Homework 6

Group 45

Prachi Patel

Jignasuben Vekariya

424-394-6096 (Tel. of Prachi)

857-262-0803 (Tel of Jignasu)

patel.prachi2@northeastern.edu

vekariya.j@northeastern.edu

Percentage of Effort Contributed by Student 1: 50%

Percentage of Effort Contributed by Student 2: 50%

Signature of Student 1: Prachi Patel

Signature of Student 2: J.R.Vekariya

Submission Date: 04/27/2022

Homework_6 (1) (1)

April 27, 2022

1 Homework 6

Before you start: Read Chapter 10 Logistic Regression and Chapter 11 Neural Networks in the textbook.

Note: Please enter the code along with your comments in the **TODO** section.

Alternative solutions are always welcomed.

[ ]:
```
# # Please remove # and run the following code if you have an error while␣
↪importing the dataset
# !pip install --upgrade openpyxl
```

1.1 Part 1: Logistic Regression

1.1.1 Problem 1 - Financial Condition of Banks

The file **Banks.csv** includes data on a sample of 20 banks.

The "Financial Condition" column records the judgment of an expert on the financial condition of each bank. This response variable takes one of two possible values—weak or strong—according to the financial condition of the bank.

The predictors are two ratios used in the financial analysis of banks: TotLns&Lses/Assets is the ratio of total loans and leases to total assets and TotExp/Assets is the ratio of total expenses to total assets.

The target is to classify the financial condition of a new bank using the two ratios.

[126]:
```
import plotly.graph_objs as go
import plotly.express as px
from plotly.subplots import make_subplots

import pandas as pd              # Loading the required packages required for␣
↪analysis
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import math
import random
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Ridge

from collections import Counter
from sklearn.neural_network import MLPRegressor

import scipy.stats as ss
import sklearn.preprocessing as sp

from sklearn.multiclass import OneVsOneClassifier

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

from sklearn import svm
from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score,r2_score,mean_squared_error


from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer


#\
#!pip3 install catboost
import warnings
warnings.filterwarnings('ignore')
sns.set_style('darkgrid')
cmap = sns.cm.mako_r
%matplotlib inline

from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import␣
↪BaggingRegressor,AdaBoostRegressor,GradientBoostingRegressor
from sklearn.naive_bayes import MultinomialNB
from sklearn.decomposition import PCA
```

```
from sklearn.pipeline import make_pipeline
from sklearn.datasets import load_iris

from six.moves import input

!pip install mord
!pip install dmba
from dmba import classificationSummary, gainsChart, liftChart
from dmba.metric import AIC_score
import tensorflow as tf

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense
```

```
Requirement already satisfied: mord in /usr/local/lib/python3.7/dist-packages
(0.6)
Requirement already satisfied: dmba in /usr/local/lib/python3.7/dist-packages
(0.1.0)
```

[18]:

```
# Import the dataset
from google.colab import files
file = files.upload()
df_bank = pd.read_csv("Banks.csv")
df_bank.head()
```

```
<IPython.core.display.HTML object>
```

```
Saving Banks.csv to Banks (1).csv
```

[18]:

	Obs	Financial Condition	TotCap/Assets	TotExp/Assets	TotLns&Lses/Assets
0	1	1	9.7	0.12	0.65
1	2	1	1.0	0.11	0.62
2	3	1	6.9	0.09	1.02
3	4	1	5.8	0.10	0.67
4	5	1	4.3	0.11	0.69

TODO 1

Run a logistic regression model (on the entire dataset) that models the status of a bank as a function of the two financial measures provided.

Specify the success class as weak (this is similar to creating a dummy that is 1 for financially weak banks and 0 otherwise), and use the default cutoff value of 0.5.

Let's assume, Weak financial condition=1, strong financial condition=0

[127]:
```python
y = df_bank['Financial Condition']
x = df_bank.drop(columns=['Obs','Financial Condition', 'TotCap/Assets'])
log_reg = LogisticRegression(penalty="l2", C=1e42, solver='liblinear')
log_reg.fit(x,y)
print('Intercept ', log_reg.intercept_[0])
coef=pd.DataFrame({'Coefficient': log_reg.coef_[0]}, index=x.columns)

print(pd.DataFrame({'Coefficient': log_reg.coef_[0]}, index=x.columns))
```

```
Intercept  -14.720832806179043
                    Coefficient
TotExp/Assets         89.832567
TotLns&Lses/Assets     8.371267
```

TODO 2

Write the estimated equation that associates the financial condition of a bank with its two predictors in three formats:

a. The logit as a function of the predictors

b. The odds as a function of the predictors

c. The probability as a function of the predictors

x1 = TotExp/Assets, x2 = TotLns&Lses/Assets, e=2.71828

A. logit = -14.72083281 + (89.83256659 * x1) + (8.37126718 * x2)

B. odds(weak) = e^(-14.72083281 + (89.83256659 * x1) + (8.37126718*x2))

C. p(weak) = 1/(1 + (e^-1(-14.72083281 + (89.83256659 * x1) + (8.37126718*x2))))

TODO 3

Consider a new bank whose total loans and leases/assets ratio = 0.6 and total expenses/assets ratio = 0.11.

From your logistic regression model, estimate the following four quantities for this bank:

the logit, the odds, the probability of being financially weak, and the classification of the bank (use cutoff = 0.5).

[128]:
```python
logit=log_reg.intercept_[0]+(coef.Coefficient[0]*0.11)+(coef.Coefficient[1]*0.6)
print("logit: ", logit)
odds= math.exp(logit)
print("odds: ", odds)
probability=odds/(1+odds)
print("probability: ", probability)
if (probability<0.5):
  print(" Bank is Financially Strong")
else:
  print("Bank is Financially weak")
```

```
logit:  0.18350982427661666
odds:  1.2014267685026223
probability:  0.5457491412806863
Bank is Financially weak
```

TODO 4

We use a cutoff value of 0.5 to classify a record based on propensity.

Instead, if we want to classify the record using the odds or logit, what value should we take as a cutoff?

[129]:
```
cutoffs = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
new_pred = []
for i in cutoffs:
  new_pred=[]
  predict = (log_reg.predict_proba(x) >= i).astype(int)
  for k in predict:
    if k[0] == 1:
      new_pred.append(0)
    else:
      new_pred.append(1)
print("Accuracy for Cutoff {}: {}".format(i,accuracy_score(y, new_pred)))
```

```
Accuracy for Cutoff 0.9: 0.85
```

TODO 5

When a bank with in poor financial condition is misclassified as financially strong, the misclassification cost is much higher than a financially strong bank misclassified as weak.

To minimize the expected cost of misclassification, should the cutoff value for classification (which is currently at 0.5) be increased or decreased?

In order to minimise the expected cost of misclassification the cutoff value for the classification should be decreased.A bank that is financially strong may be considered financially weak,which will not affect the bank much,however a financially weak bank has more to loose if it is assumed to be financially strong.Hence reducing the cutoff will reduce the expected cost of misclassification.

1.1.2 Problem 2 - Identifying Good System Administrators

A management consultant is studying the roles played by experience and training in a system administrator's ability to complete a set of tasks in a specified amount of time. In particular, the consultant is interested in discriminating between administrators who are able to complete given tasks within a specified time and those who are not.

Data are collected on the performance of 75 randomly selected administrators. They are stored in the file **SystemAdministrators.csv**.

The variable Experience measures months of full-time system administrator experience, while Training measures the number of relevant training credits. The outcome variable Completed is either Yes or No, according to whether or not the administrator completed the tasks.

[6]:
```
# Import the dataset
from google.colab import files
file = files.upload()
df1 = pd.read_csv("SystemAdministrators.csv")
df1.head()
```

<IPython.core.display.HTML object>

Saving SystemAdministrators.csv to SystemAdministrators.csv

[6]:

	Experience	Training	Completed task
0	10.9	4	Yes
1	9.9	4	Yes
2	10.4	6	Yes
3	13.7	6	Yes
4	9.4	8	Yes

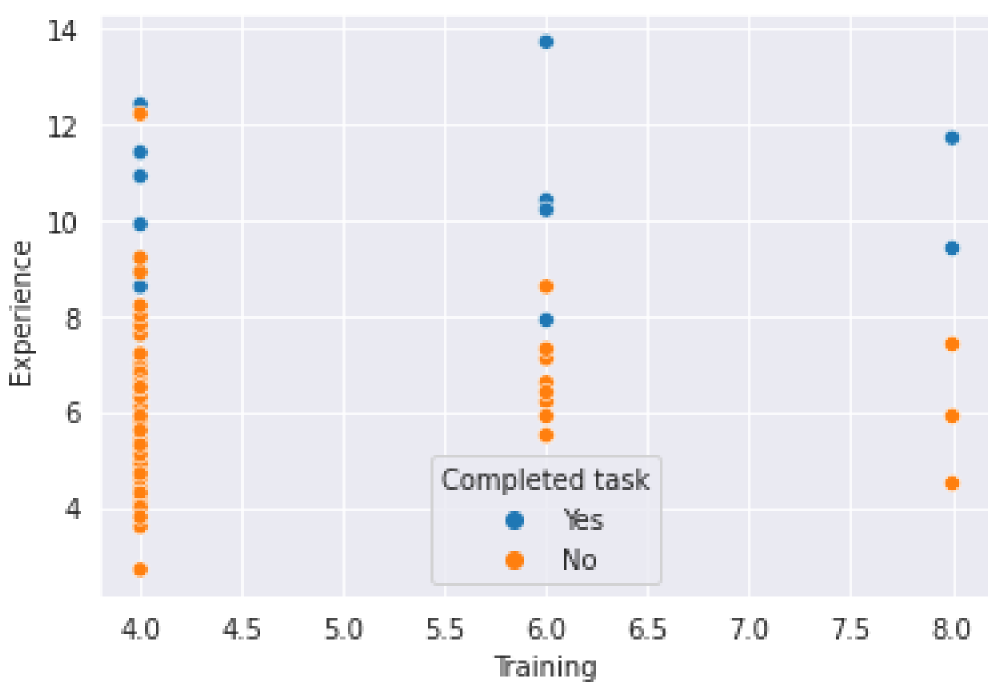
TODO 1

Create a scatter plot of Experience vs. Training using color or symbol to distinguish the administrators' task completion statues.

Which predictor(s) appear(s) potentially useful for the classifying task?

[130]:
```
Scatter_plot= sns.
↪scatterplot(x='Training',y='Experience',data=df1,hue='Completed task')
Scatter_plot
```

[130]: <matplotlib.axes._subplots.AxesSubplot at 0x7f952902c110>



Indented block

TODO 2

Run a logistic regression model with both predictors using the entire dataset as training data. Among those who completed the task, what is the percentage of administrators incorrectly classified as failing to complete the task?

[131]:
```
y=df1['Completed task']
X=df1.drop(['Completed task'], axis=1)
model_log=LogisticRegression()
model_log.fit(X,y)
classes={'No','Yes'}
classificationSummary(y, model_log.predict(X), class_names=classes)
```

```
Confusion Matrix (Accuracy 0.9067)

       Prediction
Actual Yes  No
   Yes  58   2
    No   5  10
```

[132]:
```
cm=confusion_matrix(y, model_log.predict(X))
tn=cm[0,0]
fp=cm[0,1]
tp=cm[1,1]
fn=cm[1,0]
print('The percentage of administators incorrectly classified as failing to␣
 ↪complete the task',cm)
```

```
The percentage of administators incorrectly classified as failing to complete
the task [[58  2]
 [ 5 10]]
```

58 is indicative of the individuals that correctly completed the task and were correctly predicted while 2 of them correctly completed the task but were incorrectly predicted as failing to complete the task.So,around 2.6 % were incorrectly predicted as failing to complete the task.

TODO 3

To decrease the percentage in TODO 2, should we increase or decrease the cutoff probability?

The cutoff probability needs to be increased.

TODO 4

How much experience must be accumulated by a administrator with 4 years of training before his or her estimated probability of completing the task exceeds 0.5?

Since the given training period is 4 years and p>0.5

[133]:
```
p=0.5
odds=p/(1-p)
logit=math.log(odds)
n0=model_log.intercept_[0]
n1=model_log.coef_[0,0]
n2=model_log.coef_[0,1]
exp=(logit-n0-(n2*4))/n1
print('The administrator should have', round(exp,3), 'years experience, so that␣
↪his or her estimated probability')
```

```
The administrator should have 9.168 years experience, so that his or her
estimated probability
```

1.2 Part 2: Neural Network

1.2.1 Problem 3 - Car Sales

Consider the data on used cars (**ToyotaCorolla.csv**) with 1436 records and details on 38 attributes, including Price, Age, KM, HP, and other specifications. The goal is to predict the price of a used Toyota Corolla based on its specifications.

[178]:
```
# Import the dataset
from google.colab import files
file = files.upload()
df3 = pd.read_csv("ToyotaCorolla.csv", encoding = 'unicode_escape')
df3.head()
```

```
<IPython.core.display.HTML object>
```

```
Saving ToyotaCorolla.csv to ToyotaCorolla (2).csv
```

[178]:
```
   Id                                          Model  Price  Age_08_04  \
0   1  TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors  13500         23
1   2  TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors  13750         23
2   3  TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors  13950         24
3   4  TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors  14950         26
4   5    TOYOTA Corolla 2.0 D4D HATCHB SOL 2/3-Doors  13750         30

   Mfg_Month  Mfg_Year     KM Fuel_Type  HP  Met_Color  … Powered_Windows  \
0         10      2002  46986    Diesel  90          1  …               1
1         10      2002  72937    Diesel  90          1  …               0
2          9      2002  41711    Diesel  90          1  …               0
3          7      2002  48000    Diesel  90          0  …               0
4          3      2002  38500    Diesel  90          0  …               1

   Power_Steering  Radio  Mistlamps  Sport_Model  Backseat_Divider  \
0               1      0          0            0                 1
1               1      0          0            0                 1
2               1      0          0            0                 1
```

```
3                1      0           0           0                 1
4                1      0           1           0                 1

   Metallic_Rim  Radio_cassette  Parking_Assistant  Tow_Bar
0             0               0                  0        0
1             0               0                  0        0
2             0               0                  0        0
3             0               0                  0        0
4             0               0                  0        0

[5 rows x 39 columns]
```

TODO 1

Fit a neural network model with the following specifications:

- Use predictors Age_08_04, KM, Fuel_Type, HP, Automatic, Doors, Quarterly_Tax, Mfr_Guarantee, Guarantee_Period, Airco, Automatic_airco, CD_Player,Powered_Windows, Sport_Model, and Tow_Bar
- Partition the data into 80% training and 20% validation
- Scale the numerical predictor and outcome; convert categorical predictors to dummies
- The neural network should have one single hidden layer with two nodes

Present the model summary and performance on the training and validation sets.

[179]:
```python
x1 = ['Age_08_04', 'KM', 'Fuel_Type', 'HP', 'Automatic', 'Doors',␣
↪'Quarterly_Tax', 'Mfr_Guarantee', 'Guarantee_Period', 'Airco',
             'Automatic_airco', 'CD_Player', 'Powered_Windows', 'Sport_Model',␣
↪'Tow_Bar','Price']
x1_data = df3[x1]
x1_data.head()
```

[179]:
```
   Age_08_04     KM Fuel_Type  HP  Automatic  Doors  Quarterly_Tax  \
0         23  46986    Diesel  90          0      3            210
1         23  72937    Diesel  90          0      3            210
2         24  41711    Diesel  90          0      3            210
3         26  48000    Diesel  90          0      3            210
4         30  38500    Diesel  90          0      3            210

   Mfr_Guarantee  Guarantee_Period  Airco  Automatic_airco  CD_Player  \
0              0                 3      0                0          0
1              0                 3      1                0          1
2              1                 3      0                0          0
3              1                 3      0                0          0
4              1                 3      1                0          0

   Powered_Windows  Sport_Model  Tow_Bar  Price
0                1            0        0  13500
1                0            0        0  13750
```

```
2               0             0        0  13950
3               0             0        0  14950
4               1             0        0  13750
```

[180]:
```
x1_data.describe()
```

[180]:
```
          Age_08_04             KM           HP    Automatic        Doors  \
count   1436.000000    1436.000000  1436.000000  1436.000000  1436.000000
mean      55.947075   68533.259749   101.502089     0.055710     4.033426
std       18.599988   37506.448872    14.981080     0.229441     0.952677
min        1.000000       1.000000    69.000000     0.000000     2.000000
25%       44.000000   43000.000000    90.000000     0.000000     3.000000
50%       61.000000   63389.500000   110.000000     0.000000     4.000000
75%       70.000000   87020.750000   110.000000     0.000000     5.000000
max       80.000000  243000.000000   192.000000     1.000000     5.000000

       Quarterly_Tax  Mfr_Guarantee  Guarantee_Period        Airco  \
count    1436.000000    1436.000000       1436.000000  1436.000000
mean       87.122563       0.409471          3.815460     0.508357
std        41.128611       0.491907          3.011025     0.500104
min        19.000000       0.000000          3.000000     0.000000
25%        69.000000       0.000000          3.000000     0.000000
50%        85.000000       0.000000          3.000000     1.000000
75%        85.000000       1.000000          3.000000     1.000000
max       283.000000       1.000000         36.000000     1.000000

       Automatic_airco    CD_Player  Powered_Windows  Sport_Model  \
count      1436.000000  1436.000000      1436.000000  1436.000000
mean          0.056407     0.218663         0.561978     0.300139
std           0.230786     0.413483         0.496317     0.458478
min           0.000000     0.000000         0.000000     0.000000
25%           0.000000     0.000000         0.000000     0.000000
50%           0.000000     0.000000         1.000000     0.000000
75%           0.000000     0.000000         1.000000     1.000000
max           1.000000     1.000000         1.000000     1.000000

           Tow_Bar         Price
count  1436.000000   1436.000000
mean      0.277855  10730.824513
std       0.448098   3626.964585
min       0.000000   4350.000000
25%       0.000000   8450.000000
50%       0.000000   9900.000000
75%       1.000000  11950.000000
max       1.000000  32500.000000
```

[181]:
```
x1_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1436 entries, 0 to 1435
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Age_08_04         1436 non-null   int64 
 1   KM                1436 non-null   int64 
 2   Fuel_Type         1436 non-null   object
 3   HP                1436 non-null   int64 
 4   Automatic         1436 non-null   int64 
 5   Doors             1436 non-null   int64 
 6   Quarterly_Tax     1436 non-null   int64 
 7   Mfr_Guarantee     1436 non-null   int64 
 8   Guarantee_Period  1436 non-null   int64 
 9   Airco             1436 non-null   int64 
 10  Automatic_airco   1436 non-null   int64 
 11  CD_Player         1436 non-null   int64 
 12  Powered_Windows   1436 non-null   int64 
 13  Sport_Model       1436 non-null   int64 
 14  Tow_Bar           1436 non-null   int64 
 15  Price             1436 non-null   int64 
dtypes: int64(15), object(1)
memory usage: 179.6+ KB
```

[182]:
```python
x1_data.isnull().sum()
x1_data['Fuel_Type'].value_counts()
```

[182]:
```
Petrol    1264
Diesel     155
CNG         17
Name: Fuel_Type, dtype: int64
```

[183]:
```python
f_dum= pd.get_dummies(x1_data['Fuel_Type'])
f_dum.head()
```

[183]:

	CNG	Diesel	Petrol
0	0	1	0
1	0	1	0
2	0	1	0
3	0	1	0
4	0	1	0

[184]:
```python
x1_data = pd.concat([x1_data,f_dum], axis=1)
x1_data.head()
```

[184]:

	Age_08_04	KM	Fuel_Type	HP	Automatic	Doors	Quarterly_Tax	\
0	23	46986	Diesel	90	0	3	210	
1	23	72937	Diesel	90	0	3	210	

```
2          24  41711     Diesel   90           0       3            210
3          26  48000     Diesel   90           0       3            210
4          30  38500     Diesel   90           0       3            210

   Mfr_Guarantee  Guarantee_Period  Airco  Automatic_airco  CD_Player  \
0              0                 3      0                0          0
1              0                 3      1                0          1
2              1                 3      0                0          0
3              1                 3      0                0          0
4              1                 3      1                0          0

   Powered_Windows  Sport_Model  Tow_Bar  Price  CNG  Diesel  Petrol
0                1            0        0  13500    0       1       0
1                0            0        0  13750    0       1       0
2                0            0        0  13950    0       1       0
3                0            0        0  14950    0       1       0
4                1            0        0  13750    0       1       0
```

[185]:
```
x1_data=x1_data.drop(['Fuel_Type'], axis=1)
x1_data.head()
```

[185]:
```
   Age_08_04     KM  HP  Automatic  Doors  Quarterly_Tax  Mfr_Guarantee  \
0         23  46986  90          0      3            210              0
1         23  72937  90          0      3            210              0
2         24  41711  90          0      3            210              1
3         26  48000  90          0      3            210              1
4         30  38500  90          0      3            210              1

   Guarantee_Period  Airco  Automatic_airco  CD_Player  Powered_Windows  \
0                 3      0                0          0                1
1                 3      1                0          1                0
2                 3      0                0          0                0
3                 3      0                0          0                0
4                 3      1                0          0                1

   Sport_Model  Tow_Bar  Price  CNG  Diesel  Petrol
0            0        0  13500    0       1       0
1            0        0  13750    0       1       0
2            0        0  13950    0       1       0
3            0        0  14950    0       1       0
4            0        0  13750    0       1       0
```

[193]:
```
x1_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1436 entries, 0 to 1435
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
```

```
---  ------            --------------  -----
 0   Age_08_04         1436 non-null   float64
 1   KM                1436 non-null   float64
 2   HP                1436 non-null   float64
 3   Automatic         1436 non-null   int64
 4   Doors             1436 non-null   float64
 5   Quarterly_Tax     1436 non-null   float64
 6   Mfr_Guarantee     1436 non-null   int64
 7   Guarantee_Period  1436 non-null   float64
 8   Airco             1436 non-null   int64
 9   Automatic_airco   1436 non-null   int64
 10  CD_Player         1436 non-null   int64
 11  Powered_Windows   1436 non-null   int64
 12  Sport_Model       1436 non-null   int64
 13  Tow_Bar           1436 non-null   int64
 14  Price             1436 non-null   float64
 15  CNG               1436 non-null   uint8
 16  Diesel            1436 non-null   uint8
 17  Petrol            1436 non-null   uint8
dtypes: float64(7), int64(8), uint8(3)
memory usage: 172.6 KB
```

[194]:
```
x1_data.isnull().sum()
```

[194]:
```
Age_08_04           0
KM                  0
HP                  0
Automatic           0
Doors               0
Quarterly_Tax       0
Mfr_Guarantee       0
Guarantee_Period    0
Airco               0
Automatic_airco     0
CD_Player           0
Powered_Windows     0
Sport_Model         0
Tow_Bar             0
Price               0
CNG                 0
Diesel              0
Petrol              0
dtype: int64
```

[195]:
```
num =␣
 ↪x1_data[['Age_08_04','KM','HP','Doors','Quarterly_Tax','Guarantee_Period','Price']]
num.head()
```

[195]:
```
   Age_08_04        KM        HP     Doors  Quarterly_Tax  Guarantee_Period  \
0  -1.771966 -0.574695 -0.768042 -1.085139        2.98868         -0.270919
1  -1.771966  0.117454 -0.768042 -1.085139        2.98868         -0.270919
2  -1.718184 -0.715386 -0.768042 -1.085139        2.98868         -0.270919
3  -1.610620 -0.547650 -0.768042 -1.085139        2.98868         -0.270919
4  -1.395491 -0.801028 -0.768042 -1.085139        2.98868         -0.270919

      Price
0  0.763763
1  0.832715
2  0.887877
3  1.163685
4  0.832715
```

[196]:
```
scaler = StandardScaler().fit(num)
x1_data[['Age_08_04','KM','HP','Doors','Quarterly_Tax','Guarantee_Period','Price']]␣
 ↪= scaler.transform(num)
x1_data.head()
```

[196]:
```
   Age_08_04        KM        HP  Automatic     Doors  Quarterly_Tax  \
0  -1.771966 -0.574695 -0.768042          0 -1.085139        2.98868
1  -1.771966  0.117454 -0.768042          0 -1.085139        2.98868
2  -1.718184 -0.715386 -0.768042          0 -1.085139        2.98868
3  -1.610620 -0.547650 -0.768042          0 -1.085139        2.98868
4  -1.395491 -0.801028 -0.768042          0 -1.085139        2.98868

   Mfr_Guarantee  Guarantee_Period  Airco  Automatic_airco  CD_Player  \
0              0         -0.270919      0                0          0
1              0         -0.270919      1                0          1
2              1         -0.270919      0                0          0
3              1         -0.270919      0                0          0
4              1         -0.270919      1                0          0

   Powered_Windows  Sport_Model  Tow_Bar     Price  CNG  Diesel  Petrol
0                1            0        0  0.763763    0       1       0
1                0            0        0  0.832715    0       1       0
2                0            0        0  0.887877    0       1       0
3                0            0        0  1.163685    0       1       0
4                1            0        0  0.832715    0       1       0
```

[197]:
```
b= x1_data['Price']
a = x1_data.drop("Price",axis=1)
a.head()
```

[197]:
```
   Age_08_04        KM        HP  Automatic     Doors  Quarterly_Tax  \
0  -1.771966 -0.574695 -0.768042          0 -1.085139        2.98868
1  -1.771966  0.117454 -0.768042          0 -1.085139        2.98868
```

```
2  -1.718184 -0.715386 -0.768042          0 -1.085139        2.98868
3  -1.610620 -0.547650 -0.768042          0 -1.085139        2.98868
4  -1.395491 -0.801028 -0.768042          0 -1.085139        2.98868

   Mfr_Guarantee  Guarantee_Period  Airco  Automatic_airco  CD_Player  \
0              0         -0.270919      0                0          0
1              0         -0.270919      1                0          1
2              1         -0.270919      0                0          0
3              1         -0.270919      0                0          0
4              1         -0.270919      1                0          0

   Powered_Windows  Sport_Model  Tow_Bar  CNG  Diesel  Petrol
0                1            0        0    0       1       0
1                0            0        0    0       1       0
2                0            0        0    0       1       0
3                0            0        0    0       1       0
4                1            0        0    0       1       0
```

[198]:
```
b.head()
```

[198]:
```
0    0.763763
1    0.832715
2    0.887877
3    1.163685
4    0.832715
Name: Price, dtype: float64
```

[199]:
```
X_train, X_valid,y_train,y_valid = train_test_split(a, b,test_size=0.20,␣
↪random_state=1)
```

Model 1:

[201]:
```
regression = MLPRegressor(hidden_layer_sizes=(2), random_state=1).fit(X_train,␣
↪y_train)
regression.predict(X_valid)
```

[201]:
```
array([ 4.13137273e-01, -2.78643090e-01,  9.81301075e-01, -1.02931365e+00,
        2.04481649e-01,  2.10965084e+00, -6.56250054e-01, -2.20704720e-01,
       -6.44300638e-01,  1.52723088e-01, -2.17465381e-01, -4.65552513e-01,
       -6.95288710e-01, -6.09565306e-01, -1.11309156e+00, -5.79798635e-01,
       -1.11309156e+00, -8.21241817e-01,  1.82584280e+00, -1.11309156e+00,
        1.27206418e+00, -9.82955762e-01, -5.65679096e-01, -9.84839771e-01,
        4.86708214e-01,  2.73867219e+00, -6.74847923e-01, -2.44774803e-01,
        2.24758923e+00, -3.84419394e-01, -7.91467681e-01, -6.95397648e-01,
        5.43392685e-01,  1.93647388e+00,  4.57893513e+00,  2.95400801e-02,
       -1.80152453e-01, -8.26706043e-01, -5.88177427e-01,  1.80091563e+00,
        7.15184732e-01, -3.83952975e-01, -6.30583922e-01, -6.12443878e-01,
       -6.28500808e-01, -3.77439981e-01,  3.23780163e-01, -4.70126522e-01,
```

```
-1.11309156e+00, -9.98950485e-01, -6.63469366e-01,  2.87509992e-01,
-6.89036619e-01, -9.30942418e-01, -4.95356274e-01,  2.28195811e+00,
 1.13352614e+00,  3.25799156e-01, -3.11402640e-01, -1.11309156e+00,
 1.39194953e+00, -4.50820040e-01, -3.72944961e-01,  2.47918207e-01,
-9.97782817e-02,  1.21202546e+00,  1.57000466e-01, -3.75160079e-01,
-1.62125313e-01,  1.28981056e-01, -1.01652836e+00, -2.00602563e-01,
-8.43599199e-01, -7.01383483e-01, -9.65097633e-01,  1.04862752e+00,
-6.44889121e-01, -5.64566790e-01, -5.06435444e-01,  5.65033344e-01,
 4.13966559e-01, -9.00031209e-01, -1.11309156e+00, -2.63606958e-02,
-4.00957512e-01, -5.30014427e-01,  3.93839111e-01,  1.51906676e+00,
 3.74874983e-01, -1.03060873e+00, -3.53594542e-01,  3.76864121e-02,
-3.17109226e-01, -2.33828194e-01, -7.07642202e-01,  2.93250507e-01,
 6.43353049e-01,  7.95571953e-02,  6.52302942e-02,  3.00363411e-02,
-6.35241909e-01, -2.16106274e-01, -5.50445630e-02, -7.35771465e-01,
 2.60290005e-01,  6.86405966e-01, -8.22935320e-02, -7.55798257e-01,
-2.74290529e-01, -8.39129724e-02, -8.33112376e-01,  6.06167920e-01,
-6.09088343e-01, -1.11309156e+00, -1.80053509e-01,  8.08275632e-01,
-1.11309156e+00, -5.74157952e-01, -4.89373395e-01, -1.04943673e+00,
 2.18523056e+00,  2.58133426e+00, -6.75086794e-01,  2.45512353e-01,
-1.00748554e+00,  1.38396321e+00, -1.10412926e+00,  1.02395253e-01,
-1.02309999e-01,  2.74620571e+00,  2.37153986e+00, -6.16733478e-01,
-3.54631745e-01, -3.60237660e-01,  3.24709991e-01, -4.74478493e-01,
-2.27802760e-02, -5.28273310e-01,  8.31269762e-01, -5.76214565e-01,
-4.27549642e-01, -4.33133757e-01, -1.78151107e-01, -8.95581243e-01,
-1.43955634e-02,  2.58422005e-01,  1.77491952e+00, -2.93372851e-01,
-4.22539750e-01, -1.11309156e+00, -7.87991913e-01, -1.02975763e+00,
 2.40301543e+00, -9.62539923e-01, -1.11309156e+00,  1.40090317e+00,
 1.25501320e-01, -7.44910958e-01, -7.25166745e-01,  6.94531572e-01,
-2.65723140e-01,  4.44022611e-01, -1.11309156e+00, -7.88585436e-01,
-8.08260264e-01, -1.11309156e+00, -1.48785706e-01, -3.90557062e-01,
-9.08920469e-01,  4.91179466e-01, -3.87958021e-01,  2.77213933e+00,
-3.35651117e-01,  1.42605876e-01,  3.41939374e+00, -2.13833811e-01,
-7.29425293e-01, -3.02068477e-01, -9.41895454e-01, -8.15630220e-01,
 6.69539567e-01,  1.83556507e+00, -9.82125934e-01, -9.42514559e-01,
-3.61426241e-01, -6.24242446e-01, -5.00539914e-01, -2.61466300e-01,
-1.90353386e-01,  1.92292983e+00,  7.67912788e-01, -6.59916617e-01,
 1.38296247e+00, -2.05388006e-01, -9.55391820e-01, -4.34641096e-01,
-5.22162380e-01, -7.85571577e-01, -4.69155458e-01, -8.22204101e-01,
-8.67271420e-01,  1.56644393e-02,  2.27260545e+00, -7.78495372e-01,
 5.01188200e-01, -5.94242835e-01,  4.04691494e-02, -7.20163742e-01,
-5.50535254e-01,  1.54817869e+00, -2.08472040e-01, -6.87519881e-01,
-3.07884993e-01, -4.77866341e-01,  9.16490634e-01, -9.27010350e-01,
-5.67493638e-01,  7.13492203e-01,  5.05911857e-01, -3.31215902e-01,
 2.28056838e-01, -4.05408963e-01, -6.12724695e-01, -7.50869932e-01,
 3.83058974e-01, -3.42473250e-01,  1.77993728e+00, -2.22394598e-01,
-2.18229913e-01, -1.49142773e-01,  7.64863987e-01, -6.70394217e-01,
 1.74458872e+00,  5.48889315e-03, -5.28447694e-04, -7.08677035e-01,
```

```
 2.48441287e+00, -6.72499698e-01,  3.95151709e-01, -4.97085850e-01,
-4.38406339e-01, -4.10449214e-01,  1.09662771e+00,  1.18483383e+00,
 9.91547021e-02, -3.15543691e-01, -1.99483817e-01, -7.33447447e-01,
-6.87465138e-01,  5.60388592e-01, -6.86232405e-01,  4.80704494e-01,
 2.95811167e+00,  5.04192191e-01, -3.39537753e-01, -5.10284023e-01,
-3.47971589e-01, -8.34002867e-01, -7.61888218e-01,  4.37925147e-01,
-2.18337389e-01,  1.55091498e-01, -1.06450993e+00,  2.87535273e+00,
 4.54524680e-01,  1.16980506e+00, -3.34500318e-01, -9.86970422e-02,
-1.11309156e+00, -5.22010099e-01, -1.75275564e-01,  2.52492814e+00,
 1.04157882e+00, -9.14123378e-02,  2.04324990e+00, -9.51885299e-02,
-7.02860758e-01, -3.72655073e-01, -5.25242810e-01, -5.39762778e-01,
 1.73431969e+00, -7.79168268e-01, -9.36091524e-01, -7.87689491e-01,
 6.10319380e-01,  1.57616622e+00, -6.60962078e-01, -4.37817118e-01])
```

[202]:
```
regression.score(X_train, y_train)
```

[202]:
```
0.8881522743121312
```

[203]:
```
regression.score(X_valid, y_valid)
```

[203]:
```
0.9029320302838734
```

High values of r2 is indicative of a good model

TODO 2

Repeat the process, changing the number of hidden layers and nodes to {single layer with five nodes}, {two layers, five nodes in each layer}.

Comment on the performance of the three models above.

Model 2 (SIngle layer with 5 nodes)

[204]:
```
r2 = MLPRegressor(hidden_layer_sizes=(5), random_state=1).fit(X_train, y_train)
r2.predict(X_valid)
```

[204]:
```
array([ 0.23236521, -0.20808668,  0.89367006, -0.92655803,  0.23168687,
        1.90487523, -0.84932051, -0.50547726, -0.90924789,  0.07565079,
       -0.33580599, -0.3105919 , -0.68290915, -0.46946362, -0.81387794,
       -0.80068424, -0.87310017, -0.84754929,  1.550843  , -0.66164489,
        1.11037193, -0.64435851, -0.41912993, -0.6764134 ,  0.47814467,
        2.62310348, -0.66361995, -0.15759763,  2.29825755, -0.27975717,
       -0.73580543, -0.82693975,  0.5236992 ,  1.71722841,  4.50440188,
       -0.12345581, -0.12975393, -0.61015918, -0.44328287,  1.73598824,
        0.59564834, -0.43322167, -0.76986565, -0.60232365, -0.79869015,
       -0.25085791,  0.22645084, -0.38355131, -0.82902161, -0.62685907,
       -0.86419335,  0.05419903, -0.53556665, -0.62762179, -0.69303409,
        2.13590979,  0.69517411,  0.22807367, -0.35028311, -0.87667689,
        1.41780154, -0.7686446 , -0.67783688,  0.27048428, -0.06568927,
        1.1840564 ,  0.03421106, -0.57950541, -0.17338955, -0.17717222,
```

```
       -0.61850424, -0.27670906, -1.05618426, -0.96331634, -0.65568604,
        0.9408194 , -0.4533249 , -0.5661388 , -0.77446669,  0.57395884,
        0.31494424, -0.98953979, -0.85615348, -0.02442847, -0.55134697,
       -0.76186358,  0.87944755,  1.45344391,  0.26140647, -1.25889001,
       -0.2303684 , -0.10271356, -0.16205408, -0.3364574 , -0.66410338,
        0.1821588 ,  0.5502532 ,  0.05450607,  0.08341154, -0.04662108,
       -0.96948788, -0.19877145, -0.17043431, -0.6764134 ,  0.32492171,
        0.86605057, -0.08094655, -0.59197899, -0.43676631, -0.2521074 ,
       -1.06491662,  0.65013983, -0.9649683 , -0.8329975 , -0.24516994,
        0.76094401, -0.71837839, -0.65509403, -0.61299321, -0.61461195,
        2.12534096,  3.08747046, -0.55046563,  0.16114465, -0.89060788,
        0.95397562, -0.70002139, -0.05147177, -0.06357385,  2.79574261,
        2.62972254, -0.75655389, -0.56432865, -0.49580895,  0.35116539,
       -0.59554202, -0.03902334, -0.37466684,  0.76061851, -0.6764134 ,
       -0.57945878, -0.11840718, -0.29334301, -0.95543555, -0.10696205,
        0.22299788,  1.46724886, -0.26895515, -0.15036809, -0.85674724,
       -1.00607855, -0.6764134 ,  1.88355069, -0.57206374, -0.67438584,
        2.00738207,  0.02169663, -0.85899237, -0.86554261,  0.53350567,
       -0.37869606,  0.44515499, -0.92124981, -0.67016496, -0.64655589,
       -0.46381857,  0.09729428, -0.56355103, -1.0017914 ,  0.51802238,
       -0.59150672,  2.89530261, -0.2498926 ,  0.08355206,  3.75286377,
       -0.16003083, -0.60786683, -0.42477446, -0.90854343, -0.76205529,
        0.66085814,  1.79636388, -0.62791377, -1.09315039, -0.50094213,
       -0.94484588, -0.31995417, -0.3878602 , -0.36286836,  1.64057178,
        0.70580482, -0.67247259,  1.48854482, -0.33362603, -0.6311571 ,
       -0.37993569, -0.44288884, -0.6764134 , -0.41012794, -0.92084771,
       -0.65330502, -0.08651082,  2.23089131, -0.63566996,  0.63547443,
       -0.75081997,  0.13838668, -0.82227601, -0.60759293,  1.63205976,
       -0.3246709 , -0.64680701, -0.18779366, -0.59122524,  0.78007869,
       -0.79921819, -0.31307653,  0.94546666,  0.94952085, -0.32398667,
        0.37166833, -0.66055358, -0.54782377, -0.75045621,  0.58180946,
       -0.12351008,  1.71232914, -0.42454193, -0.18626812, -0.36361051,
        0.71299134, -0.81785878,  1.72325531, -0.01289127, -0.13193451,
       -0.81309896,  2.65289603, -0.6764134 ,  0.36964809, -0.4379183 ,
       -0.70650393, -0.50866415,  1.07238236,  1.10205641,  0.02665728,
       -0.22964132, -0.3442441 , -0.60874851, -0.70532612,  0.45256386,
       -0.47844625,  0.50213625,  3.13496135,  1.00936622, -0.34128549,
       -0.24138141, -0.52596273, -0.82164419, -0.75697468,  0.26855054,
       -0.25332322,  0.18628307, -0.77502097,  3.21815616,  0.46330928,
        1.21880664, -0.23996349, -0.12594826, -0.95597518, -0.43734385,
       -0.0807367 ,  2.11160286,  0.98559823, -0.25013762,  2.31005396,
       -0.28001896, -0.67452929, -0.36659735, -0.3782435 , -0.89635928,
        1.64307342, -0.72957003, -0.89623324, -0.88976469,  0.57570952,
        1.62846414, -0.5835138 , -0.75234747])
```

[205]:
```
r2.score(X_train, y_train)
```

[205]: 0.8888120738085448

[206]:
```
r2.score(X_valid, y_valid)
```

[206]: 0.900462935100548

High values of r2 is indicatice of having a good performing model.

Model 3 (Two Layers with 5 nodes)

[207]:
```
r3 = MLPRegressor(hidden_layer_sizes=(5,5), random_state=1).fit(X_train,␣
↪y_train)
r3.predict(X_valid)
```

[207]:
```
array([ 2.22405613e-01, -1.70176023e-01,  1.04152492e+00, -7.26240742e-01,
        1.79226735e-01,  2.01510324e+00, -6.87272858e-01, -6.38028716e-01,
       -6.88024616e-01, -5.97698541e-01, -4.57874265e-01, -4.22713102e-01,
       -7.65832794e-01, -7.30644211e-01, -7.65798581e-01, -6.74928480e-01,
       -7.38756084e-01, -6.28824821e-01,  1.63295099e+00, -7.65832794e-01,
        1.20302218e+00, -4.00697653e-01, -4.43484120e-01, -7.65832794e-01,
        5.95640471e-01,  2.52425740e+00, -7.42166485e-01, -3.17632581e-01,
        2.08699116e+00, -8.32950198e-01, -6.91141080e-01, -7.01903589e-01,
        7.03706092e-01,  1.77363285e+00,  4.24891659e+00, -2.92118976e-01,
       -2.94197711e-01, -4.55937287e-01, -5.04145234e-01,  1.69342387e+00,
        5.70883677e-01, -4.82749123e-01, -8.06326601e-01, -6.23828164e-01,
       -5.69001368e-01, -3.04082018e-01,  1.94453072e-01, -4.57292477e-01,
       -7.65464007e-01, -7.52527077e-01, -8.28836173e-01, -2.53357259e-02,
       -2.35796161e-01, -7.15492687e-01, -7.11474695e-01,  2.10213576e+00,
        1.03156652e+00,  3.18713604e-01, -8.48253604e-01, -7.62561955e-01,
        1.42656570e+00, -7.99338533e-01, -6.87549612e-01,  1.99660104e-01,
       -3.19735715e-01,  1.19280343e+00, -4.71005672e-03, -6.87025723e-01,
       -2.26540827e-01, -1.42200871e-01, -7.65832794e-01, -1.41551319e-01,
       -7.36982789e-01, -7.12046410e-01, -7.37356902e-01,  1.07778033e+00,
       -3.72941428e-01, -7.07534735e-01, -8.41389120e-01,  5.01187464e-01,
        4.34351229e-01, -6.58031923e-01, -7.65832794e-01, -5.11906997e-02,
       -3.68571479e-01, -6.97876053e-01,  9.35245520e-01,  1.49242740e+00,
        1.92834385e-01, -7.59909559e-01, -4.91101585e-01, -1.21407120e-01,
       -8.27362990e-01, -2.48095530e-01, -7.42915684e-01, -1.83714609e-03,
        5.44693957e-01, -1.37774482e-03,  7.29276013e-02,  3.84381501e-02,
       -6.67184549e-01, -2.15777927e-01, -5.07511107e-01, -7.65832794e-01,
        1.58256394e-01,  8.64780309e-01, -4.31981367e-04, -7.08398454e-01,
       -7.17891164e-01, -4.11135409e-01, -7.42280232e-01,  8.59403563e-01,
       -6.91003407e-01, -7.17330634e-01, -1.66081899e-01,  9.02431611e-01,
       -7.38390655e-01, -7.10213148e-01, -3.25870660e-01, -7.65832794e-01,
        2.29356510e+00,  3.14128139e+00, -5.57658616e-01,  2.53039597e-01,
       -7.56729517e-01,  1.25467164e+00, -7.65832794e-01, -2.08390746e-01,
        8.83837407e-02,  2.70036811e+00,  2.55891881e+00, -8.76021200e-01,
       -4.53256699e-01, -5.58174074e-01,  3.26537040e-01, -2.66865364e-01,
```

```
-1.42973593e-02, -5.83407619e-01,  7.56881625e-01, -7.08319550e-01,
-6.68939269e-01, -3.36208066e-01, -2.01978218e-01, -6.45651430e-01,
-2.18308901e-01,  1.94926574e-01,  1.80585012e+00, -3.92179169e-01,
-5.74212911e-01, -7.51610139e-01, -6.76016864e-01, -7.65832794e-01,
 2.07540895e+00, -5.16695646e-01, -7.65832794e-01,  1.89508900e+00,
-7.76709486e-03, -6.98517218e-01, -6.82537022e-01,  4.37428033e-01,
-8.32533458e-01,  3.49500904e-01, -7.32391763e-01, -7.19469866e-01,
-7.11393826e-01, -7.82992886e-01,  8.53381751e-02, -3.99923381e-01,
-7.00936108e-01,  4.57452360e-01, -4.33808057e-01,  2.76150637e+00,
-1.30241440e-01,  2.08175888e-01,  3.49118218e+00, -4.60171850e-01,
-7.65115212e-01, -6.92451425e-01, -7.06119886e-01, -7.65832794e-01,
 7.21219574e-01,  1.77201464e+00, -7.56418792e-01, -8.33618025e-01,
-5.51140392e-01, -7.14381782e-01, -4.58358275e-01, -3.87069288e-01,
-4.17852835e-01,  1.78655413e+00,  8.34402332e-01, -7.36581721e-01,
 1.23388937e+00, -1.52320398e-01, -7.38804348e-01, -4.02751805e-01,
-7.12852354e-01, -7.65832794e-01, -5.93590916e-01, -6.58025805e-01,
-7.65832794e-01, -2.67009615e-01,  2.28632380e+00, -7.54875148e-01,
 5.54432535e-01, -6.71762480e-01,  1.02544218e-01, -7.12284354e-01,
-6.69007079e-01,  1.84080047e+00, -2.80823785e-01, -7.15302063e-01,
 1.26566624e-01, -8.00041631e-01,  8.97494820e-01, -8.09218514e-01,
-5.68102056e-01,  9.21034091e-01,  1.15339465e+00, -5.13886178e-01,
 3.08604900e-01, -9.19145628e-01, -7.58798357e-01, -9.86060175e-01,
 7.35292244e-01, -4.97683064e-02,  1.95946820e+00, -4.51691253e-01,
-3.06762760e-01, -4.17742489e-01,  8.28021569e-01, -7.10714769e-01,
 1.64612455e+00,  6.51178272e-02, -2.38818664e-01, -7.70545124e-01,
 2.60714368e+00, -7.65832794e-01,  1.65770345e-01, -2.44689054e-01,
-6.57996806e-01, -1.74196515e-01,  1.26083641e+00,  1.09634395e+00,
-3.65425868e-01, -1.13273447e-01, -4.26978921e-01, -4.80520224e-01,
-6.26061361e-01,  2.40505535e-01, -6.89939848e-01,  1.90847608e-01,
 2.93420362e+00,  1.16339066e+00, -7.39146881e-01, -6.92185022e-01,
-6.78141635e-01, -7.83185410e-01, -7.36225339e-01,  3.11554466e-01,
-1.54010887e-01,  6.79193868e-02, -7.65832794e-01,  3.28950300e+00,
 6.10945046e-01,  1.29551267e+00, -1.69389578e-01, -1.99917627e-02,
-9.43371885e-01, -7.25476738e-01, -1.63588114e-01,  2.20374461e+00,
 1.10173731e+00, -3.90987156e-01,  2.31911046e+00, -4.28625756e-01,
-7.65832794e-01, -3.74989767e-01, -7.41337437e-01, -7.08158249e-01,
 1.63900264e+00, -6.95840989e-01, -7.06222697e-01, -7.56319287e-01,
 8.63734640e-01,  1.66371905e+00, -7.43226451e-01, -6.89579112e-01])
```

[208]:
```
r3.score(X_train, y_train)
```

[208]: 0.8887536213334735

[209]:
```
r3.score(X_valid, y_valid)
```

[209]: 0.8830729021882556

Higher values of 0.88 is indicative of a good NN.

Comparing the three models the best model is model 2 with a value of 0.88,followed by model 3 and then model 1.All three models have a very small difference and are generally good performing models.However Model 2 comrising of a single layer with 5 nodes performs the best as it a value which is closest to 1.