Name : B.M.G.G.K. Rajapaksha

Index No. : S14210

Faculty : Faculty of Science, Bioinformatics

CS 4104 – Data Analytics – Assignment 2

a. Screenshots with an explanation of the tools you used for the above training process.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
recall_score, accuracy_score, precision_score
import matplotlib.pyplot as plt
```

pandas – It is a fast powerful, flexible and easy to use open source data analysis and manipulation tool which can be used for python programming. In here it is used to represent the training and testing datasets using pandas dataframe structure.

```
datasets = pd.ExcelFile('SCS4204_IS4103_CS4104 _dataset.xlsx')
training = pd.read_excel(datasets, 'Training Dataset')
testing = pd.read_excel(datasets, 'Testing Dataset')
```

Output

```
Training dataset
      ID  Age Gender    TB   DB  ALK SGPT SGOT   TP  ALB AG_Ratio Class
0      1   65  Female   0.7  0.1  187   16   18  6.8  3.3      0.9   Yes
1      2   62    Male  10.9  5.5  699   64  100  7.5  3.2     0.74   Yes
2      3   62    Male   7.3  4.1  490   60   68    7  3.3     0.89   Yes
3      4   58    Male     1  0.4  182   14   20  6.8  3.4        1   Yes
4      5   72    Male   3.9    2  195   27   59  7.3  2.4      0.4   Yes
..   ...  ...     ...   ...  ...  ...  ...  ...  ...  ...      ...   ...
578  579   60    Male   0.5  0.1  500   20   34  5.9  1.6     0.37    No
579  580   40    Male   0.6  0.1   98   35   31    6  3.2      1.1   Yes
580  581   52    Male   0.8  0.2  245   48   49  6.4  3.2        1   Yes
581  582   31    Male   1.3  0.5  184   29   32  6.8  3.4        1   Yes
582  583   38    Male     1  0.3  216   21   24  7.3  4.4      1.5    No

[583 rows x 12 columns]

Testing dataset
      ID  Age  Gender    TB   DB  ALK SGPT SGOT   TP  ALB AG_Ratio Class
0      1   65  Female   0.7  0.1  187   16   18  6.8  3.3      0.9   Yes
1      2   62    Male  10.9  5.5  699   64  100  7.5  3.2     0.74   Yes
2      3   62    Male   7.3  4.1  490   60   68    7  3.3     0.89   Yes
3      4   58    Male     1  0.4  182   14   20  6.8  3.4        1   Yes
4      5   72    Male   3.9    2  195   27   59  7.3  2.4      0.4   Yes
..   ...  ...     ...   ...  ...  ...  ...  ...  ...  ...      ...   ...
```

numpy – NumPy is a fundamental package for scientific computing with python. It supports for large, multi – dimensional arrays and matrices.

Here **np.NaN** was used which is the IEEE 754 floating-point representation for Not a Number. It was used under data pre-processing part to replace '?' values from the dataset.

```
# Replace '?' values from the mean value of the respective columns
(training dataset)

empty_not_accepted = ['Age', 'Gender', 'TB', 'DB', 'ALK', 'SGPT', 'SGOT',
'TP', 'ALB', 'AG_Ratio', 'Class']

accepted = empty_not_accepted

for column in accepted:
    if column not in ('Gender', 'Class'):
        training[column] = training[column].replace('?', np.NaN)
        mean = int(training[column].mean(skipna=True))
        training[column] = training[column].replace(np.NaN, mean)

    else:
        training[column] = training[column].replace('?', np.NaN)
        training = training.dropna(how='any', axis=0)
```

sklearn – Sklearn is a highly recommended and wide using machine learning library package for the python programming. It consists set of machine learning tools that are used in most machine learning problems.

sklearn.preprocessing – This package provide several common utility functions and transformer classes to change raw feature vectors into more suitable representation.

standardScaler – This is a class under **sklearn.preprocessing** to standardize features by removing the mean and scaling to unit variance. Standardising of the datasets is essential requirement for machine learning estimators.

standardScaler calculates standard score of a sample x (here x = training and testing datasets) as follows;

Standard score (Z) = (x-u)/s

Where;

- u = mean of the training samples or zero if with_mean = False. (in this problem, with_mean = True (default))
- s = standard deviation of the training sample or one if with_std=False (in this problem, with_std = True (default))

```
sc_X = StandardScaler()
train_x = sc_X.fit_transform(train_x)
test_x = sc_X.transform(test_x)

print('Standardized Training X data\n', train_x, '\n')
```

Output

```
Standardized Training X data
 [[ 1.25209764  1.76228085 -0.42024231 ...  0.29123986  0.20450104
  -0.1392656 ]
 [ 1.06663704 -0.56744644  1.22427792 ...  0.93754212  0.07833797
  -0.63794871]
 [ 1.06663704 -0.56744644  0.64385901 ...  0.47589765  0.20450104
  -0.17043329]
 ...
 [ 0.44843504 -0.56744644 -0.40411957 ... -0.07807572  0.07833797
   0.17241134]
 [-0.84978917 -0.56744644 -0.32350583 ...  0.29123986  0.33066412
   0.17241134]
 [-0.41704777 -0.56744644 -0.37187407 ...  0.75288433  1.59229488
   1.73079606]]
```

sklearn.model_selection – It is a method for setting a blueprint to analyse data and then using it to measure new data.

sklearn.neighbors – This provides functionality for unsupervised and supervised neighbours – based learning methods.

KNeighborsClassifier – This is the classifier that implement the k-nearest neighbours vote under sklearn.neighbors. Classification is computed from a simple majority vote of the nearest neighbours of each point. The majority voted class is selected as the class of the candidate data point.

```
# Define the model
knn = KNeighborsClassifier()
```

GridSearchCV – This is a library function under sklearn.model_selection. It is used to loop through predefined hypeparameters and fit the model (here knn) on the training dataset. At the end it gives the best set of hyperparameters from the given set.

Used parameters in GridSearchCV

- **cv** = Class validation. This determines the cross validation splitting strategy. Here 10 is used as the cv value. Hence it split the dataset into 10 equal sets. Then select one set as a testing set and others as the training set. This is repeats until it covers the all 10 folds.
- **hyperparameters;**
    - leaf_size (controls the minimum no. of points in a given node)
    - n_neighbors (number of nearest neighbours)
    - p (power parameter for the Minkkowski metric)

  are selected as hyperparameters which want to tune.

```
# List of Hyperparameters that we want to tune

leaf_size = list(range(1, 30))
n_neighbors = list(range(1, 30))
```

```python
p = [1, 2]

# get hyperparameters into a dictionary

hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)

# Define the model
knn = KNeighborsClassifier()

# Use GridSearch
clf = GridSearchCV(knn, hyperparameters, cv=10)   # cv = class validation

# here cv = 10 means we have to divide the dataset into 5 sets/folds

# Fit the model

best_model = clf.fit(train_x, train_y)

print("Estimated best hyperparameters \n")
```

Output

```
Estimated best hyperparameters

Best leaf_size: 1
Best p: 2
Best n_neighbors: 29
```

```
Predicted y values for test data
 [1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 1 1 0 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1
 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1
 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

sklearn.metrics – This module implements functions assessing prediction error for specific purposes.

confusion_matrix – Compute confusion matrix to evaluate the accuracy of a classification. According to the definition;
Confusion matrix is such that $C_{i,j}$ = no of observations known to be in group i and predicted to be in group j.

Since this problem is a binary classification,

- $C_{0,0}$ = true negative'
- $C_{1,0}$ = false negative
- $C_{1,1}$ = true positive
- $C_{0,1}$ = false positive

```
# Evaluate model
cm = confusion_matrix(test_y, pred_y)
print('Confusion matrix \n', cm, '\n')
```

Output

```
Confusion matrix
 [[ 22  68]
 [ 14 206]]
```

ConfusionMatrixDisplay – For confusion matrix visualization.

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['No',
'Yes'])
```

matplotlib – Is a plotting library for python programming.

matplotlob.pyplot – pyplot is an API(Aplication Programming Interface) for python's matplotlib that effectively makes matplotlib a viable open source alternative to MATLAB. Here these two are used to visualize the confusion matrix as a plot.

```
# Confusion matrix plot
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['No',
'Yes'])
disp.plot()
plt.show()
```

recall_score, accuracy_score, precision_score – These are used to calculate the Specificity, Accuracy and Precision of the KNN model.

```
print('Accuracy : ', round((accuracy_score(test_y, pred_y)*100), 2), '%')
print('Precision (Positive Predictive Value) : ',
round((precision_score(test_y, pred_y)*100), 2), '%')
print('Sensitivity (Hit rate/Recall) : ', round((tp/(tp+fn))*100, 2), '%')
print('Specificity : ', round((recall_score(test_y, pred_y)*100), 2), '%')
print('Error rate : ', round(((fp+fn)/(tp+fn+fp+tn))*100, 2), '%')
```

b. Brief explanation of the pre-processing steps you followed.

1. First, the nominal data columns (Gender and Class) were selected and each nominal categories were replaced by 0 and 1 in both training and testing datasets.
   - Male = 1
   - Female = 0
   - No = 0

- Yes = 1

```
# give numerical labels for 'Gender' and 'Class' attribute values in
testing dataset

training['Gender'].replace(['Male', 'Female'], [0, 1], inplace=True)
training['Class'].replace(['Yes', 'No'], [1, 0], inplace=True)

testing['Gender'].replace(['Male', 'Female'], [0, 1], inplace=True)
testing['Class'].replace(['Yes', 'No'], [1, 0], inplace=True)
```

2. Then the complete two datasets were scanned through a loop to check the '?' symbolic data entries these entries are replaced by **NaN** values. Except the **Gender** and **Class** columns, all other Columns were scanned again through a loop and replace all NaN values from the mean values of their respective column.

   Replacement of the NaN values in Gender and Class columns by their means is not meaningful. Hence these rows were completely removed from the training and testing datasets.

```
# Replace '?' values from the mean value of the respective columns
(training dataset)

empty_not_accepted = ['Age', 'Gender', 'TB', 'DB', 'ALK', 'SGPT', 'SGOT',
'TP', 'ALB', 'AG_Ratio', 'Class']

accepted = empty_not_accepted

for column in accepted:
    if column not in ('Gender', 'Class'):
        training[column] = training[column].replace('?', np.NaN)
        mean = int(training[column].mean(skipna=True))
        training[column] = training[column].replace(np.NaN, mean)

    else:
        training[column] = training[column].replace('?', np.NaN)
        training = training.dropna(how='any', axis=0)

# Replace '?' values from the mean value of the respective columns (testing
dataset)

empty_not_accepted = ['Age', 'Gender', 'TB', 'DB', 'ALK', 'SGPT', 'SGOT',
'TP', 'ALB', 'AG_Ratio', 'Class']

accepted = empty_not_accepted

for column in accepted:
    if column not in ('Gender', 'Class'):
        testing[column] = testing[column].replace('?', np.NaN)
        mean = int(testing[column].mean(skipna=True))
        testing[column] = testing[column].replace(np.NaN, mean)
    else:
        testing[column] = testing[column].replace('?', np.NaN)
        testing = testing.dropna(how='any', axis=0)
```

3. Both training and testing datasets were spit into X and Y variables**.** Only the **Class** column was selected in to the Y variable and rest of the columns except the **Id** column of the both training and testing data sets were selected to the X.

```
# Split the training dataset into x and y

train_x = training.iloc[:, 1:11]  # [all rows,column 1-11]
train_y = training.iloc[:, 11]  # [all rows,column 11]

print('First 10 values of train_x\n', train_x.head(10), '\n')
print('First 10 values of train_y\n', train_y.head(10), '\n')

# Split the testing dataset into x and y

test_x = testing.iloc[:, 1:11]  # [all rows,column 1-11]
test_y = testing.iloc[:, 11]  # [all rows,column 11]

print('First 10 values of test_x\n', test_x.head(10), '\n')
print('First 10 values of test_y\n', test_y.head(10), '\n')
```

4. Finally both training and testing X data sets were standardized by using **standardScaler()** function. Standardising of the datasets is essential requirement for machine learning estimators.

   standardScaler calculates standard score of a sample x (here x = training and testing datasets) as follows;

   Standard score (Z) = (x-u)/s

   Where;

   - u = mean of the training samples or zero if with_mean = False. (in this problem, with_mean = True (default))
   - s = standard deviation of the training sample or one if with_std=False (in this problem, with_std = True (default))

```
# Feature Scaling

sc_X = StandardScaler()
train_x = sc_X.fit_transform(train_x)
test_x = sc_X.transform(test_x)

print('Standardized Training X data\n', train_x, '\n')
```

c. Generated *Confusion matrix* for the Test dataset.

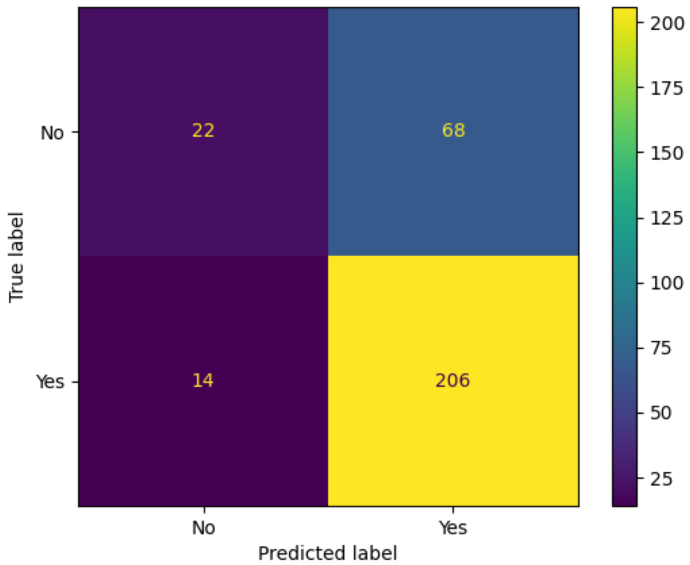
```
# predict the test set results
pred_y = clf.predict(test_x)
print('Predicted y values for test data \n', pred_y, '\n')
```

```
# Evaluate model
cm = confusion_matrix(test_y, pred_y)
print('Confusion matrix \n', cm, '\n')

# Confusion matrix plot
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['No',
'Yes'])
disp.plot()
plt.show()
```

Output

```
Confusion matrix
 [[ 22  68]
 [ 14 206]]
```



d. List of below measures calculated for the Test dataset.

I. Accuracy : 73.55%
II. Precision : 75.18%
III. Sensitivity : 24.44%
IV. Specificity : 93.64%
V. Error Rate : 26.45%

```
print('Accuracy : ', round((accuracy_score(test_y, pred_y)*100), 2), '%')
print('Precision (Positive Predictive Value) : ',
round((precision_score(test_y, pred_y)*100), 2), '%')
print('Sensitivity (Hit rate/Recall) : ', round((tp/(tp+fn))*100, 2), '%')
```

```
print('Specificity : ', round((recall_score(test_y, pred_y)*100), 2), '%')
print('Error rate : ', round(((fp+fn)/(tp+fn+fp+tn))*100, 2), '%')
```

```
Accuracy :  73.55 %
Precision (Positive Predictive Value) :  75.18 %
Sensitivity (Hit rate/Recall) :  24.44 %
Specificity :  93.64 %
Error rate :  26.45 %
```

Manual calculations

```
# tn = true negative = 22
# tp = true positive = 206
# fn = false negative = 14
# fp = false positive = 68

# Manual calculations

# print('Accuracy : ', round(((tp+tn)/(tp+fn+fp+tn))*100, 2), '%')
# print('Precision (Positive Predictive Value) : ', round((tp/(tp+fp))*100,
2), '%')
# print('Sensitivity (Hit rate/Recall) : ', round((tp/(tp+fn))*100, 2),
'%')
# print('Specificity : ', round((tn/(tn+fp))*100, 2), '%')
# print('FN Rate (Miss rate) : ', round((fn/(tp+fn))*100, 2), '%')
# print('TP Rate (False Alarm Rate) : ', round((fp/(fp+tn))*100, 2), '%')
# print('Error rate : ', round(((fp+fn)/(tp+fn+fp+tn))*100, 2), '%')
```

e. Append your full code lines at the end of the PDF file

```
"""
Name         : B.M.G.G.K. Rajapaksha
Faculty      : Faculty of Science, Bioinformatics
Index NO.    : S14210
Date         : 25/07/2022

Task         : Assignment 2; A supervised machine learning algorithm (KNN)
to train and test the
               SCS4204_IS4103_CS4104 _dataset.xlsx dataset

Input        : 1. SCS4204_IS4103_CS4104 _dataset.xlsx dataset
                  (It contains training and testing data)

Output       : 1. Accuracy
               2. Precision
               3. Sensitivity
               4. Specificity
               5. Error Rate

"""

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
```

```
recall_score, accuracy_score, precision_score
import matplotlib.pyplot as plt

# load the datasets

datasets = pd.ExcelFile('SCS4204_IS4103_CS4104 _dataset.xlsx')
training = pd.read_excel(datasets, 'Training Dataset')
testing = pd.read_excel(datasets, 'Testing Dataset')

print('Training dataset\n', training, '\n')
print('Testing dataset\n', testing, '\n')

print('Length of the testing dataset - ', len(testing))
print('Length of the training dataset - ', len(training), '\n')

# Pre processing the dataset
# convert categorical values into numerical values
# both 'Gender' and 'Class' attributes are categorical variable. Hence need
to convert them into numerical values before
# preprocessing

# give numerical labels for 'Gender' and 'Class' attribute values in
testing dataset

training['Gender'].replace(['Male', 'Female'], [0, 1], inplace=True)
training['Class'].replace(['Yes', 'No'], [1, 0], inplace=True)

testing['Gender'].replace(['Male', 'Female'], [0, 1], inplace=True)
testing['Class'].replace(['Yes', 'No'], [1, 0], inplace=True)

# Replace '?' values from the mean value of the respective columns
(training dataset)

empty_not_accepted = ['Age', 'Gender', 'TB', 'DB', 'ALK', 'SGPT', 'SGOT',
'TP', 'ALB', 'AG_Ratio', 'Class']

accepted = empty_not_accepted

for column in accepted:
    if column not in ('Gender', 'Class'):
        training[column] = training[column].replace('?', np.NaN)
        mean = int(training[column].mean(skipna=True))
        training[column] = training[column].replace(np.NaN, mean)

    else:
        training[column] = training[column].replace('?', np.NaN)
        training = training.dropna(how='any', axis=0)

# Replace '?' values from the mean value of the respective columns (testing
dataset)

empty_not_accepted = ['Age', 'Gender', 'TB', 'DB', 'ALK', 'SGPT', 'SGOT',
'TP', 'ALB', 'AG_Ratio', 'Class']

accepted = empty_not_accepted

for column in accepted:
    if column not in ('Gender', 'Class'):
        testing[column] = testing[column].replace('?', np.NaN)
        mean = int(testing[column].mean(skipna=True))
        testing[column] = testing[column].replace(np.NaN, mean)
```

```
    else:
        testing[column] = testing[column].replace('?', np.NaN)
        testing = testing.dropna(how='any', axis=0)

# Split the training dataset into x and y

train_x = training.iloc[:, 1:11]  # [all rows,column 1-11]
train_y = training.iloc[:, 11]  # [all rows,column 11]

print('First 10 values of train_x\n', train_x.head(10), '\n')
print('First 10 values of train_y\n', train_y.head(10), '\n')

# Split the testing dataset into x and y

test_x = testing.iloc[:, 1:11]  # [all rows,column 1-11]
test_y = testing.iloc[:, 11]  # [all rows,column 11]

print('First 10 values of test_x\n', test_x.head(10), '\n')
print('First 10 values of test_y\n', test_y.head(10), '\n')

# Feature Scaling

sc_X = StandardScaler()
train_x = sc_X.fit_transform(train_x)
test_x = sc_X.transform(test_x)

print('Standardized Training X data\n', train_x, '\n')

# hyperparameter tuning

from sklearn.model_selection import GridSearchCV

# List of Hyperparameters that we want to tune

leaf_size = list(range(1, 30))
n_neighbors = list(range(1, 30))
p = [1, 2]

# get hyperparameters into a dictionary

hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)

# Define the model
knn = KNeighborsClassifier()

# Use GridSearch
clf = GridSearchCV(knn, hyperparameters, cv=10)  # cv = class validation

# here cv = 10 means we have to divide the dataset into 5 sets/folds

# Fit the model

best_model = clf.fit(train_x, train_y)

print("Estimated best hyperparameters \n")

print('Best leaf_size:',
best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:',
best_model.best_estimator_.get_params()['n_neighbors'], '\n')
```

```
# predict the test set results
pred_y = clf.predict(test_x)
print('Predicted y values for test data \n', pred_y, '\n')

# Evaluate model
cm = confusion_matrix(test_y, pred_y)
print('Confusion matrix \n', cm, '\n')

# Confusion matrix plot
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['No',
'Yes'])
disp.plot()
plt.show()

tn, fp, fn, tp = cm.ravel()

# tn = true negative
# tp = true positive
# fn = false negative
# fp = false positive

# Manual calculations

# print('Accuracy : ', round(((tp+tn)/(tp+fn+fp+tn))*100, 2), '%')
# print('Precision (Positive Predictive Value) : ', round((tp/(tp+fp))*100,
2), '%')
# print('Sensitivity (Hit rate/Recall) : ', round((tp/(tp+fn))*100, 2),
'%')
# print('Specificity : ', round((tn/(tn+fp))*100, 2), '%')
# print('FN Rate (Miss rate) : ', round((fn/(tp+fn))*100, 2), '%')
# print('TP Rate (False Alarm Rate) : ', round((fp/(fp+tn))*100, 2), '%')
# print('Error rate : ', round(((fp+fn)/(tp+fn+fp+tn))*100, 2), '%')

# same calculations using sklearn

print('Accuracy : ', round((accuracy_score(test_y, pred_y)*100), 2), '%')
print('Precision (Positive Predictive Value) : ',
round((precision_score(test_y, pred_y)*100), 2), '%')
print('Sensitivity (Hit rate/Recall) : ', round((tp/(tp+fn))*100, 2), '%')
print('Specificity : ', round((recall_score(test_y, pred_y)*100), 2), '%')
print('Error rate : ', round(((fp+fn)/(tp+fn+fp+tn))*100, 2), '%')
```

S14210

Output

```
Training dataset
       ID  Age  Gender    TB   DB  ALK SGPT SGOT   TP  ALB AG_Ratio Class
0       1   65  Female   0.7  0.1  187   16   18  6.8  3.3      0.9   Yes
1       2   62    Male  10.9  5.5  699   64  100  7.5  3.2     0.74   Yes
2       3   62    Male   7.3  4.1  490   60   68    7  3.3     0.89   Yes
3       4   58    Male     1  0.4  182   14   20  6.8  3.4        1   Yes
4       5   72    Male   3.9    2  195   27   59  7.3  2.4      0.4   Yes
..    ...  ...     ...   ...  ...  ...  ...  ...  ...  ...      ...   ...
578   579   60    Male   0.5  0.1  500   20   34  5.9  1.6     0.37    No
579   580   40    Male   0.6  0.1   98   35   31    6  3.2      1.1   Yes
580   581   52    Male   0.8  0.2  245   48   49  6.4  3.2        1   Yes
581   582   31    Male   1.3  0.5  184   29   32  6.8  3.4        1   Yes
582   583   38    Male     1  0.3  216   21   24  7.3  4.4      1.5    No

[583 rows x 12 columns]

Testing dataset
       ID  Age  Gender    TB   DB  ALK SGPT SGOT   TP  ALB AG_Ratio Class
0       1   65  Female   0.7  0.1  187   16   18  6.8  3.3      0.9   Yes
1       2   62    Male  10.9  5.5  699   64  100  7.5  3.2     0.74   Yes
2       3   62    Male   7.3  4.1  490   60   68    7  3.3     0.89   Yes
3       4   58    Male     1  0.4  182   14   20  6.8  3.4        1   Yes
4       5   72    Male   3.9    2  195   27   59  7.3  2.4      0.4   Yes
..    ...  ...     ...   ...  ...  ...  ...  ...  ...  ...      ...   ...
306   307   52    Male   2.7    ?  251   20   40    6  1.7     0.39   Yes
307   308   90    Male   1.1  0.3  215   46  134  6.9    3      0.7   Yes
308   309   45  Female  15.6  9.5  134   54  125  5.6    4      2.5   Yes
309   310   52    Male   0.8  0.2  245   48   49  6.4  3.2        1   Yes
310   311   45    Male   1.3  0.5  184   29   32  6.8  3.4        1   Yes
```

S14210

```
Length of the testing dataset -  311
Length of the training dataset -  583

First 10 values of train_x
    Age  Gender    TB   DB    ALK  SGPT   SGOT   TP  ALB  AG_Ratio
0   65       1   0.7  0.1  187.0  16.0   18.0  6.8  3.3      0.90
1   62       0  10.9  5.5  699.0  64.0  100.0  7.5  3.2      0.74
2   62       0   7.3  4.1  490.0  60.0   68.0  7.0  3.3      0.89
3   58       0   1.0  0.4  182.0  14.0   20.0  6.8  3.4      1.00
4   72       0   3.9  2.0  195.0  27.0   59.0  7.3  2.4      0.40
5   46       0   1.8  0.7  208.0  19.0   14.0  7.6  4.4      1.30
6   26       1   0.9  0.2  154.0  16.0   12.0  7.0  3.5      1.00
7   29       1   0.9  0.3  202.0  14.0   11.0  6.7  3.6      1.10
8   17       0   0.9  0.3  202.0  22.0   19.0  7.4  4.1      1.20
9   55       0   0.7  0.2  290.0  53.0   58.0  6.8  3.4      1.00

First 10 values of train_y
 0    1
1    1
2    1
3    1
4    1
5    1
6    1
7    1
8    0
9    1

First 10 values of test_x
    Age  Gender    TB   DB    ALK  SGPT   SGOT   TP  ALB  AG_Ratio
0   65       1   0.7  0.1  187.0  16.0   18.0  6.8  3.3      0.90
1   62       0  10.9  5.5  699.0  64.0  100.0  7.5  3.2      0.74
2   62       0   7.3  4.1  490.0  60.0   68.0  7.0  3.3      0.89
3   58       0   1.0  0.4  182.0  14.0   20.0  6.8  3.4      1.00
4   72       0   3.9  2.0  195.0  27.0   59.0  7.3  2.4      0.40
5   30       1   0.9  0.3  202.0  15.0   11.0  6.7  3.1      1.10
6   17       0   0.9  0.3  277.0  22.0   19.0  7.4  4.1      1.20
7   55       0   0.7  0.2  290.0  53.0   58.0  6.8  3.4      1.00
8   64       0   0.9  0.3  310.0  61.0   58.0  7.0  3.4      0.90
9   25       0   0.6  0.1  183.0  91.0   53.0  5.5  2.3      0.70

First 10 values of test_y
 0    1
1    1
2    1
3    1
4    1
5    1
6    0
7    1
8    0
9    0

Standardized Training X data
 [[ 1.25209764  1.76228085 -0.42024231 ...  0.29123986  0.20450104
   -0.1392656 ]
 [ 1.06663704 -0.56744644  1.22427792 ...  0.93754212  0.07833797
   -0.63794871]
 [ 1.06663704 -0.56744644  0.64385901 ...  0.47589765  0.20450104
   -0.17043329]
 ...
 [ 0.44843504 -0.56744644 -0.40411957 ... -0.07807572  0.07833797
    0.17241134]
 [-0.84978917 -0.56744644 -0.32350583 ...  0.29123986  0.33066412
    0.17241134]
 [-0.41704777 -0.56744644 -0.37187407 ...  0.75288433  1.59229488
    1.73079606]]

Estimated best hyperparameters

Best leaf_size: 1
Best p: 2
Best n_neighbors: 29
```

S14210

```
Predicted y values for test data
 [1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 1 1 0 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1
 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1
 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

Confusion matrix
 [[ 22  68]
 [ 14 206]]

Accuracy :  73.55 %
Precision (Positive Predictive Value) :  75.18 %
Sensitivity (Hit rate/Recall) :  93.64 %
Specificity :  93.64 %
Error rate :  26.45 %
```