```
# PART A
# Implement logistic regression using gradient descent in Python.
# Once you have written the code, test your model by fitting it to the dataset g
iven below.
# For this stage, do not implement regularization, cross validation, the cost fu
nction or use the sklearn library until later.
# The purpose of this part of the coursework is to get your logistic regression
 code running and to validate your model’s results with a small dataset with a k
nown solution.
# Refer to the lecture slides on Logistic Regression to see the solution you sho
uld be obtaining with your model.

import matplotlib.pyplot as plt
import numpy as np
import math

X = np.array([[0.50],[0.75],[1.00],[1.25],[1.50],[1.75],[1.75],[2.00],[2.25],[2.
50],[2.75], [3.00],[3.25],[3.50],[4.00],[4.25],[4.50],[4.75],[5.00],[5.50]])
Y = np.array([0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1])

def logisticFunc(theta0, theta1, x):
    return float(1) / (1 + np.exp(-(theta0 + theta1 * x)))

def gradientDescent(a):
    tempTheta0 = 0
    tempTheta1 = 0
    theta0 = 0
    theta1 = 0
    iterations = 0

    while True:
        T0_sum = 0
        T1_sum = 0
        for i1 in range(0, X.size):
            T0_sum = (T0_sum + ((logisticFunc(tempTheta0, tempTheta1, X[i1]) - Y[
]) * 1))
            T1_sum = (T1_sum + ((logisticFunc(tempTheta0, tempTheta1, X[i1]) - Y[
]) * X[i1]))

        tempTheta0 = (tempTheta0 - (a * T0_sum))
        tempTheta1 = (tempTheta1 - (a * T1_sum))

        diffT0 = abs(tempTheta0-theta0)
        diffT1 = abs(tempTheta1-theta1)

        if(iterations > 5000):
            print "Theta0 & Theta1 = ", theta0, theta1
            print "Number of iterations till convergence = ", iterations
            print "Value of alpha = ", a
            return theta0, theta1

        if((diffT0 < 0.0001) and (diffT1 < 0.0001)):
            print "Theta0 & Theta1 = ", theta0, theta1
            print "Number of iterations till convergence = ", iterations
            print "Value of alpha = ", a
            return theta0, theta1

        iterations += 1
        theta0 = tempTheta0
        theta1 = tempTheta1
```

```
T0, T1 = gradientDescent(0.09009)
```

```
Theta0 & Theta1 =  [-4.07773037] [ 1.50459548]
Number of iterations till convergence =  2712
Value of alpha =  0.09009
```

```
# PART B
# Compute and save the cost function for each iteration of your gradient descent
 algorithm.
# Plot the cost function over the iteration number to answer the following quest
ions.

import matplotlib.pyplot as plt
import numpy as np

X = np.array([[0.50],[0.75],[1.00],[1.25],[1.50],[1.75],[1.75],[2.00],[2.25],[2.
50],[2.75], [3.00],[3.25],[3.50],[4.00],[4.25],[4.50],[4.75],[5.00],[5.50]])
Y = np.array([0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1])

# Logistic
def logisticFunc(theta0, theta1, x):
    return float(1) / (1 + np.exp(-(theta0 + theta1 * x)))

# Cost function
def costFunc(theta0, theta1, x, y):
    h = logisticFunc(theta0,theta1, x)
    return -y*np.log(h) - (1-y)*np.log(1-h)

def gradientDescent(a):
    tempTheta0 = 0
    tempTheta1 = 0
    theta0 = 0
    theta1 = 0
    iterations = 0
    costIter = np.zeros(0)

    while True:
        costSum = 0
        T0_sum = 0
        T1_sum = 0

        for i1 in range(0, X.size):
            costSum += costFunc(tempTheta0, tempTheta1, X[i1], Y[i1])
            T0_sum = (T0_sum + ((logisticFunc(tempTheta0, tempTheta1, X[i1]) - Y[
]) * 1))
            T1_sum = (T1_sum + ((logisticFunc(tempTheta0, tempTheta1, X[i1]) - Y[
]) * X[i1]))

        tempTheta0 = (tempTheta0 - (a * T0_sum))
        tempTheta1 = (tempTheta1 - (a * T1_sum))

        diffT0 = abs(tempTheta0-theta0)
        diffT1 = abs(tempTheta1-theta1)

        if(iterations > 5000):
#              print "Theta0 & Theta1 = ", theta0, theta1
#              print "Does not converge before iteration = ", iterations
#              print "Value of alpha = ", a
            return theta0, theta1, costIter, iterations

        if((diffT0 < 0.0001) and (diffT1 < 0.0001)):
#              print "Theta0 & Theta1 = ", theta0, theta1
#              print "Number of iterations till convergence = ", iterations
#              print "Value of alpha = ", a
            return theta0, theta1, costIter, iterations
```

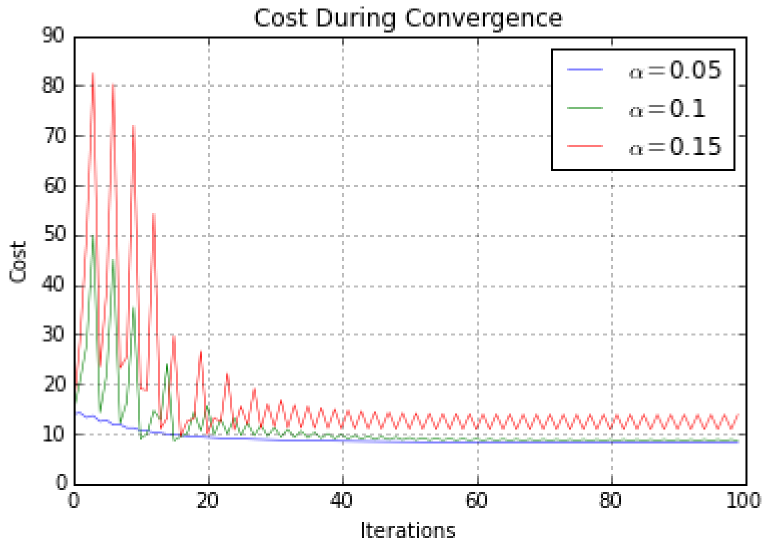
```
        theta0 = tempTheta0
        theta1 = tempTheta1
        costIter = np.append(costIter, costSum)
        iterations += 1

plt.title('Cost During Convergence')
plt.xlabel('Iterations')
plt.ylabel('Cost')

plt.grid(True)

for i in range (0, 3):
    alpha = (i+1) * 0.05
    T0, T1, costIter, iterations = gradientDescent(alpha)
    plt.plot(np.arange( 100 ),  costIter[:100], linewidth = 0.5, label=r'$\alpha
=$%s'%(alpha))

plt.legend(loc='upper right')
plt.show()
```



In [1]:

```
# PART C
# Use the sklearn library function to validate your results.
# The function linear_model.SGDClassifier implements gradient descent and the fu
nction argument loss = ‘log’ applies logistic regression.

import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model

X = np.array([[0.50],[0.75],[1.00],[1.25],[1.50],[1.75],[1.75],[2.00],[2.25],[2.
50],[2.75], [3.00],[3.25],[3.50],[4.00],[4.25],[4.50],[4.75],[5.00],[5.50]])
Y = np.array([0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1])

theta0 = -407
theta1 = 150
numIter = 1

while(True):

    clf = linear_model.SGDClassifier(loss='log', n_iter=numIter)
    clf.fit(X, Y)

    if(numIter > 10000):
        print "Model did not converge within 10000 iterations"
        break

    if(round(clf.intercept_*100) == theta0 and round(clf.coef_*100) == theta1):
        break

    numIter+= 1


print "Number of Iterations Required for Convergence: ", numIter

print "Theta0: ", clf.intercept_
print "Theta1: ", clf.coef_

print "Model prediction for 5 hours of study: ", clf.predict(5)
```

```
Number of Iterations Required for Convergence:  4782
Theta0:  [-4.07341959]
Theta1:  [[ 1.49918078]]
Model prediction for 5 hours of study:  [1]
```

In [ ]: