SCS3201 – Machine Learning and Neural Computing

Assignment 1

B.A.A.D. Bandara

Index Number – 19000091

Table of Contents

Table of Contents 2

1. Multivariate Linear Regression 3

1.1. Introduction to Data Set 3

1.2. Data Preprocessing 3

1.3. Model Fitting 4

1.4. Evaluation 5

2. Logistic Regression 6

2.1. Introduction to Data Set 6

2.2. Data Preprocessing 6

2.3. Model Fitting 7

2.4. Evaluation 8

1. Multivariate Linear Regression

1.1. Introduction to Data Set

Data set: Concrete Compressive Strength Data Set

https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength

Number of instances 1030

Number of attributes 9

This data set can be used to predict the Concrete Compressive Strength using following 8 input attributes.

Input attributes

- Cement -- quantitative -- kg in a m3 mixture
- Blast Furnace Slag -- quantitative -- kg in a m3 mixture
- Fly Ash -- quantitative -- kg in a m3 mixture
- Water -- quantitative -- kg in a m3 mixture
- Super plasticizer -- quantitative -- kg in a m3 mixture
- Coarse Aggregate -- quantitative -- kg in a m3 mixture
- Fine Aggregate -- quantitative -- kg in a m3 mixture
- Age -- quantitative -- Day (1~365)

Output attribute

- Concrete compressive strength -- quantitative – Mpa

1.2. Data Preprocessing

After analyzing the data set, I found that the data set did not have any missing values. For make sure there are no missing values I remove it also according to following

19000091_LogisticRegression.py 19000091_LinearRegression.py

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#data imported to the file. Data gathering
concrete_compression_strength = pd.read_excel("Concrete_Data.xls")

#if there are any missing values remove them
concrete_compression_strength = concrete_compression_strength.dropna()

```

Also all the variables are of real numbers and are not categorical. So, the data set was ready to be used do a linear regression.

1.3. Model Fitting

First all the input attributes set into the X axis. Output (predicting) attribute set into the Y axis.

Then we split the data set random as 75% percent of data as training data and 25% percent of data as testing data

Configuration parameters:

Learning rate (α): 0.0001

After that trained the model with Linear Regression with xTrain data and yTrain data

Codes are below about model fitting

```python
#Data preprocessing is not required because all the attribute values have real/integer values

x = concrete_compression_strength.iloc[ : , : 8 ]
y = concrete_compression_strength.iloc[ : , 8 ]


#seperating traning and testing data randomly from the data frame
from sklearn.model_selection import train_test_split

xTrain , xTest , yTrain , yTest = train_test_split( x , y )


#Training the Linear regression Model using training data
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

trainedDataModel = make_pipeline( StandardScaler() , SGDRegressor( alpha = 0.0001 ) ).fit( xTrain , yTrain )
```

1.4. Evaluation

Then predict the results with using trained data model and xTest data.

For evaluate the accuracy of trained model first calculate Mean Accuracy by using xTest, yTest and trained data model.

Also calculated Root Mean Squared Error that relative for the data set. Therefore also calculated the Normalized Root Mean Squared Error to evaluate the data model.

Codes are below about evaluation.

```
#Predicting results for xTest variables using trained data model
predictedResults = trainedDataModel.predict( xTest )

#Testing accuracy of trained model
meanAccuracy = trainedDataModel.score( xTest , yTest )
print( "Accuracy of model : " , meanAccuracy * 100 )


#Calculating mean squared error
from sklearn.metrics import mean_squared_error
meanSquaredError = mean_squared_error( yTest , predictedResults )
print( "Root mean squared error : " , meanSquaredError** ( 0.5 ) )

maxValue = max(concrete_compression_strength['concrete_compre_stren'])
minValue = min(concrete_compression_strength['concrete_compre_stren'])
print( "Normalized Root mean squared error : " , (meanSquaredError** ( 0.5 ))/(maxValue-minValue)  )
```

```
Notes/3rd Year 1st Semester/SCS3201 - Machine
Learning and Neural Computing/Assignment 01')
Accuracy of model :  60.48925091266153
Root mean squared error :  10.921730741883927
Normalized Root mean squared error :
0.1360668021276686
```

According to those, predictions of these trained model absolutely not correct. Accuracy more than 60. It means lot of predictions can be correct, but we can't guarantee that very well. Normalized root mean squared error is about 0.136. That is a good value. However this get smaller model will be better in predictions

2. Logistic Regression

2.1. Introduction to Data Set

Data set: Car Evaluation Data Set

https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

Number of instances 1728

Number of attributes 6

This data set can be used to predict the Car Class (Evaluate) using following 6 input attributes.

Input attributes

- Buying price: vhigh, high, med, low.
- Maintenance price: vhigh, high, med, low.
- Number of doors: 2, 3, 4, 5more.
- Person capacity: 2, 4, more.
- The size of luggage boot: small, med, big.
- Estimated safety of the car: low, med, high.

Output variables

- Car class – qualitative: unacc, acc, good, vgood

2.2. Data Preprocessing

After analyzing the data set, I found that the data set did not have any missing values. For make sure there are no missing values I remove it also according to following

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#data imported to the file. Data gathering
car_evaluvation = pd.read_csv("car.csv")

#if there are any missing values remove them
car_evaluvation = car_evaluvation.dropna()
```

But all the variables attributes were categorical. So, I had to convert them to numerical values by using dictionaries and encoding the values in all the variables as following

```
#data preprocessing
buyingAndmaintDataClassification ={
    'vhigh' : 3,
    'high' : 2,
    'med' : 1,
    'low' : 0
    }

car_evaluvation['buying'] = car_evaluvation['buying'].apply( lambda id : buyingAndmaintDataClassification [id])

car_evaluvation['maint'] = car_evaluvation['maint'].apply( lambda id : buyingAndmaintDataClassification [id])


doorsDataClassification ={
    '2' : 2,
    '3' : 3,
    '4' : 4,
    '5more' : 5
    }

car_evaluvation['doors'] = car_evaluvation['doors'].apply( lambda id : doorsDataClassification [id])


personsDataClassification ={
    '2' : 2,
    '4' : 4,
    'more' : 5
    }

car_evaluvation['persons'] = car_evaluvation['persons'].apply( lambda id : personsDataClassification [id])
```

```
lug_bootDataClassification ={
    'small' : 1,
    'med' : 2,
    'big' : 3
    }

car_evaluvation['lug_boot'] = car_evaluvation['lug_boot'].apply( lambda id : lug_bootDataClassification [id])


safetyDataClassification ={
    'low' : 1,
    'med' : 2,
    'high' : 3
    }

car_evaluvation['safety'] = car_evaluvation['safety'].apply( lambda id : safetyDataClassification [id])


car_classDataClassification ={
    'unacc' : 1,
    'acc' : 2,
    'good' : 3,
    'vgood' : 4
    }

car_evaluvation['car_class'] = car_evaluvation['car_class'].apply( lambda id : car_classDataClassification [id])
```

2.3. Model Fitting

First all the input attributes set into the X axis. Output (predicting) attribute set into the Y axis.

Then we split the data set random as 75% percent of data as training data and 25% percent of data as testing data

After that trained the model with Linear Regression with xTrain data and yTrain data

Codes are below about model fitting

```
x = car_evaluvation.iloc[ : , : 6 ]
y = car_evaluvation.iloc[ : , 6 ]


#seperating traning and testing data randomly from the data frame

from sklearn.model_selection import train_test_split

xTrain , xTest , yTrain , yTest = train_test_split( x , y )


#Training the Logistic regression Model using training data
from sklearn.linear_model import SGDClassifier

trainedDataModel = SGDClassifier( ).fit( xTrain , yTrain )
```

2.4. Evaluation

Then predict the results with using trained data model and xTest data.

For evaluate the accuracy of trained model first calculate Mean Accuracy by using xTest, yTest and trained data model.

Also get an idea about data model get a classification report by using yTest data and predicted results which get by using trained data model and xTest data

Codes are below about evaluation.

```
#Predicting results for xTest variables using trained data model
predictedResults = trainedDataModel.predict( xTest )


#Testing accuracy of trained model
meanAccuracy = trainedDataModel.score( xTest , yTest )
print( "Accuracy of model : " , meanAccuracy * 100 , " %" )


#Generating prediction report
from sklearn.metrics import classification_report
predictionReport = classification_report( yTest , predictedResults )
print( "Prediction report : " )
print( predictionReport )
```

```
Accuracy of model :  85.18518518518519  %
Prediction report :
              precision    recall  f1-score   support

           1       0.92      0.92      0.92       297
           2       0.70      0.68      0.69        95
           3       0.71      0.57      0.63        21
           4       0.74      0.89      0.81        19

    accuracy                           0.85       432
   macro avg       0.77      0.77      0.76       432
weighted avg       0.85      0.85      0.85       432
```

According to those, predictions of these trained model absolutely not correct. Accuracy more than 85 percent. It means more than lot of predictions can be correct, but we can't guarantee that every prediction is correct. There can be wrong predictions.

The classification report gives us following evaluations about trained data model's predicting skills.

- Precision – We can see that what percent of models predictions were correct
- Recall – We can see that what percent of the positive cases the model caught
- F1 score – We can see that what present of positive predictions were correct
- Support – The number of actual occurrences of the class in the predicted dataset.