ELG 5255: Applied Machine Learning

Assignment 1

By: Group 13

Kishita Pakhrani

David Talson

Question 1

Answer:

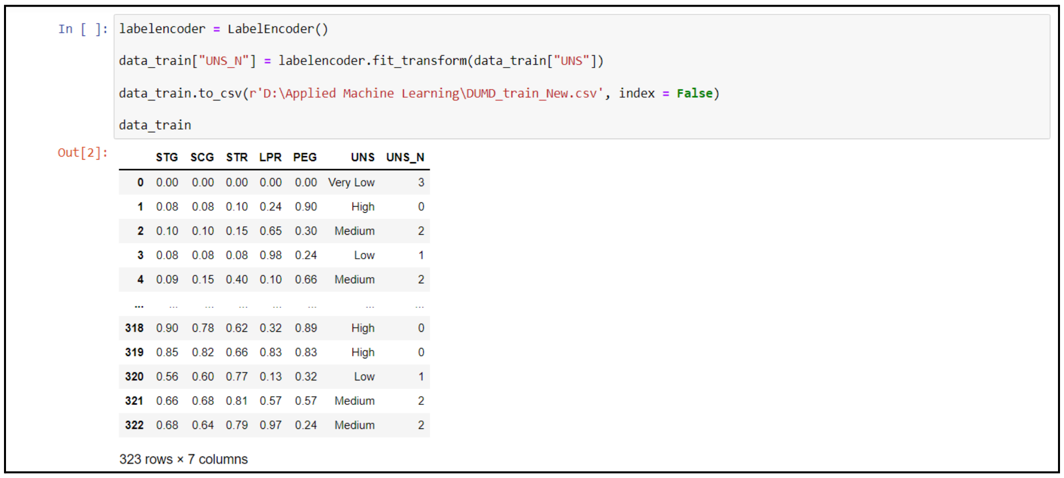
(A) We convert the categorical class labels under the “UNS” column to numerical values and save them in a new column called “UNS_N” by using LabelEncoder.

Here, 0 = High

1 = Low

2 = Medium

3 = Very Low



```
In [ ]: labelencoder = LabelEncoder()

        data_train["UNS_N"] = labelencoder.fit_transform(data_train["UNS"])

        data_train.to_csv(r'D:\Applied Machine Learning\DUMD_train_New.csv', index = False)

        data_train
```

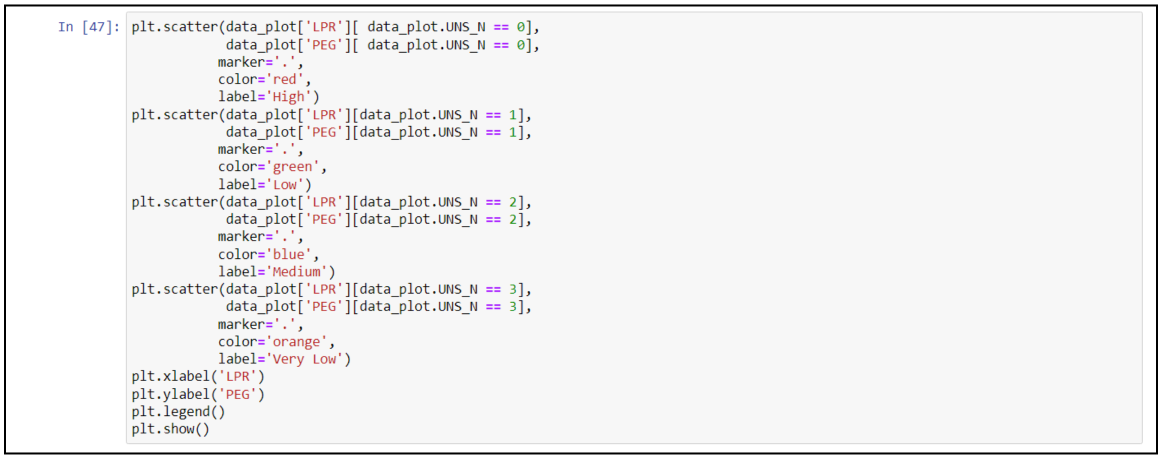
```
Out[2]:
```

	STG	SCG	STR	LPR	PEG	UNS	UNS_N
0	0.00	0.00	0.00	0.00	0.00	Very Low	3
1	0.08	0.08	0.10	0.24	0.90	High	0
2	0.10	0.10	0.15	0.65	0.30	Medium	2
3	0.08	0.08	0.08	0.98	0.24	Low	1
4	0.09	0.15	0.40	0.10	0.66	Medium	2
...	...	...	...	...	...	...	...
318	0.90	0.78	0.62	0.32	0.89	High	0
319	0.85	0.82	0.66	0.83	0.83	High	0
320	0.56	0.60	0.77	0.13	0.32	Low	1
321	0.66	0.68	0.81	0.57	0.57	Medium	2
322	0.68	0.64	0.79	0.97	0.24	Medium	2

323 rows × 7 columns

Fig 1 . LabelEncoder Code

(B) The two features that we chose are LPR and PEG. Looking at other features like STG and SCG we found out that in some cases there are the same values for different classes. This can confuse the model and will provide less accuracy. LPR and PEG had different values each time and by increasing one or the other there is a direct impact on the class label. Thus, we chose LPR and PEG as our two features for classification.



```
In [47]: plt.scatter(data_plot['LPR'][ data_plot.UNS_N == 0],
                     data_plot['PEG'][ data_plot.UNS_N == 0],
                    marker='.',
                    color='red',
                    label='High')
         plt.scatter(data_plot['LPR'][data_plot.UNS_N == 1],
                     data_plot['PEG'][data_plot.UNS_N == 1],
                    marker='.',
                    color='green',
                    label='Low')
         plt.scatter(data_plot['LPR'][data_plot.UNS_N == 2],
                     data_plot['PEG'][data_plot.UNS_N == 2],
                    marker='.',
                    color='blue',
                    label='Medium')
         plt.scatter(data_plot['LPR'][data_plot.UNS_N == 3],
                     data_plot['PEG'][data_plot.UNS_N == 3],
                    marker='.',
                    color='orange',
                    label='Very Low')
         plt.xlabel('LPR')
         plt.ylabel('PEG')
         plt.legend()
         plt.show()
```

Fig 2. Data Plot code

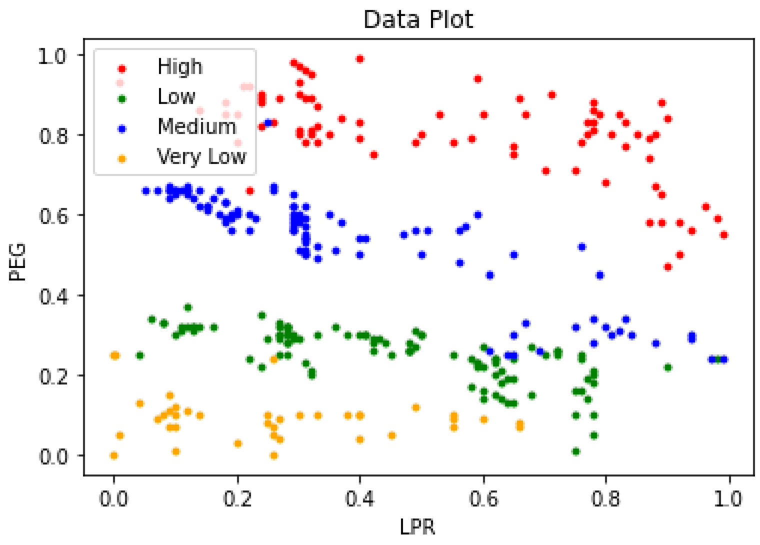
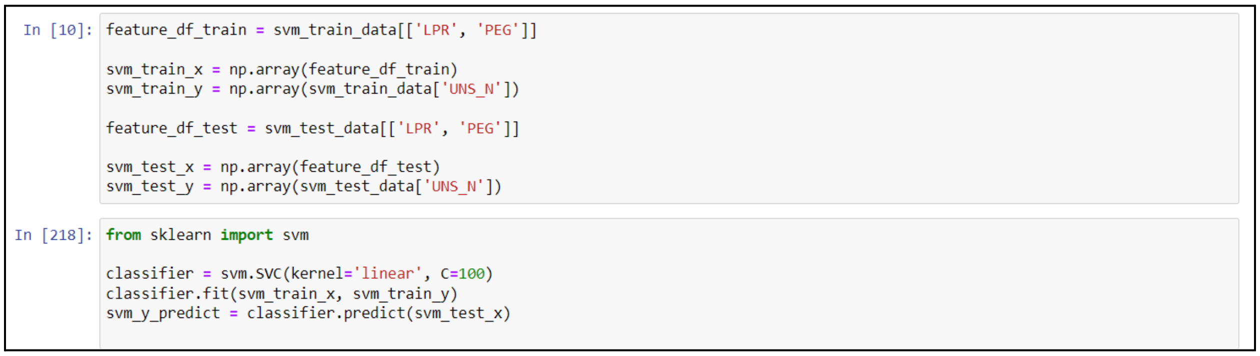


Fig 3. Data Plot Output

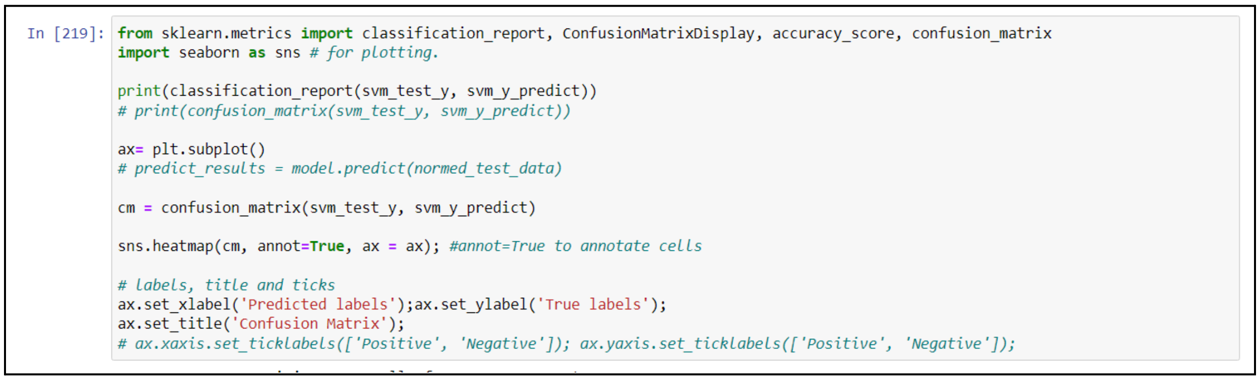
(C) SVM:

The below code snippets show the SVM classifier and accuracy, confusion matrix and decision boundaries for SVM Classifier.



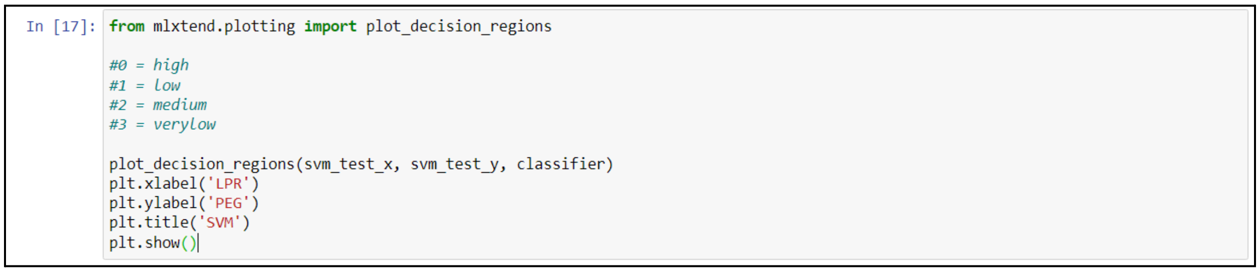
```
In [10]: feature_df_train = svm_train_data[['LPR', 'PEG']]

         svm_train_x = np.array(feature_df_train)
         svm_train_y = np.array(svm_train_data['UNS_N'])

         feature_df_test = svm_test_data[['LPR', 'PEG']]

         svm_test_x = np.array(feature_df_test)
         svm_test_y = np.array(svm_test_data['UNS_N'])

In [218]: from sklearn import svm

          classifier = svm.SVC(kernel='linear', C=100)
          classifier.fit(svm_train_x, svm_train_y)
          svm_y_predict = classifier.predict(svm_test_x)
```

Fig 4. SVM Classifier Code



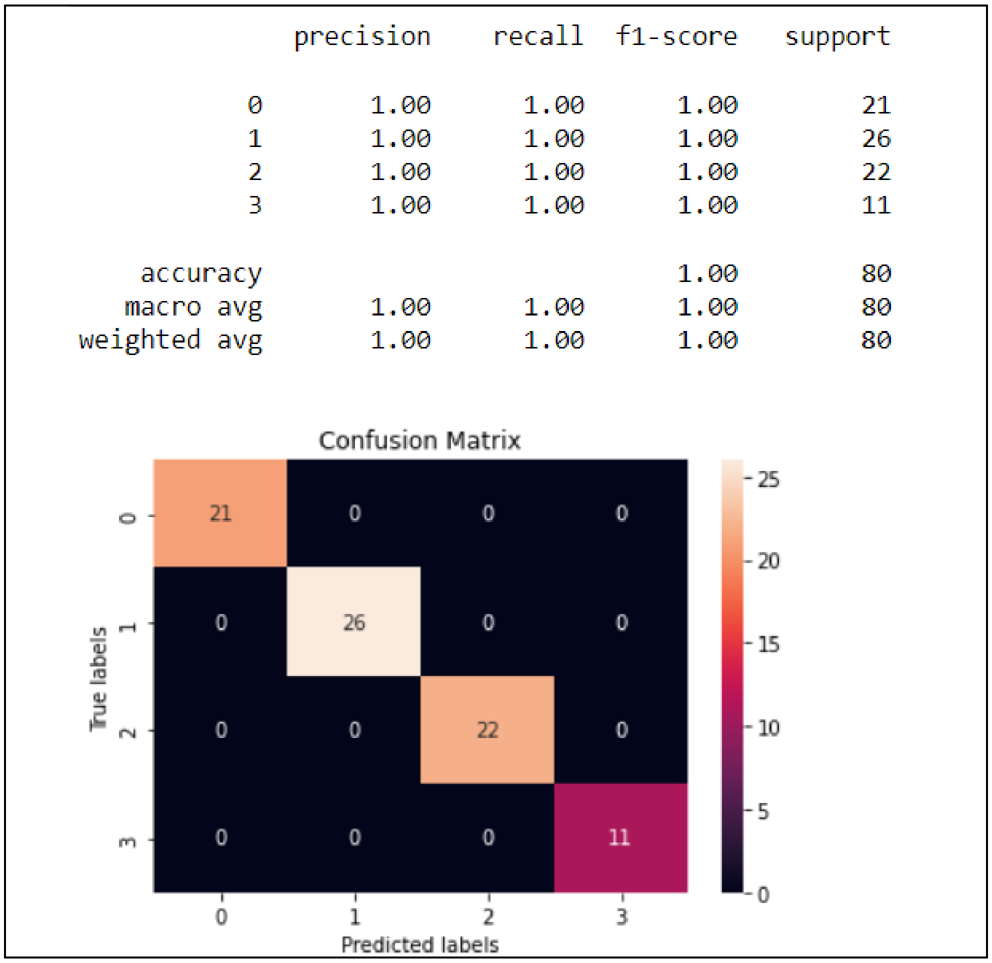
```
In [219]: from sklearn.metrics import classification_report, ConfusionMatrixDisplay, accuracy_score, confusion_matrix
          import seaborn as sns # for plotting.

          print(classification_report(svm_test_y, svm_y_predict))
          # print(confusion_matrix(svm_test_y, svm_y_predict))

          ax= plt.subplot()
          # predict_results = model.predict(normed_test_data)

          cm = confusion_matrix(svm_test_y, svm_y_predict)

          sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

          # labels, title and ticks
          ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
          ax.set_title('Confusion Matrix');
          # ax.xaxis.set_ticklabels(['Positive', 'Negative']); ax.yaxis.set_ticklabels(['Positive', 'Negative']);
```

Fig 5. SVM classification report code



```
In [17]: from mlxtend.plotting import plot_decision_regions

         #0 = high
         #1 = low
         #2 = medium
         #3 = verylow

         plot_decision_regions(svm_test_x, svm_test_y, classifier)
         plt.xlabel('LPR')
         plt.ylabel('PEG')
         plt.title('SVM')
         plt.show()
```

Fig 7. SVM Decision Boundary Plotting Code



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      1.00      1.00        26
           2       1.00      1.00      1.00        22
           3       1.00      1.00      1.00        11

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80
```

Fig 6. Accuracy and Confusion Matrix for SVM Classifier

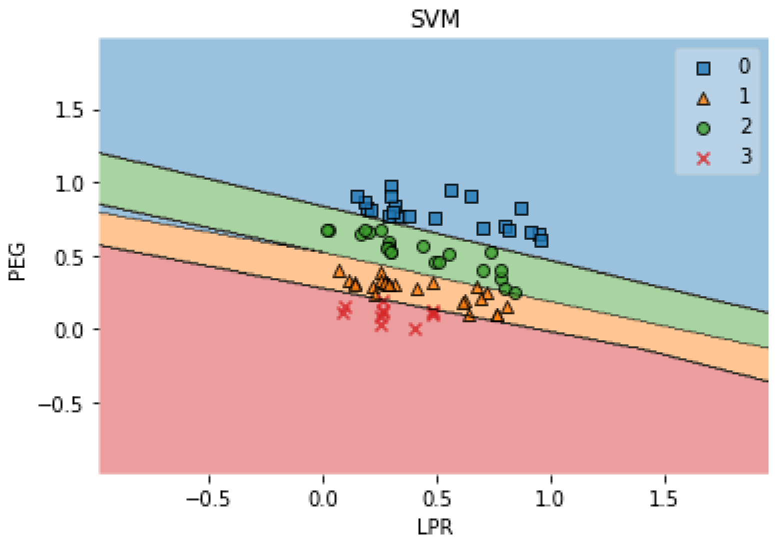
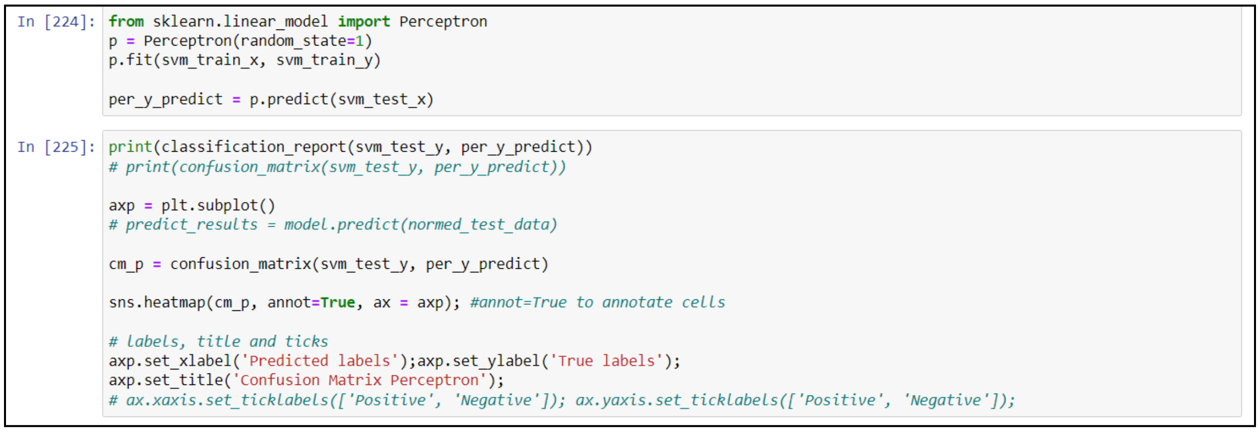


Fig 8. Decision Boundary for SVM

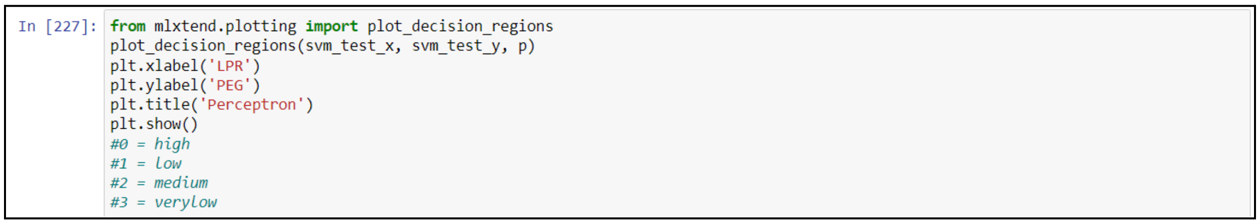
Perceptron:

The below code snippets show the Perceptron classifier and accuracy, confusion matrix and decision boundaries for Perceptron classifier.



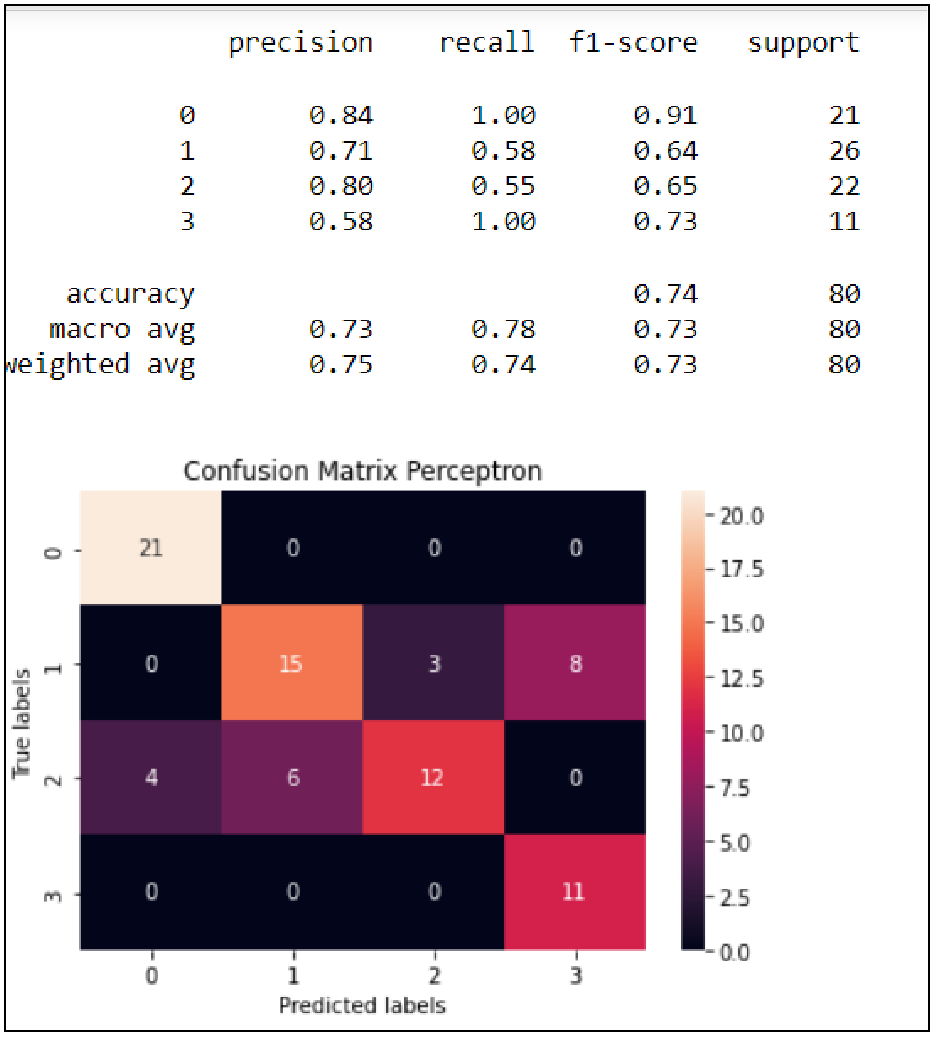
```
In [224]: from sklearn.linear_model import Perceptron
          p = Perceptron(random_state=1)
          p.fit(svm_train_x, svm_train_y)

          per_y_predict = p.predict(svm_test_x)

In [225]: print(classification_report(svm_test_y, per_y_predict))
          # print(confusion_matrix(svm_test_y, per_y_predict))

          axp = plt.subplot()
          # predict_results = model.predict(normed_test_data)

          cm_p = confusion_matrix(svm_test_y, per_y_predict)

          sns.heatmap(cm_p, annot=True, ax = axp); #annot=True to annotate cells

          # labels, title and ticks
          axp.set_xlabel('Predicted labels');axp.set_ylabel('True labels');
          axp.set_title('Confusion Matrix Perceptron');
          # ax.xaxis.set_ticklabels(['Positive', 'Negative']); ax.yaxis.set_ticklabels(['Positive', 'Negative']);
```

Fig 9. Perceptron Classifier and Classification report Code



```
In [227]: from mlxtend.plotting import plot_decision_regions
          plot_decision_regions(svm_test_x, svm_test_y, p)
          plt.xlabel('LPR')
          plt.ylabel('PEG')
          plt.title('Perceptron')
          plt.show()
          #0 = high
          #1 = low
          #2 = medium
          #3 = verylow
```

Fig 10. Perceptron Decision Boundary Plotting Code



```
              precision    recall  f1-score   support

           0       0.84      1.00      0.91        21
           1       0.71      0.58      0.64        26
           2       0.80      0.55      0.65        22
           3       0.58      1.00      0.73        11

    accuracy                           0.74        80
   macro avg       0.73      0.78      0.73        80
weighted avg       0.75      0.74      0.73        80
```

Fig 11. Accuracy and Confusion Matrix for Perceptron Classifier

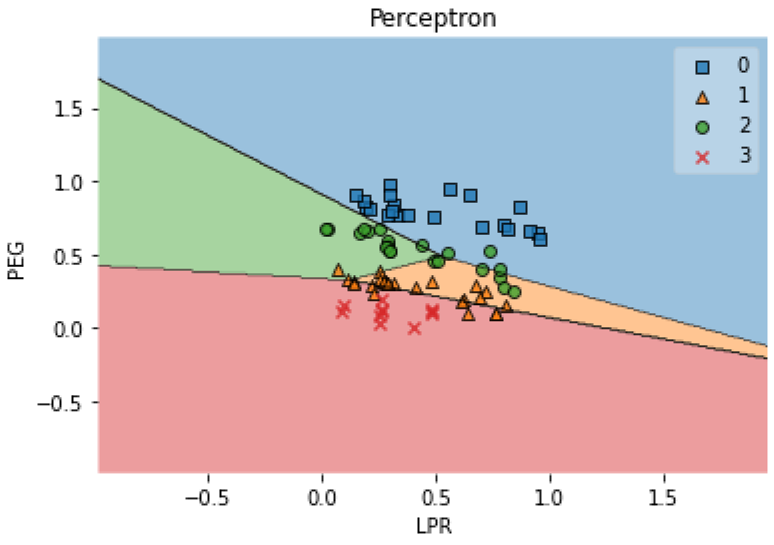


Fig 12. Decision Boundary for Perceptron

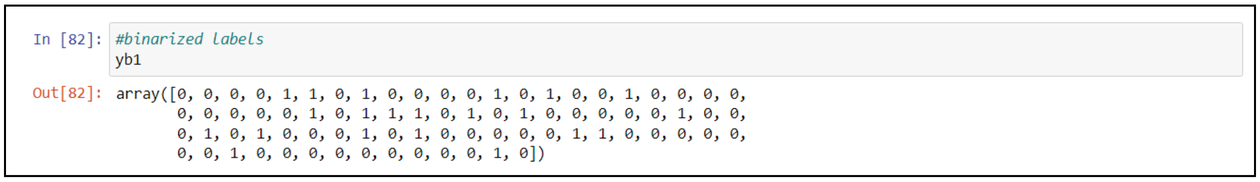
Question 2

Answer:

(A) For the binary classifiers we have chosen RBF SVM model instead of linear because RBF provided better accuracy results compared to the linear SVM Model.

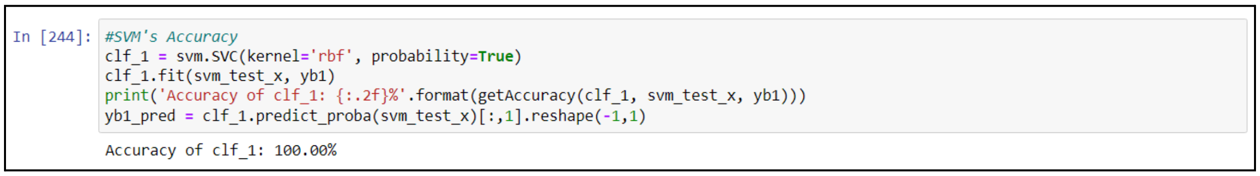
<u>Class = High - Binary Classifier</u>

The model gives 100% accuracy with the RBF SVM model. This can be a case of overfitting.



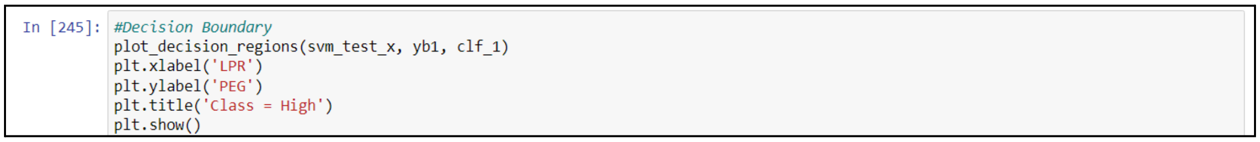
```
In [82]:  #binarized labels
          yb1

Out[82]: array([0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
                0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
                0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0])
```

Fig 13. Binarized labels for class high



```
In [244]: #SVM's Accuracy
          clf_1 = svm.SVC(kernel='rbf', probability=True)
          clf_1.fit(svm_test_x, yb1)
          print('Accuracy of clf_1: {:.2f}%'.format(getAccuracy(clf_1, svm_test_x, yb1)))
          yb1_pred = clf_1.predict_proba(svm_test_x)[:,1].reshape(-1,1)

          Accuracy of clf_1: 100.00%
```

Fig 14. Accuracy for class high



```
In [245]: #Decision Boundary
          plot_decision_regions(svm_test_x, yb1, clf_1)
          plt.xlabel('LPR')
          plt.ylabel('PEG')
          plt.title('Class = High')
          plt.show()
```

Fig 15. Data plot code for class high

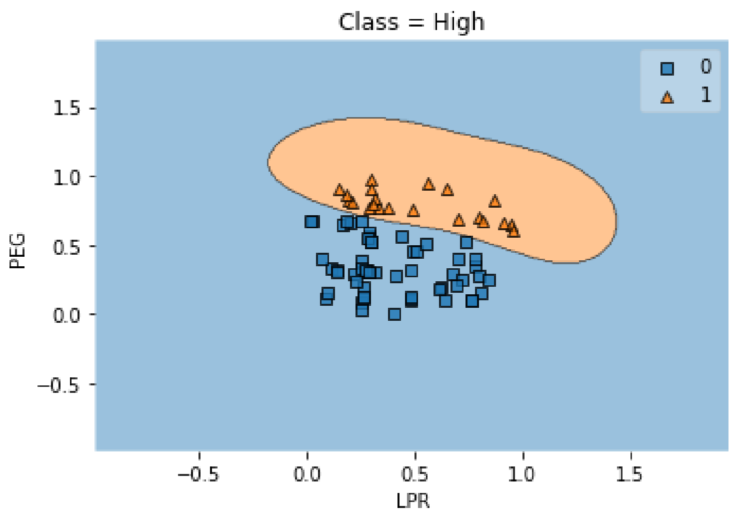
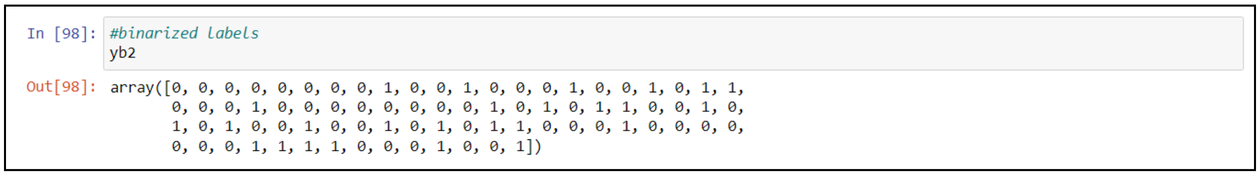


Fig 16. Decision Boundary for class high binarized label

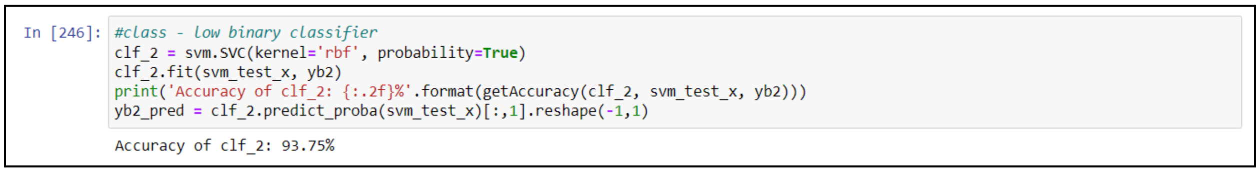
Class = Low - Binary Classifier

The model gives 93.75% accuracy with the RBF SVM model. This means the model is performing well but the accuracy can be improved with more samples.



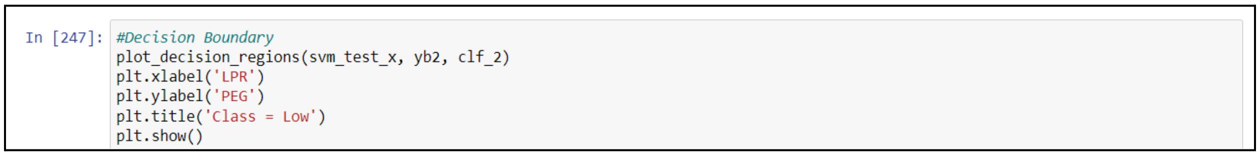
```
In [98]: #binarized labels
         yb2

Out[98]: array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1,
                0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0,
                1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
                0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1])
```

Fig 17. Binarized labels for class low



```
In [246]: #class - low binary classifier
          clf_2 = svm.SVC(kernel='rbf', probability=True)
          clf_2.fit(svm_test_x, yb2)
          print('Accuracy of clf_2: {:.2f}%'.format(getAccuracy(clf_2, svm_test_x, yb2)))
          yb2_pred = clf_2.predict_proba(svm_test_x)[:,1].reshape(-1,1)

          Accuracy of clf_2: 93.75%
```

Fig 18. Accuracy for class low



```
In [247]: #Decision Boundary
          plot_decision_regions(svm_test_x, yb2, clf_2)
          plt.xlabel('LPR')
          plt.ylabel('PEG')
          plt.title('Class = Low')
          plt.show()
```

Fig 19. Data plot code for class low

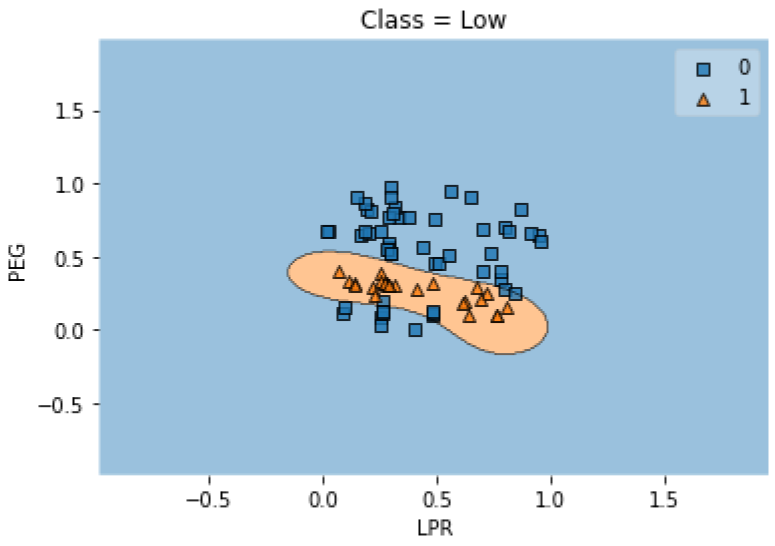
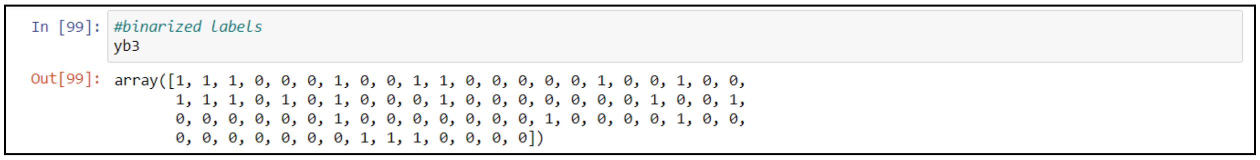


Fig 20. Decision Boundary for class low binarized label

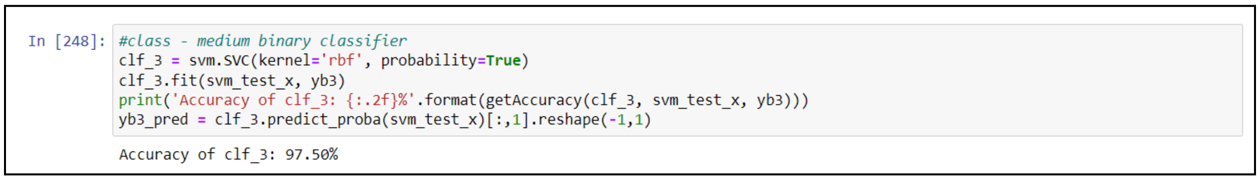
<u>Class = Medium - Binary Classifier</u>

The model gives 97.50% accuracy with the RBF SVM model. This is a good accuracy result. This means the model is not neither underfitting nor overfitting.



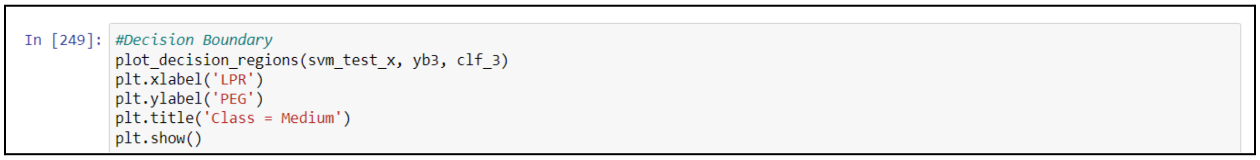
```
In [99]: #binarized labels
         yb3

Out[99]: array([1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
                1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
                0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0])
```

Fig 21. Binarized labels for class medium



```
In [248]: #class - medium binary classifier
          clf_3 = svm.SVC(kernel='rbf', probability=True)
          clf_3.fit(svm_test_x, yb3)
          print('Accuracy of clf_3: {:.2f}%'.format(getAccuracy(clf_3, svm_test_x, yb3)))
          yb3_pred = clf_3.predict_proba(svm_test_x)[:,1].reshape(-1,1)

          Accuracy of clf_3: 97.50%
```

Fig 22. Accuracy for class medium



```
In [249]: #Decision Boundary
          plot_decision_regions(svm_test_x, yb3, clf_3)
          plt.xlabel('LPR')
          plt.ylabel('PEG')
          plt.title('Class = Medium')
          plt.show()
```

Fig 23. Data plot code for class medium

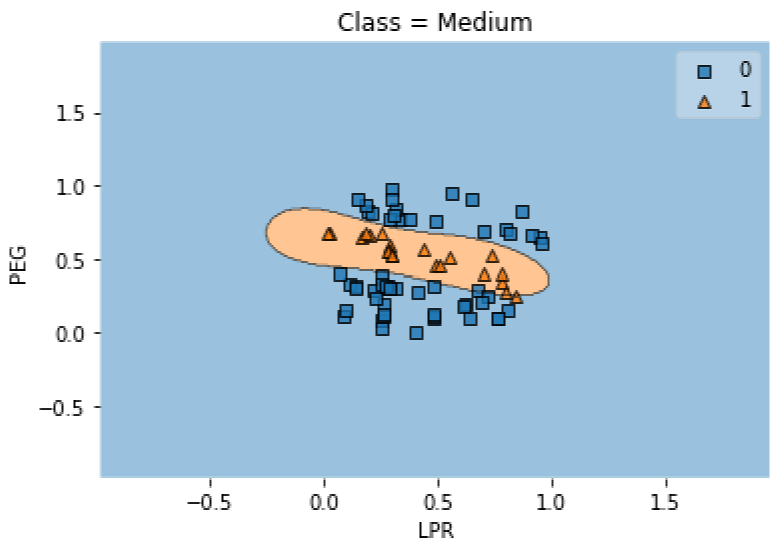
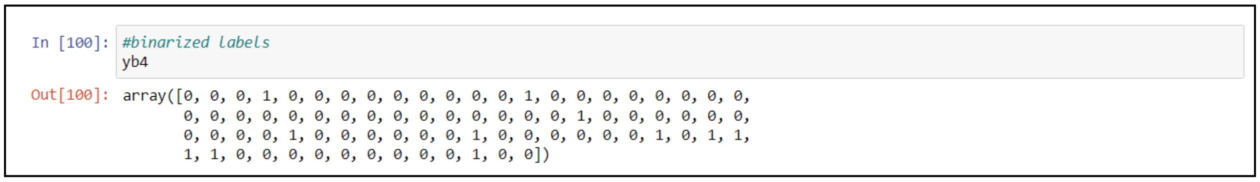


Fig 24. Decision Boundary for class medium binarized label

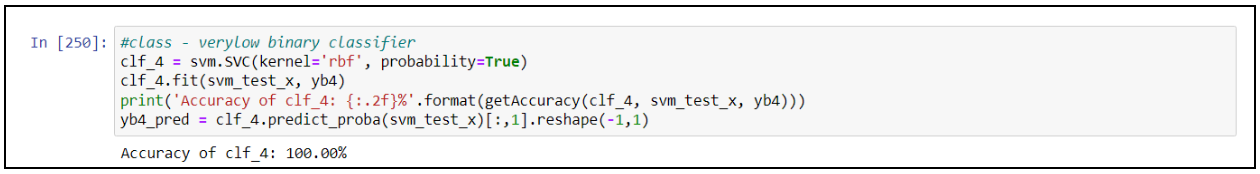
Class = Very Low - Binary Classifier

The model gives 100% accuracy with the RBF SVM model. This can be a case of overfitting.



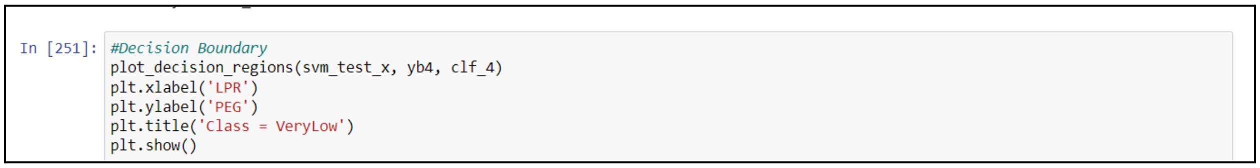
```
In [100]: #binarized labels
          yb4

Out[100]: array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
                 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0])
```

Fig 25. Binarized labels for class verylow



```
In [250]: #class - verylow binary classifier
          clf_4 = svm.SVC(kernel='rbf', probability=True)
          clf_4.fit(svm_test_x, yb4)
          print('Accuracy of clf_4: {:.2f}%'.format(getAccuracy(clf_4, svm_test_x, yb4)))
          yb4_pred = clf_4.predict_proba(svm_test_x)[:,1].reshape(-1,1)

          Accuracy of clf_4: 100.00%
```

Fig 26. Accuracy for class verylow



```
In [251]: #Decision Boundary
          plot_decision_regions(svm_test_x, yb4, clf_4)
          plt.xlabel('LPR')
          plt.ylabel('PEG')
          plt.title('Class = VeryLow')
          plt.show()
```

Fig 27. Data plot code for class verylow

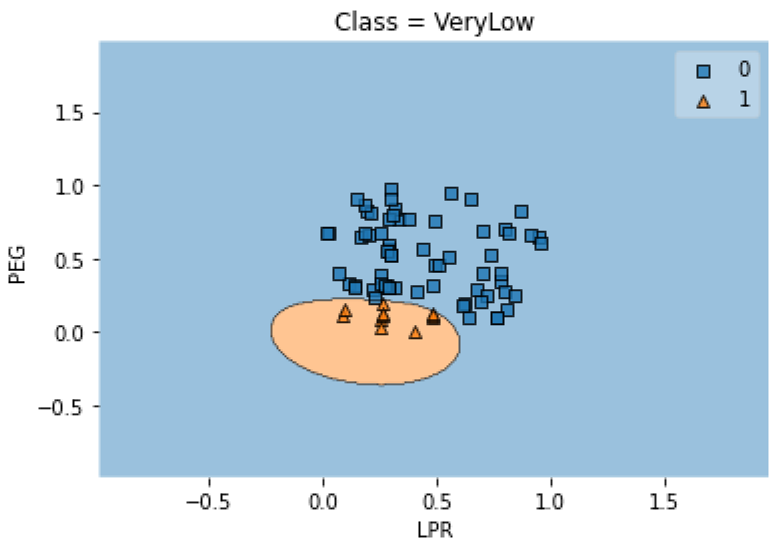
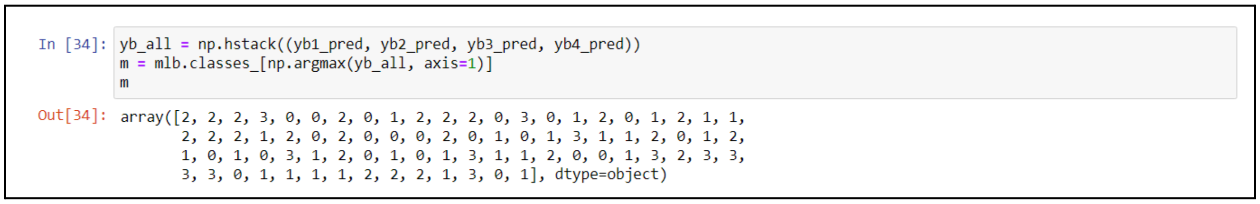


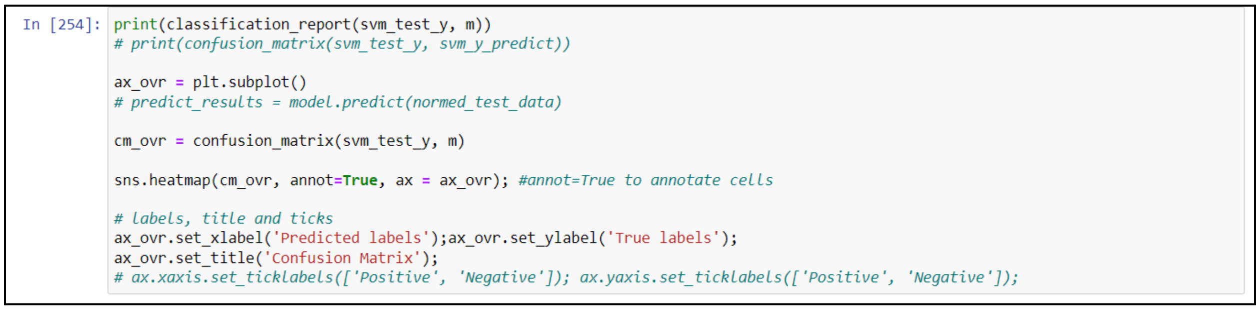
Fig 28. Decision Boundary for class verylow binarized label

(B) The final predicted labels obtained are as follows:



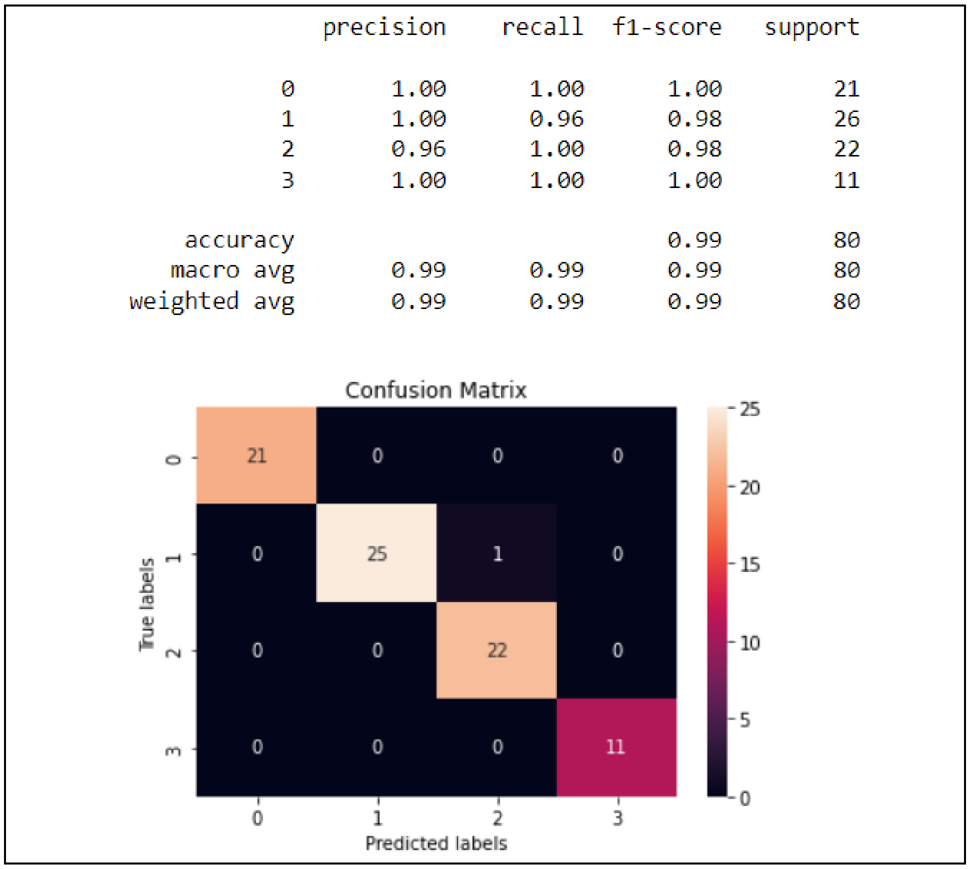
```
In [34]: yb_all = np.hstack((yb1_pred, yb2_pred, yb3_pred, yb4_pred))
         m = mlb.classes_[np.argmax(yb_all, axis=1)]
         m

Out[34]: array([2, 2, 2, 3, 0, 0, 2, 0, 1, 2, 2, 2, 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, 3, 3,
                3, 3, 0, 1, 1, 1, 1, 2, 2, 2, 1, 3, 0, 1], dtype=object)
```

Fig 29.Aggregate predicted labels



```
In [254]: print(classification_report(svm_test_y, m))
          # print(confusion_matrix(svm_test_y, svm_y_predict))

          ax_ovr = plt.subplot()
          # predict_results = model.predict(normed_test_data)

          cm_ovr = confusion_matrix(svm_test_y, m)

          sns.heatmap(cm_ovr, annot=True, ax = ax_ovr); #annot=True to annotate cells

          # labels, title and ticks
          ax_ovr.set_xlabel('Predicted labels');ax_ovr.set_ylabel('True labels');
          ax_ovr.set_title('Confusion Matrix');
          # ax.xaxis.set_ticklabels(['Positive', 'Negative']); ax.yaxis.set_ticklabels(['Positive', 'Negative']);
```

Fig 30. Aggregate Classification report Code



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      0.96      0.98        26
           2       0.96      1.00      0.98        22
           3       1.00      1.00      1.00        11

    accuracy                           0.99        80
   macro avg       0.99      0.99      0.99        80
weighted avg       0.99      0.99      0.99        80
```

Fig 31. Accuracy and Confusion Matrix for Aggregated Predicted labels

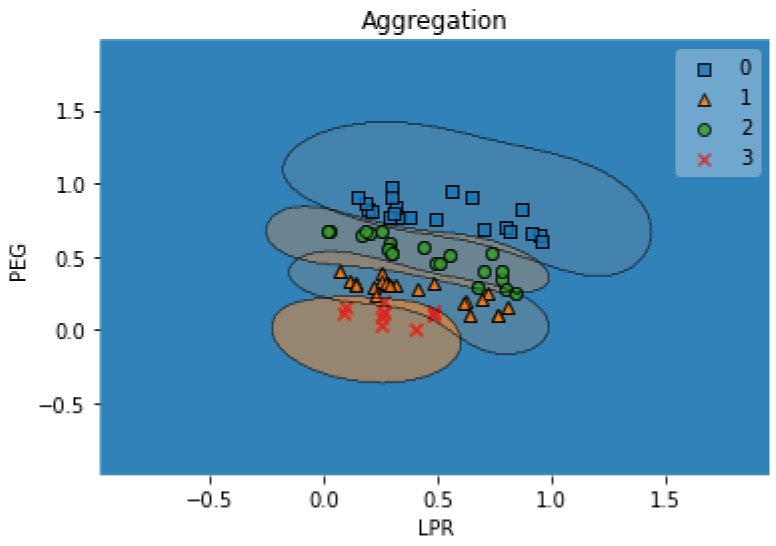


Fig 32. Decision Boundary for aggregate predicted labels

(C) <u>Models (Perceptron, SVM)</u>

Using the SVM-Linear model we got 100% accuracy. The model was able to classify the test data perfectly and had proper decision boundaries as well. But using the same dataset and training the Perceptron we get less accurate results. The Perceptron yields an accuracy of 73.75%. This is very less compared to the accuracy of SVM. The reason for variation in performance between the above techniques can be due to how error and stop are calculated. The Perceptron makes no attempt to optimize separation "distance." It's fine as long as it discovers a hyperplane that connects the two sets. SVM, on the other hand, tries to maximize the "support vector," which is the distance between two sample points that are closest to each other.

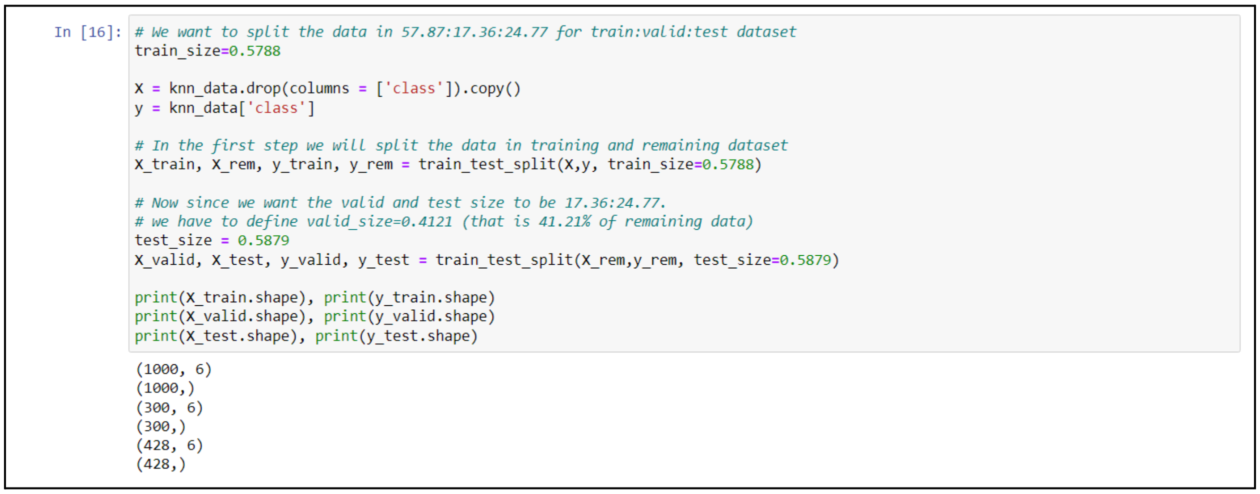
<u>OvR and Aggregated results</u>

Looking at individual binary classifiers, the accuracy varies 90% to 100%. But when we look at the aggregate results we get 99% accuracy. This means using binary classification first and then taking the aggregate yields better results than just classifying a multiclass problem using the SVM model. Also, the rbf is a better way to classify data when it comes to binary classification and the linear model gives better results for a multiclass classification.

Question 3

Answer:

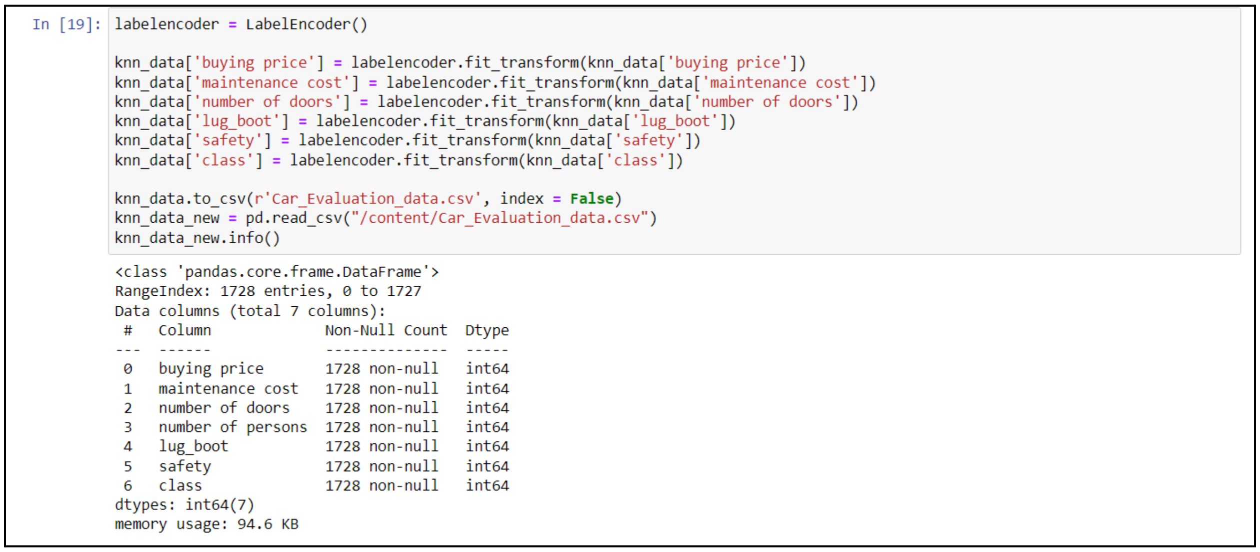
(A)



```
In [16]: # We want to split the data in 57.87:17.36:24.77 for train:valid:test dataset
         train_size=0.5788

         X = knn_data.drop(columns = ['class']).copy()
         y = knn_data['class']

         # In the first step we will split the data in training and remaining dataset
         X_train, X_rem, y_train, y_rem = train_test_split(X,y, train_size=0.5788)

         # Now since we want the valid and test size to be 17.36:24.77.
         # we have to define valid_size=0.4121 (that is 41.21% of remaining data)
         test_size = 0.5879
         X_valid, X_test, y_valid, y_test = train_test_split(X_rem,y_rem, test_size=0.5879)

         print(X_train.shape), print(y_train.shape)
         print(X_valid.shape), print(y_valid.shape)
         print(X_test.shape), print(y_test.shape)

         (1000, 6)
         (1000,)
         (300, 6)
         (300,)
         (428, 6)
         (428,)
```

Fig 33. Code for splitting the dataset into train, valid and test dataset

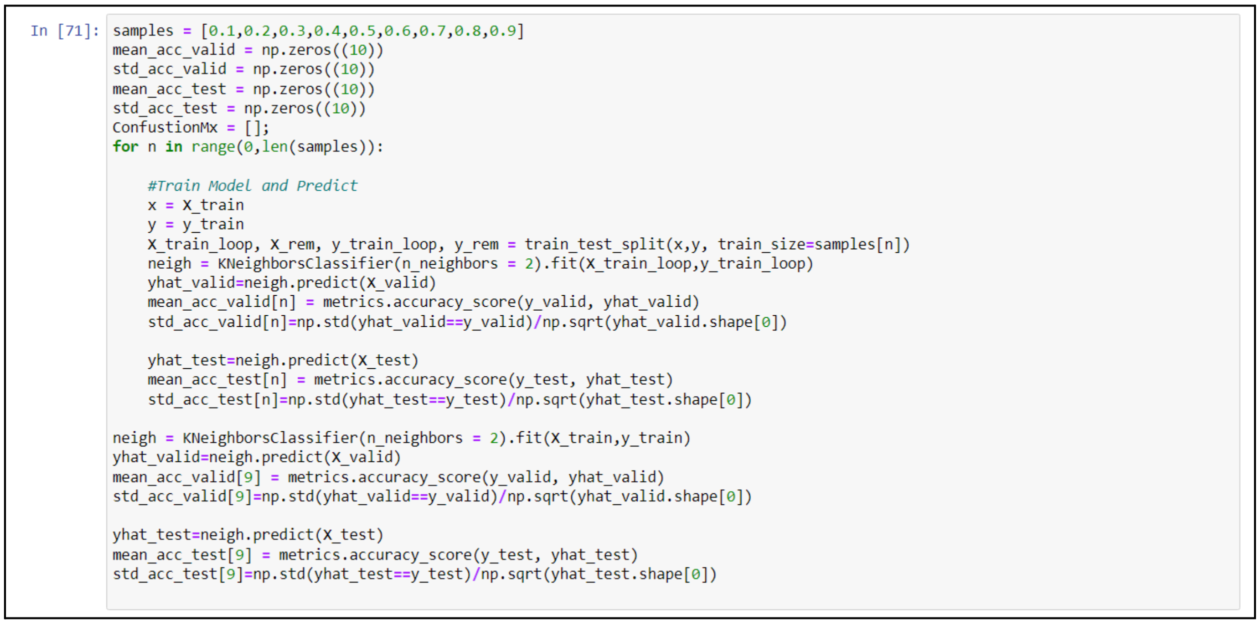
(B)



```
In [19]: labelencoder = LabelEncoder()

         knn_data['buying price'] = labelencoder.fit_transform(knn_data['buying price'])
         knn_data['maintenance cost'] = labelencoder.fit_transform(knn_data['maintenance cost'])
         knn_data['number of doors'] = labelencoder.fit_transform(knn_data['number of doors'])
         knn_data['lug_boot'] = labelencoder.fit_transform(knn_data['lug_boot'])
         knn_data['safety'] = labelencoder.fit_transform(knn_data['safety'])
         knn_data['class'] = labelencoder.fit_transform(knn_data['class'])

         knn_data.to_csv(r'Car_Evaluation_data.csv', index = False)
         knn_data_new = pd.read_csv("/content/Car_Evaluation_data.csv")
         knn_data_new.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 1728 entries, 0 to 1727
         Data columns (total 7 columns):
          #   Column             Non-Null Count  Dtype
         ---  ------             --------------  -----
          0   buying price       1728 non-null   int64
          1   maintenance cost   1728 non-null   int64
          2   number of doors    1728 non-null   int64
          3   number of persons  1728 non-null   int64
          4   lug_boot           1728 non-null   int64
          5   safety             1728 non-null   int64
          6   class              1728 non-null   int64
         dtypes: int64(7)
         memory usage: 94.6 KB
```

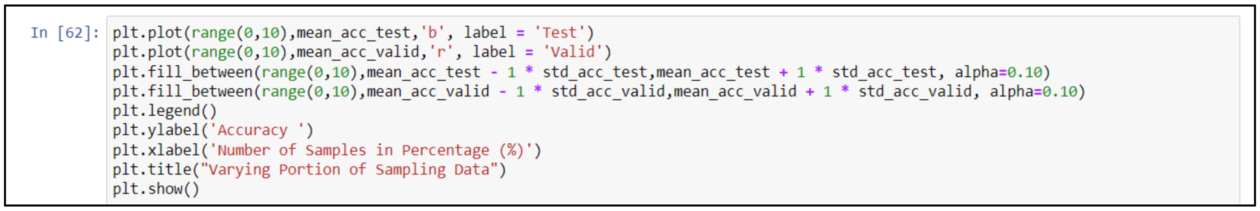
Fig 34. Code for transforming strings values to number

(C)



```
In [71]: samples = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
         mean_acc_valid = np.zeros((10))
         std_acc_valid = np.zeros((10))
         mean_acc_test = np.zeros((10))
         std_acc_test = np.zeros((10))
         ConfustionMx = [];
         for n in range(0,len(samples)):

             #Train Model and Predict
             x = X_train
             y = y_train
             X_train_loop, X_rem, y_train_loop, y_rem = train_test_split(x,y, train_size=samples[n])
             neigh = KNeighborsClassifier(n_neighbors = 2).fit(X_train_loop,y_train_loop)
             yhat_valid=neigh.predict(X_valid)
             mean_acc_valid[n] = metrics.accuracy_score(y_valid, yhat_valid)
             std_acc_valid[n]=np.std(yhat_valid==y_valid)/np.sqrt(yhat_valid.shape[0])

             yhat_test=neigh.predict(X_test)
             mean_acc_test[n] = metrics.accuracy_score(y_test, yhat_test)
             std_acc_test[n]=np.std(yhat_test==y_test)/np.sqrt(yhat_test.shape[0])

         neigh = KNeighborsClassifier(n_neighbors = 2).fit(X_train,y_train)
         yhat_valid=neigh.predict(X_valid)
         mean_acc_valid[9] = metrics.accuracy_score(y_valid, yhat_valid)
         std_acc_valid[9]=np.std(yhat_valid==y_valid)/np.sqrt(yhat_valid.shape[0])

         yhat_test=neigh.predict(X_test)
         mean_acc_test[9] = metrics.accuracy_score(y_test, yhat_test)
         std_acc_test[9]=np.std(yhat_test==y_test)/np.sqrt(yhat_test.shape[0])
```

Fig 35. Varying Sample data code



```
In [62]: plt.plot(range(0,10),mean_acc_test,'b', label = 'Test')
         plt.plot(range(0,10),mean_acc_valid,'r', label = 'Valid')
         plt.fill_between(range(0,10),mean_acc_test - 1 * std_acc_test,mean_acc_test + 1 * std_acc_test, alpha=0.10)
         plt.fill_between(range(0,10),mean_acc_valid - 1 * std_acc_valid,mean_acc_valid + 1 * std_acc_valid, alpha=0.10)
         plt.legend()
         plt.ylabel('Accuracy ')
         plt.xlabel('Number of Samples in Percentage (%)')
         plt.title("Varying Portion of Sampling Data")
         plt.show()
```

Fig 36. Code for plotting results for varying sample data

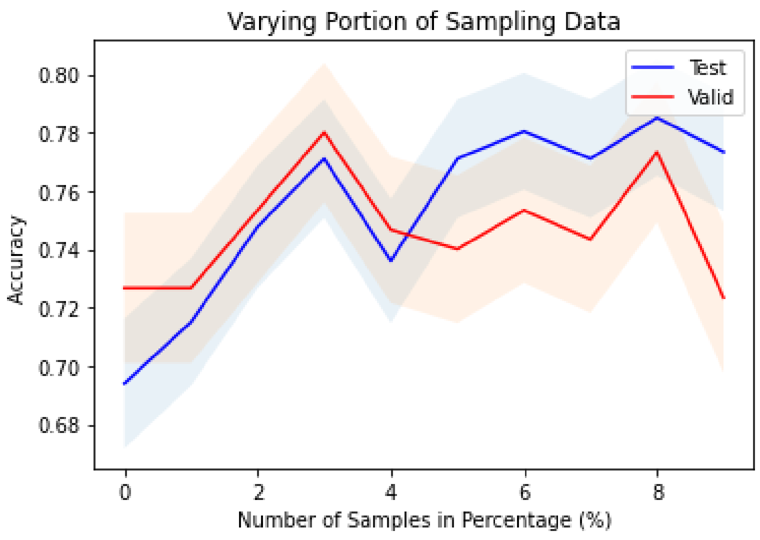
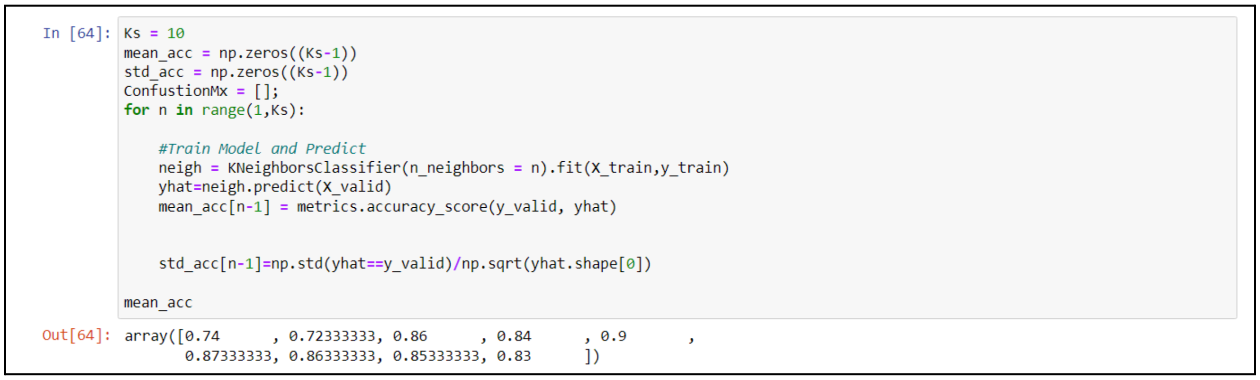


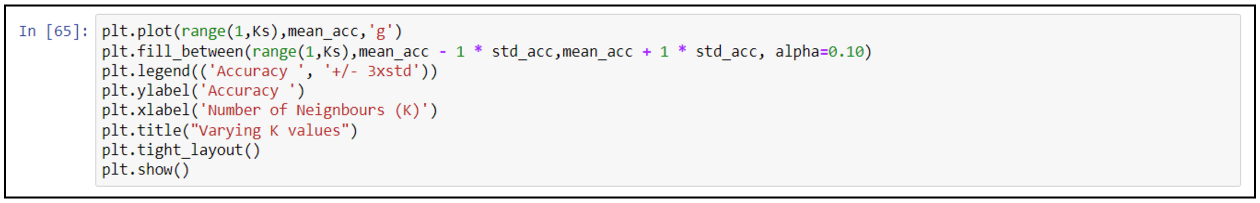
Fig 37. Output showing test and valid dataset accuracies for different portions of sample data

(D)



```
In [64]: Ks = 10
         mean_acc = np.zeros((Ks-1))
         std_acc = np.zeros((Ks-1))
         ConfustionMx = [];
         for n in range(1,Ks):

             #Train Model and Predict
             neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
             yhat=neigh.predict(X_valid)
             mean_acc[n-1] = metrics.accuracy_score(y_valid, yhat)


             std_acc[n-1]=np.std(yhat==y_valid)/np.sqrt(yhat.shape[0])

         mean_acc

Out[64]: array([0.74      , 0.72333333, 0.86      , 0.84      , 0.9       ,
                0.87333333, 0.86333333, 0.85333333, 0.83      ])
```

Fig 38. Code for varying K values.



```
In [65]: plt.plot(range(1,Ks),mean_acc,'g')
         plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
         plt.legend(('Accuracy ', '+/- 3xstd'))
         plt.ylabel('Accuracy ')
         plt.xlabel('Number of Neignbours (K)')
         plt.title("Varying K values")
         plt.tight_layout()
         plt.show()
```

Fig 39. Code for plotting accuracies for different K values

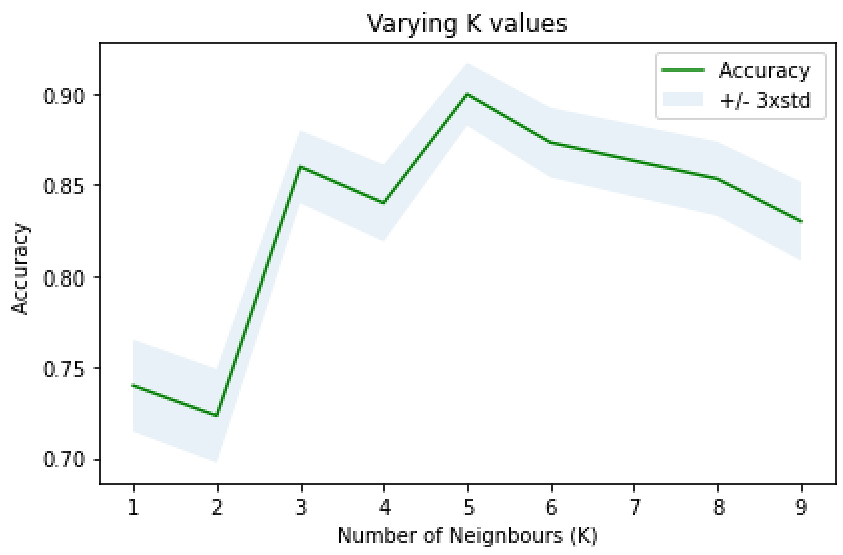


Fig 40. Output showing dataset accuracy for different K values

(E) Varying Sample Data

From the graph obtained we can see that the accuracy increases as the sample data increases upto a certain value. Then there is dip at 40%. The accuracy starts decreasing as the sample data starts increasing. After that again similar results are seen as before the dip. The valid dataset accuries vary more as compared to the test dataset. Test dataset starts showing stable accuracies after 50%. The valid dataset accuracy shows an increase till 80% sample data and then starts decreasing again.

Varying K values

The accuracy of the model increases as the K values increase. At K=5 it reaches its maximum accuracy. After that the accuracy of the model starts decreasing as the K value increases. The best accuracy of 90% is provided when K=5. Hence having a larger value of K can have a negative impact on the accuracy of the model. We can see that we had 4 classes and K=5 gives a good result. Thus, the K value should be near the number of classes to get better accuracy.