19SE02IT058 SEIT4013

Practical-08

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
```

In [2]:
```
#%% import dataset
data = pd.read_csv("data (1).csv")
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)
```

In [3]:
```
# %% normalization
x = (x_data -np.min(x_data))/(np.max(x_data)-np.min(x_data)).values
```

In [4]:
```
# %%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=0.15, random_

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, 483)
x test:  (30, 86) y
train:   (483,)  y
test:  (86,)
```

In [5]:
```
# %%initialize
# lets initialize parameters
# So what we need is dimension 4096 that is number of pixels as a parameter for o
def initialize_weights_and_bias(dimension):     w = np.full((dimension,1),0.01)
b = 0.0     return w, b
```

In [6]:
```
#%% sigmoid
# calculation of z
#z = np.dot(w.T,x_train)+b
def sigmoid(z):     y_head =
1/(1+np.exp(-z))     return
y_head #y_head = sigmoid(5)
```

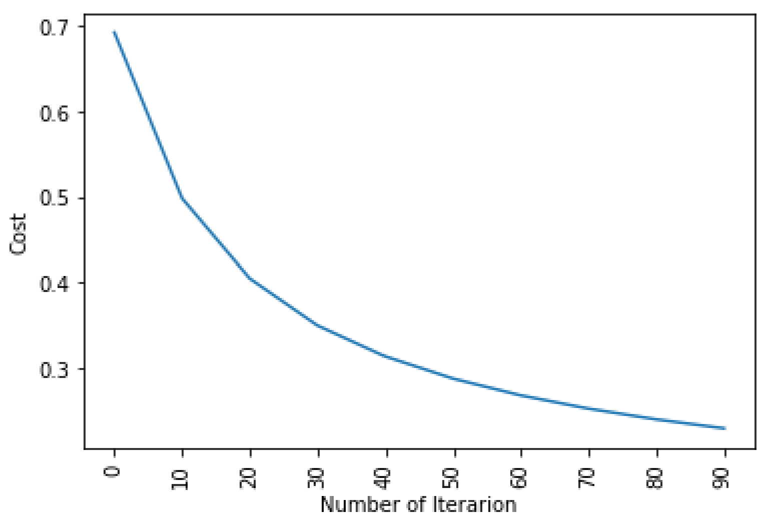
In [7]:
```
#%% forward and backward
# In backward propagation we will use y_head that found in forward progation
# Therefore instead of writing backward propagation method, lets combine forward
def forward_backward_propagation(w,b,x_train,y_train):
    # forward propagation     z
= np.dot(w.T,x_train) + 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 scalin
    # backward propagation
    derivative_weight = (np.dot(x_train,((y_head-y_train).T)))/x_train.shape[1] #
derivative_bias = np.sum(y_head-y_train)/x_train.shape[1]                 # x
gradients = {"derivative_weight": derivative_weight,"derivative_bias": deriva
return cost,gradients
```

In [8]:
```
#%%# Updating(learning) parameters 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
```

In [9]:
```
#%%  # prediction 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
# predict(parameters["weight"],parameters["bias"],x_test)
```

In [10]:
```
# %%
def logistic_regression(x_train, y_train, x_test, y_test, learning_rate ,  num_it
    # 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_ra

    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
print("test accuracy: {} %".format(100 - np.mean(np.abs(y_prediction_test - y

logistic_regression(x_train, y_train, x_test, y_test,learning_rate = 1, num_itera
```

```
Cost after iteration 0: 0.692836
Cost after iteration 10: 0.498576
Cost after iteration 20: 0.404996
Cost after iteration 30: 0.350059
Cost after iteration 40: 0.313747
Cost after iteration 50: 0.287767
Cost after iteration 60: 0.268114
Cost after iteration 70: 0.252627
Cost after iteration 80: 0.240036 Cost
after iteration 90: 0.229543
```




```
train accuracy: 94.40993788819875 % test
accuracy: 94.18604651162791 %
```

In [11]:
```
# sklearn from sklearn import
linear_model
logreg = linear_model.LogisticRegression(random_state = 42,max_iter= 150)
print("test accuracy: {} ".format(logreg.fit(x_train.T,
y_train.T).score(x_test.T print("train accuracy: {}
".format(logreg.fit(x_train.T, y_train.T).score(x_train
```

```
test accuracy: 0.9767441860465116  train
accuracy: 0.968944099378882
```

In [ ]: