Assignment 1
COL 774: Machine Learning

Gonçalo Alves$^{[2022VST9500]}$

Indian Institute of Technology Delhi
info@iitd.ac.in
https://home.iitd.ac.in

1 Introduction

This report is meant to inform about the implementation of four algorithms: Linear Regression, SGD, Logistic Regression and GDA. Each algorithm has its own dedicated section where the implementation and its results will be discussed in detail.

2 Linear Regression

In this assignment, we were tasked with developing a variation of Linear Regression, known as Least Squares Linear Regression, to predict the density of a wine ($y \epsilon R$) based on it's acidity ($x \epsilon R$). Data was first normalized according to zero mean and unit variation.

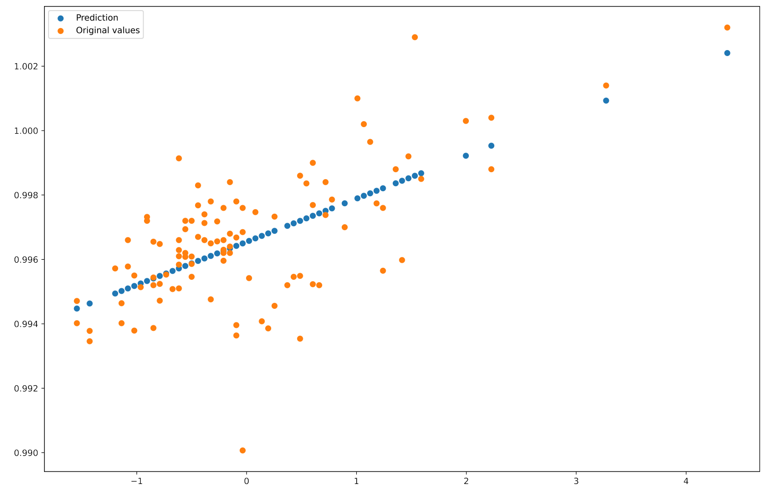
We used batch gradient descent method to optimize the cost function, J(θ):

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (y^i - h_\theta(x^i))^2 \tag{1}$$

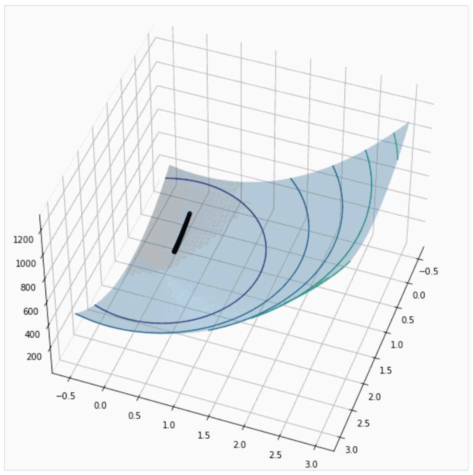
The initial parameter values were the following: $\theta = 0$ and $\alpha = 0.01$. The decision of the learning rate is crucial to obtain a good balance between convergence and speed: a lower learning rate implies that the steps taken towards optimization are smaller and as such, the algorithm takes a longer time to converge; a higher learning rate causes the step to be to big which induces divergence of the function. This way, testing was done with learning rates of [1e-5, 1e-2, 1e-1, 1e1] and after a careful study, 1e-2 was chosen as the best suited learning rate for our problem (1-e1 had similar results) and as such, all results presented will be considering this learning rate. There must also be a stopping criteria for the algorithm. A good criteria is to stop the optimization process when the gradient of the cost function is close to 0 (1e-8). Additionally, a large number of iterations was also used as an alternative stopping criteria, to prevent long executions. These criteria are good enough, even if they can only guarantee local minima, because the cost function is convex.

The results were the following: $J(\theta_{initial}) = 0.4966279047145$ and $J(\theta_{final}) = 1.197530238073229e-06$, 947 iterations

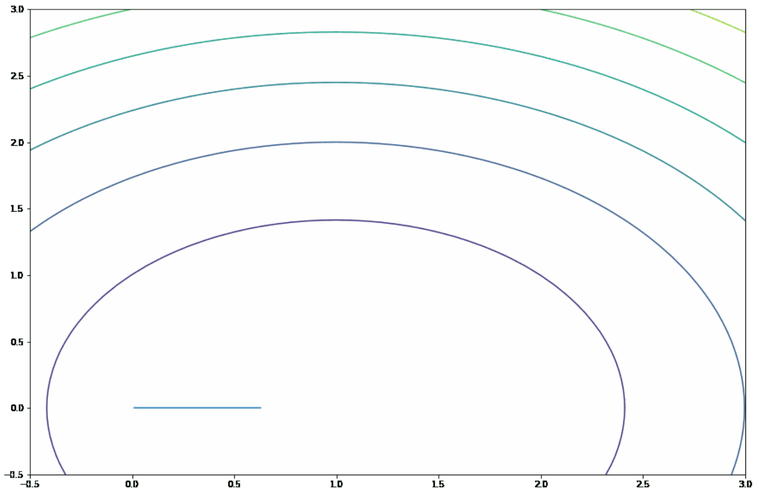
In the following figure, it is possible to see how well our model fits, in comparison to the actual values.



The following 3D plot shows the evolution of the cost function, in black, at each step.



The following contour plot is similar to the one above but in 2D fashion.


3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

3 SGD

In this assignment, we had to generate data, using a probabilistic interpretation of Linear Regression, and then apply SGD to optimize $J(\theta)$. Our data follows the following sampling:

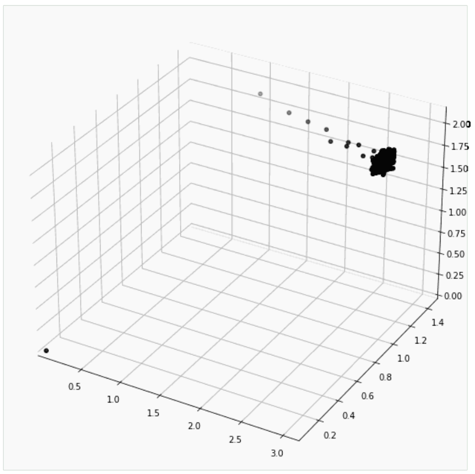
$$y^i = \theta_0 + \theta 1(x_1)^i + \theta_2(x_2)^i + \epsilon^i \tag{2}$$

where $\epsilon \sim N(0, \delta^2)$

We then use SGD to learn the original prediction from the data that we generated. Although very similar to the cost function above, this time we will only go through r, the batch size, samples instead of m, which leads to faster convergence.

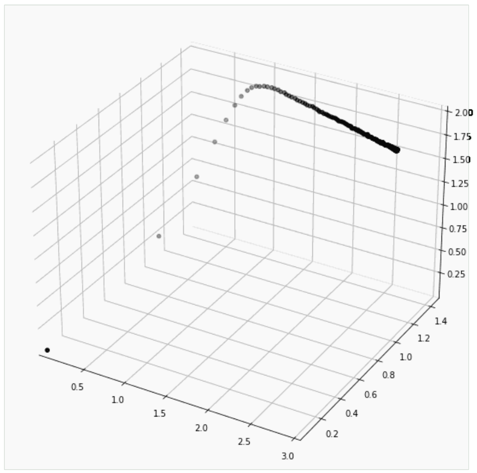
We were tasked with experimenting with different batch sizes, $r \epsilon 1, 100, 10000, 1000000$, and fixed learning rate, 0.001.

For batch size **1**, the algorithm takes 1 million steps per epoch, therefore the parameters are updated for each data point. The figure below shows how θ converges:



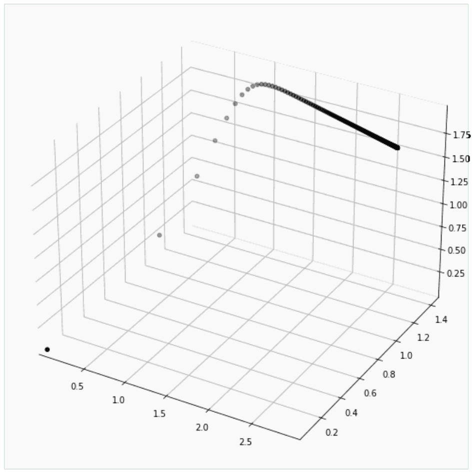
As we can see, the convergence is very fast. This is because we don't go through the entire data every time we do an update. But, because we calculating the cost with only one data point, instead of the calculating the total cost, we can see that there the path is random. To see the performance of different batch sizes, the test error on the original prediction, with $\theta = 0$, was calculated: 0.9829469215. The lower the difference between the original test error and the one from our model, the better the model. The final results were: $\theta_f = [2.98253243, 1.03542041, 1.9526828]$,*test error = 1.1665294847119572* and the difference between errors was: 0.1835825632119572.

For batch size **100**, the parameters are updated for each 100 data point. The figure below shows how θ converges:



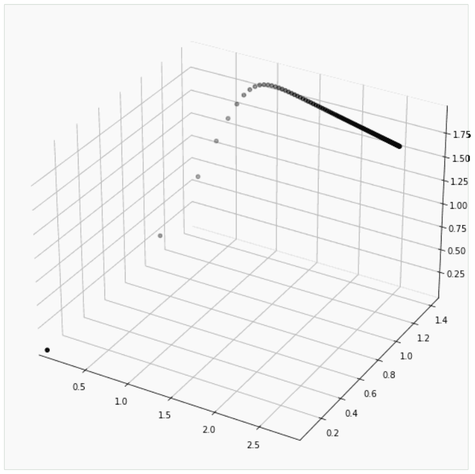
As we can see, the convergence is very fast as well, but there is less randomness in the path, comparing to batch size = 1. The final results were: $\theta_f = [3.00189627, 0.99801062, 1.99770664]$,*test error = 0.983837158548581* and the difference between errors was: 0.000890237048581044.

For batch size **10000**, the parameters are updated for each 10000 data point. The figure below shows how θ converges:



As we can see, the convergence starts being slower, but there is almost no randomness in the path. The final results were: $\theta_f = [2.99121302, 1.00110481, 1.99882841]$,*test error = 0.9832436233590488* and the difference between errors was: 0.0002967018590488424.

For batch size **1000000**, the parameters are updated for each 1000000 data point. The figure below shows how θ converges:



As we can see, the convergence is the slowest, but there seems to be no randomness in the path. The final results were: $\theta_f = [2.88902918, 1.02353158, 1.99151893]$,*test error = 1.0174952791590102* and the difference between errors was: 0.03454835765901021.

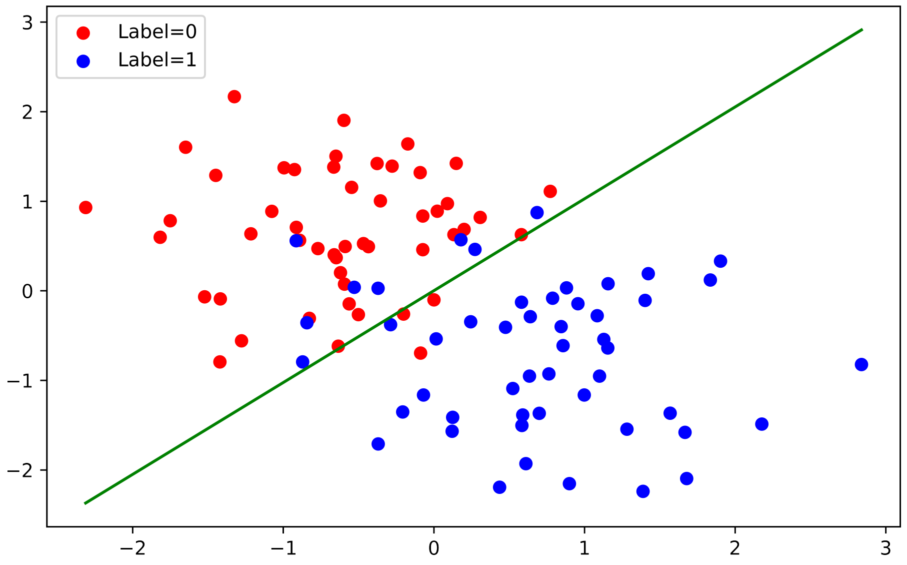
4 Logistic Regression

In this assignment, we were tasked with developing an implementation of Logistic Regression, a model used for classification of classes of data. We used the Newton's Method for optimizing the negative LL (log-likelihood) of the cost function. We normalized the dataset given to zero mean and unit variance, which only had two class labels and two features. LL of Logistic Regression is: $LL(\theta) = \sum_{i=1}^{m} y^i log(h_\theta(x^i)) + (1-y^i)log(1-h_\theta(x^i)))$

Newton's Method uses second order derivation for optimization of LL. This way, we calculate the Hessian of the gradient of LL: $H_\theta(LL(\theta)) = x^T.diag(\delta(x.\theta)(1-\delta(x.\theta))).x$ and we update with step: $\theta^{t+1} = \theta^t - H^{-1}\nabla_\theta(LL(\theta))$.

Initial parameters used were $\theta = [0, 0, 0]$.

Below, we can see the decision boundary learnt by the model, with $\theta_f = [-0.00064394, 0.00921424, -0.00898329]$.

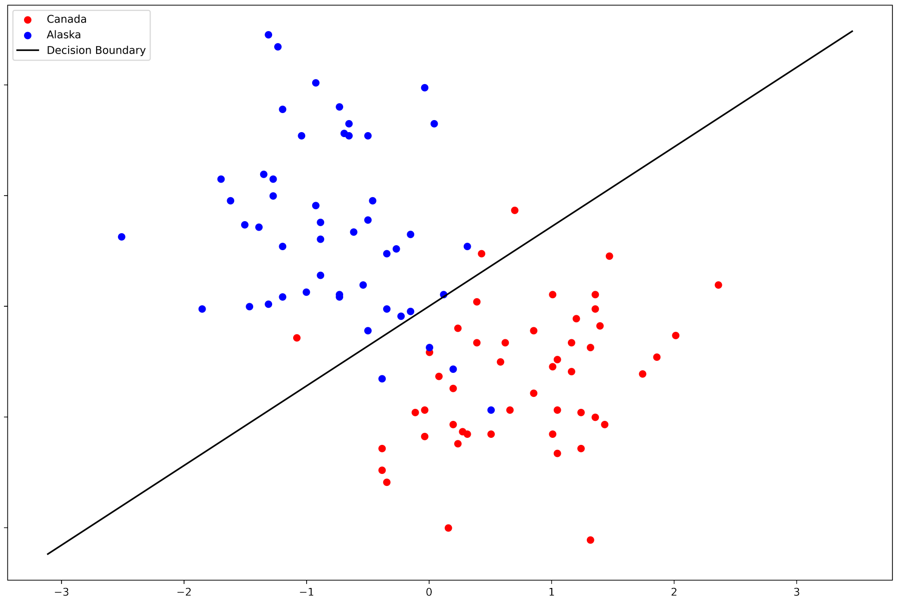


5 GDA

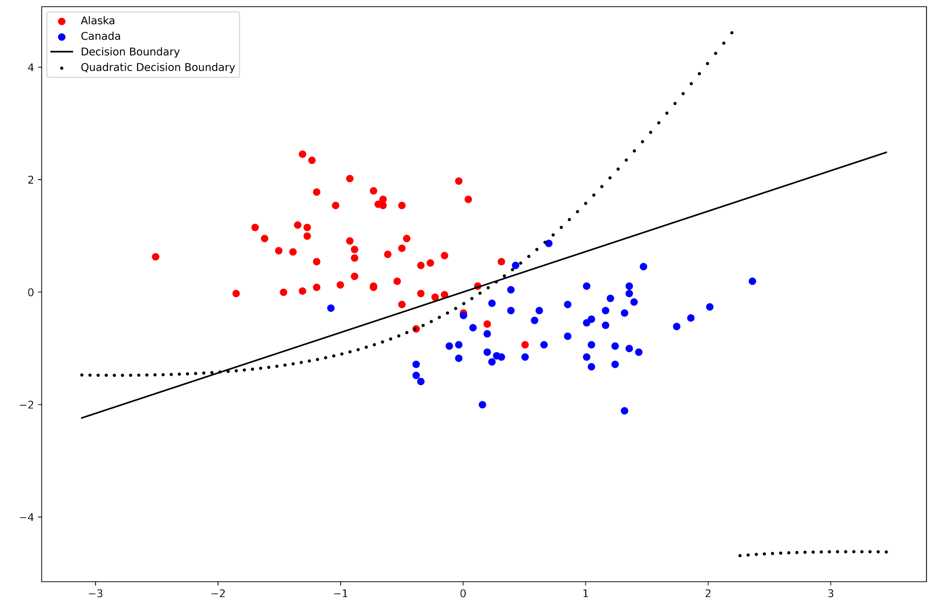
In this assignment, we were tasked with developing an implementation of GDA, to classify classes of salmon from Alaska and Canada.We normalized the dataset given to zero mean and unit variance, which only had two class labels, Alaska (class 0) and Canada (class 1) and two features. The equations used to compute the means of distribution and covariance are:

$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^i = 0\}x^i}{\sum_{i=1}^{m} 1\{y^i = 0\}}, \mu_1 = \frac{\sum_{i=1}^{m} 1\{y^i = 1\}x^i}{\sum_{i=1}^{m} 1\{y^i = 1\}}, \sum = \frac{\sum_{i=1}^{m}(x^i - \mu_{y^i})(x^i - \mu_{y^i})^T}{m} \tag{3}$$

First, we assume that both classes have the same covariance, so we will get a linear decision boundary. We got the following results: $\mu_0 = [-0.75529433, 0.68509431]$, $\mu_1 = [0.75529433, -0.68509431]$ and $\sum = [[0.42953048 -0.02247228][-0.02247228 0.53064579]]$. Below we can see the training data with the respective linear boundary obtained.



Without the covariance, assumption, the decision boundary turns into a quadratic one. We got the following results: $\sum_0 = [[0.38158978 -0.15486516][-0.15486516 0.64773717]]$ and $\sum_1 = [[0.47747117 0.1099206][0.1099206 0.41355441]]$, with the means remaining the same. Below we can see the training data with the respective quadratic boundary obtained and the previous linear boundary.



As we can see in the figure, the quadratic boundary does a better job in separating the two classes. This is because the quadratic boundary, unlike the linear boundary, has information about the covariance of each class.