**Week 9**

**Question 1**

**Part A**

Here is the code for Q1 A

Graphical user interface, text, application, email

Description automatically generated

**Part B**

For this part we take x = 3

Alpha = 1

Chart

Description automatically generated

Alpha = 0.1

Chart, scatter chart

Description automatically generated

Alpha = 0.01

Chart

Description automatically generated

For alpha = 1, we can see that x jumps from -3 to 3, making the change in x at every iteration at 8.

For alpha = 0.1, we find the estimation to be closer to the actual value than the one for alpha = 1 with more iterations

For alpha = 0.01, the estimation is closer than the ones before

**Part C**

Here is the code for Q1 C

Text

Description automatically generated

**Part D**

For alpha = 1 and 0.1

**Chart

Description automatically generated**

For alpha = 0.01

**Chart

Description automatically generated**

**Question 2**

**Part A**

According to the given dataset, the best approach we should be using is the “Bag of words” approach.

For this approach, we proceed to map each remaining word into a feature vector where we use integer values such that we can apply our functions on the data.

Finally, we apply our logistic regression methods with the following loss function to train our model:

m∑mi=1log(1 +e−y(i)θTx(i))

**Part B**

First of all, we separate reviews solely based on text and not on other factors such as stars.

We also assume that the data we have been provided is clean and valid. Invalid data and data with null parameters (like reviews with no text) can cause our model to be trained incorrectly.

Finally, we assume that irrelevant words have either been removed from the feature vector or truncated and the words in the vector are completely relevant. This point can help improve the quality of model after training.

**Part C**

First of all, we would sample from the training data again with replacement and train our model based on this sample. With this result, we can find a set of parameters (with the help of some data analysis) which can minimize the cost function for this data.

Thus, we can use those parameters to estimate the distribution of our values by building a confidence interval.

**Code Used**

import numpy as np

import random

import matplotlib.pyplot as plt

fig = 1

def fx(x):

return (x\*x)-1

def dfdx(x):

return 2\*x

def gradientDescent(x, color, alpha, iters):

\_ = plt.figure(fig) #\_ used as a variable here

for \_ in range(iters):

currentX = x

x = currentX - alpha \* dfdx(currentX)

plt.plot(x, fx(x), color)

return x

def randomVicinity(x, color, alpha, iters):

\_ = plt.figure(fig)

for \_ in range(iters):

randomStep = random.uniform(-1\*alpha, alpha)

if fx(x+randomStep) < fx(x):

x = x + randomStep

plt.plot(x, fx(x), color)

return x

#Q1 b

x = 3

x = gradientDescent(x, 'g.', 1, 1000)

print("Minimum at " + str(x))

print("Alpha = 1")

fig = fig+1

x = 3

x = gradientDescent(x, 'b.', 0.1, 1000)

print("Minimum at " + str(x))

print("Alpha = 0.1")

fig = fig+1

x = 3

x = gradientDescent(x, 'r.', 0.01, 1000)

print("Minimum at " + str(x))

print("Alpha = 0.01")

fig = fig+1

#Q1 C

x = 3

x = randomVicinity(x, 'g.', 1, 1000)

print("R1 Minimum at " + str(x))

fig = fig+1

x = 3

x = randomVicinity(x, 'g.', 0.1, 1000)

print("R1 Minimum at " + str(x))

fig = fig+1

x = 3

x = randomVicinity(x, 'g.', 0.01, 1000)

print("R1 Minimum at " + str(x))

fig = fig+1

plt.show()