REPORT

CHIRAG MOHAPATRA
2018CS50403

---

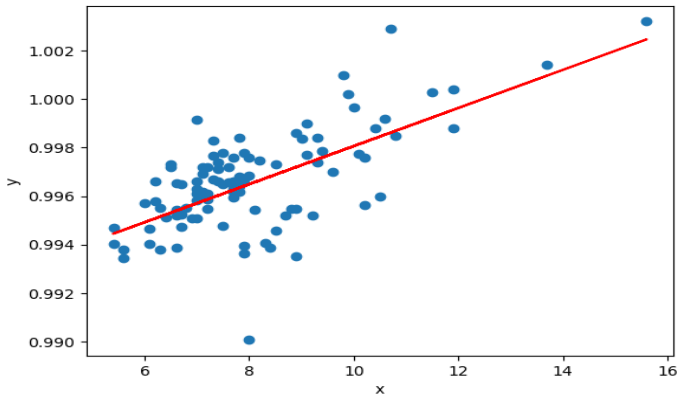
1. In this question, we implement batch gradient descent with linear regression.

   (a) After some testing, these are my final results:

   Learning rate eta: 0.01
   Stopping parameter epsilon: 1e-10
   Number of epochs required to converge: 918
   Weights learned by model: [[0.99653605 0.00135781]]

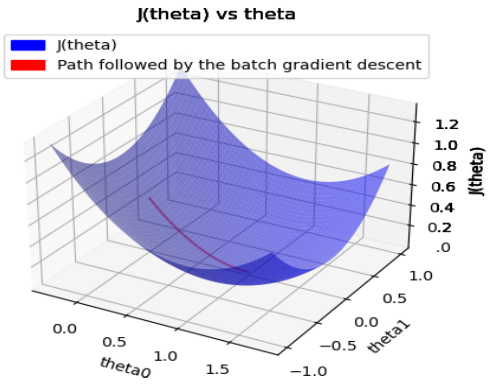
   My model converges when:
   $|J(\theta^{t+1}) - J(\theta^t)| \leq \text{epsilon} = 10^{-10}$

   (b) This is the plot of the hypothesis along with the training data.

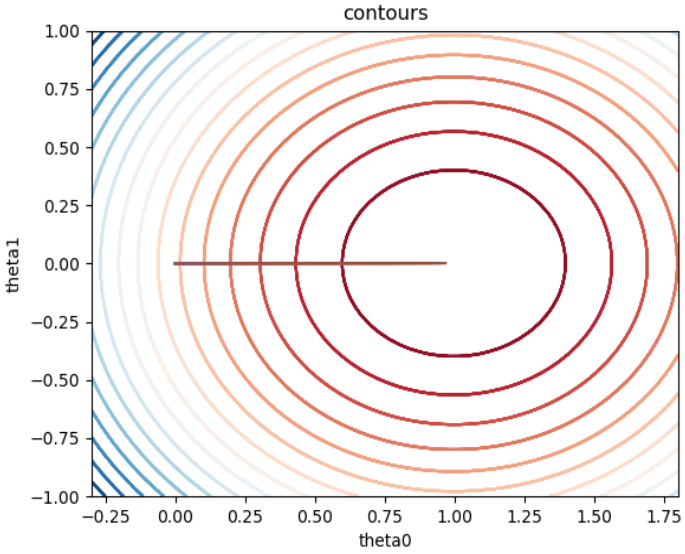


   Here, the red line shows the hypothesis function. From the plot, it is clear that our model provides a good linear approximation.

   (c) I animated the 3D mesh by plotting at different iterations and it is apparent how J(theta) descends to a minima. This is the final plot:



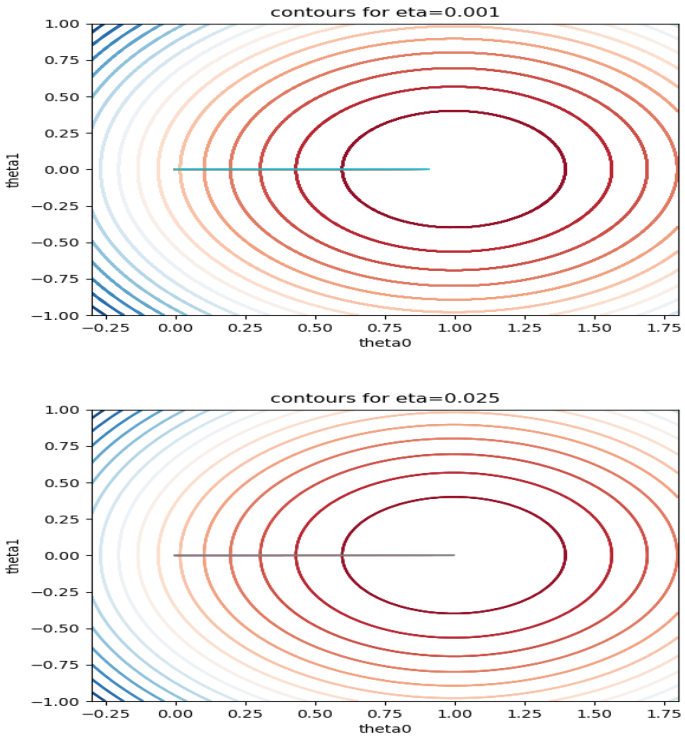
   (d) The contours of the error function are as shown:

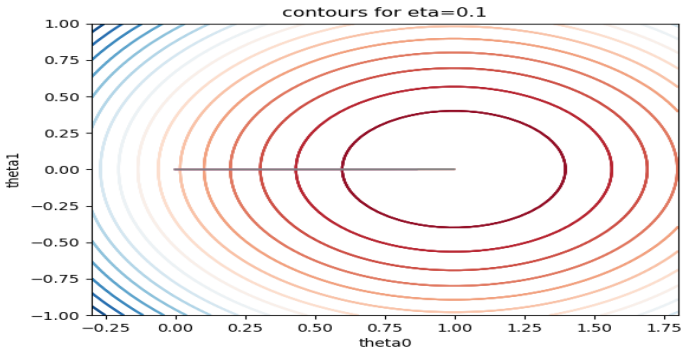


The thetas start at (0,0) and move towards (0.996,0.0013) which are the final weights learnt by model.

(e) All the different learning rates converge to almost the same weights as expected since our stopping criteria is uniform.

The main difference is that 0.001 converges very slowly while 0.1 moves very fast and may overshoot the minima.





Note: In the above figures, it may appear that 0.001 is not converging but that is not the case, it is just that I ran the plot for a fixed number of iterations and 0.001 learning rate could not converge within that time.

After testing, I found that for the data, my learning rate of 0.01 converges at a decent pace with minimal chance of overshooting.

2. This question implements stochastic gradient descent with linear regression.
   (a) One million data points were generated for y with:
   $y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \epsilon$
   (b) After much testing, I have developed different convergence criteria for each batch size:
   - For batch size 1: I used the same convergence criteria as in Andrew Ng's video-
     $|J_l(\theta^{t+1}) - J_l(\theta^t)| \leq \epsilon = 0.001$ where $J_l(\theta)$ denotes the average cost of last 1000 iterations.
     The stats are:
     Weights learned: [[3.01786788] [1.01012585] [2.06498334]]
     Time taken: 5.7392637729644775 seconds
     Number of iterations: 56000
     Squared error difference from original hypothesis: 0.0015482094291194848
     As we can see, the weights learned are quite close to the original (3,1,2).
   - For batch size 100: Here also I took the average of last 1000 iterations except ϵ is set as 0.002 here.
     The stats are:
     Weights learned: [[2.86125288] [1.03185816] [1.97985145]]
     Time taken: 5.403133869171143 seconds
     Number of iterations: 11000
     Squared error difference from original hypothesis: 0.0068905563391756615
     The weights learned here are also very good.
   - For batch size 10000: Here I used a variation of the convergence criteria that sir taught in class except instead of considering k consecutive, I considered any k iterations where k=400.
     In other words, if $|J_l(\theta^{t+1}) - J_l(\theta^t)| \leq \epsilon = 0.001$ where $J_l(\theta)$ denotes the cost of an iteration is satisfied k=400 times, then I consider it converged.
     I tested it multiple times and it works great(in retrospect, this is somewhat similar to considering number of epochs).
     These are the stats:
     Weights learned: [[2.97751059] [1.0041928] [1.99750886]]
     Time taken: 4.978362560272217 seconds
     Number of iterations: 17462
     Squared error difference from original hypothesis: 0.00017651962281546878
     The results are very good here.

- For batch size 1000000: This is basically batch gradient descent. I set ϵ as 10^{-5} in this case. On average, this takes about 80-85 seconds to converge.
  Weights learned: [[2.64260287] [1.07757146] [1.97314924]]
  Time taken: 67.36932563781738 seconds
  Number of iterations: 7550
  Squared error difference from original hypothesis: 0.04482366855935691
  Here, the results are not that good which is to be expected with such huge batch size.

(c) This is the mean square error on the test data with the weights of our learned model:
Test error: 0.9829469215000001
Running test data on batch size 1:
Error: 1.1916402702804165

Running test data on batch size 100:
Error: 1.0592651911556976

Running test data on batch size 10000:
Error: 0.9844746432881966

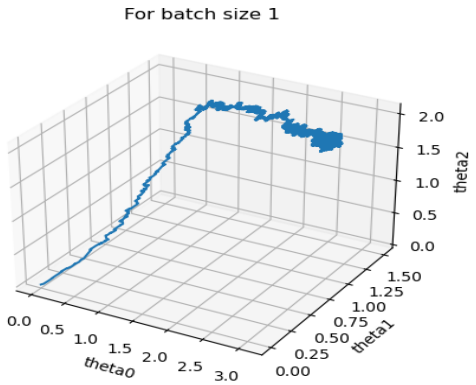
Running test data on batch size 1000000:
Error: 1.3504081428392378

Note: Here batch size 10000 seems to be performing extraordinarily well but that is not always the case since our distribution has random noise. Mostly, batches 1,100 and 10000 show similar results with 1 needing the most iterations followed by 100 then 10000 and finally 1 million.

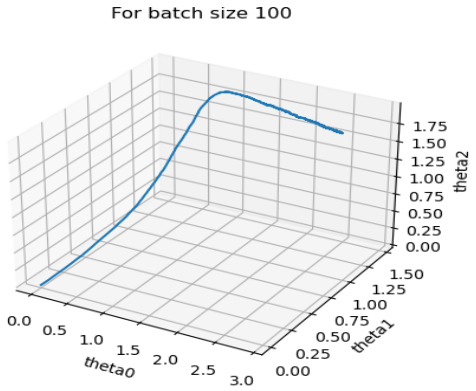
Analysing our results with the different convergence criteria that we came up with, it is apparent that:

- Batch size 1 million performs worst in terms of time to converge and also the error. However, it takes the least number of iterations which is as expected since this becomes Batch Gradient Descent.
- For the other batch sizes, the results are mostly comparable with some randomness arising because of the random noises.
  We discover that batch size 1 always requires the most number of iterations which is not surprising.
  The times to converge are comparable in the three cases(albeit batch size 10000 has a vastly different convergence criteria).

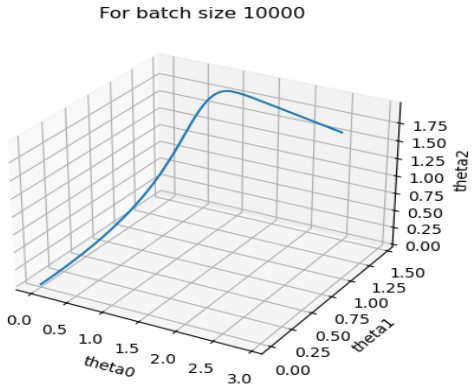
(d) These are the movements of θ:



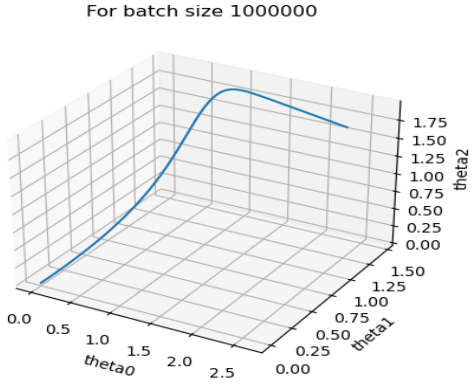
For batch size 1, the movements plotted make intuitive sense since there would be randomness in the selection of batches and so there are irregularities, however as a whole they converge.



Here, there are slight irregularities, however not as much as the batch size 1 case which makes sense since our data is randomly shuffled.



The number of irregularities decrease and this makes sense because in a batch, the irregularities balance out and we get a proper distribution.



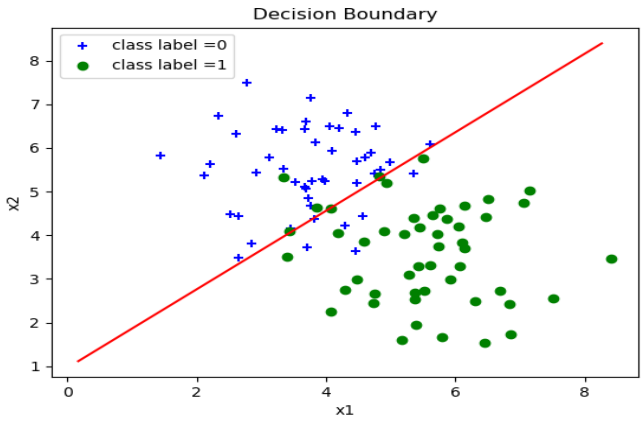
Finally, the 1 million batch size plot is the smoothest plot.

3. In this question, we implement a logistic regression classifier and train using Newton's method of convergence. I calculated the j,kth element of the Hessian analytically.

$H_{jk} = \frac{\partial L(\theta)}{\partial \theta_j \theta_k} = \quad (-1) * \sum_{i=1}^{m} {x_j}^{(i)} {x_k}^{(i)} \frac{e^{-\theta^T x_i}}{(1+e^{-\theta^T x_i})^2}$

(a) After calculating the Hessian, the update follows the Newton's method and the weights converge. The convergence criteria I consider is that:
$|{\theta_j}^{(t+1)} - {\theta_j}^{(t)}| \leq \text{delta} = 10^{-7} \ \forall j$
The weights I learned are:
[[0.46722676 2.55770122 -2.78143761]]

(b) The decision boundary is linear, it is infact $\theta^T x = 0$. This is the plot:

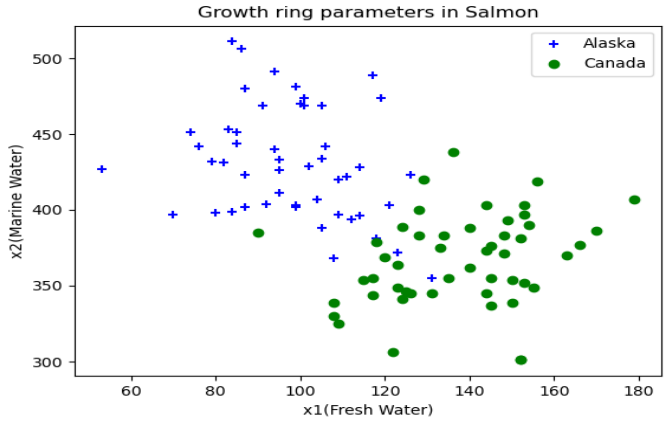


For those samples lying above the boundary, the probability of them being 1 is less.

4. In this question, we had to implement a Gaussian Discriminant Analysis Model.

(a) We already calculated the parameters analytically in class. Using those formulas, these are the parameters I get:
phi = 0.5
mu0 = [[-0.75529433 0.68509431]]
mu1 = [[0.75529433 -0.68509431]]
sigma = [[0.42953048 -0.02247228] [-0.02247228 0.53064579]]
where 0 label indicates Alaska and 1 indicates Canada.

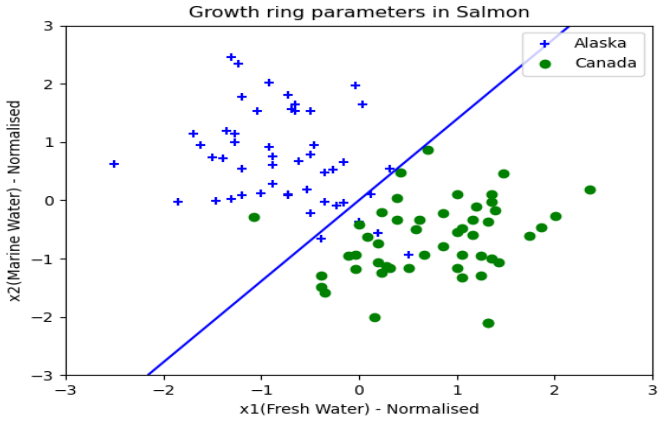
(b) This is the plot of the training data:



(c) Using the parameters, we can calculate the decision boundary,
The equation of the decision boundary is:

$2log(\frac{\phi}{1-\phi}) + log(\frac{|\Sigma_0|}{|\Sigma_1|}) = x^T({\Sigma_1}^{-1} - {\Sigma_0}^{-1})x - 2x^T({\Sigma_1}^{-1}\mu_1 - {\Sigma_0}^{-1}\mu_0) + {\mu_1}^T\Sigma_1\mu_1 - {\mu_0}^T\Sigma_0\mu_0$

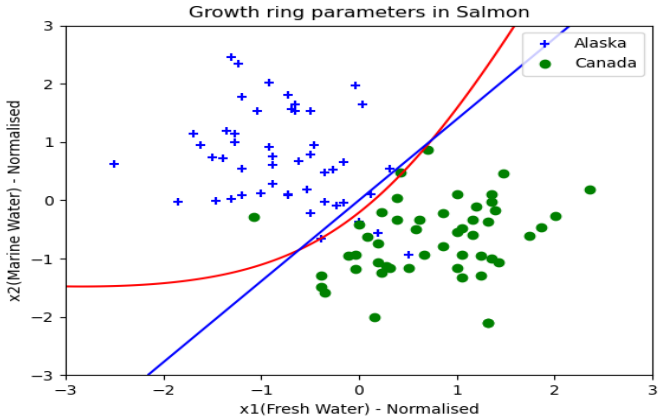
Assuming parameter tying, this equation is linear and so, we get a linear decision boundary. This is the plot of this decision boundary:



(d) If we don't assume parameter tying, we get separate Σ_0 and Σ_1. These are the parameters I obatined in the general case:

phi = 0.5
mu0 = [[-0.75529433 0.68509431]]
mu1 = [[0.75529433 -0.68509431]]
sigma0 = [[0.38158978 -0.15486516] [-0.15486516 0.64773717]]
sigma1 = [[0.47747117 0.1099206] [0.1099206 0.41355441]]

(e) Without the parameter tying assumption, the decision boundary is quadratic as described in the equation in part c. This is the plot of this boundary:



The quadratic decision boundary is in red while the linear is in blue.

(f) As far as dividing the points is concerned, both boundaries are not that much different though this may be only in this example.

We can deduce that the linear boundary was clearly made making more assumptions, namely parameter tying. Thus, the linear decision boundary may have the tendency to underfit in some scenarios.

The quadratic boundary is in that sense more general.

So, if our model may have Gaussian distribution, then GDA is a good way of classification.