BBM469: Data Intensive Applications Lab. **(Due: 01/05/2020)**

Submission Assignment #2

Name: Doğukan Berat KARATAŞ, ID: 21527142

1 Introduction

The purpose of this assignment is to be able to predict with the highest accuracy whether or not another cancer patient data with the same characteristics will come in the future by using the model created with the Breast Cancer dataset. To create such a model, it is necessary to preprocess your data, train the model with several classification methods, and choose the method that achieves the highest result. I will be explaining these steps in detail below.

2 Dataset

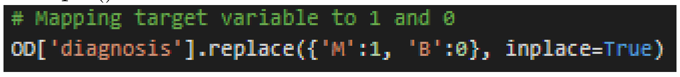
Our data set contains 569 rows and 33 columns. Attribute information:

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter$\hat{2}$ / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension (”coastline approximation” - 1)

The mean, standard error of these features and ”worst” or largest (average of the three largest values) were calculated for each image and 30 features were obtained. For example, field 3 is the Mean Radius, field 13 is the Radius SE, field 23 is the Worst Radius.

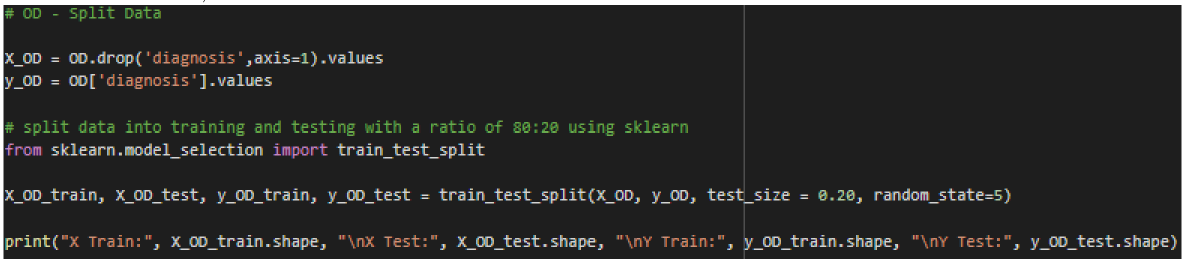
3 Data Preprocessing

First of all, I examined the dataset with the ”info()” method and saw that there are 3 different data types of columns. The column ”id” has the value ’int’, the column ’diagnosis’ has the value ’object’ and the other columns have the value ’float64’.
It was also seen in the same method that the column named ”Unnamed: 32” has ’NaN’ values. So I deleted the column ”Unnamed: 32” with the values ’NaN’ and deleted that column as the ”id” column would not work for me when creating my model.
Finally, I have the ”diagnosis” column with object values. I know that this column has unique values with the ”unique()” function. So I converted this column to int values using the following method:


```
# Mapping target variable to 1 and 0
OD['diagnosis'].replace({'M':1, 'B':0}, inplace=True)
```

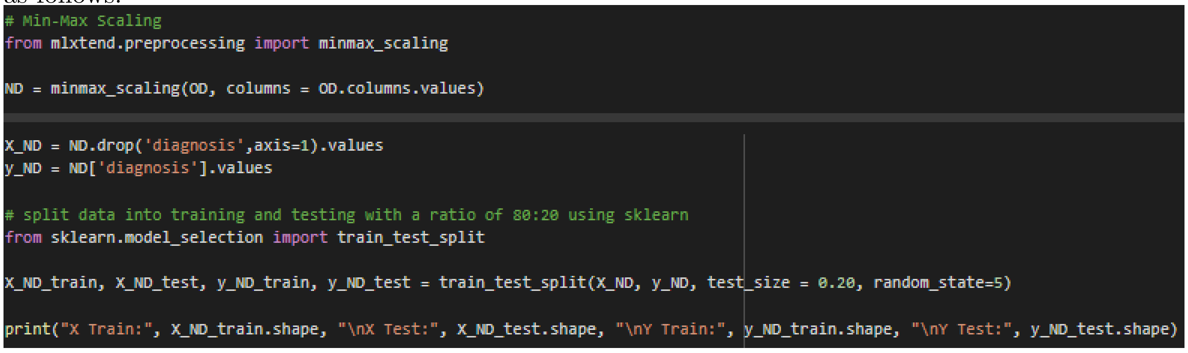
4 Data Splitting

Now I need to reserve the data for classification. Since I will use 80% of the data in my hand as a train set and 20% as a test set, I have divided it as follows:



```
# OD - Split Data

X_OD = OD.drop('diagnosis',axis=1).values
y_OD = OD['diagnosis'].values

# split data into training and testing with a ratio of 80:20 using sklearn
from sklearn.model_selection import train_test_split

X_OD_train, X_OD_test, y_OD_train, y_OD_test = train_test_split(X_OD, y_OD, test_size = 0.20, random_state=5)

print("X Train:", X_OD_train.shape, "\nX Test:", X_OD_test.shape, "\nY Train:", y_OD_train.shape, "\nY Test:", y_OD_test.shape)
```

I have Original Dataset (OD) dedicated train and test sets. It was also necessary to normalize this dataset according to the min-max normalization and keep it as a separate dataset (Normalized Dataset (ND)). I did it as follows:



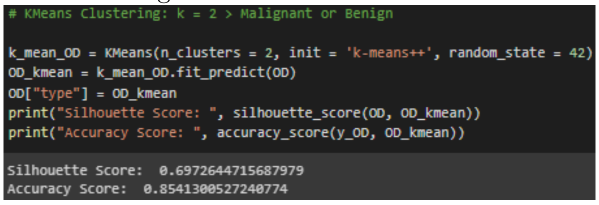
```
# Min-Max Scaling
from mlxtend.preprocessing import minmax_scaling

ND = minmax_scaling(OD, columns = OD.columns.values)
```

```
X_ND = ND.drop('diagnosis',axis=1).values
y_ND = ND['diagnosis'].values

# split data into training and testing with a ratio of 80:20 using sklearn
from sklearn.model_selection import train_test_split

X_ND_train, X_ND_test, y_ND_train, y_ND_test = train_test_split(X_ND, y_ND, test_size = 0.20, random_state=5)

print("X Train:", X_ND_train.shape, "\nX Test:", X_ND_test.shape, "\nY Train:", y_ND_train.shape, "\nY Test:", y_ND_test.shape)
```

5 Clustering

Currently, I have OD and ND datasets. I used the K-means clustering method to cluster them. Here I have given the value of 'n_cluster' 2, which is my class size.

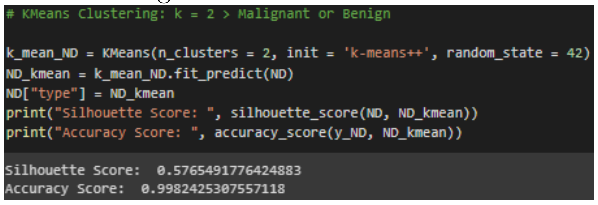
The clustering for OD:



```
# KMeans Clustering: k = 2 > Malignant or Benign

k_mean_OD = KMeans(n_clusters = 2, init = 'k-means++', random_state = 42)
OD_kmean = k_mean_OD.fit_predict(OD)
OD["type"] = OD_kmean
print("Silhouette Score: ", silhouette_score(OD, OD_kmean))
print("Accuracy Score: ", accuracy_score(y_OD, OD_kmean))
```

```
Silhouette Score:  0.6972644715687979
Accuracy Score:  0.8541300527240774
```

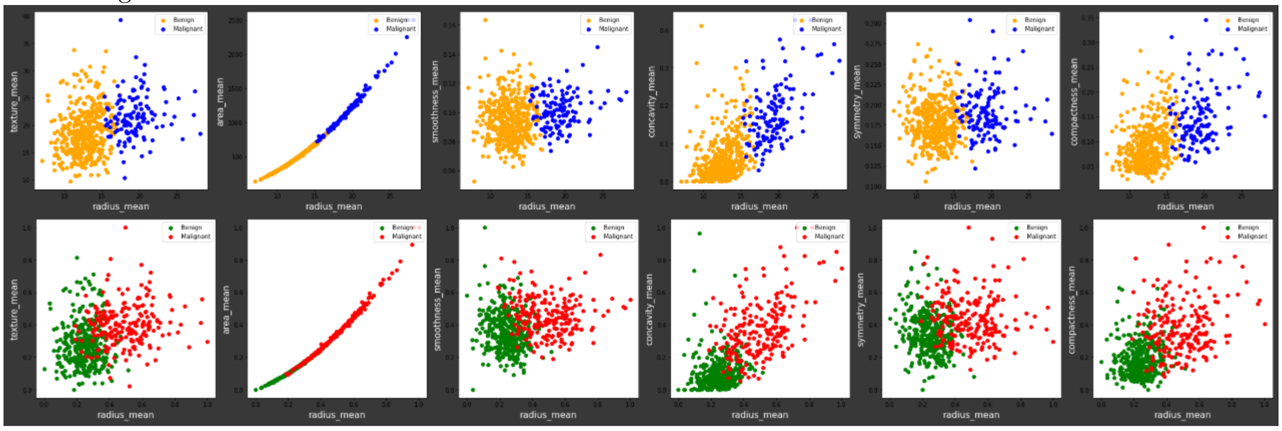
The clustering for ND:



```
# KMeans Clustering: k = 2 > Malignant or Benign

k_mean_ND = KMeans(n_clusters = 2, init = 'k-means++', random_state = 42)
ND_kmean = k_mean_ND.fit_predict(ND)
ND["type"] = ND_kmean
print("Silhouette Score: ", silhouette_score(ND, ND_kmean))
print("Accuracy Score: ", accuracy_score(y_ND, ND_kmean))
```

```
Silhouette Score:  0.5765491776424883
Accuracy Score:  0.9982425307557118
```

Visualizing Clusters For Some Features:



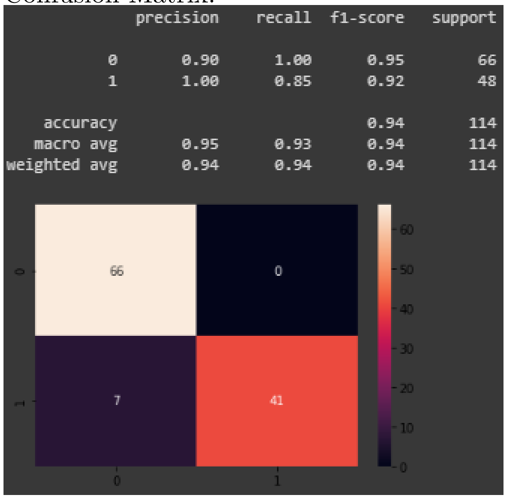
First Plots (orange-blue) are OD clusters. Second Plots (green-red) are ND clusters.

6 Classification

I have parts of OD and ND datasets devoted to train and test datasets. I created them using the Support Vector Machine algorithm and tested them with my test data.

6.1 Standard Vector Machine (SVM) for OD Dataset

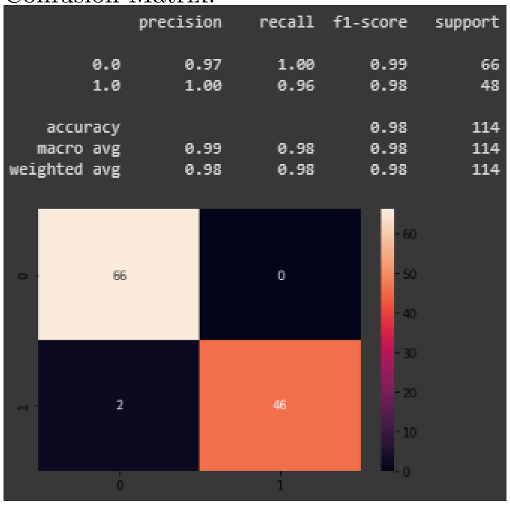
When I trained the Original Data Set with SVM, I had a nice accuracy rate of 94%.
Confusion Matrix:



```
              precision    recall  f1-score   support

           0       0.90      1.00      0.95        66
           1       1.00      0.85      0.92        48

    accuracy                           0.94       114
   macro avg       0.95      0.93      0.94       114
weighted avg       0.94      0.94      0.94       114
```

6.2 Standard Vector Machine (SVM) for ND Dataset

I achieved the highest accuracy rate of 98% with SVM among the different classification methods I train Normalized Dataset.
Confusion Matrix:



```
              precision    recall  f1-score   support

         0.0       0.97      1.00      0.99        66
         1.0       1.00      0.96      0.98        48

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```