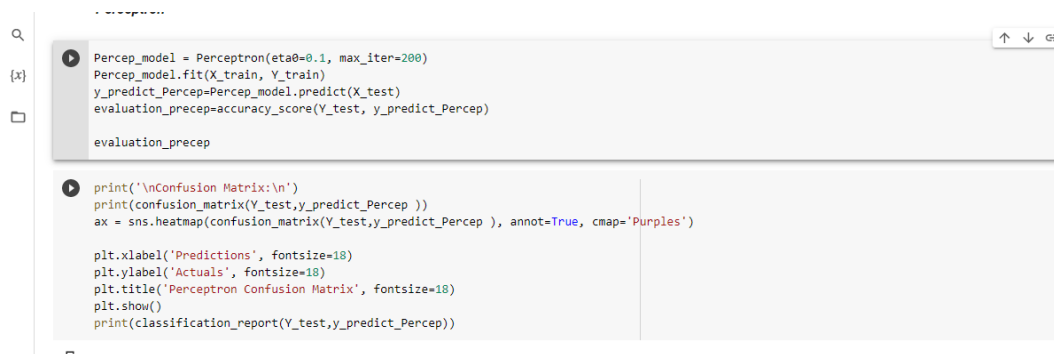
0s completed at 10:27 PM
```



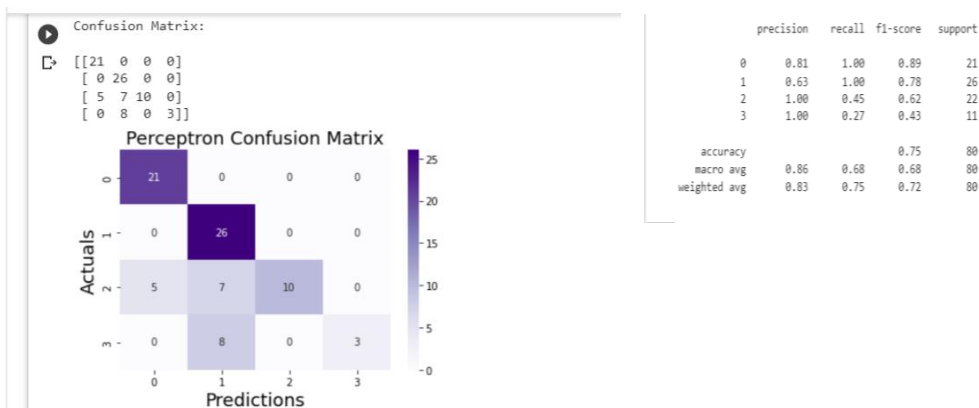


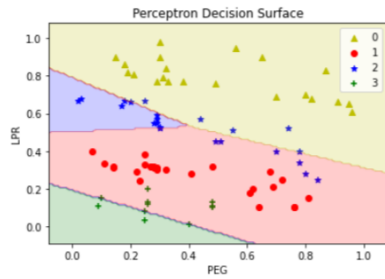
Perceptron Model:

We work on the perceptron to get the accuracy of the model to make classification in classes 0,1,2,3 (High, Low, Medium, Very Low) the model give us accuracy 75%



Output of the model:





OVR SVM:

Made binarize labels for Train labels and Test labels by MultiLabelBinarizer class using that Obtain the binarized labels as shown :

```

• Binarize y train and y test

[ ] mlb = MultiLabelBinarizer()
    yb_train = mlb.fit_transform(Y_TRAIN.reshape((-1,1)))
    yb_test = mlb.fit_transform(Y_TEST.reshape((-1,1)))

[ ] yb1_train = yb_train[:,0]
    yb2_train = yb_train[:,1]
    yb3_train = yb_train[:,2]
    yb4_train = yb_train[:,3]

    yb1_test = yb_test[:,0]
    yb2_test = yb_test[:,1]
    yb3_test = yb_test[:,2]
    yb4_test = yb_test[:,3]

```

- SVM First Classifier Plot decision boundary and Confusion Matrix and Accuracy: 100.00% we get the accuracy of the model and compare with other data.

```

• model 1

[ ] clf_1 = svm.SVC(kernel='linear', probability=True)
    clf_1.fit(X_train, yb1_train)
    y1_pred = clf_1.predict(X_test)
    print('Accuracy of clf_1: {:.2f}%'.format(getAccuracy(clf_1, X_test, yb1_test)))
    y1_pred_proba = clf_1.predict_proba(X_test)[1,:].reshape(-1,1)
    print('\nConfusion Matrix OVR:\n')
    confusion_matrix_binary_classes(yb1_test, y1_pred)
    plot_binary_classifier(X_test, yb1_test, clf_1)

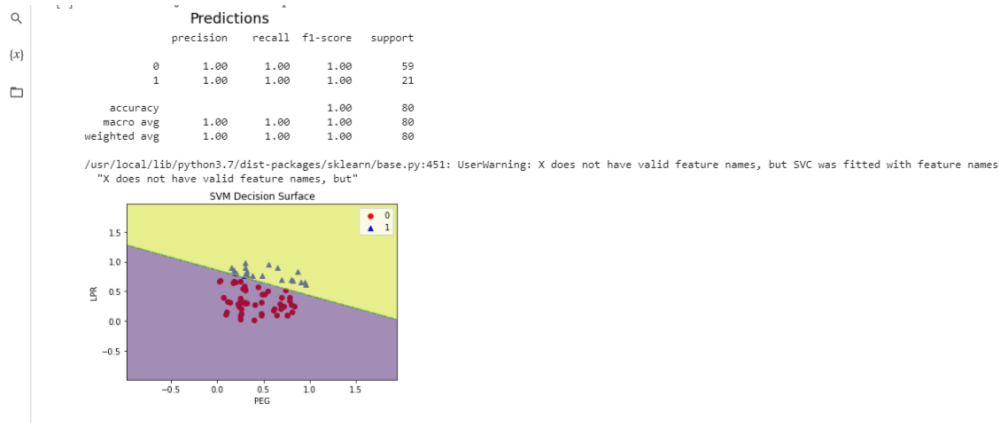
[ ] Accuracy of clf_1: 100.00%

Confusion Matrix OVR:
[[59  0]
 [ 0 21]]

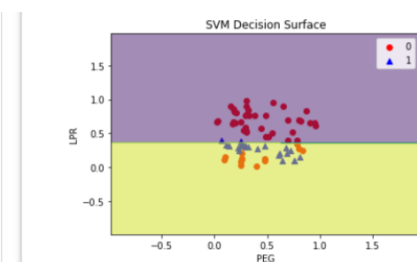
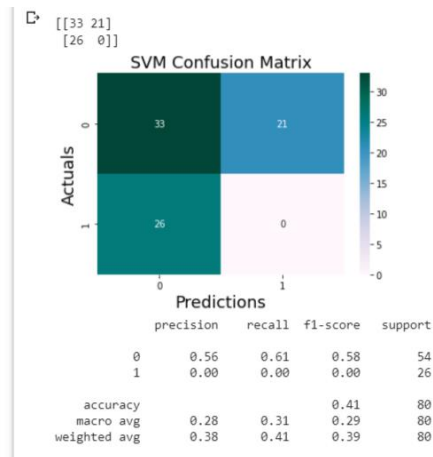
SVM Confusion Matrix

```

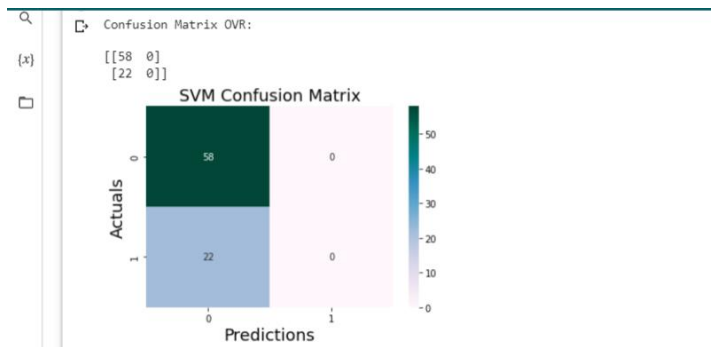
The SVM Confusion Matrix heatmap shows the results of the SVM classifier. The x-axis is 'Predictions' (0, 1) and the y-axis is 'Actuals' (0, 1). The matrix values are: True Positives (TP) = 59, True Negatives (TN) = 21, False Positives (FP) = 0, and False Negatives (FN) = 0. The color scale ranges from 0 (light pink) to 50 (dark green).



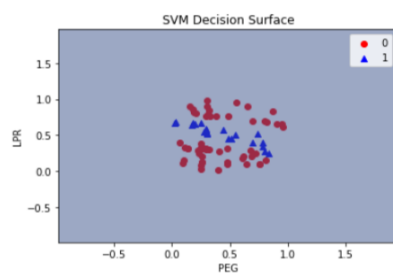
- SVM Second Classifier Plot decision boundary and Confusion Matrix and Accuracy: 80.00%



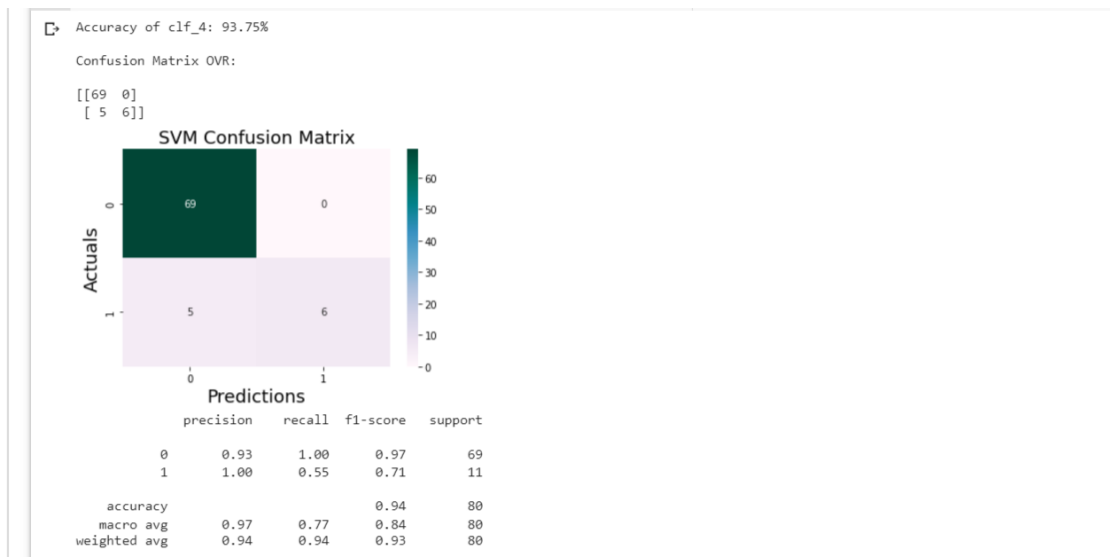
- SVM Third Classifier Plot decision boundary and Confusion Matrix and Accuracy: 72.50%

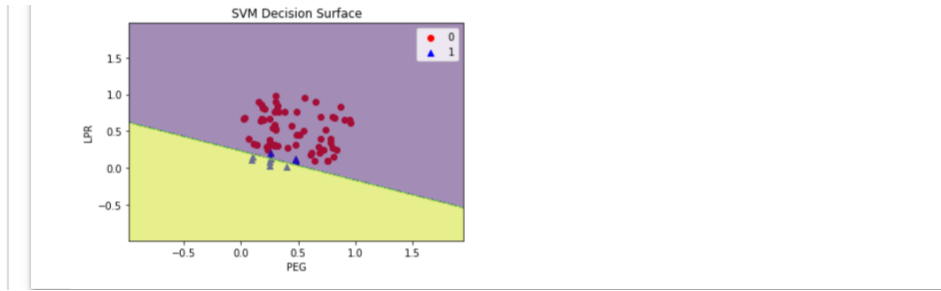


	precision	recall	f1-score	support
0	0.72	1.00	0.84	58
1	0.00	0.00	0.00	22
accuracy			0.73	80
macro avg	0.36	0.50	0.42	80
weighted avg	0.53	0.72	0.61	80

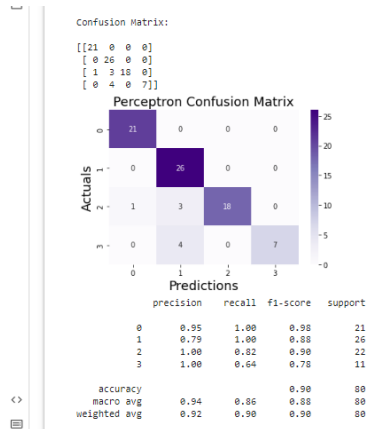


- SVM fourth Classifier Plot decision boundary and Confusion Matrix and Accuracy: 93.75%





→The final model accuracy is : 90%



OVO

Partition labeled oneVsOne data and binarize them for trianing

```

X1_train, X2_train, X3_train, X4_train, X5_train, X6_train, = [], [], [], [], [], []
y1_train, y2_train, y3_train, y4_train, y5_train, y6_train, = [], [], [], [], [], []

for i in range(len(X_train)):
    # (0,1) if the target labels are 0 or 1
    if y_train[i] == 0 or y_train[i] == 1:
        X1_train.append(X_train[i])
        y1_train.append(y_train[i])

    # (0, 2)
    if y_train[i] == 0 or y_train[i] == 2:
        X2_train.append(X_train[i])
        y2_train.append(y_train[i])

    # (0, 3)
    if y_train[i] == 0 or y_train[i] == 3:
        X3_train.append(X_train[i])
        y3_train.append(y_train[i])

    # (1, 2)
    if y_train[i] == 1 or y_train[i] == 2:
        X4_train.append(X_train[i])
        y4_train.append(y_train[i])

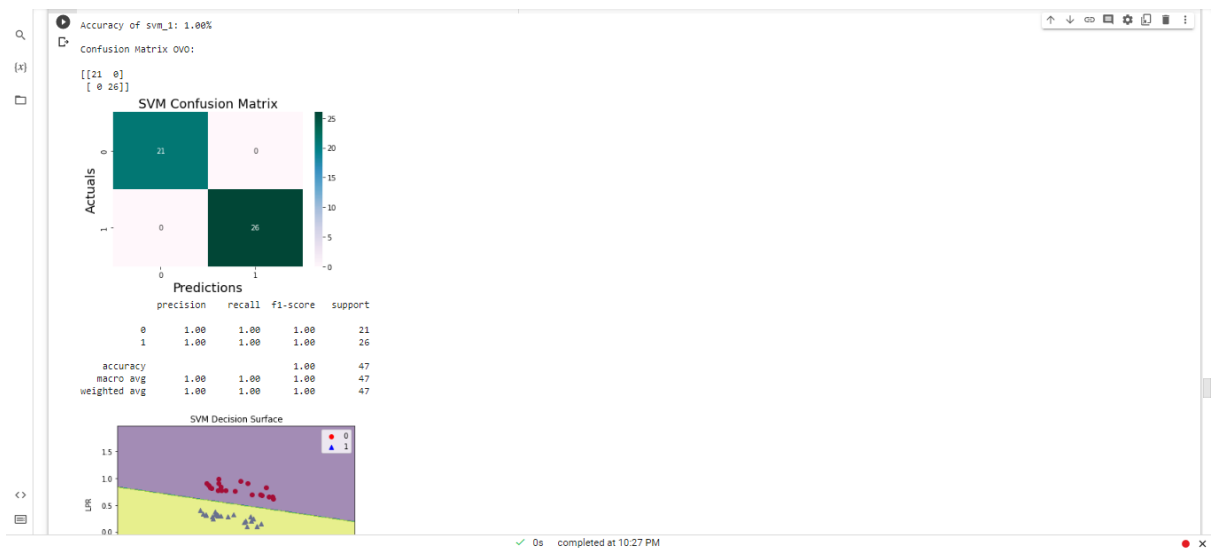
    # (1, 3)
    if y_train[i] == 1 or y_train[i] == 3:
        X5_train.append(X_train[i])
        y5_train.append(y_train[i])

    # (2, 3)
    if y_train[i] == 2 or y_train[i] == 3:
        X6_train.append(X_train[i])
        y6_train.append(y_train[i])

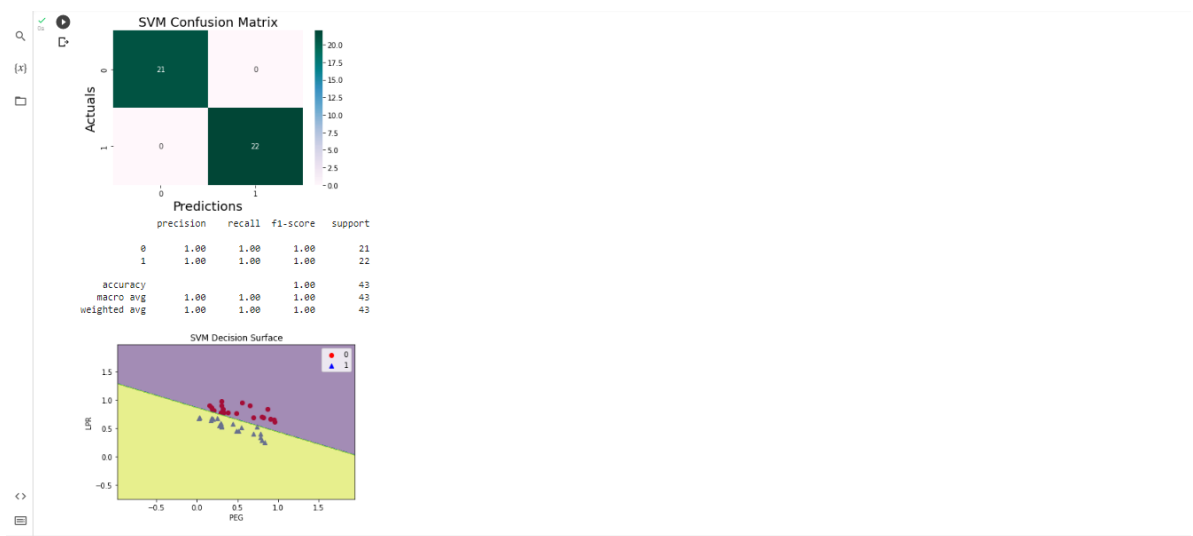
# Binarize the y labels
mlb= MultiLabelBinarizer()
y1_train= mlb.fit_transform(np.array(y1_train).reshape((-1,1)))[:,1]
y2_train= mlb.fit_transform(np.array(y2_train).reshape((-1,1)))[:,1]
y3_train= mlb.fit_transform(np.array(y3_train).reshape((-1,1)))[:,1]
y4_train= mlb.fit_transform(np.array(y4_train).reshape((-1,1)))[:,1]
y5_train= mlb.fit_transform(np.array(y5_train).reshape((-1,1)))[:,1]
y6_train= mlb.fit_transform(np.array(y6_train).reshape((-1,1)))[:,1]

```

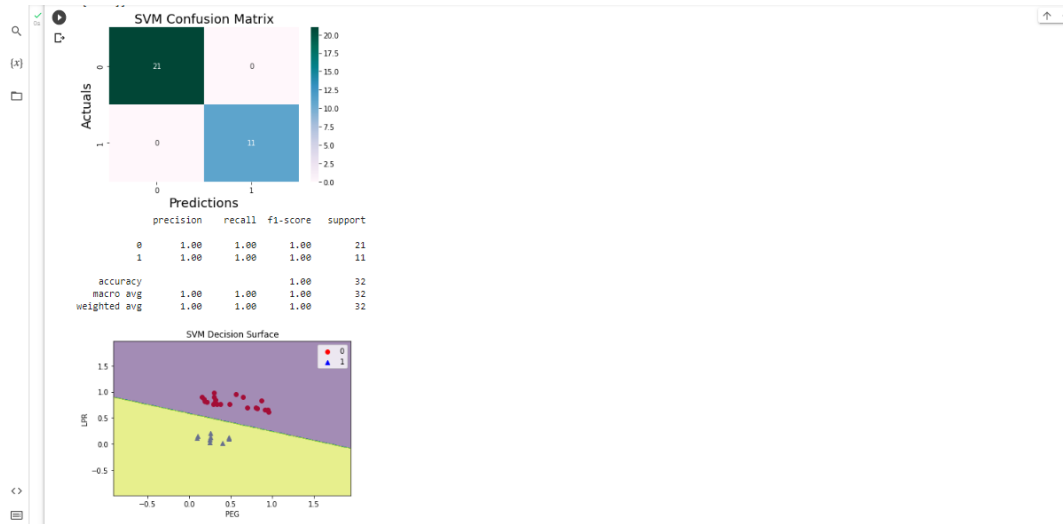
- SVM First Classifier Model (0 , 1) Plot decision boundary and Confusion Matrix and Accuracy: 100.00%



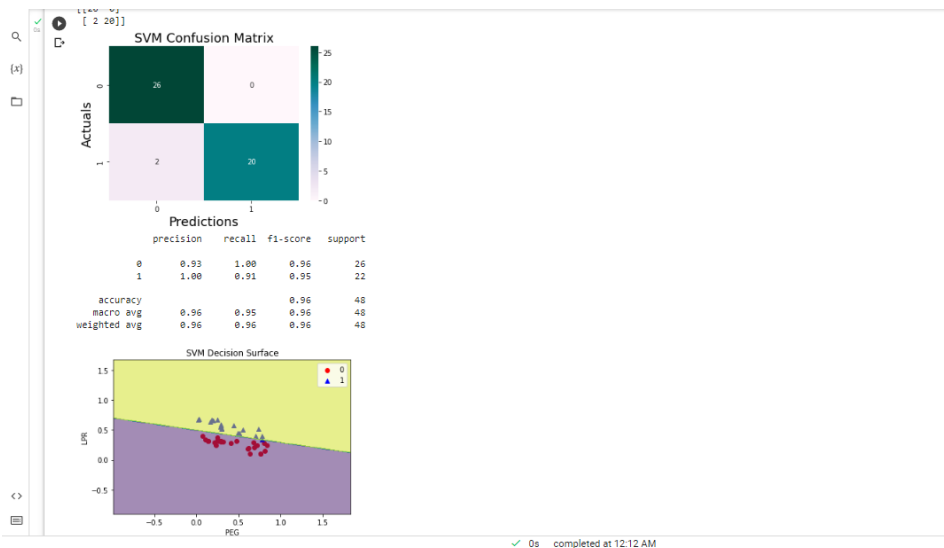
- SVM First Classifier Model (0 , 2) Plot decision boundary and Confusion Matrix and Accuracy: 100.00%



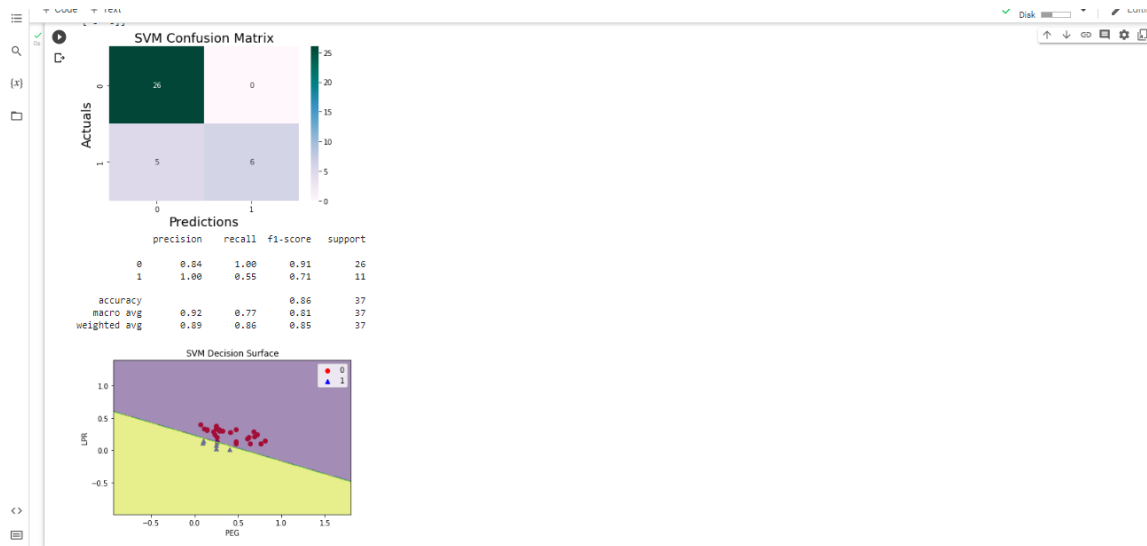
- SVM First Classifier Model (0 , 3) Plot decision boundary and Confusion Matrix and Accuracy: 100.00%



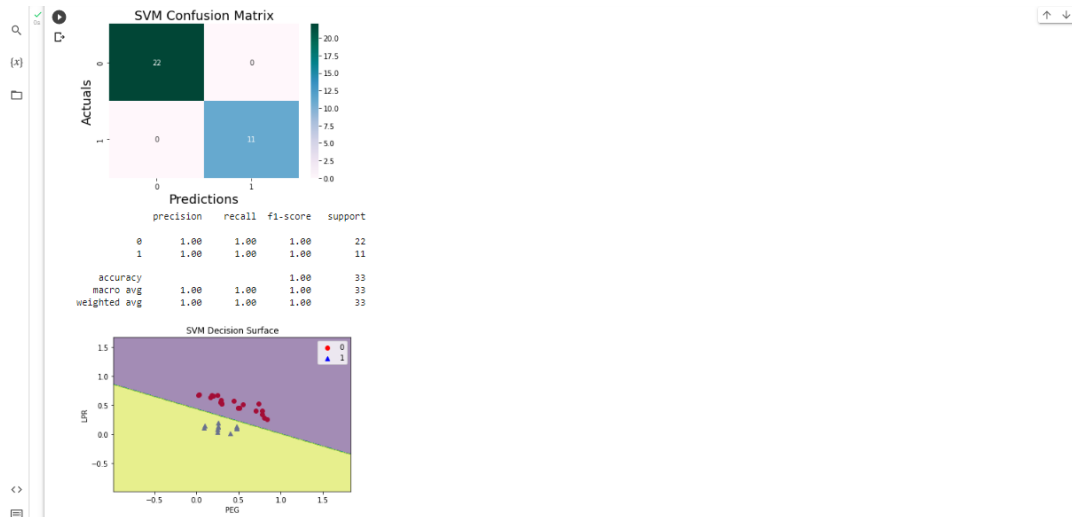
- SVM First Classifier Model (1 , 2) Plot decision boundary and Confusion Matrix and Accuracy: 95 . 83%



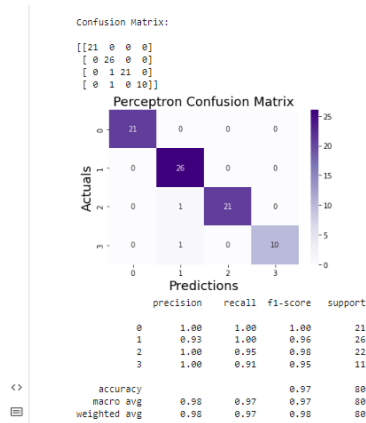
- SVM First Classifier Model (1 , 3) Plot decision boundary and Confusion Matrix and Accuracy: 86 . 49%



- SVM First Classifier Model (2 , 3) Plot decision boundary and Confusion Matrix and Accuracy 100.00%



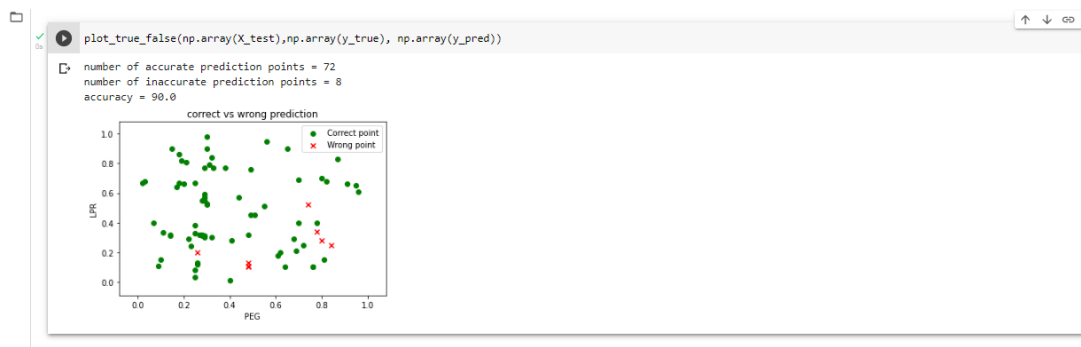
→The final model accuracy is : 97%



Using Argmax for Aggregation:

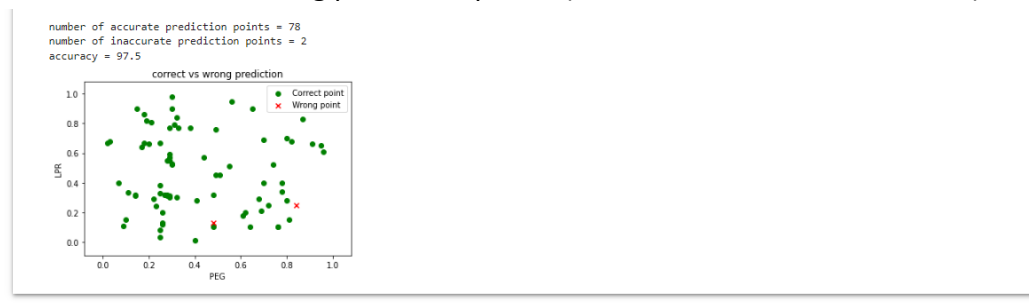
Argmax OVR Result is:

- Accuracy: 90.0
- Correct and wrong prediction points plot as Shown:



Argmax OVO Result is:

- Accuracy: 97.5
- Correct and wrong prediction points (Matched= 78 ,Not Matched=2) as shown :



→As shown above performance of OVO is better than OVR in using Argmax as An aggregation method.

Conclusion:

- Compare between **SVM** model accuracy and **Perceptron** model accuracy. Some models can give good prediction in the data and other models give less prediction, in our data SVM gives us better prediction (91%) than perceptron (75%)
- **OvR Model** : We learned how to make binary classifier on data to build model that make classifications on our labeled data and get different prediction .
- **Argmax** : Help us to select the label that has the highest confidence score.
- **Aggregation Strategy** : For OVR simply we get the final class label with the highest probability using argmax. For OVO we get the labels by calculating the mean of the probabilities for each class and selecting the highest one using argmax
- **OVO vs OVR**: OVR is simpler and uses lower computational power compared with OVO. OVO has higher accuracy (97.5%) compared with OVR (90.0%) but it consumes a lot of computational power and usually has models to train more than OVR.