60004210155
Jigar Siddhpura
C22

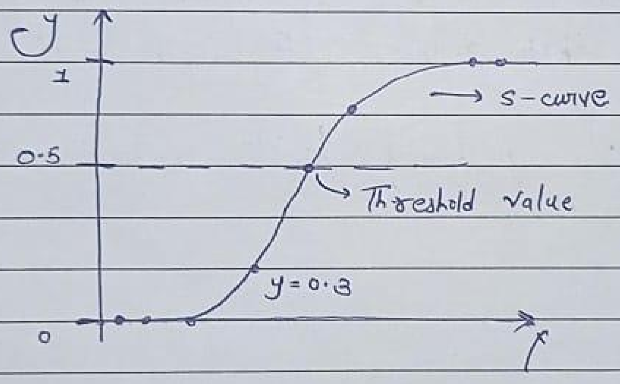
Experiment 3 - Logistic Regression

Aim: To implement Logistic Regression

Theory:

It is one of the popular ML algorithms, which comes under supervised learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. It predicts the output of a categorical dependent variable. Thus, the outcome must be categorical / discrete. It can be either 0/1, True / False, etc.

Logistic function:



- Sigmoid function — Logistic function uses sigmoid function to model the probability starting that a given input belongs to a particular category. Sigmoid function maps any real-valued number to a value ranging from 0 & 1.

- Cost function — It is derived from maximum likelihood estimation. It measures the difference between the predicted probability & actual label.

- Gradient descent — Goal of logistic regression is to minimize the cost function by adjusting the parameters (θ) using optimization algos like gradient descent. It iteratively updates the parameters in the direction that reduces the cost function

- Conclusion — We performed logistic regression

Code :

```
[19] import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
```

```
data = pd.read_csv("/content/gdrive/MyDrive/ML/breast_cancer.csv")
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_w
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	184.60	20
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	158.80	19
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	152.50	17
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	98.87	5
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	152.20	15

5 rows × 33 columns

```
data.drop(['Unnamed: 32',"id"], axis=1, inplace=True)
data.diagnosis = [1 if each == "M" else 0 for each in data.diagnosis]
y = data.diagnosis.values
x_data = data.drop(['diagnosis'], axis=1)
```

```
[5] # normalization
    x = (x_data -np.min(x_data))/(np.max(x_data)-np.min(x_data)).values
```

```
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:84: FutureWarning: In a future version, DataFrame.min(axis=None) will return a scalar min over the entire DataFrame. 1
  return reduction(axis=axis, out=out, **passkwargs)
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:84: FutureWarning: In a future version, DataFrame.max(axis=None) will return a scalar max over the entire DataFrame. 1
  return reduction(axis=axis, out=out, **passkwargs)
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:84: FutureWarning: In a future version, DataFrame.min(axis=None) will return a scalar min over the entire DataFrame. 1
  return reduction(axis=axis, out=out, **passkwargs)
```

```
[12] # train test split
     from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=10)

     # .T = transpose
     x_train = x_train.T
     x_test = x_test.T
     y_train = y_train.T
     y_test = y_test.T

     print("x train: ",x_train.shape)
     print("x test: ",x_test.shape)
     print("y train: ",y_train.shape)
     print("y test: ",y_test.shape)
```

```
x train:  (30, 455)
x test:  (30, 114)
y train:  (455,)
y test:  (114,)
```

```
[13] def initialize_weights_and_bias(dimension):
         w = np.full((dimension,1),0.01)
         b = 0.0
         return w, b
```

```
[14] def sigmoid(z):
         y_head = 1/(1+np.exp(-z))
         return y_head
```

```
def forward_backward_propagation(w,b,x_train,y_train):
    # forward propagation
    z = np.dot(w.T,x_train) + b
    y_head = sigmoid(z)
    loss = -y_train*np.log(y_head)-(1-y_train)*np.log(1-y_head)
    cost = (np.sum(loss))/x_train.shape[1]      # x_train.shape[1]  is for scaling
    # backward propagation
    derivative_weight = (np.dot(x_train,((y_head-y_train).T)))/x_train.shape[1] # x_train.shape[1]  is for scaling
    derivative_bias = np.sum(y_head-y_train)/x_train.shape[1]                 # x_train.shape[1]  is for scaling
    gradients = {"derivative_weight": derivative_weight,"derivative_bias": derivative_bias}
    return cost,gradients
```

```
def update(w, b, x_train, y_train, learning_rate,number_of_iterarion):
    cost_list = []
    cost_list2 = []
    index = []
    # updating(learning) parameters is number_of_iterarion times
    for i in range(number_of_iterarion):
        # make forward and backward propagation and find cost and gradients
        cost,gradients = forward_backward_propagation(w,b,x_train,y_train)
        cost_list.append(cost)
        # lets update
        w = w - learning_rate * gradients["derivative_weight"]
        b = b - learning_rate * gradients["derivative_bias"]
        if i % 10 == 0:
            cost_list2.append(cost)
            index.append(i)
            print ("Cost after iteration %i: %f" %(i, cost))
    # we update(learn) parameters weights and bias
    parameters = {"weight": w,"bias": b}
    plt.plot(index,cost_list2)
    plt.xticks(index,rotation='vertical')
    plt.xlabel("Number of Iterarion")
    plt.ylabel("Cost")
    plt.show()
    return parameters, gradients, cost_list
```

```
[17] def predict(w,b,x_test):
    # x_test is a input for forward propagation
    z = sigmoid(np.dot(w.T,x_test)+b)
    Y_prediction = np.zeros((1,x_test.shape[1]))
    # if z is bigger than 0.5, our prediction is sign one (y_head=1),
    # if z is smaller than 0.5, our prediction is sign zero (y_head=0),
    for i in range(z.shape[1]):
        if z[0,i]<= 0.5:
            Y_prediction[0,i] = 0
        else:
            Y_prediction[0,i] = 1

    return Y_prediction
```

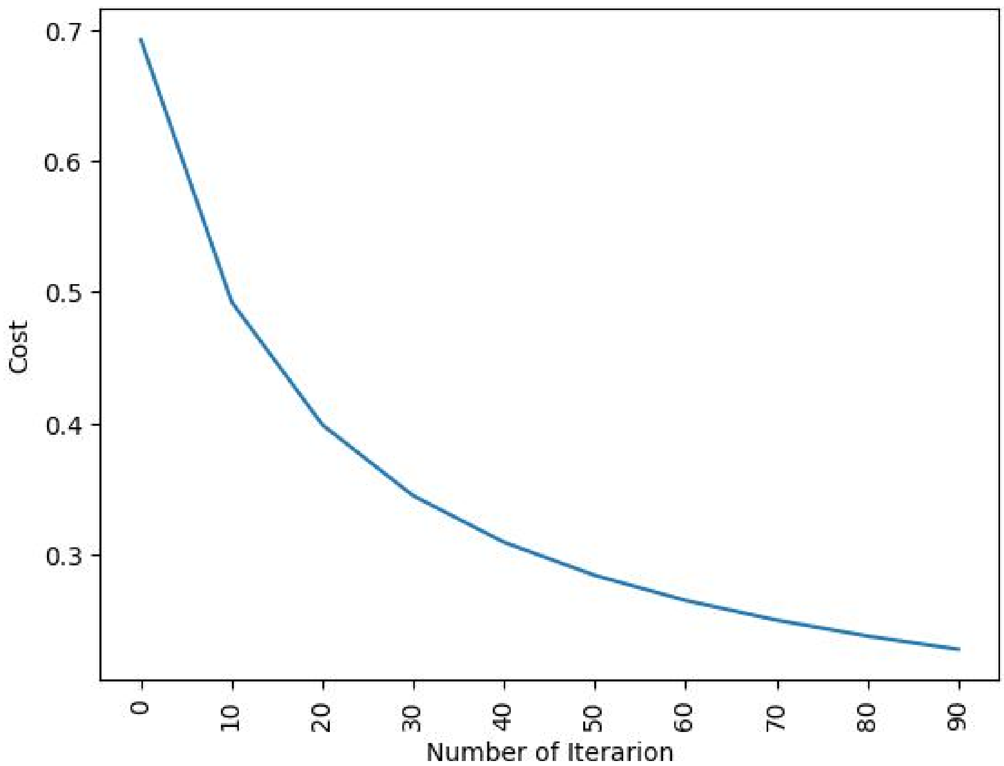
```
[20] def logistic_regression(x_train, y_train, x_test, y_test, learning_rate ,  num_iterations):
    # initialize
    dimension =  x_train.shape[0]  # that is 4096
    w,b = initialize_weights_and_bias(dimension)
    # do not change learning rate
    parameters, gradients, cost_list = update(w, b, x_train, y_train, learning_rate,num_iterations)

    y_prediction_test = predict(parameters["weight"],parameters["bias"],x_test)
    y_prediction_train = predict(parameters["weight"],parameters["bias"],x_train)

    # Print train/test Errors
    print("train accuracy: {} %".format(100 - np.mean(np.abs(y_prediction_train - y_train)) * 100))
    print("test accuracy: {} %".format(100 - np.mean(np.abs(y_prediction_test - y_test)) * 100))

logistic_regression(x_train, y_train, x_test, y_test,learning_rate = 1, num_iterations = 100)
```

```
Cost after iteration 0: 0.692304
Cost after iteration 10: 0.492536
Cost after iteration 20: 0.398952
Cost after iteration 30: 0.344921
Cost after iteration 40: 0.309501
Cost after iteration 50: 0.284275
Cost after iteration 60: 0.265246
Cost after iteration 70: 0.250279
Cost after iteration 80: 0.238126
Cost after iteration 90: 0.228008
```



train accuracy: 94.28571428571429 %
test accuracy: 95.6140350877193 %