COL774 Assignment 1

Name : Ujjwal Mehta

Entry No. : 2020CS10401

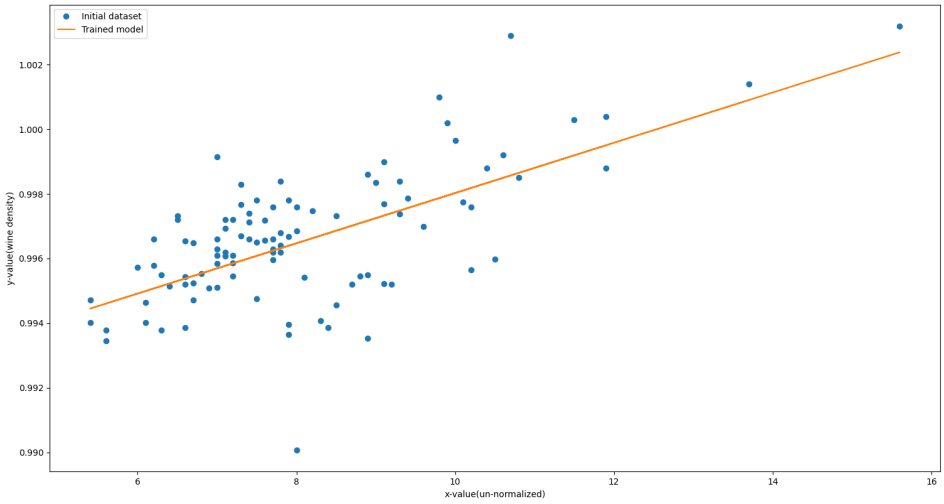
In this assignment we trained different models using various machine learning algorithms. The description of each model(question) is given below.

Question 1

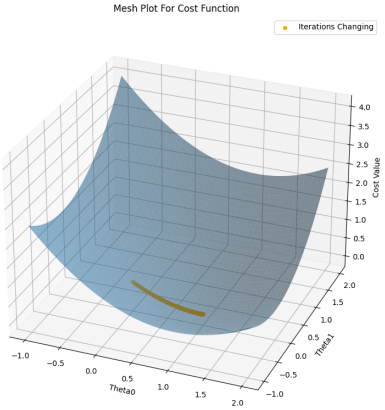
In this question we trained our linear regression model using batch gradient descent algorithm. For the implementation of this question we first read the training data and store it in a numpy array. Then we normalize(using normalize function made in code) the data to get new normalized data with a mean of 0 and sigma of 1. Then we define 2 more function named cost_function and cost_gradient in order to calculate the cost function and its gradient value at a given θ value for the normalized data. Finally we implement the batch gradient descent algorithm by continuously going in opposite direction of gradient until we meet a convergence criteria and then we stop and get the final trained θ values.

1. The learning rate which I chose for training the model is **0.01**, the stopping criteria is when in a single iteration the absolute difference of all the θ_i values is $<$ **0.000001** and the final set of trained parameters is θ = [**0.99652102,0.00134006**] in the general representation.

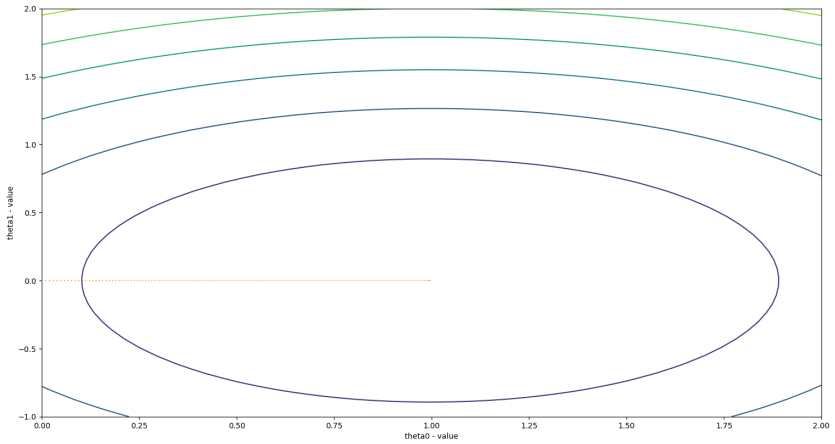
2. The plot of data on 2D graph along with the hypothesis function is shown in the below image :-



3. The mesh plot displaying the error (cost) function for various changing value of θ is shown below :-

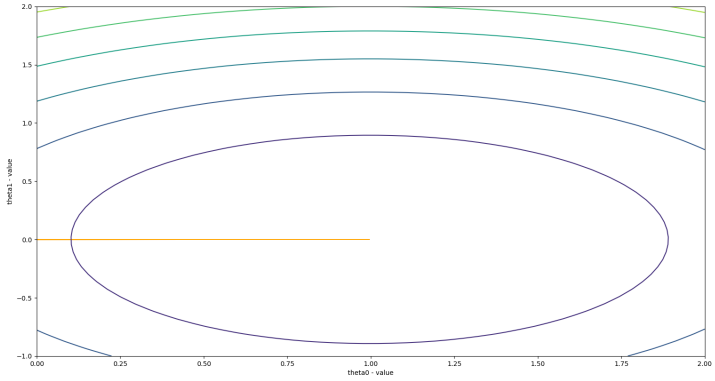


4. The contour plot of the above gradient descent algorithm is shown below for changing θ :-

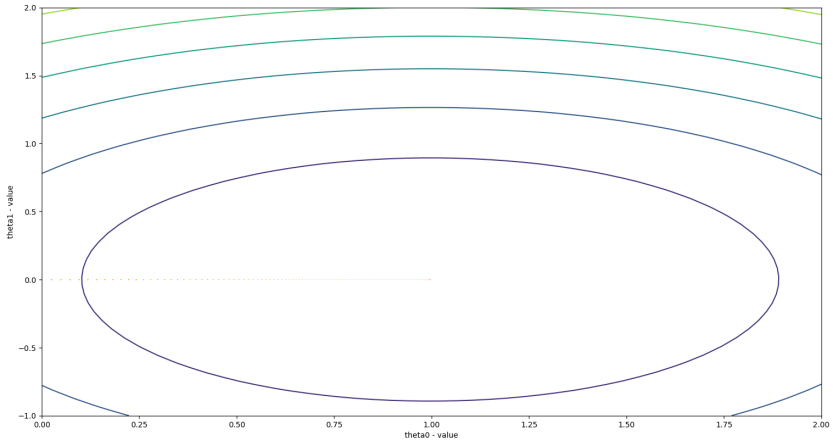


5. When the above part of contour plotting is repeated with varying learning rates then we get the following plots :-

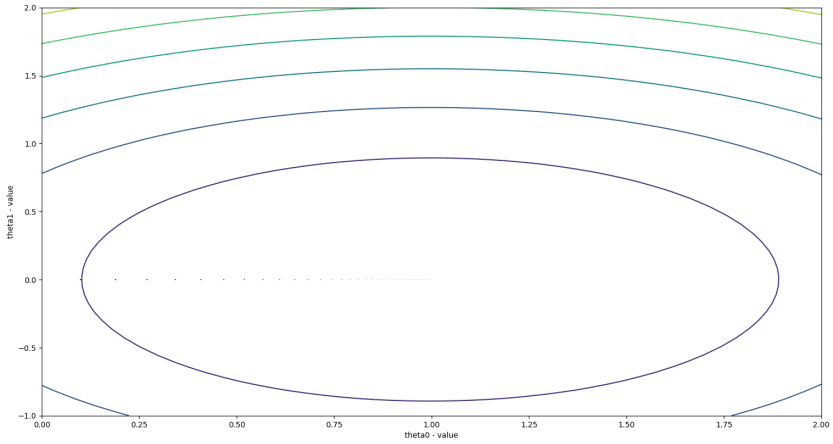
(a) $\eta = 0.001$:-



(b) $\eta = 0.025$:-



(c) $\eta = 0.1$:-



The observation that we can see here is that when we increase the value of η then the step difference between 2 consecutive iterations increases and it converges in less number of iterations (which can be seen from the above

contour plots).

Question 2

In this question we implemented sampling by adding gaussian noises and then stochastic gradient descent to train a synthetic data generated using normal distribution. First we generate 1 million data sets from normal distribution using the inbuilt function of numpy library and then we implement cost_func_batch and cost_derivative_batch functions to get the value of cost function and its gradient for a given batch size and starting index in a shuffled numpy array. Then we finally implement our main stochastic gradient descent algorithm in which we first shuffle our data set and then change our parameter θ according to the gradient of current batch taken in the while loop. Now in order to check for the convergence in this algorithm **I have taken then average value of θ over some avg_iterations (which is a function parameter as it may change with batch size) and when the norm of difference of change in this θ_{avg} and θ becomes less then the stopping criteria (also a function parameter) then I will stop (also when the number of total iterations exceed a certain limit then also I stop the loop as with high batch sizes it will become slow otherwise) and the return the obtained θ value**.

1. We have sampled the data set and added noises in the y_i values using gaussian noises.
2. The various values of trained θ obtained when training with varying batch sizes is shown below :-

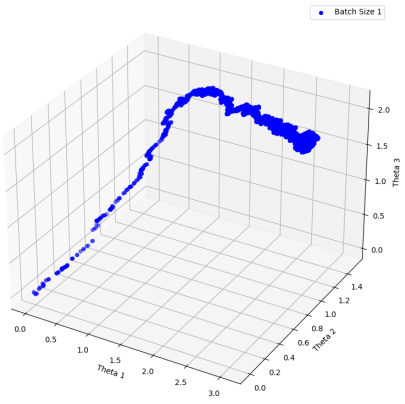
(a) When the batch size is 1 then the value of θ obtained are [**2.97398998,1.06484068,1.94679814**] with a stopping criteria value of **0.0001** (avg size is 1000 iterations for convergence).
(b) When the batch size is 100 then the value of θ obtained are [**2.95709555,1.01050213,1.99719291**] with a stopping criteria value of **0.0001** (avg size is 1000 iterations for convergence).
(c) When the batch size is 10000 then the value of θ obtained are [**0.25924896,0.96042591,0.50952641**] with a stopping criteria value of **0.001**(avg size of 10 iterations for convergence).
(d) When the batch size is 1000000 then the value of θ obtained are [**0.00400129,0.01598149,0.00397306**] with a stopping criteria value of **0.001** (avg size of 1 iteration for convergence).
3. The different algorithms for varying batch size will converge to the same parameter (if they are allowed to run for very long time) but since here I have kept the number of iterations low for high batch size (else the runtime will be too large) so we can see some difference in the obtained values of θ. The run time depends on the batch size and the number of iterations so though the jump might be big in large batch size but then the computation time for each jump will be quite large. The error cost for the 10000 sample data given along with the assignment are as follows for varying batch size :-

(a) Batch Size 1 : Error = 1.3467612
(b) Batch Size 100 : Error = 0.989251
(c) Batch Size 10000 : Error = 113.84862406

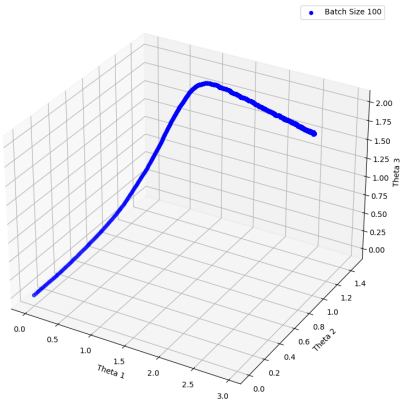
(d) Batch Size 1000000 : Error = 251.55152371

4. The plot showing movement of θ_j for varying batch sizes are shown below :-

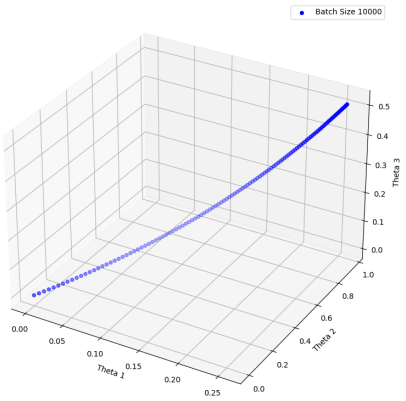
(a) Batch Size = 1 :-



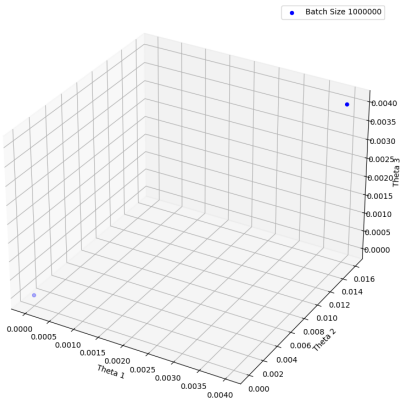
(b) Batch Size = 100 :-



(c) Batch Size = 10000 :-



(d) Batch Size = 1000000(only 2 points there cause stopped early):-



Here we can see that as we increase the batch size the zig-zag movement of variation of θ_j decreases and the

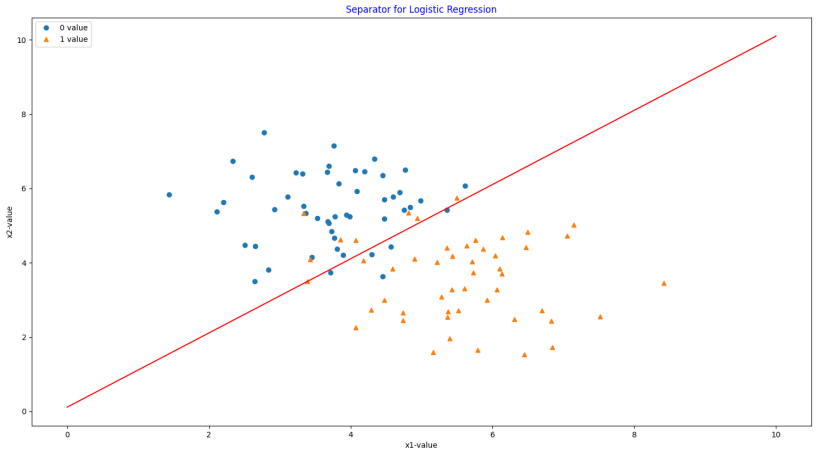
graph tends to become more smooth.

Question 3

In this questions we implemented logistic regression using Newton's method and for implementing that first read the training data from the csv file using numpy and then we normalize it using the similar procedure as question 1 and then we find the hessian H for a given theta value and given training data and this is evaluated using XDX^T where D is a diagonal matrix where $a_{ii} = (h_\theta(x^i))(1 - h_\theta(x^i))$ and X is the training input data matrix. Then we implement Newton's method in which we try to find the root of the gradient of gradient of negative of $LL(\theta)$ which is computed by decreasing θ at each step by matrix product of H^{-1} and gradient of negative of $LL(\theta)$. Then finally we stop by using convergence technique similar to question 1.

1. The trained coefficients (θ value) is [**0.40125304, 2.5885472 ,-2.7255878**] in general notation.

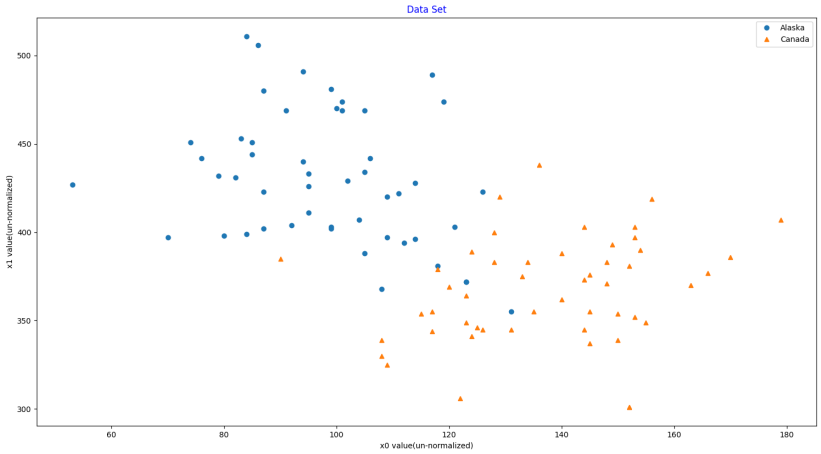
2. The plot of data containing the training data set as well as the decision boundary is given below :-



Question 4

In this question we implemented gaussian discriminant analysis model to find the linear as well as quadratic separator to classify given data and tell whether it belongs to Alaska or Canada. In order to implement this we first calculate the values of ϕ , μ_0 and μ_1 using the equations described in the lecture and we also evaluate the value of Σ as well as Σ_0 and Σ_1 (for the quadratic boundary). Then we finally calculate the boundary by equating the probability of both the classes (Alaska and Canada) as equal.

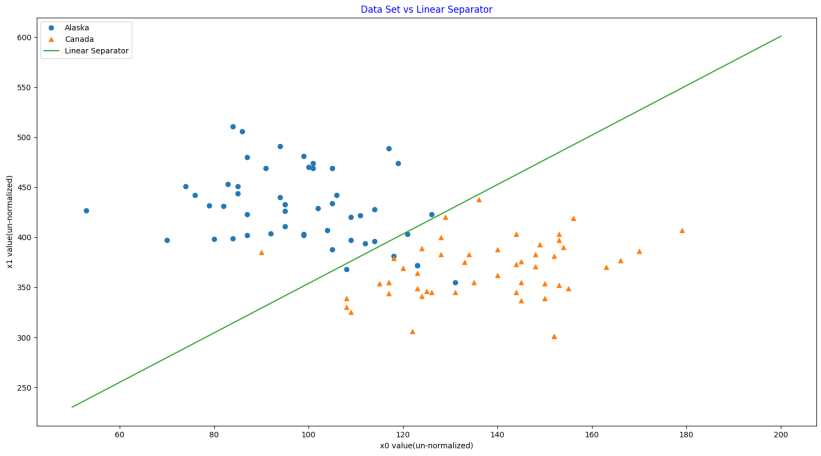
1. In this part the values of μ_0 **is [-0.75529428, 0.68509396],** μ_1 **is [0.75529443, -0.68509459] and** Σ **is [[0.42953051,-0.02247228], [-0.02247228, 0.53064575]].**
2. The plot of the given data set without any separator is shown below :-



3. Here as discussed in the class we can calculate the decision boundary by equating equal probabilities for both classes Alaska and Canada and by doing so we will get the linear equation which is given by (note that I am writing a reduced form of the equation) (we can observe that μ_0 is approximately negative of μ_1) :-

$$x^T \Sigma^{-1} \mu_0 = 0$$

Now the plot depicting this linear separator is given below :-



4. If we consider different covariance matrices then the values obtained of μ_0 **is [-0.75529428, 0.68509396],** μ_1 **is [0.75529443, -0.68509459],** Σ_0 **is [[0.3815898, -0.15486515], [-0.15486515, 0.64773712]] and** Σ_1 **is [[0.47747121,0.1099206], [0.1099206,0.41355438]].**

5. The equation of quadratic boundary as described in the class can be obtained in a similar manner as done in linear separator case and here the equation is given by :-

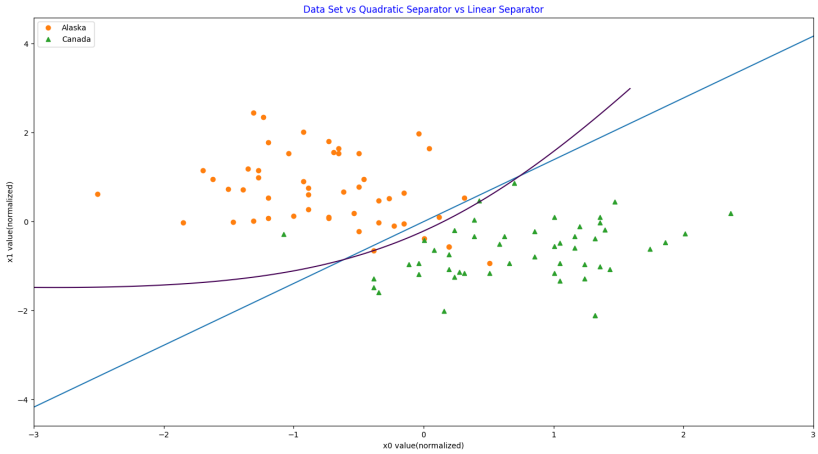
$x^T(\Sigma_0^{-1} - \Sigma_1^{-1})x + 2x^T(\Sigma_1^{-1}\mu_1 - \Sigma_0^{-1}\mu_0) + K = 0$

where K is given by :-

$K = (\mu_0^T\Sigma_0^{-1}\mu_0 - \mu_1^T\Sigma_1^{-1}\mu_1) + c$ where c is given by :-

$c = \log_(\det(\Sigma_0)/\det(\Sigma_1))$

The plot of the quadratic separator along with the data set and linear separator is shown below :-



6. After carefully observing the linear as well as the quadratic boundary we can say that the quadratic fitting is better and will provide us with a better model when the number of points in the data set are increased.