Assignment 1
COL 774 Machine Learning
Submitted by : Hardeep Kaur
2019CS10354

Q.1 Implement batch gradient descent method for optimizing J(θ).

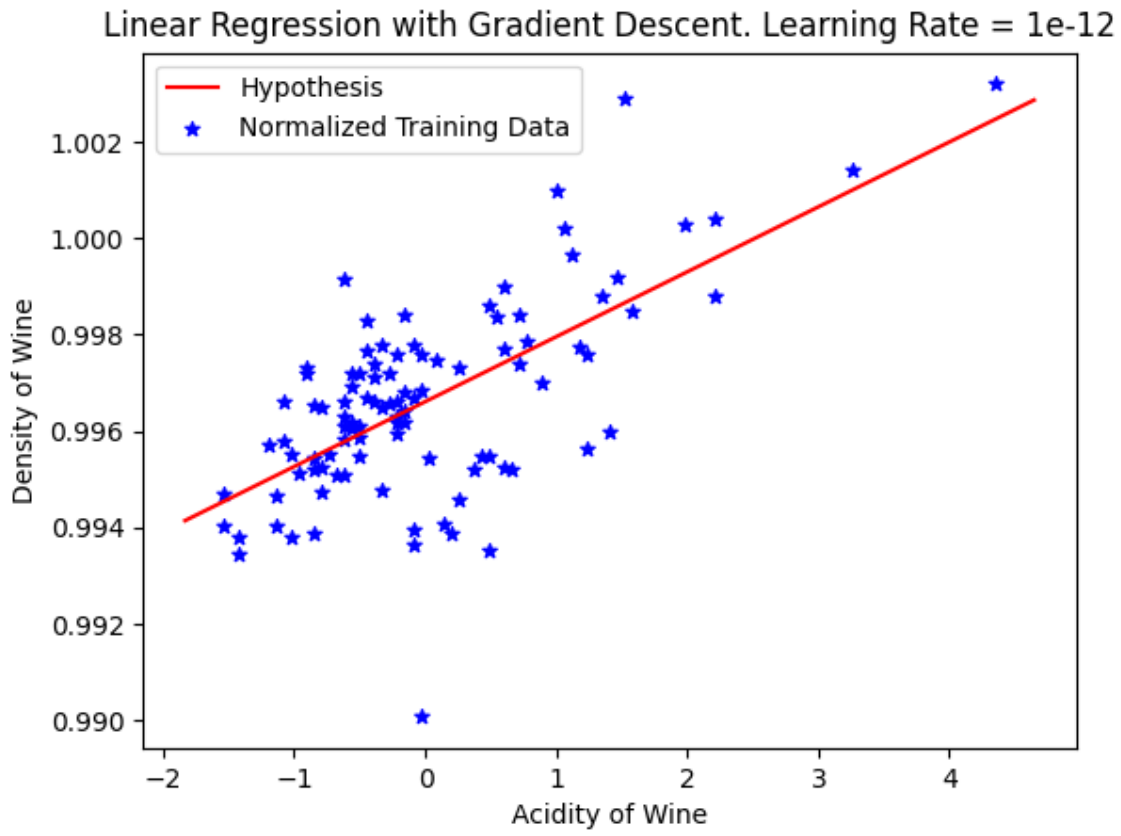
(a) Code in file : LinearRegression.py

Learning rate : 0.01

Stopping Criteria :
Declare convergence when the difference between consecutive cost values fall below 1e-12.

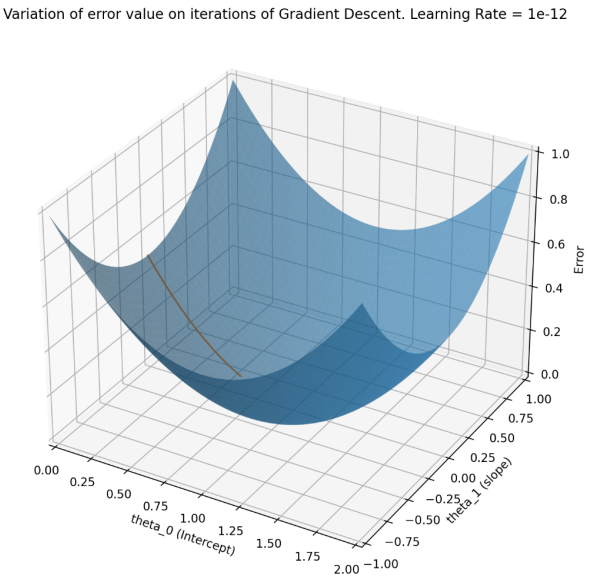
Parameters obtained :
$\theta_0 = 0.9867$
$\theta_1 = 0.0013$

(b) Plot the data and hypothesis function on a 2d graph



(c) Draw a 3d mesh for the error function and display the error values using current set of parameters at each iteration of gradient descent.

Below is a snapshot from the animation. Run the code to see full animation. (Maximize window for better visualisation). Recording is also attached in same directory as ‘ErrorAnimation.mp4’.



(d) Draw the contours and show the error value at each iteration.

Recordings are attached as ‘ContourAnimation_η.mp4’ where η is the learning rate.

Learning Rate	Parameters
0.001	θ_0 = 0.996588 θ_1 = 0.001346
0.01	θ_0 = 0.996610 θ_1 = 0.001346
0.025	θ_0 = 0.996613 θ_1 = 0.001346
0.1	θ_0 =0.996617 θ_1 = 0.001346

We observed that time taken to converge reduced greatly as we increased the learning rate. The parameters learnt in each case were almost similar.

Q.2 (a) Sample one million data points with theta vector = (3,1,2). Refer to question for complete statement.
Code for part (a) is in file ‘SampleData.py’.
The data generated is stored in the same directory as ‘SampleX.csv’ and ‘SampleY.csv’.

(b) Implement stochastic gradient descent method.
Code for part (b) in StochasticGradient.py
(c) Learning Rate, η = 0.001
In the table given below, k represents the number of iterations over which average was taken while deciding convergence, i.e., the cost value averaged over last k iterations would be compared with the cost value averaged over next k iterations.
Note that the iterations on x axis of plots refer to the number of groups of k iterations each.

Batch size (r)	k	δ	Parameters	#iterations	Error on test data
1	1000	0.1	θ_0 = 1.535516 θ_1 = 1.407037 θ_2 = 1.685160	28000	14.5580920
100	1000	0.01	θ_0 = 3.330238 θ_1 = 0.989423 θ_2 = 2.010873	9000	1.038496
10,000	1000	0.001	θ_0 = 2.954686 θ_1 = 1.010624 θ_2 = 1.997922	9000	0.989192
10,00,000	1	0.000001	θ_0 = 2.947233 θ_1 = 1.006069 θ_2 = 1.998378	6658	0.98605970

Error on original hypothesis,i.e., theta = [3,1,2], error = 0.9829469
We observe that models with larger batch sizes are more accurate, with error almost similar to that on original hypothesis.

Parameters learnt by varying the batch size are almost the same and are close to the initial parameters used to generate data for batch sizes 100 and greater. For r = 1, the parameters vary significantly from the original hypothesis. This may be due to the smaller value of delta (which was needed for the program to terminate within a reasonable time).
Relative speed of convergence increases with increasing batch size since each iteration needs to perform more calculations (on a bigger batch of examples).

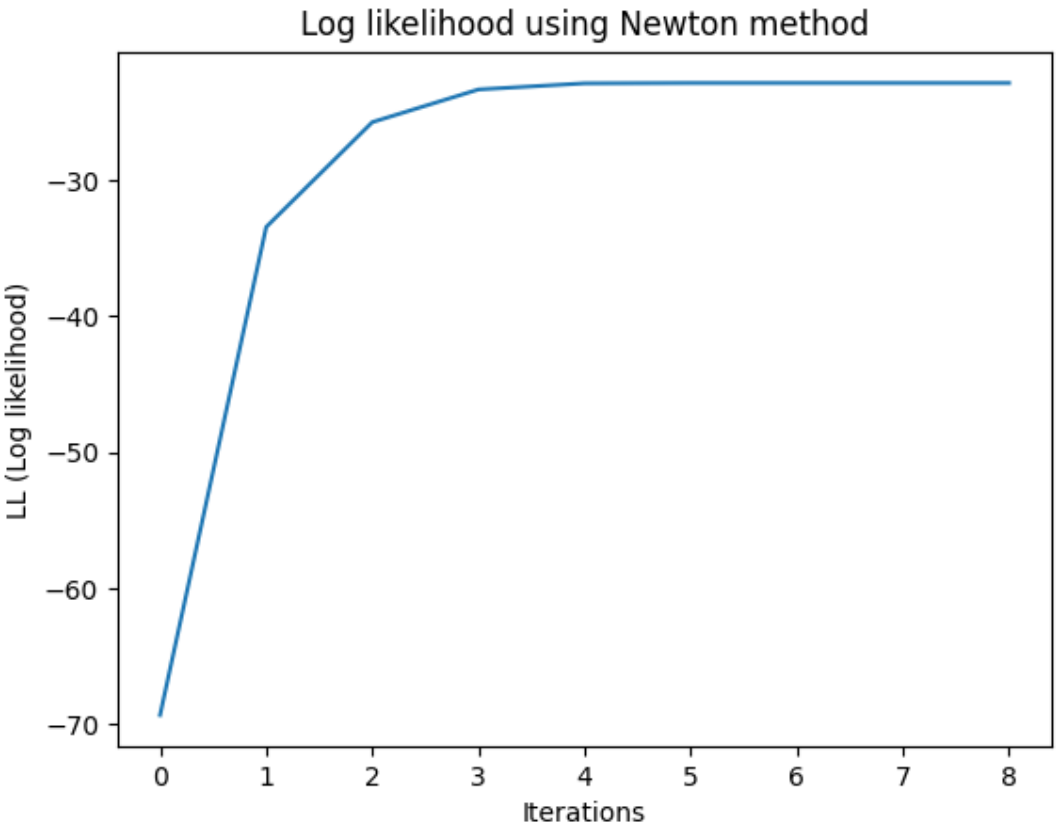
(d) In case of batch size = 1, theta values attained were very random and the movement was not always directed towards the final values. As batch size increases, the movement becomes more and more directed to optimal values. Run the code to see the animation. Snapshots attached in directory.

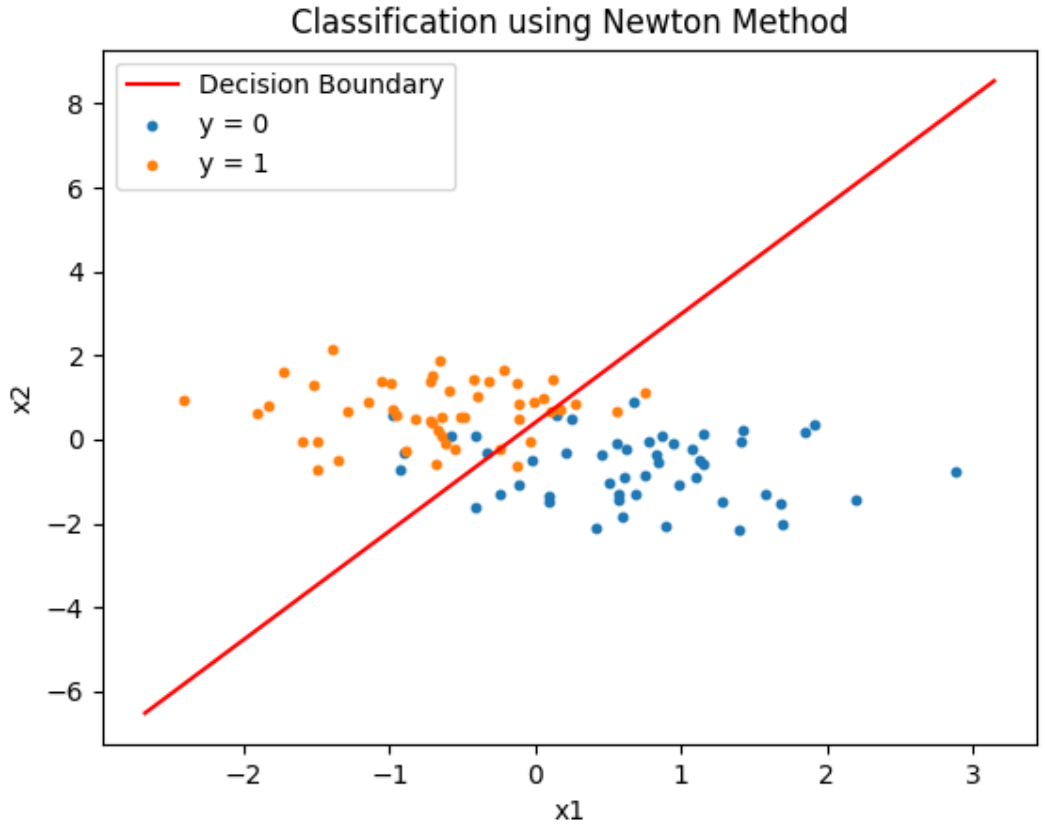
Q.3 (a) Implement Newton's method for classification.
Code in 'NewtonMethod.py'

Parameters obtained :

θ_0 = 0.401253
θ_1 = 2.588547
θ_2 = -2.725588

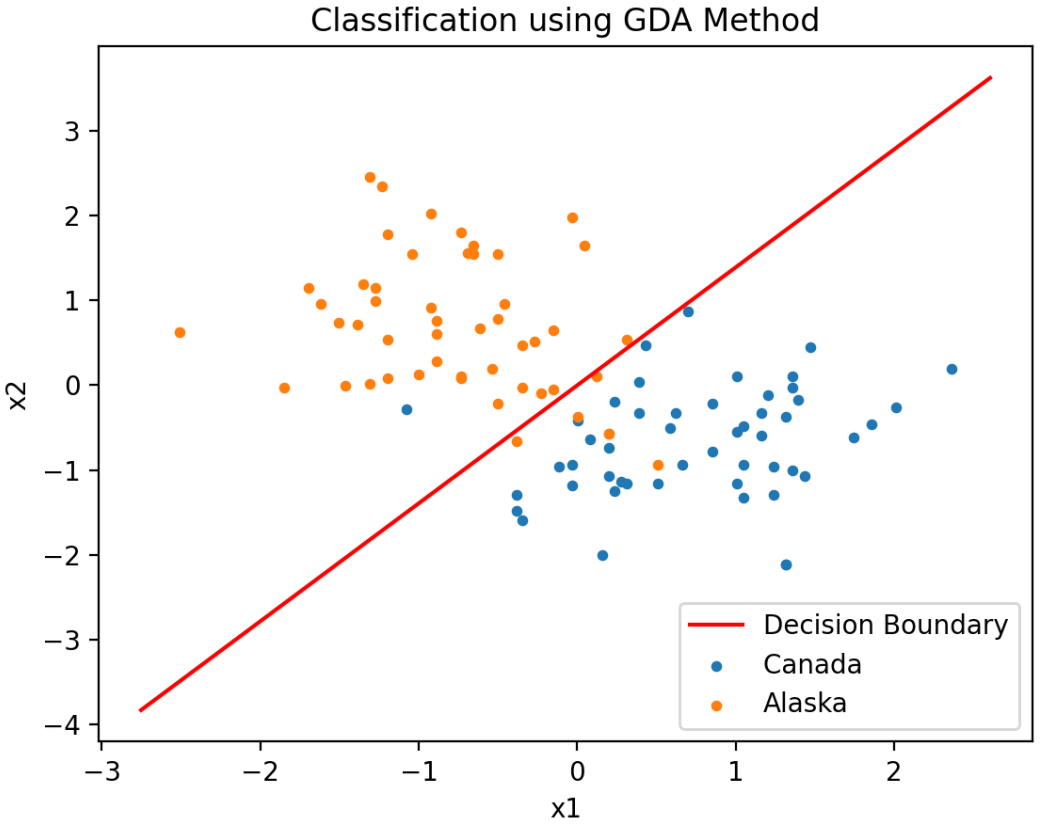
(b) Refer to the plots given below :


Log likelihood using Newton method
LL (Log likelihood)
−30
−40
−50
−60
−70
0
1
2
3
4
5
6
7
8
Iterations


Classification using Newton Method
Decision Boundary
y = 0
y = 1
x2
8
6
4
2
0
−2
−4
−6
−2
−1
0
1
2
3
x1

Q.4 (a) Implement GDA.

Code in file ‘GDA.py’
(b)



μ_0 = [-0.7553, 0.6851]
μ_1 = [0.7552, -0.6851]
Σ = [[0.42953048 -0.02247228]
[-0.02247228 0.53064579]]

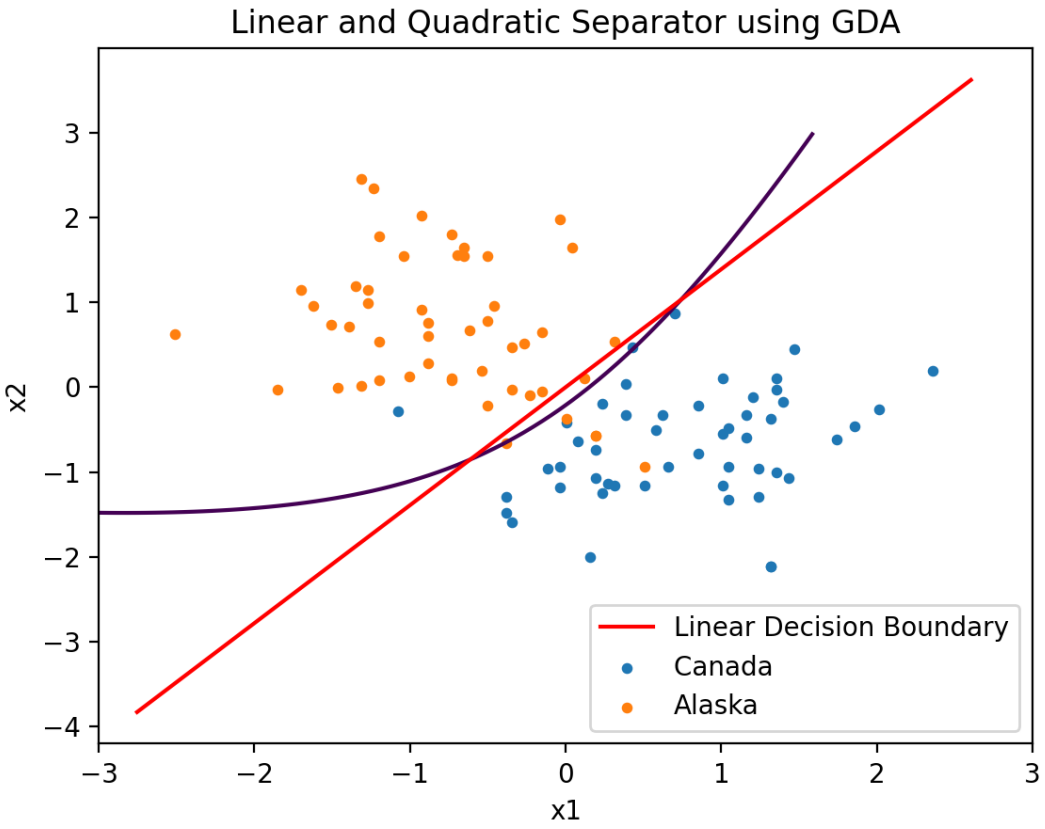
We obtain a linear decision boundary corresponding to the following equation :

$$- [(\mu 1 - \mu 0)' \Sigma^{-1} X - \mu 1' \Sigma^{-1} \mu 1 + \mu 0' \Sigma^{-1} \mu 0 - \log((1 - \Phi)/\Phi)]$$

(d) μ_0 = [-0.75529433, 0.68509431]
μ_1 = [0.75529433, -0.68509431]
Σ_0 = [[0.38158978 -0.15486516]
[-0.15486516 0.64773717]]
Σ_1 = [[0.47747117 0.1099206]
[0.1099206 0.41355441]]

(e) We obtain a quadratic decision boundary corresponding to the following equation :

$$0.5 [X'(\Sigma_1^{-1} - \Sigma_0^{-1})X - 2(\mu 1' \Sigma_1^{-1} - \mu 0' \Sigma_0^{-1}) + \mu 1' \Sigma_1^{-1} \mu 1 - \mu 0' \Sigma_0^{-1} \mu 0] + \log((1 - \Phi)/\Phi)$$



(f) Quadratic boundary classifies the data better than the linear one, due to the reason that it contains more information. But using more parameters may lead to overfitting of data and it may perform less on the test data. Though the linear boundary is simpler, it classifies the data almost equally well.