COL774: Assignment 1 Report

Aniruddha Chatterjee

2021MT10896

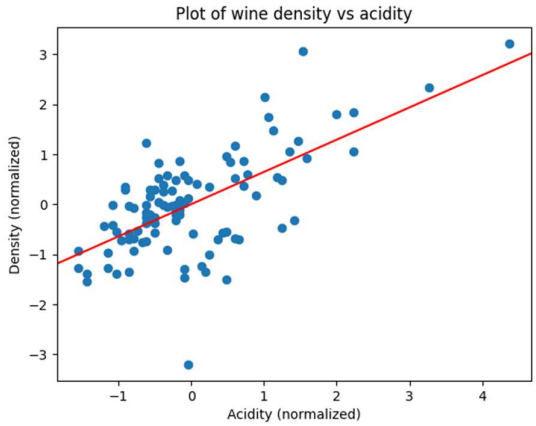
Q1. Linear Regression

(a) The default run of Batch Gradient Descent used the parameters:

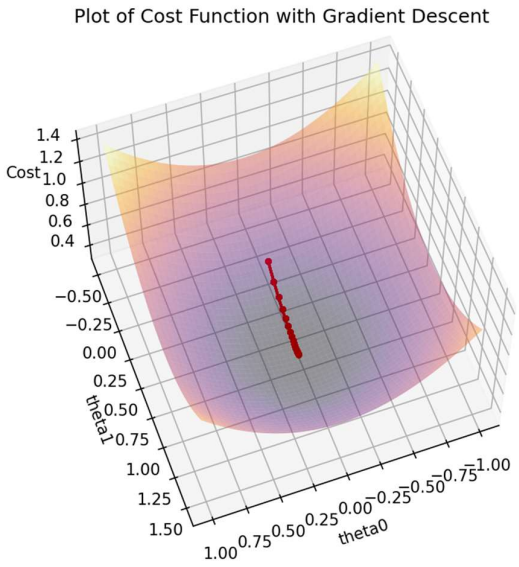
· Learning rate (η) = 0.01 & · Tolerance (ε) =0.0001

Wherein, we obtain θ = [0.00000, 0.64515]

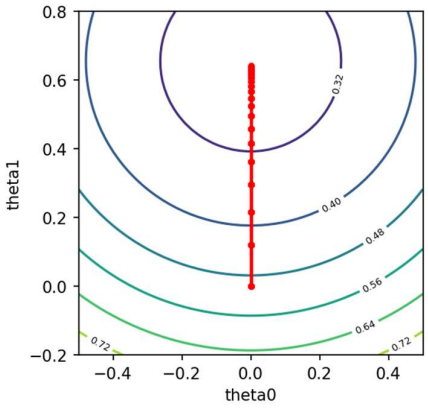
(b) Final Plot of Hypothesis v/s Data:



(c) Path taken by Gradient Descent:



(d) We plot contours to denote the cost function in 2D plane:

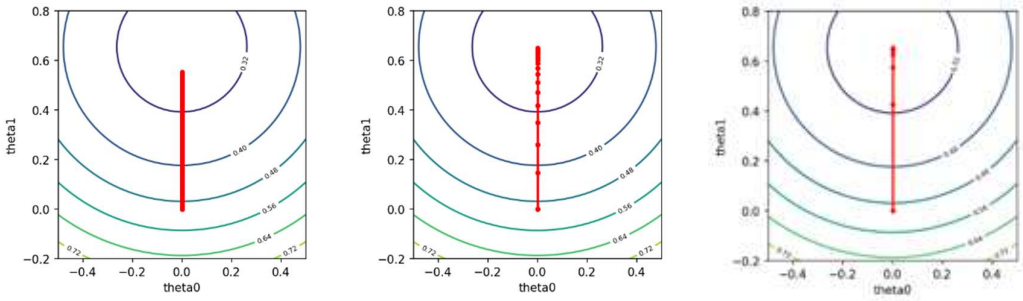


Here, the default learning rate:

- η = 0.01

converges towards the center of the contours.

(e) For varying values of learning rate, we see different rates of convergence towards minima:



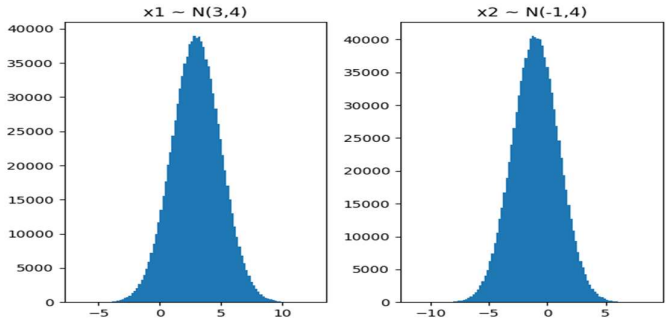
η = 0.001 η = 0.025 η = 0.1

For η = 0.001, the convergence is very slow, and the algorithm doesn't quite reach the minima within time, although it reaches a nearby point.

For η = 0.025 and 0.1, the algorithm converges in cost. While the former moves at a comfortable pace (like the default η = 0.01), the latter takes large steps each iteration, and may overshoot in certain cases.

Q2. Sampling and Stochastic Gradient Descent

(a) Distribution of a random x1 ~ N(3,4) and x2 ~ N(−1,4) would resemble:



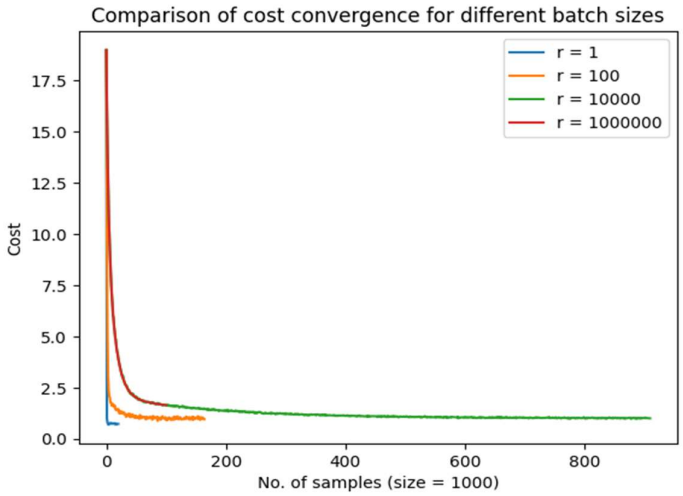
Adding noise to the hypothesis, i.e.-

- $y = h(\theta) + \varepsilon = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \varepsilon$,where $\varepsilon \sim N(0,2)$

(b) Setting the convergence triggers at a certain sample size (about 1000, as suggested in the video) helps prevent the SGD algorithm from terminating at a local stage.

We obtain the following results for different batch sizes:

r = 100	Final theta = [[2.96563182] [1.00332964] [2.00407675]] Avg cost at convergence= 0.9506933324152567 No. of iterations = 1641 No. of samples = 165 No. of epoch = 0 Final cost = 1.0029535382628643
r = 1	Final theta = [[2.95095036] [0.9280175] [2.01483959]] Avg cost at convergence= 0.7269041569463789 No. of iterations = 20001 No. of samples = 21 No. of epoch = 0 Final cost = 1.0521452775445512
r = 10000	Final theta = [[2.76315501] [1.05388153] [1.98230996]] Avg cost at convergence= 0.9999568561006242 No. of iterations = 911 No. of samples = 911 No. of epoch = 9 Final cost = 1.010467922818379
r - 1000000	Max epoch exceeded Final theta = [[0.87959161] [1.45079991] [1.80912212]] Avg cost at convergence= 1.6452558810758158 No. of iterations = 101 No. of samples = 101 No. of epoch = 100 Final cost = 1.6452558810758162



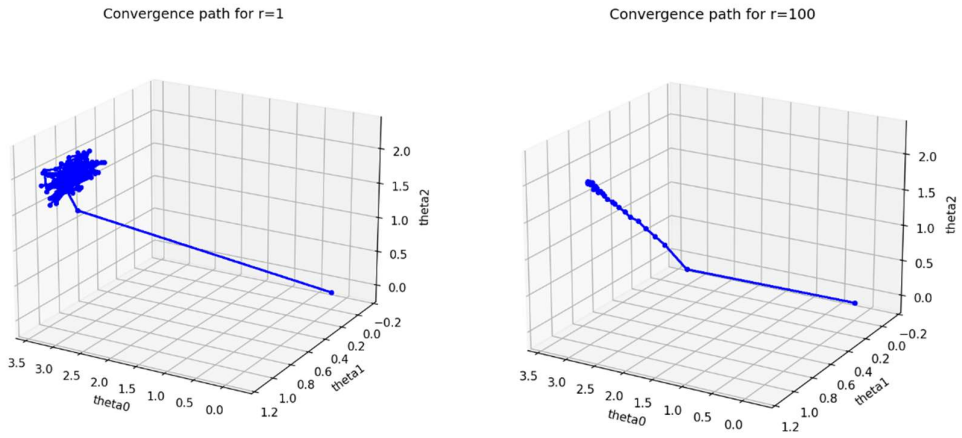
(c) Different batch sizes converge at different rates, as we can see from the number of samples (of size 1000) being tested. While they try to converge to the same value i.e. theta = [[3], [1], [2]], there is randomness in smaller batches, and convergence sometimes deviates from minima.

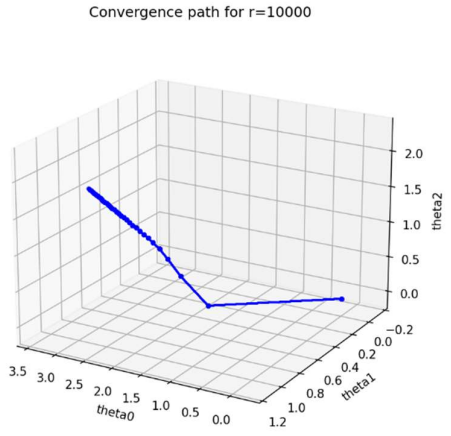
We can also observe that for smaller batch sizes, the average costs are suspiciously low. This may be because the smaller the 'r', the higher is the local bias. Thus, the average cost is quite low even when the actual cost is oscillating around the minima. As can be seen for r = 1.

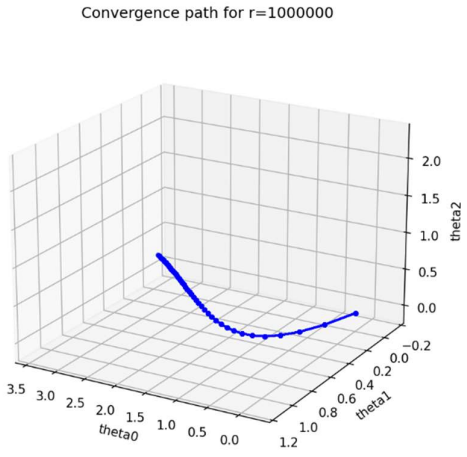
For r = 1000000, we are essentially performing Batch Gradient Descent. Due to the large size, the algorithm is unable to converge in time. Thus, the cost (which is equivalent to actual cost, unlike r = 1) is very high.

We obtain best results for r = 100, 10000, 1, 1000000 in that order. Convergence is fastest in order of smaller batch sizes.

(d) The movement of θ as the parameters are updated (until convergence) for varying batch sizes is as follows:







As the size of the batch increases, the movement of θ is smoother (and slower). This is indicative of the fact that randomness decreases due to the selection of samples from the dataset.

We obtain poor results from the extreme cases:

- r = 1, because the algorithm moves very randomly near the actual convergence point, as can be seen from the graph. Thus, often terminates suboptimally.
- r = 1000000, because the algorithm basically acts as a Batch Gradient Descent (batch size is same as size of dataset) and is very slow in updating the parameters. Thus, the algorithm terminates prematurely as it exceeds the epoch limit.

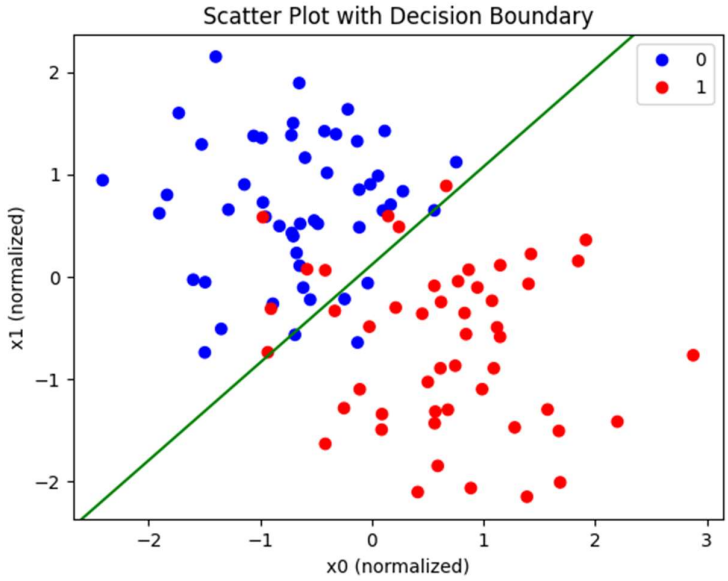
Q3. Logistic Regression

(a) The Log-Likelihood function is:

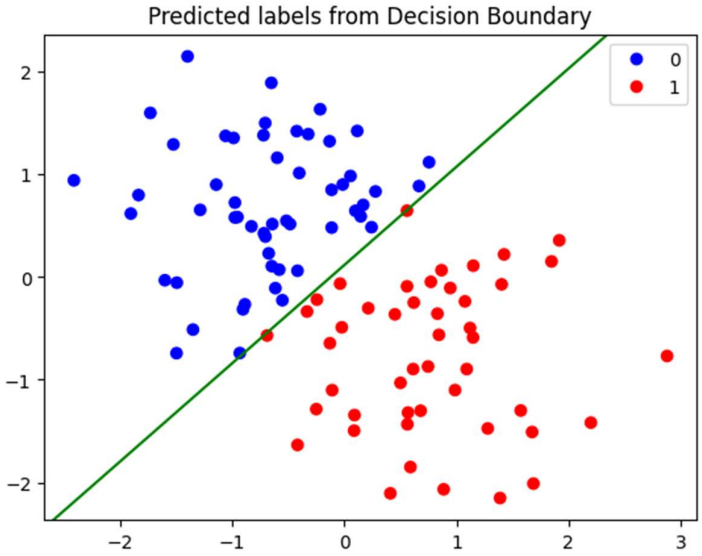
=> $L(\theta) = \sum_{i=1}^{m} y^{(i)} \log h\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h\theta(x^{(i)}))$

- The fitting line has the final coefficients θ = [0.26742, 2.19161, -2.28893]
  Using the stopping criteria ε = 0.0001

(b) Plot of Decision Boundary with actual labelled data:



Plot of Decision Boundary showing predicted labels:

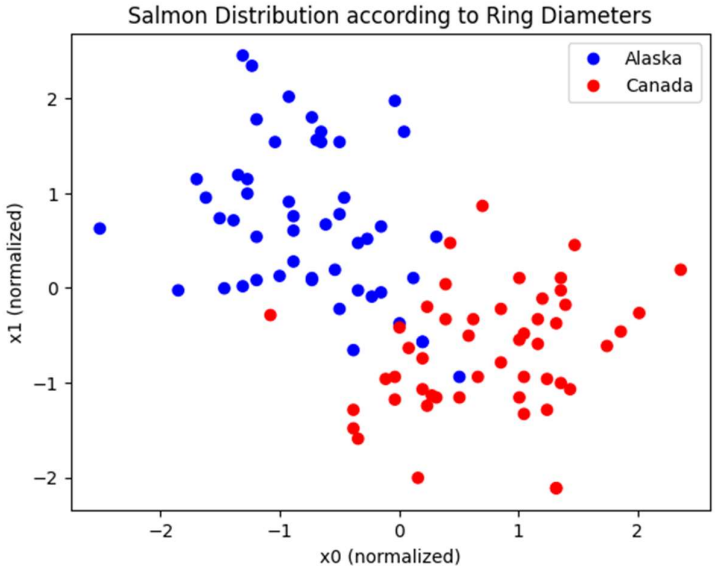


Q4. Gaussian Discriminant Analysis

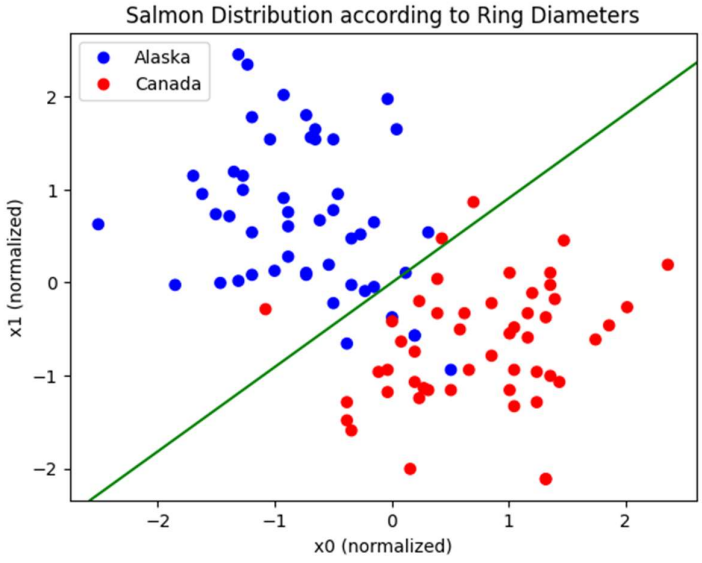
(a) Using the same Covariance matrix for both classes, we obtain parameters:

- $\phi = 0.5$
- $\mu_0 = [-0.75529 \;\; 0.68509]$
- $\mu_1 = [\,0.75529 \;\; -0.68509]$
- $\Sigma = \begin{bmatrix} 0.42953 & -0.02247 \\ -0.02247 & 0.53064 \end{bmatrix}$

(b) Plot of labelled data with respect to input features:
Y = 0 is Alaska while Y = 1 is Canada.



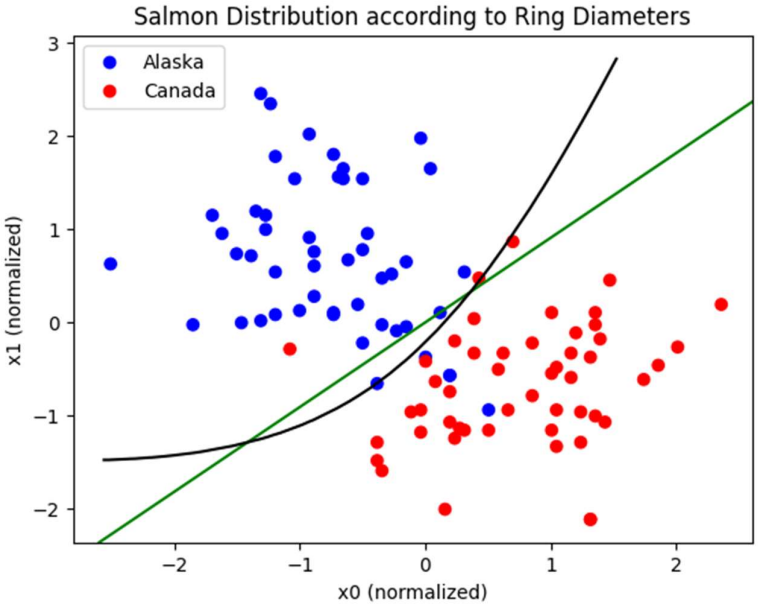
(c) The plotted boundary is linear, given as:
=> $X^T \Sigma^{-1}(\mu_1 - \mu_0) - 0.5(\mu_1{}^T \Sigma^{-1} \mu_1 - \mu_0{}^T \Sigma^{-1} \mu_0) = \ln(\emptyset/1\text{-}\emptyset)$, here $\emptyset = 0.5 = 50/100$



(d) Using separate Covariance matrices for labels, we obtain:

- $\mu_0 = [-0.75529 \;\; 0.68509]$
- $\mu_1 = [\,0.75529 \;\; -0.68509]$
- $\Sigma_0 = \begin{bmatrix} 0.38158 & -0.15486 \\ -0.15486 & 0.64773 \end{bmatrix}$
- $\Sigma_1 = \begin{bmatrix} 0.47747 & 0.10992 \\ 0.10992 & 0.41355 \end{bmatrix}$

(e) The quadratic boundary is a hyperbolic conic, given as:



(f) The linear boundary generated from a common Covariance matrix performs fairly well, having 7-8 misclassifications.

Whereas the quadratic boundary generated from GDA, with separate Σ_0 and Σ_1 for the labels 'Alaska' and 'Canada', performs marginally better with fewer misclassifications.

But, it also requires much more computational time and has a risk of overfitting, which may not be justified by performance gains. Ultimately, it depends on the type of data and features being used.