COL774 Assignment 1

Navneel Singhal

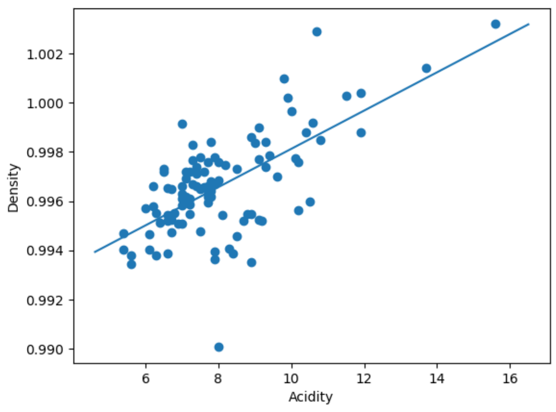
November 6, 2020

Contents

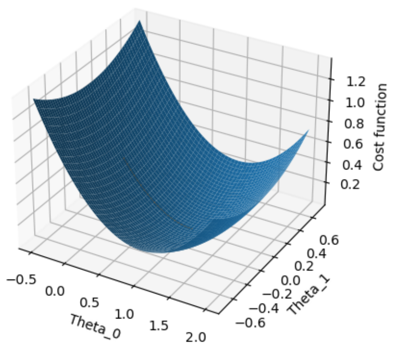
1 Linear Regression

1. In this problem, I implemented linear regression as discussed in class, and vectorized it for better performance. This was done by noting that using matrix multiplication, we can find the vector of all predictions by the current θ. Then subtracting the original values, we can get a vector of errors, which can be then used to find the gradient as well as the value of the cost function (this is faster than the Python for-loop since `numpy` is implemented internally in a faster language). The learning rate chosen is $\eta = 10^{-2}$ and the stopping criteria is that the change in cost function becomes less than $\varepsilon = 10^{-15}$. The parameters learnt by the algorithm are given in the output directory, and are roughly equal to $\begin{bmatrix} 0.996619 \\ 0.001340 \end{bmatrix}$.

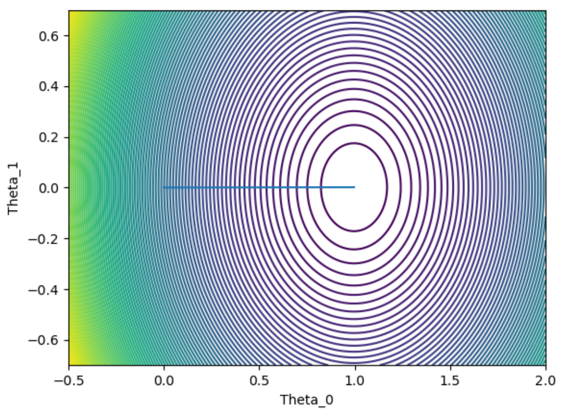
2. The plotting was done via `matplotlib`. This plot is as follows:



3. The plotting was done via `matplotlib`'s animation feature, and the final state of the loss function plot is as follows:

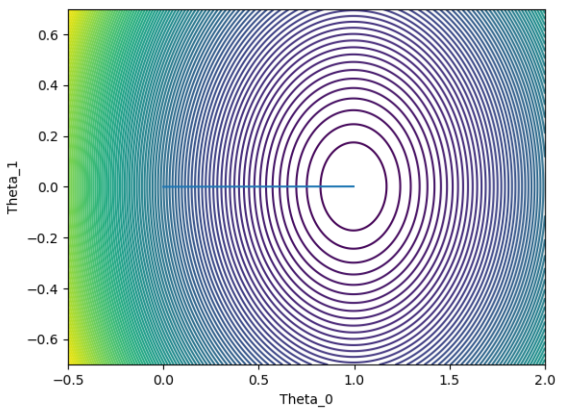


4. The plotting was done by the contour function in `matplotlib`. The final state of the loss function contour plot is as follows:

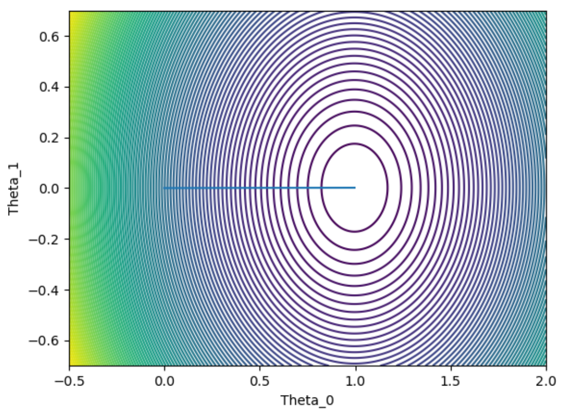


5. Note that here the value of ε is still the same as in the previous parts. The final state of the loss function contour plot for

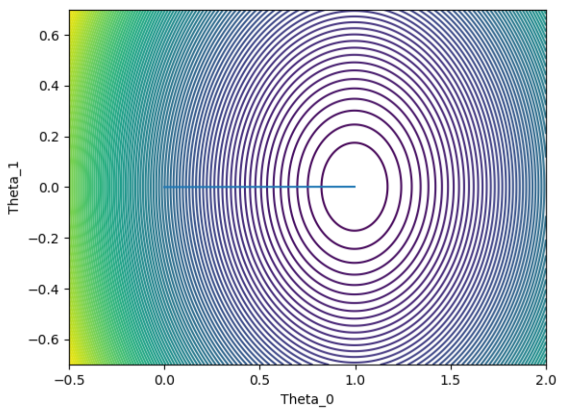
   (a) $\eta = 0.001$



   (b) $\eta = 0.025$



(c) $\eta = 0.1$



2 Stochastic Gradient Descent

1. This part was done using sampling from `numpy`'s normal distribution generation function. I assumed that in $\mathcal{N}(a, b)$, a is the mean and b is the variance. Note that the noise must have mean 0, which was implicitly assumed.

2. The convergence criteria for each case was, for the sake of comparison later on, taken to be the same for each of the parts. Convergence is decided if and only if for any $k = 3$ consecutive updates, the difference between consecutive cost functions becomes at most $\varepsilon = 2 \times 10^{-3}$. If this ever happens at any point in an epoch, we have converged and break the loop after or during that epoch. Note that such a large value of ε and such a small value for k was taken because otherwise the last two batch sizes take an inordinate amount of time to run (even the current parameters take a lot of time, of the order of a few minutes), thereby showing that vanilla gradient descent with small-ish learning rates is infeasible on large datasets, and even large datasets with large batch sizes.

   Note that there was another run with the value of ε being 1×10^{-4}, and a timeout was done to see how accurate the results are in the case for equal time.

   For the case of $\varepsilon = 2 \times 10^{-3}$, the following are the values:

   (a) Batch size 1: $\theta = \begin{bmatrix} 2.93941713 \\ 0.97888709 \\ 2.06029244 \end{bmatrix}$, 196289 iterations.

   (b) Batch size 100: $\theta = \begin{bmatrix} 3.00111533 \\ 1.0048262 \\ 1.99858323 \end{bmatrix}$, 792468 iterations.

   (c) Batch size 10000: $\theta = \begin{bmatrix} 2.66568549 \\ 1.07266765 \\ 1.9748256 \end{bmatrix}$, 7781 iterations.

(d) Batch size 1000000: $\theta = \begin{bmatrix} 0.56880709 \\ 1.43268035 \\ 1.52964014 \end{bmatrix}$, 493 iterations.

Using another convergence method, by averaging out the error over the last $\min(1000, 5m/\text{batch size})$ iterations, I was able to use an epsilon value of 5×10^{-5} and it gave me better convergence for the last and the second last, but it took more time for the last batch size. For this case, the following are the theta values:

(a) Batch size 1: $\theta = \begin{bmatrix} 3.07334241 \\ 0.99156185 \\ 2.02015695 \end{bmatrix}$, 433000 iterations.

(b) Batch size 100: $\theta = \begin{bmatrix} 2.9972336 \\ 0.99823272 \\ 2.00892425 \end{bmatrix}$, 556000 iterations.

(c) Batch size 10000: $\theta = \begin{bmatrix} 2.97298606 \\ 1.00629993 \\ 1.99880584 \end{bmatrix}$, 17000 iterations.

(d) Batch size 1000000: $\theta = \begin{bmatrix} 2.10363695 \\ 1.19644536 \\ 1.93579957 \end{bmatrix}$, 4161 iterations.

3. For the second convergence method, they converge to almost the same parameter values. For the first convergence method, no, they do not converge to the same parameter values. In fact, as will be seen in the next part, the graphs for the last two batch sizes for the first method and only the last batch size for the second method indicates that the gradient descent has not completed properly for these cases, and will take a lot more time to complete and converge to the actual value. The magnitude of the difference between these learned values and the original hypothesis seems to be first decreasing and then increasing with increasing batch size after a certain batch size (this is due to the fluctuation mentioned below and early stopping of gradient descent for the latter cases). The results indicated that a batch size of 1 tends to hover around the minimum, and it is limited due to the high variance of the individual samples (the standard deviation of the average of multiple samples is inversely proportional to the inverse-square-root of the number of samples, and hence the standard deviation of a batch of 100 samples is about 10 times lesser than that of a single sample, and thus the fluctuations smoothen out and lead to a steady convergence). As already mentioned, this experiment could not have been done completely with the first method with a significantly small value of ε for the larger batch sizes because of the inhibitively long training time arising from the fluctuating issue mentioned above. It was noticed that a larger batch size corresponds to a larger time to convergence, and this was also checked by breaking early, when the condition of k consecutive successes is satisfied already.

   For the first convergence method, I also tried timing out to prevent infinite loop due to oscillations of a magnitude larger than ε, and in those results, it seems to be the case that a larger batch size is related to smoother and more definite, but slower convergence. When all of them are run for the same time, the larger batches tend to get better accuracy due to lower fluctuation.

   As far as the number of iterations is concerned, if we assume the number of iterations to be the number of updates, then they seem to go down as the batch size increases after an initial increase, and this seeming contradiction is resolved if we note that the last two examples stop much before convergence for the first method and the last example stops only slightly before convergence, and that the batch sizes grow faster than the time taken for convergence (which corresponds to the total number of iterations of the inner vectorized loop over the whole execution). The time taken till convergence is a bit random across runs, but it seems to generally increase with batch size (it increases because of the gradient descent taking time on larger batch size). The number of epochs taken for convergence increases as the batch size increases, and this can be explained because of the idea that for large amount of data, we can work with a small sample to achieve convergence quicker. Note that the training time fluctuations can be resolved by attributing it to implementation details; if we noticed that the outer loop is not vectorized, while the inner loop is, the overhead for the outer loop (which runs for $10^6/b$ times for batch size b) is the dominant bottleneck in certain cases, and we can also attribute the training time fluctuations to the randomness of the data (in particular for small batch sizes).

   As far as testing the learnt and original hypotheses is concerned, the provided data has the least error on the original hypothesis (since it is generated using the sampling procedure using the ideal parameters), and then, as the batch size increases, the accuracy first increases then decreases; it was noticed that the variance effect mentioned above takes over and the batch size of 100 gives a better result that the single-sample batch. For the latter batches, the convergence has not been completed anyway. However, using the timeout values, it was noted that the error function is monotonically decreasing for error values. So the final accuracy order for early ending is original hypothesis > batch size 100 > batch size 1 > batch size 10000 > batch size 1000000.

4. We note the following three important points about the plots:

   (a) The movement is much smoother for larger batch sizes (this happens because a large number of samples smoothens out the step randomness as mentioned above)

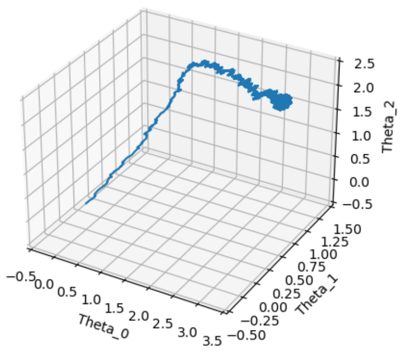
   (b) For the case where we do not have a timeout, the last two don't sufficiently converge to the correct θ and just end up stopping midway because of the learning rate and less fluctuations in the cost function values. Again, this is because of the prohibitive time.

   (c) Towards the end, the fluctuation becomes more random, and this is because in the initial parts, there is more directed motion towards the optimal parameters, while when it is close, the randomness in smaller batch sizes as discussed above crops up.

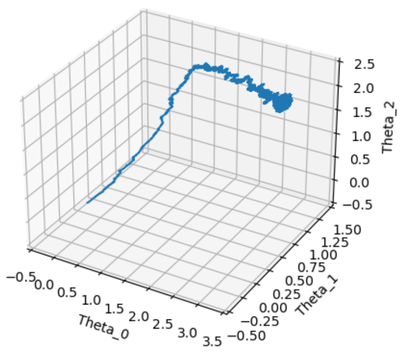
   The output parameters, errors, and iterations are stored in the output directory.

   The plots are as follows:

   (a) Batch size = 1

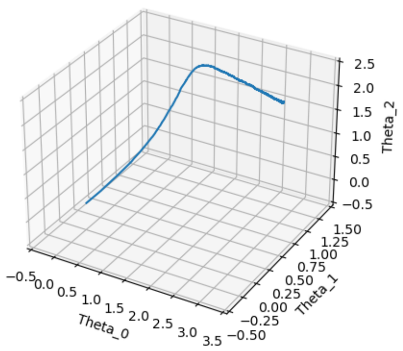


First method of convergence

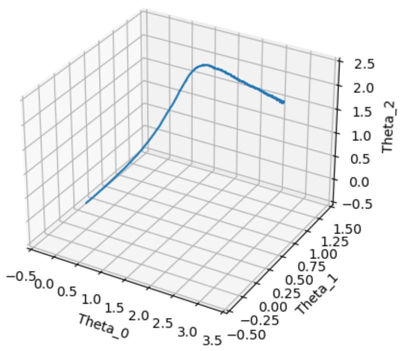


Second method of convergence

   (b) Batch size = 100

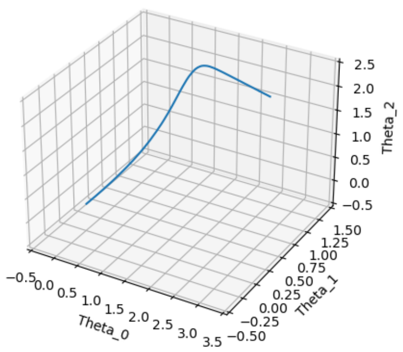


First method of convergence

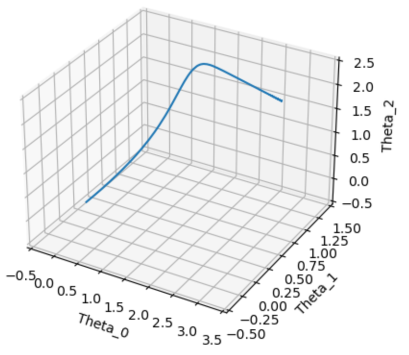


Second method of convergence

(c) Batch size = 10000

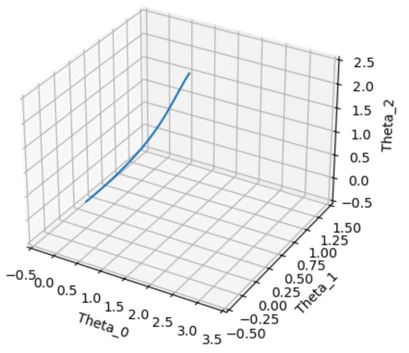


First method of convergence

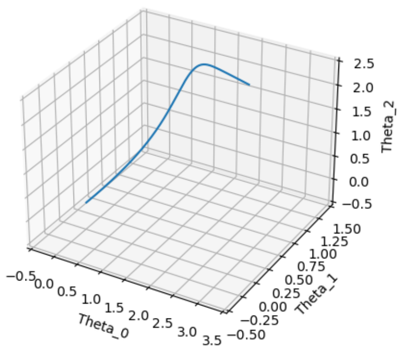


Second method of convergence

(d) Batch size = 1000000



First method of convergence



Second method of convergence

3 Logistic Regression

1. For Newton's method, we need to compute the Hessian. To do this, we need to find the double partial derivative for each pair of coefficients in θ. Doing this, we get that the Hessian is:

$$\mathcal{H} = \sum_{i=1}^{n} \frac{\exp\left(-\theta^T x^{(i)}\right)}{\left(1+\exp\left(-\theta^T x^{(i)}\right)\right)^2} \left(x^{(i)} x^{(i)T}\right)$$

We also need the gradient of the log likelihood, and it turns out to be:

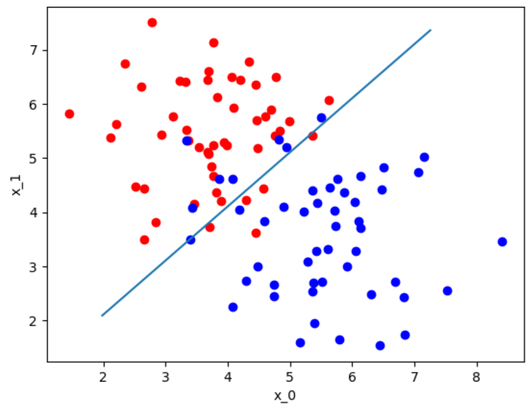
$$\nabla_\theta LL(\theta) = \sum_{i=1}^{n} \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x^{(i)}$$

The parameter update equation becomes

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \mathcal{H}^{-1}\nabla_\theta LL(\theta)$$

The value of θ learnt by the model is $\begin{bmatrix} 0.40125316 \\ 2.5885477 \\ -2.72558849 \end{bmatrix}$.

2. The required plot is as follows:



4 Gaussian Discriminant Analysis

1. The values of the means μ_0, μ_1 and the co-variance matrix Σ are computed as mentioned in the class. The values computed by the algorithm are

$$\phi = 0.5$$

$$\mu_0 = \begin{bmatrix} -0.75529433 \\ 0.68509431 \end{bmatrix}$$

$$\mu_1 = \begin{bmatrix} 0.75529433 \\ -0.68509431 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 0.42953048 & -0.02247228 \\ -0.02247228 & 0.53064579 \end{bmatrix}$$

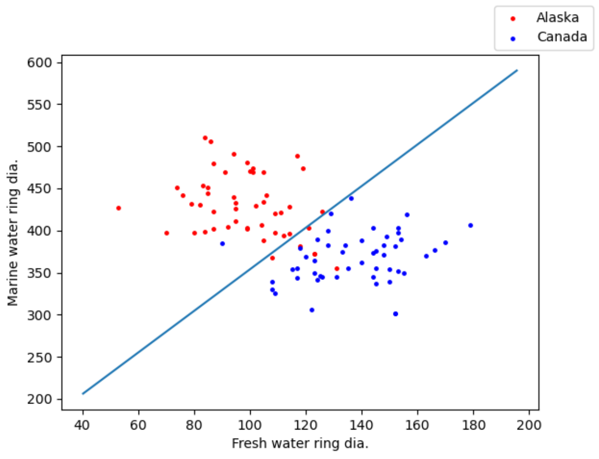
.

2. The plot is combined in the next part.

3. The equation is as follows:

$$\left(\mu_1^T - \mu_0^T\right)\Sigma^{-1}x + \log\left(\frac{\phi}{1-\phi}\right) + \frac{1}{2}\left(\mu_0^T\Sigma^{-1}\mu_0 - \mu_1^T\Sigma^{-1}\mu_1\right) = 0$$

The plot is as follows:



4. Using this formula, the values reported by the algorithm are:

$$\phi = 0.5$$

$$\mu_0 = \begin{bmatrix} -0.75529433 \\ 0.68509431 \end{bmatrix}$$

$$\mu_1 = \begin{bmatrix} 0.75529433 \\ -0.68509431 \end{bmatrix}$$

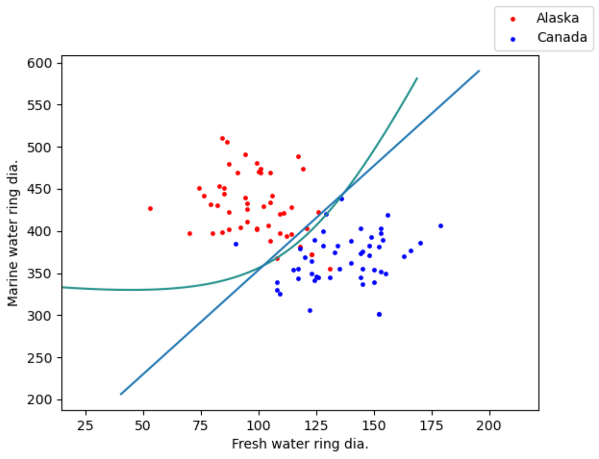
$$\Sigma_0 = \begin{bmatrix} 0.38158978 & -0.15486516 \\ -0.15486516 & 0.64773717 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 0.47747117 & 0.1099206 \\ 0.1099206 & 0.41355441 \end{bmatrix}$$

5. The equation for the quadratic boundary separating the two regions comes out as follows:

$$\frac{1}{2}x^T\left(\Sigma_0^{-1} - \Sigma_1^{-1}\right)x + \left(\mu_1^T\Sigma_1^{-1} - \mu_0^T\Sigma_0^{-1}\right)x + \log\left(\frac{\phi}{1-\phi}\right) + \frac{1}{2}\log\left(\frac{|\Sigma_0|}{|\Sigma_1|}\right) + \frac{1}{2}\left(\mu_0^T\Sigma_0^{-1}\mu_0 - \mu_1^T\Sigma_1^{-1}\mu_1\right) = 0$$

Plotting this has been done using a single contour of the function on the left in a 2D plot which corresponds to those values of x which satisfy the above equation.



6. From the plots, it can be seen that both seem to be good separators, but due to the quadratic (hyperbolic) nature of the separator, the GDA model with different covariance matrices assumes that the Alaska salmon features have a 'round' boundary and the fit to shape doesn't look like it will generalise well to more data, since even though we effectively added one more parameter to the model, the number of misclassified points hasn't been reduced as much as we could have expected.