Ahsanullah University of Science and Technology (AUST)

Department of Computer Science and Engineering

Course No. : CSE4108

Course Title: Artificial Intelligence Lab

Date of Submission: 28/03/2021

Submitted By:

Lab Group: A1

Section: A

Name: Umme Habiba

ID: 170104004

Submitted To:

Dr. S.M.A Al Mamun

Tonmoy Hossain Dihan

Sajib Kumar Saha Joy

Question: Implement Linear Regression without using Scikit-learn.

Python Code:

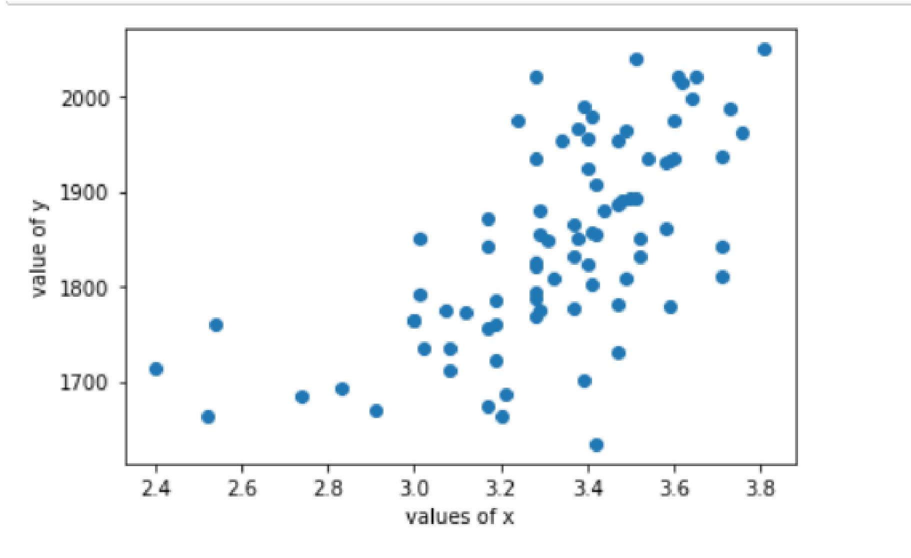
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

df=pd.read_csv("P:/Artificial Intelligence Lab(CSE4108)/Session 5/1.csv")

df.head(10)
```

Cell output:

	SAT	GPA
0	1714	2.40
1	1664	2.52
2	1760	2.54
3	1685	2.74
4	1693	2.83
5	1670	2.91
6	1764	3.00
7	1764	3.00
8	1792	3.01
9	1850	3.01

```
x=pd.array(df['GPA'])

y=pd.array(df['SAT'])

plt.scatter(x,y)

plt.xlabel("values of x")

plt.ylabel("value of y")

plt.show()
```

Cell output:



x_mean=np.mean(x)

y_mean=np.mean(y)

print("Mean for x value : %.2f" %(x_mean)," and Mean fr y values is :%.2f" %y_mean)

Cell output: Mean for x value : 3.33 and Mean for y values is :1845.27.

```
num=0
den=0
for i in range(len(x)):
   num+=(x[i]-x_mean)*(y[i]-y_mean)
   den+=(x[i]-x_mean)**2
b1=num/den
b0=y_mean-(b1*x_mean)
print("Intercept value is :%.2f " %float(b0),"& Co-efficent is : %.2f"%float(b1)
)
```

Cell output: Intercept value is :1028.64 & Co-efficent is : 245.22

```
y_pred=b0+b1*x

num_r=0

den_r=0

for i in range(len(x)):

   num_r+=(y_pred[i]-y_mean)**2

   den_r+=(y[i]-y_mean)**2

r_sq=(num_r/den_r)

print("The value of c-efficient of determination R_squre is :%.2f%%"%float(r_sq)
```

Cell output: The value of c-efficient of determination R_squre is :0.41%

Explanation: To implement linear regression without using scikit-learn, firstly we have import a csv file ,then store the value of “GPA” into x and value of ‘SAT’ into y. After that using this data have plot a graph . Then calculate the value of x mean and y mean and also the co-efficient and intercept value . Using this value have calculate the predicted values . Finally calculate the r-square value so that we can see the statistical measure of how close the data are to the fitted regression line.

Question: Implement Logistic Regression from scratch without using Scikit-learn. Run it against a dataset of choice (any dataset with over 1000 samples). Run the same algorithm with the help of Scikit-learn. Compare your implementation with Scikit-learn’s one.

Python Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets
```

```
from sklearn.model_selection import train_test_split
x, y = datasets.make_classification(n_samples=1100)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2)


class LogisticRegression:
    def __init__(self, learning_rate=0.001, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None
    def fit(self, x, y):
        n_samples, n_features = x.shape
        # init parameters
        self.weights = np.zeros(n_features)
        self.bias = 0
        # gradient descent
        for _ in range(self.n_iters):
            # approximate y with linear combination of weights and x, plus bias
            linear_model = np.dot(x, self.weights) + self.bias  #y^=wx+b
            # apply sigmoid function
            y_predicted = self._sigmoid(linear_model)
```

```
            # compute gradients
            dw = (1 / n_samples) * np.dot(x.T, (y_predicted - y))
            db = (1 / n_samples) * np.sum(y_predicted - y)
            # update parameters
            self.weights -= self.lr * dw
            self.bias -= self.lr * db
    def predict(self, x):
        linear_model = np.dot(x, self.weights) + self.bias
        y_predicted = self._sigmoid(linear_model)
        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
        return np.array(y_predicted_cls)
    def _sigmoid(self, x):
        return 1 / (1 + np.exp(-x))


def accuracy(y_true, y_pred):
    accuracy = np.sum(y_true == y_pred) / len(y_true)
    return accuracy


regressor = LogisticRegression(learning_rate=0.0001, n_iters=1000)
regressor.fit(X_train, y_train)
predictions = regressor.predict(X_test)
```

```
print("Logistic Regression  accuracy: %.2f%%"% accuracy(y_test, predictions))
```

Cell output: Logistic Regression accuracy: 0.92%.

```
# Using Scikit-learn

from sklearn.linear_model import LogisticRegression

lf = LogisticRegression(random_state=0).fit(X_train, y_train)

predictions2 = lf.predict(X_test)

print("Logistic Regression's classification accuracy is: %.2f%%" %accuracy(y_test, predictions2))
```

Cell output: Logistic Regression's classification accuracy is: 0.97%.

Explanation: To implement Logistic Regression from scratch without using S cikit-learn, firstly make a dataset which have 1100 data .Then divide this dataset into two part 80% will be training data and rest 20% will be test data. After that create a class called logisticRegression which has four functions. In fit function have compute gradients values and then update init function using this data .Prediction function predict the data using sigmoid function which being initialize in _sigmoid function. Lastly accuracy function where we calculate the logistic function accuracy. This whole process is being done without scikit-learn .But in scikit learn there have already some predefine functions that's why here we don't need to create any, just call the function an d then they will return desire logistic function accuracy.
Without using scikit-learn the accuracy is 0.92% and with using scikit-learn it s 0.97%.

Question: Make a dataset by yourself which should have enough samples and attributes and write documentation of it. Do classification or regression on it. If you want to do a classification task, implement at least five models. If you want to do regression, similarly at least five models need to be implemented. For each model get at least three performance metric scores. Implementation of cross validation is a must.

Python Code:

```
dictionary={
   'Name':'Umme Habiba',
   'Id':'170104004'
}

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from numpy import mean
from numpy import std
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

import pandas as pd
candidates = {'marks': [780,750,690,710,680,730,690,720,740,690,610,690,7
10,680,770,610,580,650,540,590,620,600,550,550,570,670,660,580,650,660,
640,620,660,660,680,650,670,580,590,690],
          'cgpa': [4,3.9,3.3,3.7,3.9,3.7,2.3,3.3,3.3,1.7,2.7,3.7,3.7,3.3,3.3,3,2.7,3.7,2.
7,2.3,3.3,2,2.3,2.7,3,3.3,3.7,2.3,3.7,3.3,3,2.7,4,3.3,3.3,2.3,2.7,3.3,1.7,3.7],
          'working_experience': [3,4,3,5,4,6,1,4,5,1,3,5,6,4,3,1,4,6,2,3,2,1,4,1,2,6,4
,2,6,5,1,2,4,6,5,1,2,1,4,5],
          'accepted': [1,1,0,1,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,0,1,0,0,0,0,1,1,0,1,1,0,0,1,
1,1,0,0,0,0,1]
          }

df = pd.DataFrame(candidates,columns= ['marks', 'cgpa','working_experience',
'accepted'])
#print (df)
df.head(10)
```

Cell output:

	marks	cgpa	working_experience	accepted
0	780	4.0	3	1
1	750	3.9	4	1
2	690	3.3	3	0
3	710	3.7	5	1
4	680	3.9	4	0
5	730	3.7	6	1
6	690	2.3	1	0
7	720	3.3	4	1
8	740	3.3	5	1
9	690	1.7	1	0

```
x = df[['marks', 'cgpa','working_experience']]
y = df['accepted']

kf = KFold(n_splits=5)

for train_index, test_index in kf.split(X):

   x_train, x_test = X.iloc[train_index], X.iloc[test_index]
   y_train, y_test = y.iloc[train_index], y.iloc[test_index]

   lf = LogisticRegression()
   lf.fit(x_train,y_train)
   prediction = lf.predict(x_test)

   print('Mean square error is : %.2f%%'%mean_squared_error(y_test, predict
ion, squared=True))

   print('Root mean square error is : %.2f%%'%mean_squared_error(y_test, p
rediction, squared=False))
   print('R-Square error is : %.2f%%'%r2_score(y_test, prediction))
   print('\n')
```

Cell output:

Mean square error is : 0.25%

Root mean square error is : 0.50%
R-Square error is : -0.07%

Mean square error is : 0.12%
Root mean square error is : 0.35%
R-Square error is : 0.50%

Mean square error is : 0.12%
Root mean square error is : 0.35%
R-Square error is : 0.33%

Mean square error is : 0.00%
Root mean square error is : 0.00%
R-Square error is : 1.00%

Mean square error is : 0.00%
Root mean square error is : 0.00%
R-Square error is : 1.00%

Explanation: To make our dataset we have taken four attributes and they are "marks,cgpa,working_experience,accepted" .We have inserted 40 samples for each attribute. I have done classification on it. For this at first separate x value s and y values .Then apply kfold which will split into five part. Then calculate the MSE,RMSE,R-square errors value to check the accuracy .