Question 3 Report

July 21, 2020

1 In his Great name

Artificial Intelligence - Regressions

1.1 Team Members

Name (Alphabetically ordered)	StdNo
Sam Asadi	9532287
Hossein Dehghanipour	9532250
Bahare Moradi	9532245

Shiraz University - Spring 2020

2 Report No. 3

2.0.1 Logistic Regression

Written By : **Hossein Dehghanipour-9532250**

3 Main purpose of Q3

The main purpose of question 3 was to have the students get familiar with the basis of logistic regression and it's apllications beside the benefit of learnign the methods of implementation.

As the question ***had not mentioned*** neighter limited us in the usage of methods of *theta* calculation, I personally prefered to use the *gradient* as the core of *theta computation* and I'll discuss the details with you in a few later lines.

3.1 Libraries

The first and most important part of a code is using the available libraries and code snippets as wisely as possible. We'd better only import the libraries that we need or we may face some performance issues.

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```python
from ExtraLibraries.Writer import Writer as Writer
```

3.2 Dynamic Handling

In order to be able to maintain and edit the code as easily as possible, we'd better to put all of our crucial variable(which we may change a lot during the process of coding) in a class which is accessible by all modules and elements outside the "main" moudule.

[2]:
```python
class CommandCenter :
    TEST_SIZE = 0.2
    TRAIN_SIZE = 1 - TEST_SIZE
    BOUNDARY_PROBABILTY = 0.5
    PLOT_SCALER = 0.5



class DataCenter :
    X_train = None
    Y_train = None
    X_test  = None
    Y_test = None
    Thetas = None
    FILE_NAME = "iris.csv"
    X = None
    Y = None
```

3.3 Basic Functions

The somplication of A.I and Logistic Regression will be gone if we try to code each and every part of the program functionally as the golden rule of programming says >Divide and Conquer, Then solve the problem piece by piece.

Here you will see some basic and important functions that are being used a lot during the program and process of bulidng a logistic regression model.

[3]:
```python
def sigmoid(z):
    return 1 / (1 + np.exp(-z))



def hypothesis(X, Thetas):
    z = np.dot(X, Thetas)
    d = sigmoid(z)
    return d



def costFunction(X, Y, Thetas):
    m = len(Y)
    h = hypothesis(X, Thetas)
    cost = Y * np.log(h) + (1 - Y) * np.log(1 - h)
    return cost.sum() / m
```

3.4 Theta Update

The main goal of our problem is to reach the best curve that would fit both our test set and train set. In order to gain such a goal , we must calculate the **derivation** of our **Cost Function** and then find the *most optimal* thetas possible. If we calculate the equation and set it equal to zero, by solving tthe equation, we would reach such a formula (*which is vectorized*) and the rest of the process would be a piece of cake.

[4]:
```
def updateThetas(X, Y, Thetas, alpha ):
    m = len(X)
    h = hypothesis(X, Thetas)
    gradient = (np.dot(X.T, h - Y) / m) * alpha
    Thetas = Thetas - gradient
    return Thetas
```

The formula above can be gained by solving the equation which is caused by derivation of the cost function. The description of the solution could be found on *Youtube* or obtained by the descriptions mentioned by ***Dr.Andrew NG - MIT University***. The description of how we have reached this formula is ***Up To You***. Feel free to search and use ___Google_.

3.5 Core

The most important part of a machine is training it. Fortunately, this part of our problem is not a big deal here due to the simplicity of the problem and not the essence of having a ***Multi Layered Nueral Network***. So the trainig part will be summarized in a few lines of code. But remeber, in order to gain a **higher accuracy** we would iterate over the training process for multiple times. (maybe 10 or 100 of thousands of time).

[5]:
```
def train(X, Y, Thetas, alpha, iters):
    costLog = []

    for i in range(iters):
        Thetas = updateThetas(X, Y, Thetas, alpha)
        cost = costFunction(X, Y, Thetas)
        costLog.append(cost)

    return Thetas, costLog
```

3.6 Accuracy

One the demands of the question was to calculate the accuracy of the algorithm and report it to the user. In the few lines of the code below, you can see that the accuracy has been calculated and logged in a varibale called `accuracy` which will be shown to the user later on.

[6]:
```
def calculate_accuracy():

    h = hypothesis(DataCenter.X_test,DataCenter.Thetas)
    def decisionBoundry(h):
        h1 = []
        for i in h:
            if ( i >= CommandCenter.BOUNDARY_PROBABILTY).any():
                h1.append(1)
            else:
                h1.append(0)
        return h1
    y_pred = decisionBoundry(h)
    accuracy = 0
    for i in range(0, len(y_pred)):
        if (y_pred[i] == DataCenter.Y_test[i]).any():
            accuracy += 1
    return (accuracy/len(DataCenter.Y_test)* 100)
```

3.7 Data Processing

I would vogorously say that the most important part of an AI problem or data science algorithm, is the *Data Processing* part. In order to gain the best results, we must ignore the corrupted data or *reform* them is possible. Fortunately the presented data set was *Nan* free and did not have any missing data. So my approach would be easier and less complicated than working with most of other data sets.

[7]:
```
def ReadData(fileName):
    dataSet = pd.read_csv(fileName)
    dataSet = dataSet[dataSet.Result != "Iris-versicolor"]
    #dataSet = dataSet[dataSet.Result != "Iris-virginica"]
    dataSet["Result"] = dataSet["Result"].apply(lambda x : 1 if x ==␣
 ↪"Iris-setosa" else 0 )

    # Create X (all the feature columns)
    X = dataSet.drop("Result", axis=1)
    DataCenter.X = X

    # Create y (the target column)
    Y = dataSet["Result"]
    DataCenter.Y = Y

    X_train, X_test, Y_train, Y_test = train_test_split(X,␣
 ↪Y,test_size=CommandCenter.TEST_SIZE)

    X_train = np.array(X_train)
    Y_train = np.array(Y_train)
    X_test = np.array(X_test)
    Y_test = np.array(Y_test)

    # Add Bias to our Features Matrix
    X_train = np.c_[np.ones((X_train.shape[0], 1)), X_train]
    X_test = np.c_[np.ones((X_test.shape[0], 1)), X_test]


    '''
     Here, our X is a two-dimensional array and y is a one-dimensional array.
     Let's make the 'y' two-dimensional to match the dimensions.
    '''
    Y_train = Y_train[:, np.newaxis]
    Y_test = Y_test[:, np.newaxis]


    return X_train , Y_train , X_test , Y_test
```

3.8 Plotter

In order to customize the calculated data , I would write a plotter class of my own. This piece of code shown blow doesn't require any technical explanations and only demands your *patience* and *familiarity* with the syntax of Python which I'm sure you have.

[10]:
```
class Plotter :

    def saveFig(plotPointer , figName , figNumbPath ):
        filerIO = Writer(figNumbPath)
        lines = filerIO.readFile()
        if ( len(lines) == 0 ):
            filerIO.append(1)
            lines[0] = 1
        plotPointer.savefig( str(figName) + str(lines[0]) + '.png')
        filerIO.clearFile()
        filerIO.append(str(int(lines[0]) + 1) )
        print("Figure saved in : " + str(figName))

    def plotter(plotInf, X0set , X1set , ResultSet , path ):
        scaler = CommandCenter.PLOT_SCALER
        X0 = ( X0set )
        weights = DataCenter.Thetas
        X1 = ( X1set )

        class_A = []
        class_B = []

        for i in range(len( ResultSet)) :
            if(ResultSet[i] == 0) :
                class_A.append(i)
            else :
                class_B.append(i)

        # getting the x co-ordinates of the decision boundary
        x_cords = np.array([min(X0) - scaler, max(X0) + scaler])
        y_cords = (weights[1] * x_cords + weights[0]) * (-1 / weights[2])
        # plot class A data with red color
        plt.scatter([X0[i] for i in class_A], [X1[i] for i in class_A],␣
↪color='blue',  label='Iris-virginica')
        # plot class B data with red color
        plt.scatter([X0[i] for i in class_B], [X1[i] for i in class_B],␣
↪color='red', label='Iris-setosa')

        # plot class B data with red color
        plt.plot(x_cords, y_cords, color = 'purple', label="Boundary")
        plt.legend()



        plt.xlabel('X0 - axis')
        plt.ylabel('X1 - axis')
        plt.title(" Logistic Regression : " + str(plotInf))

        plt.legend()
        Plotter.saveFig(plt , path + "/logistic" , "Plots/figureNumber.txt" )
        plt.show()
    # creates a new folder for each drawn plot.
    def createNewDirectory(foldNumbPath):
        import os
        filerIO = Writer(foldNumbPath)
        lines = filerIO.readFile()
        if ( len(lines) == 0 ):
            filerIO.append(1)
            lines[0] = 1
        os.mkdir( "Plots/Iteration-" +str(lines[0] ))
        path = "Plots/Iteration-" + str(lines[0])
        filerIO.clearFile()
        filerIO.append(str(int(lines[0]) + 1) )
        return path
```

3.9 Main

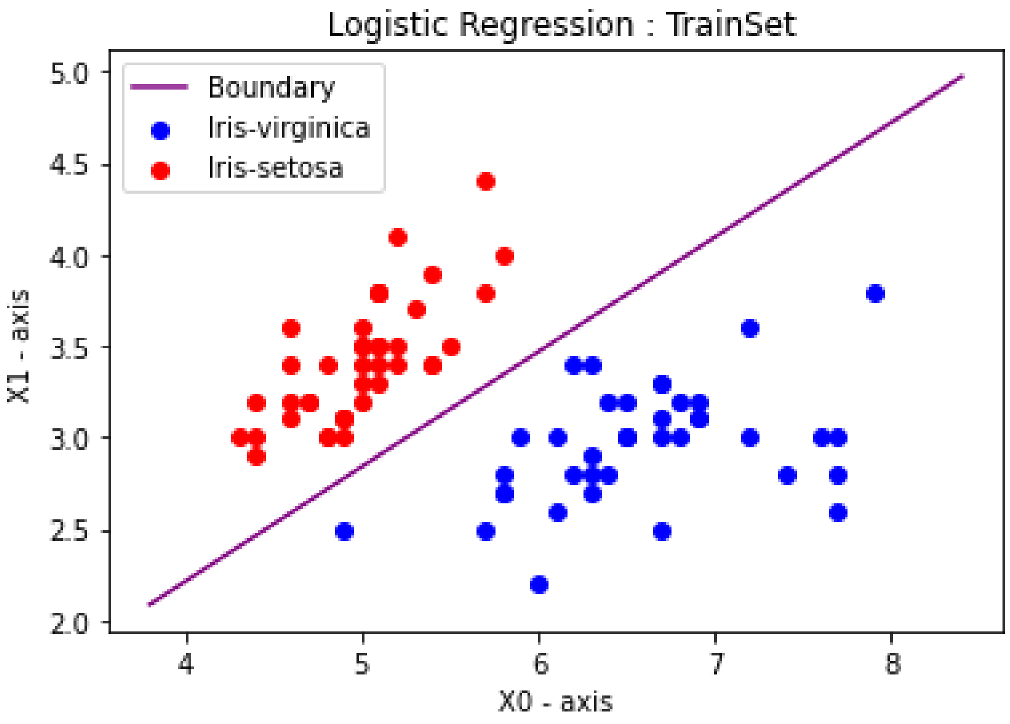
The `main` part of the code, is runnig the functions and showing the results to the user. These actions are being done in the snippet code below.

```
[11]: def showInfo(X_train,X_test, accuracy ,history):

          trainingSetLenght = len(X_train)/(len(X_train) + len(X_test)) * 100
          testSetLenght = len(X_test)/(len(X_train) + len(X_test)) * 100
          print("Training Set Length : %d Percent " %  trainingSetLenght)
          print("Test Set  Length : %d percent" % testSetLenght )
          print("Test Set  Accuracy : %d percent" %accuracy)
          for i in range(len(history)):
              if(i % 10000 == 0 ):
                  print("Error : %f | ItNo : %d" %(history[i],i))

          #S = "Training Set Length :" + str(trainingSetLenght) + " Percent \n" +␣
       ↪"Test Set  Length : "+ str(testSetLenght)+" Percent \n"  + "Test Set ␣
       ↪Accuracy :"+str(accuracy)+" percent \n"
          S = "TestSet : Accuracy :"+str(accuracy) + "%"
          print("============================")
          print("{0} + ({1})X1 + ({2})X2 = 0 " .format(DataCenter.
       ↪Thetas[0],DataCenter.Thetas[1],DataCenter.Thetas[2]))
          print("Or in another words : ")
          print("X2 = [ X1 * ( {0} )  + ({1}) ]/({2})".format(DataCenter.
       ↪Thetas[1],DataCenter.Thetas[0],DataCenter.Thetas[2]*(-1)) )
          print("============================")
          return S

      def main() :
          print("Processing ... Please Wait.")
          DataCenter.X_train , DataCenter.Y_train , DataCenter.X_test , DataCenter.
       ↪Y_test = ReadData(DataCenter.FILE_NAME)
          #initialize the Theta Matrix
          initialThetas = np.zeros((3,1))
          # Train the model
          DataCenter.Thetas, history = train(DataCenter.X_train, DataCenter.Y_train,␣
       ↪initialThetas, 0.1, 100000)
          # plot the model
          plotInf = showInfo(DataCenter.X_train,DataCenter.X_test,␣
       ↪calculate_accuracy(),history )
          path  = Plotter.createNewDirectory("Plots/folderNumber.txt")
          Plotter.plotter("TrainSet" , DataCenter.X_train[:,1] , DataCenter.X_train[:
       ↪,2] , DataCenter.Y_train ,path )
          Plotter.plotter(plotInf , DataCenter.X_test[:,1] , DataCenter.X_test[:,2] ,␣
       ↪DataCenter.Y_test ,path)
```

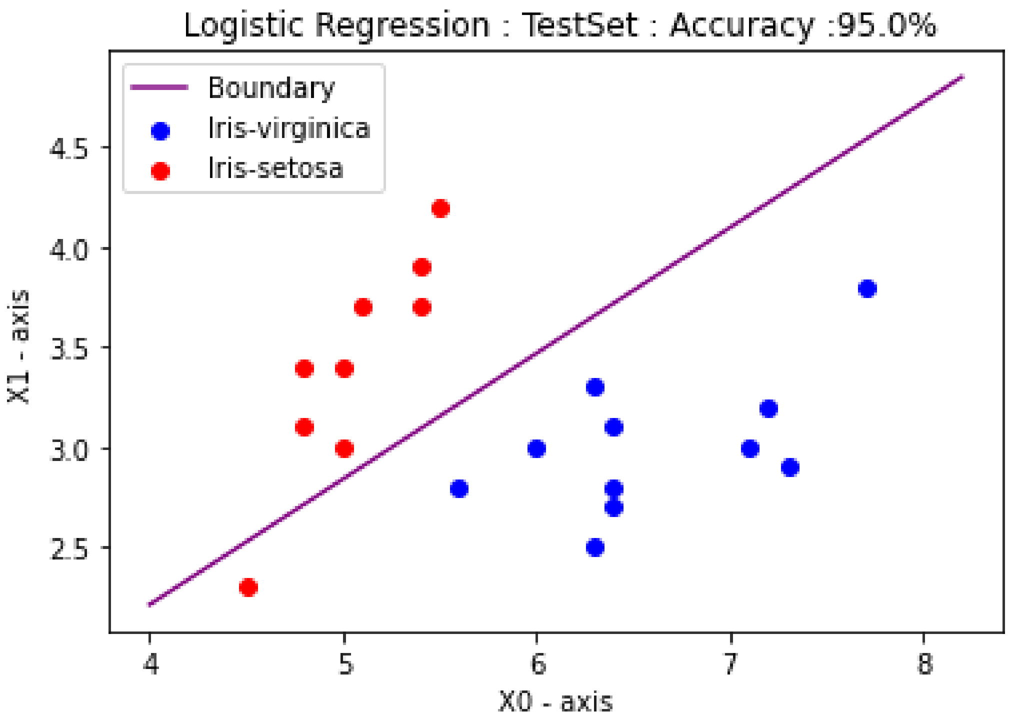
3.10 Run

```
[12]: main()
```

```
Processing … Please Wait.
Training Set Length : 80 Percent
Test Set  Length : 20 percent
Test Set  Accuracy : 95 percent
Error : -0.682713 | ItNo : 0
Error : -0.012222 | ItNo : 10000
Error : -0.007222 | ItNo : 20000
Error : -0.005236 | ItNo : 30000
Error : -0.004142 | ItNo : 40000
Error : -0.003441 | ItNo : 50000
Error : -0.002950 | ItNo : 60000
Error : -0.002587 | ItNo : 70000
Error : -0.002306 | ItNo : 80000
Error : -0.002082 | ItNo : 90000
============================
[4.90640479] + ([-10.287339])X1 + ([16.39270293])X2 = 0
Or in another words :
X2 = [ X1 * ( [-10.287339] )  + ([4.90640479]) ]/([-16.39270293])
============================
Handle Binded With Existing File Named : Plots/folderNumber.txt
Handle Binded With Existing File Named : Plots/figureNumber.txt
Figure saved in : Plots/Iteration-23/logistic
```



```
Handle Binded With Existing File Named : Plots/figureNumber.txt
Figure saved in : Plots/Iteration-23/logistic
```



3.10.1 This Document is written in Jupyter Notebook

3.10.2 With Regards, Hossein Dehghanipour

3.10.3 Spring 2020