Name : D. R. R. Wijewardene

Index No. : S14245

Faculty : Faculty of Science (Bioinformatics)

CS 4104 – Data Analytics

Assignment 02

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.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
```

- **pandas** – Pandas is a library of data analysis and manipulation tools.
  - In this implementation, it was used to open an excel file which consisted of the training and testing datasets in two separate sheets. Then each dataset was assigned to two variables by reading the respective excel sheet.

```
# Load dataset
with pd.ExcelFile('SCS4204_IS4103_CS4104 _dataset.xlsx') as dataset:
    training_dataset = pd.read_excel(dataset, sheet_name='Training Dataset')
    testing_dataset = pd.read_excel(dataset, sheet_name='Testing Dataset')
```

- **numpy** – NumPy is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays.
  - In this implementation, it was used to represent that the missing values (“?”) were NaN values (Not a Number) so that they can be easily manipulated.

```python
# Replace the '?' of missing values with the column mean
# Remove the rows with missing values for Gender or Class attributes
for column in training_dataset.columns:
    if column not in ("Gender", "Class"):
        # training data
        training_dataset[column] = training_dataset[column].replace("?", np.NaN)
        train_mean = int(training_dataset[column].mean(skipna=True))
        training_dataset[column] = training_dataset[column].replace(np.NaN, train_mean)
        # testing data
        testing_dataset[column] = testing_dataset[column].replace("?", np.NaN)
        test_mean = int(testing_dataset[column].mean(skipna=True))
        testing_dataset[column] = testing_dataset[column].replace(np.NaN, test_mean)
    else:
        training_dataset[column] = training_dataset[column].replace("?", np.NaN)
        training_dataset = training_dataset.dropna()
        testing_dataset[column] = testing_dataset[column].replace("?", np.NaN)
        testing_dataset = testing_dataset.dropna()
```

- **sklearn** – Scikit-learn is a library which provides a collection of simple and efficient tools for predictive data analysis.

  For this implementation, many tools were imported from the sklearn library.

  - **sklearn.neighbors** – The neighbors module implements the k-nearest neighbors algorithm, where the goal is to find a predefined number of training samples closest in distance to the new query, and predict the label.
    - **KNeighborsClassifier** – This is a tool which is used to create a classifier to implement the k-nearest neighbors vote according to a *k* integer value specified by the user. Learning is implemented based on a *k* number of nearest neighbors of each query point.

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

# Use gridsearch
classifier = GridSearchCV(knn, hyperparameters, cv=10)

# Fit the model
best_model = classifier.fit(x_train, y_train)

# Best hyperparameters
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 (k) :', best_model.best_estimator_.get_params()['n_neighbors'])

# Predict the test set results
y_pred = classifier.predict(x_test)
```

- **sklearn.preprocessing** – The preprocessing module consists of scaling, centering, normalization and binarization methods.
    - **StandardScaler** – This is a tool which is used to standardize features by removing the mean and scaling to unit variance.

```
# Feature scaling
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

- **sklearn.model_selection** – The model selection module consists of tools for cross-validation (evaluating estimator performance), tuning the hyper-parameters of an estimator etc.
    - **GridSearchCV** – This is a tool which does an exhaustive search over specified parameter values for an estimator.

      In this scenario, the given parameters were a range of leaf sizes, a range of k values (n_neighbors) and two different distance measures (p=1 is Manhattan and p=2 is Euclidean). This module allows the model to be created with every possible combination of parameters and provides the results of the best parameter combination.

```
# Use gridsearch
classifier = GridSearchCV(knn, hyperparameters, cv=10)

# Fit the model
best_model = classifier.fit(x_train, y_train)

# Best hyper-parameters
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 (k) :', best_model.best_estimator_.get_params()['n_neighbors'])
```

Output:

```
Best leaf_size : 1
Best p : 2
Best n_neighbors (k) : 29
```

- **sklearn.metrics** – The metrics module consists of tools related to score functions, performance metrics, pairwise metrics and distance computations.
    - **confusion_matrix** – This is a tool that is used to compute the confusion matrix, which can in turn evaluate the accuracy of the classification.

```
cm = confusion_matrix(y_test, y_pred)
```

    - **accuracy_score** – This is a tool which calculates the accuracy classification score based on whether the predicted values match the true values exactly.

```
accuracy = accuracy_score(y_test, y_pred)
```

    - **precision_score** – This is a tool which computes the precision, or the ability of the classifier not to label a negative sample as positive.

```
precision = precision_score(y_test, y_pred)
```

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

- **<u>Missing value replacement</u>**
    - The missing values were denoted by "?" in this dataset.
    - Such missing values of the attributes Age, TB, DB, ALK, SGPT, SGOT, TP, ALB and AG_Ratio were replaced by the mean value of each respective column.
    - However, since Gender and Class attributes contained categorical values, they cannot be replaced by a mean. Also, since the Class attribute was the label, it was mandatory for the data point to be used in the learning/testing process. Therefore, rows with missing data points in these two rows were removed from the dataset.
    - In both the replacement and removal processes, first the "?" was replaced by NaN (to show that the value is Null) for ease of manipulation.

```python
# Replace the '?' of missing values with the column mean
# Remove the rows with missing values for Gender or Class attributes
for column in training_dataset.columns:
    if column not in ("Gender", "Class"):
        # training data
        training_dataset[column] = training_dataset[column].replace("?", np.NaN)
        train_mean = int(training_dataset[column].mean(skipna=True))
        training_dataset[column] = training_dataset[column].replace(np.NaN, train_mean)
        # testing data
        testing_dataset[column] = testing_dataset[column].replace("?", np.NaN)
        test_mean = int(testing_dataset[column].mean(skipna=True))
        testing_dataset[column] = testing_dataset[column].replace(np.NaN, test_mean)
    else:
        training_dataset[column] = training_dataset[column].replace("?", np.NaN)
        training_dataset = training_dataset.dropna()
        testing_dataset[column] = testing_dataset[column].replace("?", np.NaN)
        testing_dataset = testing_dataset.dropna()
```

- **Converting nominal-valued attributes to numerical**
    - Since Gender and Class attributes had nominal categorical values, they were converted to numerical values.
    - In the Gender attribute, ‘Male’ was changed to 1 and ‘Female’ to 0. This conversion was necessary to train the dataset.
    - In the Class attribute, ‘Yes’ was changed to 1 and ‘No’ to 0. This conversion was not necessary for the learning process or for the prediction since the nominal values could be directly used as labels. However, to use the precision_score tool in order to find the precision of the predictions, the labels had to be numerical. Hence the conversion was done.

```python
# Convert nominal attributes to numerical attributes
def convert_to_numeric(x):
    if x == 'Male' or x == 'Yes':
        return 1
    if x == 'Female' or x == 'No':
        return 0


# Convert the gender attribute to numerical values
training_dataset['Gender'] = training_dataset['Gender'].apply(convert_to_numeric)
testing_dataset['Gender'] = testing_dataset['Gender'].apply(convert_to_numeric)

# Convert the class attribute to numerical values
training_dataset['Class'] = training_dataset['Class'].apply(convert_to_numeric)
testing_dataset['Class'] = testing_dataset['Class'].apply(convert_to_numeric)
```

- **<u>Feature scaling</u>**
  - In this multidimensional feature space, all features being on the same scale helps to locate data points. Hence the Standard Scaler tool was used to standardize the feature attribute values before the training process.

```
# Feature scaling
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

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

Output:

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

According to the default confusion matrix output of the Sci-Kit Learn module;

		Predicted Label	
		0	**1**
Actual Label	**0**	*True Negative (TN)* **22**	*False Positive (FP)* **68**
	1	*False Negative (FN)* **14**	*True Positive (TP)* **206**

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

i. **Accuracy**
ii. **Precision**
iii. **Sensitivity**
iv. **Specificity**
v. **Error Rate**

Output:

```
Accuracy    : 73.55  %
Precision   : 75.18  %
Sensitivity : 93.64  %
Specificity : 24.44  %
Error Rate  : 26.45  %
```

Manual Calculations:

TN = 22 FP = 68 FN = 14 TP = 206 P+N = 310

$$\textbf{Accuracy} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} = \frac{206 + 22}{310} = \frac{228}{310} = 0.7355 \rightarrow \mathbf{73.55}\,\%$$

$$\textbf{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = \frac{206}{206 + 68} = \frac{206}{274} = 0.7518 \rightarrow \mathbf{75.18}\,\%$$

$$\textbf{Sensitivity} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = \frac{206}{206 + 14} = \frac{206}{220} = 0.9364 \rightarrow \mathbf{93.64}\,\%$$

$$\textbf{Specificity} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = \frac{22}{22 + 68} = \frac{22}{90} = 0.2444 \rightarrow \mathbf{24.44}\,\%$$

$$\textbf{Error Rate} = \frac{\mathrm{FP} + \mathrm{FN}}{\mathrm{FP} + \mathrm{FN} + \mathrm{TP} + \mathrm{TN}} = \frac{68 + 14}{310} = \frac{82}{310} = 0.2645 \rightarrow \mathbf{26.45}\,\%$$

e. Append your full code lines.

```
"""
CS 4104 - Assignment 02
Name         : D. R. R. Wijewardene
Index No.    : S14245
"""

import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score

# Load dataset
with pd.ExcelFile('SCS4204_IS4103_CS4104 _dataset.xlsx') as dataset:
    training_dataset = pd.read_excel(dataset, sheet_name='Training Dataset')
    testing_dataset = pd.read_excel(dataset, sheet_name='Testing Dataset')

# Replace the '?' of missing values with the column mean
# Remove the rows with missing values for Gender or Class attributes
for column in training_dataset.columns:
    if column not in ("Gender", "Class"):
        # training data
        training_dataset[column] = training_dataset[column].replace("?",
np.NaN)
        train_mean = int(training_dataset[column].mean(skipna=True))
        training_dataset[column] = training_dataset[column].replace(np.NaN,
train_mean)
        # testing data
        testing_dataset[column] = testing_dataset[column].replace("?",
np.NaN)
        test_mean = int(testing_dataset[column].mean(skipna=True))
        testing_dataset[column] = testing_dataset[column].replace(np.NaN,
test_mean)
    else:
        training_dataset[column] = training_dataset[column].replace("?",
np.NaN)
        training_dataset = training_dataset.dropna()
        testing_dataset[column] = testing_dataset[column].replace("?",
np.NaN)
        testing_dataset = testing_dataset.dropna()
```

```python
# Convert nominal attributes to numerical attributes
def convert_to_numeric(x):
    if x == 'Male' or x == 'Yes':
        return 1
    if x == 'Female' or x == 'No':
        return 0


# Convert the gender attribute to numerical values
training_dataset['Gender'] =
training_dataset['Gender'].apply(convert_to_numeric)
testing_dataset['Gender'] =
testing_dataset['Gender'].apply(convert_to_numeric)

# Convert the class attribute to numerical values
training_dataset['Class'] =
training_dataset['Class'].apply(convert_to_numeric)
testing_dataset['Class'] = testing_dataset['Class'].apply(convert_to_numeric)

# Split and extract training data and labels
x_train = training_dataset.iloc[:, 1:-1]
y_train = training_dataset.iloc[:, 11]

# Split and extract testing data and labels
x_test = testing_dataset.iloc[:, 1:-1]
y_test = testing_dataset.iloc[:, 11]

# Feature scaling
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# Hyper-parameters
leaf_size = list(range(1, 50))
n_neighbors = list(range(1, 30))
p = [1, 2]
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)

# Define model
knn = KNeighborsClassifier()

# Use gridsearch
classifier = GridSearchCV(knn, hyperparameters, cv=10)
```

```python
# Fit the model
best_model = classifier.fit(x_train, y_train)

# Best hyper-parameters
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 (k) :',
best_model.best_estimator_.get_params()['n_neighbors'])

# Predict the test set results
y_pred = classifier.predict(x_test)

# Evaluate model

cm = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
error_rate = 1 - accuracy

print("\nConfusion matrix : \n", cm)
print("\nAccuracy\t: %0.2f " % (accuracy * 100), "%")
print("Precision\t: %0.2f " % (precision * 100), "%")
print("Sensitivity\t: %0.2f " % (sensitivity * 100), "%")
print("Specificity\t: %0.2f " % (specificity * 100), "%")
print("Error Rate\t: %0.2f " % (error_rate * 100), "%")
```

Output:

```
Best leaf_size : 1
Best p : 2
Best n_neighbors (k) : 29

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

Accuracy    : 73.55  %
Precision   : 75.18  %
Sensitivity : 93.64  %
Specificity : 24.44  %
Error Rate  : 26.45  %
```