ELG 5255: Applied Machine Learning

Assignment 1

Group 13

Members: Kishita Pakhrani

David Talson (8419286)

1.0 Classification Using Support Vector Machine (SVM) and Perceptron Classifiers

In this problem, Support Vector Machine (SVG) and Perceptron algorithms are applied to classify from the Data User Modelling Dataset (DUMD). The following techniques are also applied, categorical encoding, feature selection, model training and testing. In the following sections, more information is provided:

1.1 Categorical Encoding

In our training and test dataset, categorical class labels mean non-numeric values under the "UNS" column which need to be converted to numerical values because most machine learning algorithms can only work with numeric values. There are various ways to achieve this but the method used to achieve this is "Label Encoding". Label encoding is basically each converting value in the target column to a number. Label encoding was applied using scikit-learn and new values were saved in a new column called "UNS_N". The relation below shows the categorical to numerical values:

0 = High, 1 = Low, 2 = Medium, 3 = Very Low

The figure below shows the training data after label encoding:

```
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: LabelEncoding Result

1.2 Feature Selection

When selecting features for a machine it is important to avoid selecting features which contain repeated information or have low variance because it will make the model confused and less accurate. In this case, the features with the lowest variance were ‘STG’ and ‘SCG’, this meant they were discarded. Next, the correlation between the remaining features was calculated to eliminate the feature which had the highest correlation. When features have a high correlation, only one is needed to prevent information repetition. ‘STR’ and ‘LPR’ had the highest correlation so then ‘STR’ was eliminated. The final features selected are shown below:

LPR: The exam performance of user for related objects with goal object

PEG: The exam performance of user for goal objects

1.3 Class Representation

In this section, the various classes from categorical encoding in section 1.0 are plotted vs the selected features in section 1.2. This representation is used to get an idea of where the hyperplane would be placed by the model. The figure below shows a plot of the classes

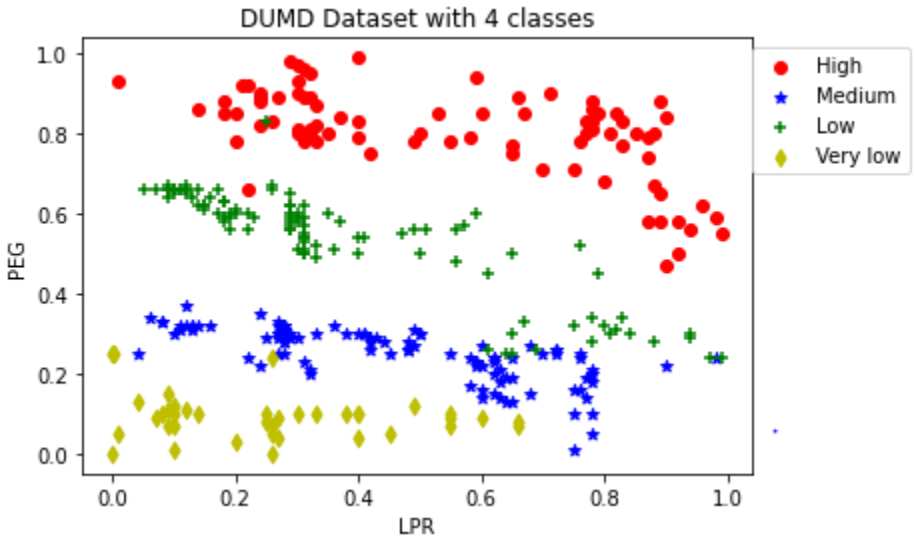

Fig 2: Data Plot Output

As can be seen from figure 2 above, each class is separately shown.

1.4 Support Vector Machine Model

In this section, the SVG algorithm is used to train a model which will predict the 'UNS' value of the test data. The SVG algorithm performs classification by determining a hyperplane which best separates the classes. This hyperplane is known as the decision boundary and is used to make decisions when test data is applied. The SVG function in scikit-learn was used to create the model and the code snippet is shown below:

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

```
def main():
  warnings.simplefilter('ignore')
  features = ['LPR','PEG']
  X_train, X_test, y_train, y_test, data_train = produceDataset(features)

  model_svg = svm.SVC(kernel='linear', decision_function_shape='ovo', C=100)
  model_svg.fit(X_train, y_train)

  y_true_svg, y_pred_svg = y_test, model_svg.predict(X_test)
```

Fig 3: SVM Classifier Code

```
print('Accuracy of model_svg: {:.2f}%'.format(getAccuracy(model_svg, X_train, y_train)))
print(classification_report(y_test, y_pred_svg))
print('\nConfusion Matrix using SVM:\n')
print(confusion_matrix(y_test, y_pred_svg))
ConfusionMatrixDisplay.from_estimator(model_svg, X_test, y_pred_svg)
plt.title("Confusion Matrix using SVG")
plt.show()
X_numpy = np.array(X_train)
y_numpy = np.array(y_train)
```

Fig 4:. SVM classification report and confusion matrix code

```
plotData(data_train, features[0], features[1])
plot_decision_regions(X_numpy, y_numpy, model_svg)
plt.title("Decision Boundaries using SVG")
plt.xlabel('LPR')
plt.ylabel('PEG')
plt.show()
```

Fig 5: SVM Decision Boundary Plotting Code

The figure below show the accuracy, classification, and confusion matrix of the SVG model:

```
Accuracy of model_svg: 96.28%
              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


Confusion Matrix using SVM:

[[21  0  0  0]
 [ 0 26  0  0]
 [ 0  0 22  0]
 [ 0  0  0 11]]
```

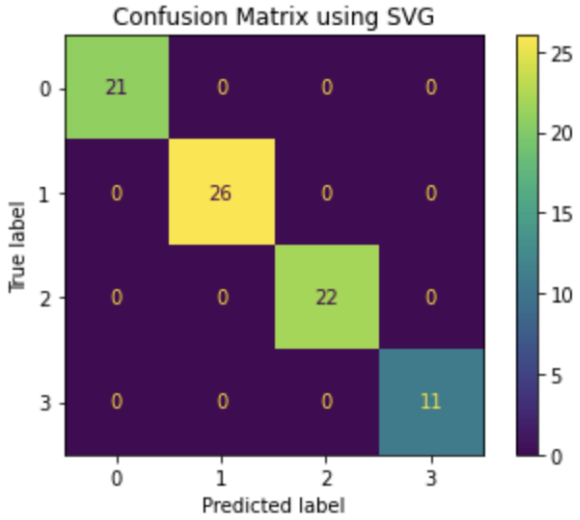

Figure 6: Model Accuracy and Confusion Matrix

The model accuracy achieved was high, this means the model is good. Also, as can be seen from the confusion matrix, the model was able to correctly predict the classes. The figure below shows the decision boundary of the model:

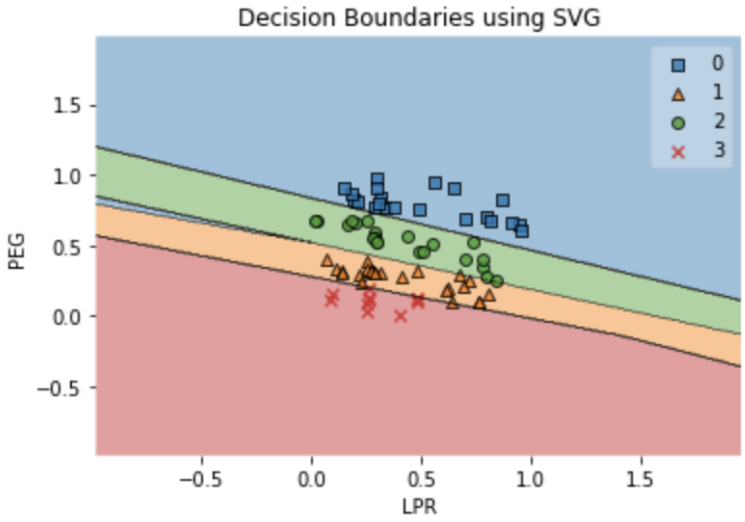

Fig 7: Decision Boundary for SVM

As can be seen from figure 7 above, the hyperplane created was accurate.

1.5 Perceptron Model

The perceptron model basically compares the weighted sum of inputs to a threshold. If the value is above the threshold, '1' is returned,, else '0' is returned. The below code snippets show the Perceptron classifier and accuracy, confusion matrix and decision boundaries for Perceptron classifier.

```
def main():
  warnings.simplefilter('ignore')
  features = ['LPR','PEG']
  X_train, X_test, y_train, y_test, data_train = produceDataset(features)

  # Using Perceptron
  model_percep = Perceptron(random_state=1)
  model_percep.fit(X_train, y_train)
  y_true_percep, y_pred_percep = y_test, model_percep.predict(X_test)
```

Fig 8: Perceptron ClassificationCode

```
print('Accuracy of model_svg: {:.2f}%'.format(getAccuracy(model_percep, X_test, y_test)))
print(classification_report(y_test, y_pred_percep))
conf = confusion_matrix(y_test, y_pred_percep)
print('\nConfusion Matrix using Perceptron:\n')
print(conf)
sns.heatmap(conf, annot=True)
plt.title('Confusion Matrix using Perceptron')
plt.xlabel('Predicted values')
plt.ylabel('Actual Values')
plt.show()
X_numpy = np.array(X_train)
y_numpy = np.array(y_train)
```

Fig 9. Perceptron Decision Boundary Plotting Code

The figure below show the accuracy, classification report, and confusion matrix of the perceptron model:

```
Accuracy of model_svg: 73.75%
              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


Confusion Matrix using Perceptron:

[[21  0  0  0]
 [ 0 15  3  8]
 [ 4  6 12  0]
 [ 0  0  0 11]]
```

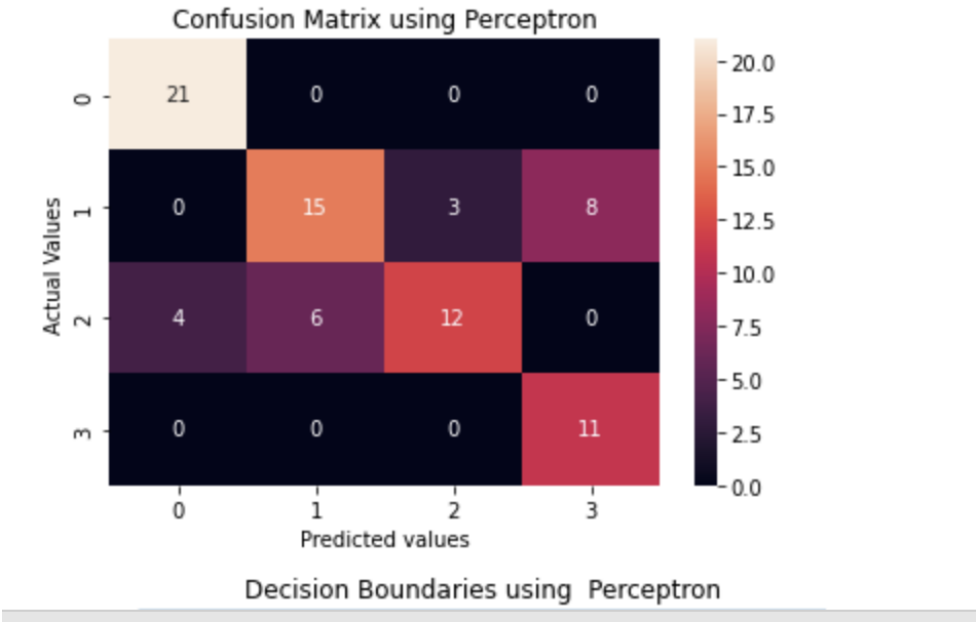

Fig 10: Accuracy and Confusion Matrix for Perceptron Classifier

As can be seen from figure 10 above the model has an average accuracy. From the confusion matrix, the model had some false positives, false negatives. The figure below shows the decision boundary of the model:

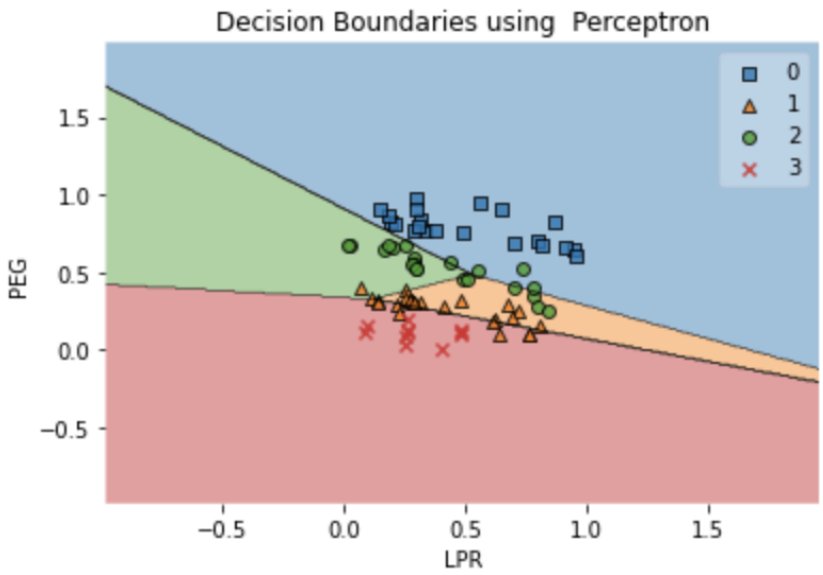

Fig 11. Decision Boundary for Perceptron

As can be seen from figure 11 above, the hyperplane created is not very accurate.

2.0 One vs Rest Support Vector Machine

In this section the One vs Rest SVG is performed. The One vs Rest SVG is a technique where the N-models are generated for N-classes instances dataset. The idea behind this technique is to create a system to tackle multi-class classification problems. Traditional classification techniques are created for binary classification so for multiple class instances the One vs Rest is used where a model is created for each class vs the rest. In the following sections, various steps are taken for each binary classifier to perform this process:

2.1 Building One vs Rest Support Vector Machine

2.1.1 Obtain Binarized labels

The binarized label was obtained using the MultiLabelBinarizer function in python. The multilabelbinarizer function which converts our classes to binary values. After

implementation, each label was stored in a variable. The figure below shows the output for ‘yb0’ which corresponds to class ‘High’

```
y0
 [0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0
 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 1 0 1 0 1 0 0 1 0 0 0 0 1 0 1 0 1 1 0 0
 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0
 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 1 1 0 1
 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 1 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0]
```

Figure 12: Multilabel Binarization

2.1.2 Model Accuracy

For the binary classifiers Radial Basis Function (RBF) was chosen for the SVM kernel instead of linear because since this is a one vs rest problem, we need to clearly define a region for each class where any point out of the region is classified as ‘rest’. Also, RBF provided better accuracy results compared to the linear SVM Model. Using the RBF, the accuracy for each class is shown.

Class: High

The model gave an accuracy of 100% accuracy which could mean a case of overfitting. The figure below shows the accuracy and decision boundary of the model:

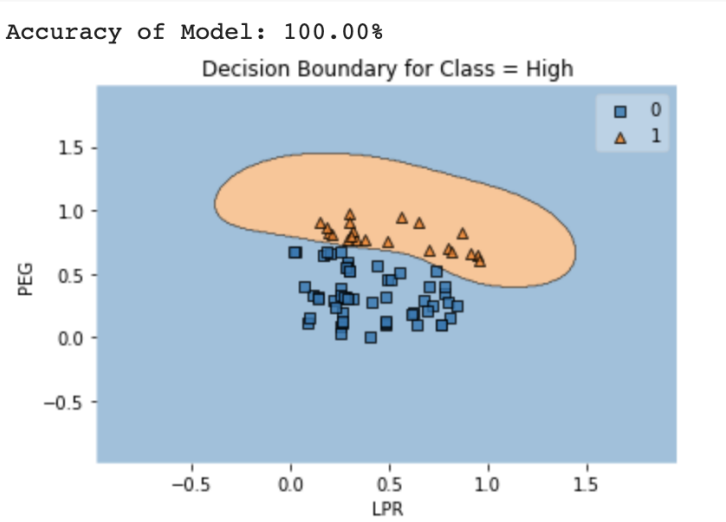

Figure 13: Accuracy and Decision Boundary for Class = High

As can be seen from figure 13 above, the model is very accurate and the hyperplane clearly separates the high class from the rest.

Class: Low

The model gives 97.5% accuracy which is good. The figure below shows the accuracy and decision boundary:

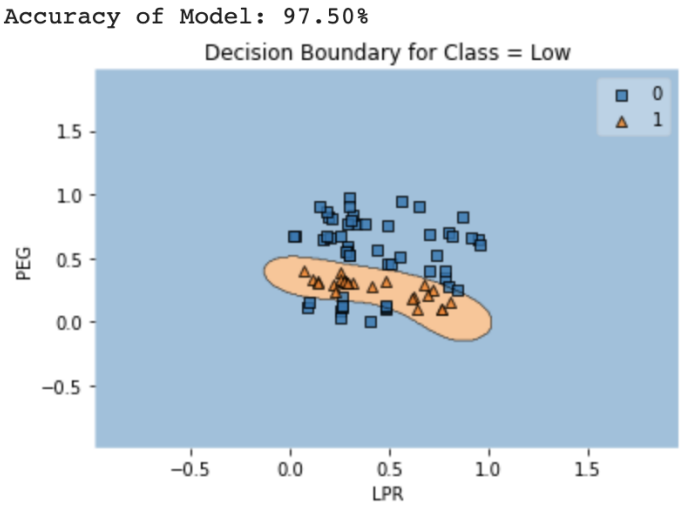

Fig 14. Decision Boundary for class low binarized label

Class: Medium

The model gave an accuracy of 100%. This could also mean overfitting. The figure below shows the accuracy and decision boundary:

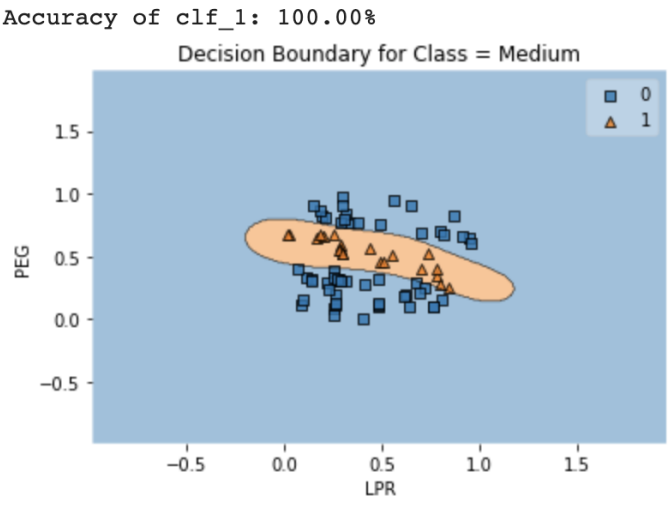

Figure 15: Decision boundary for medium class

Class: Very Low

The model gave a 98.75% accuracy. The figure below shows the accuracy and decision boundary .

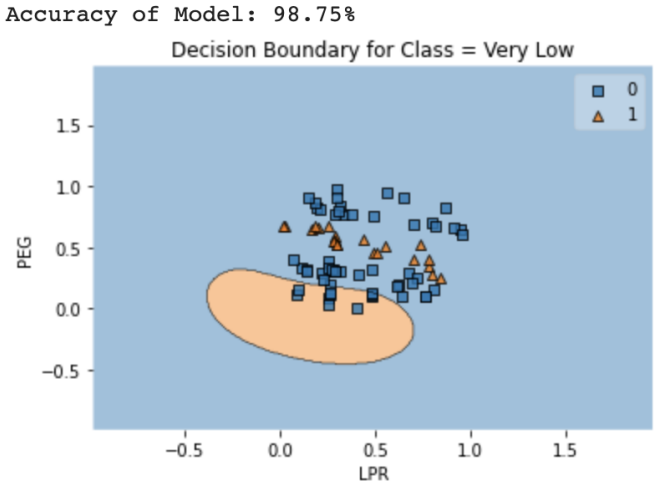

Fig 16: Decision Boundary for class very low class

2.2 Aggregate Results

The individual predicted labels can be combined to get an aggregate predicted labels. The code snippet in the figure below shows this:

```
yb_all = np.hstack((yb0_pred, yb1_pred, yb2_pred, yb3_pred))
m = mlb.classes_[np.argmax(yb_all, axis=1)]
m = m.astype(str).astype(int)
```

Fig 17: Aggregate predicted labels

As can be seen from figure 17 above, the aggregate predicted classes can be compared to the actual class value. The result of this comparison is shown the the classification report and confusion matrix below:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	21
1	0.96	1.00	0.98	26
2	1.00	1.00	1.00	22
3	1.00	0.91	0.95	11
accuracy			0.99	80
macro avg	0.99	0.98	0.98	80
weighted avg	0.99	0.99	0.99	80

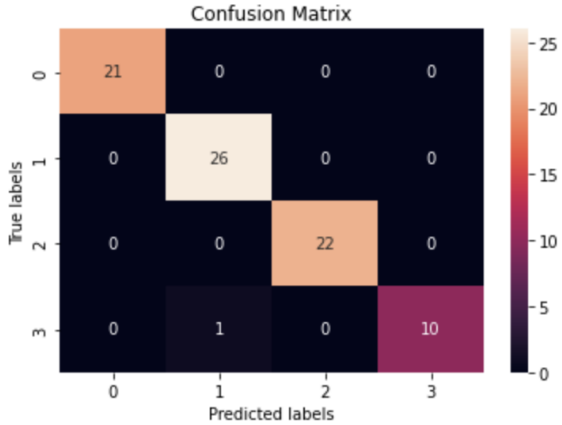

Fig 18: Accuracy and Confusion Matrix for Aggregated Predicted labels

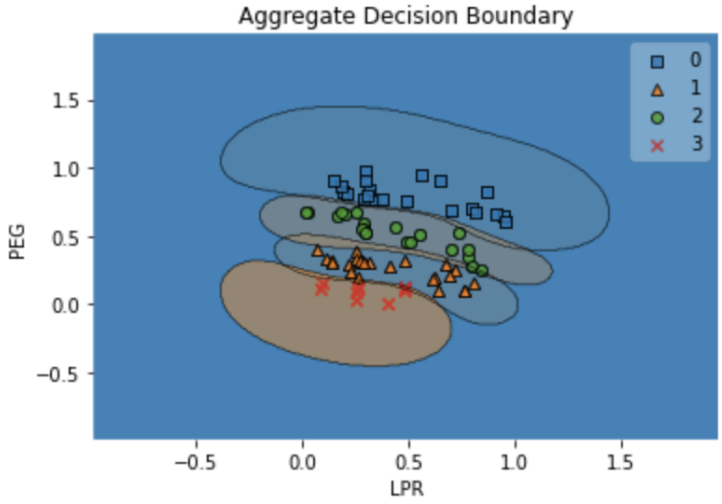

Fig 19: Decision Boundary for aggregate predicted labels

2.3 Discussion

Models (Perceptron vs SVM)

The accuracy using a linear kernel in the SVG was 100%. 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 model was less accurate. 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.

OvR and Aggregated results

Looking at individual binary classifiers, the accuracy varied 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.

3 K-Nearest Neighbors Classifier

The K-Nearest Neighbors Classifier (KNN) is a supervised learning technique where labeled input data is used to learn and create a model. In this exercise, a KNN classifier is used on the car-evaluation dataset.

3.1 Data Preparation

In this section the dataset is divided into training data, validation data, and test data. The training data will be used to create a model, the validation data is used to check how well

the model is learning, and the test data is used to test the model. The code snippet below shows how the dataset was split.

```
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 20: Code for splitting the dataset into train, valid and test dataset

As can be seen from figure 20 above, the data was successfully split. Then the string values were converted to numerical using label encoding as shown in the code snippet below:

```
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 21: Code for transforming strings values to number

Then the number of samples for the training dataset was varied to observe the impact on the accuracy of the model. The code snippet below shows the techniques used:

```
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 22: 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 23: Code for plotting results for varying sample data

The figure below show the model accuracy for varying sample data:

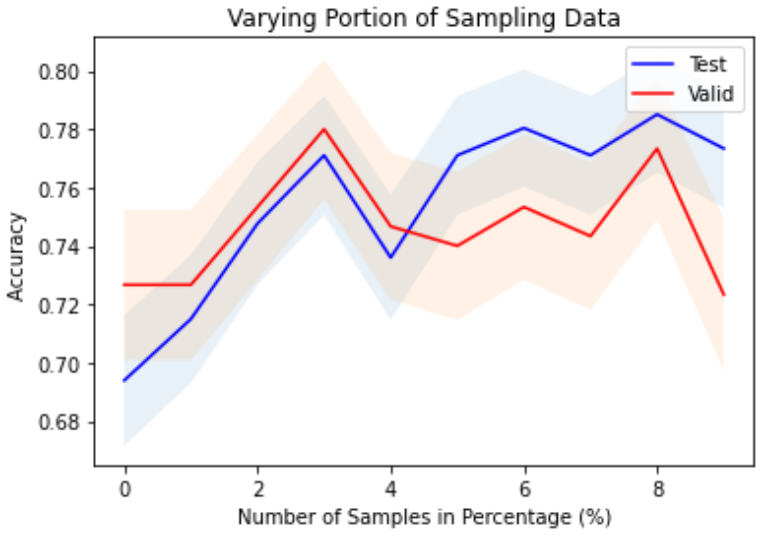

Fig 24: Output showing test and valid dataset accuracies for different portions of sample data

As can be seen from figure 24 above, the increase in sample data for training yielded higher accuracy until a certain percentage of around 80% then the accuracy drops. The code snippet below shows the technique used for varying k-values:

```
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 25: 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 26: Code for plotting accuracies for different K values

The figure below shows the accuracy for varying k-values:

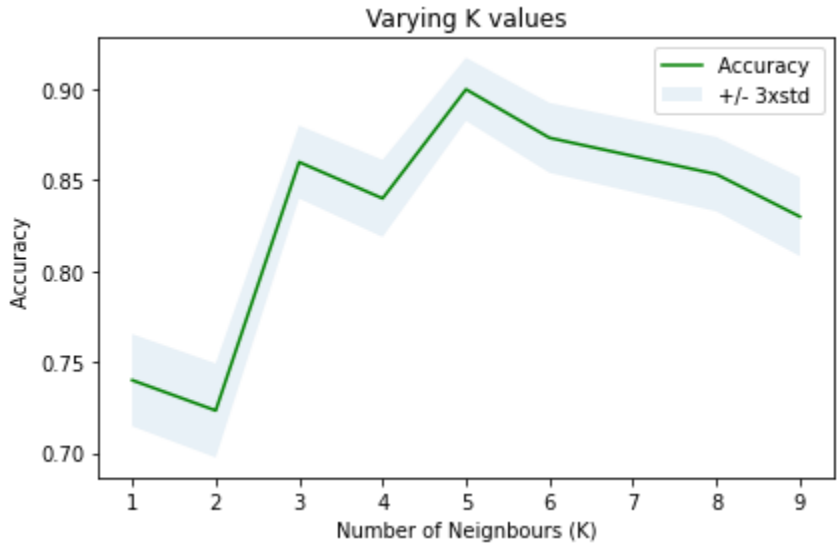

Fig 27: Output showing dataset accuracy for different K values

As can be seen from figure 27 above, increasing the k-values yielded better accuracy until a certain value around 5 then the accuracy reduces.

Varying Sample Data

From the graph obtained we can see that the accuracy increases as the sample data increases till 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.

Best K-value

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.