# COL774 Assignment 1

Aniruddha Deb
2020CS10869

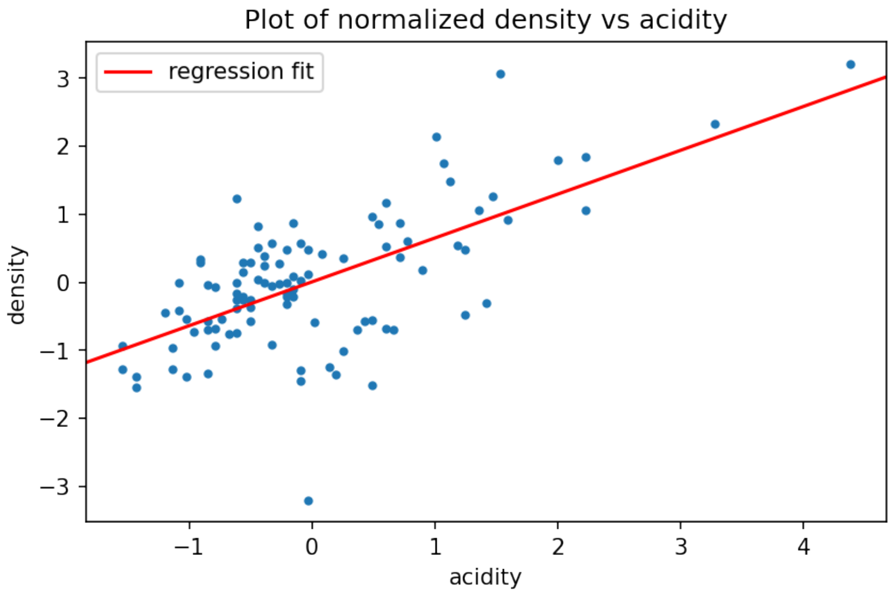
September 2022

## Linear Regression

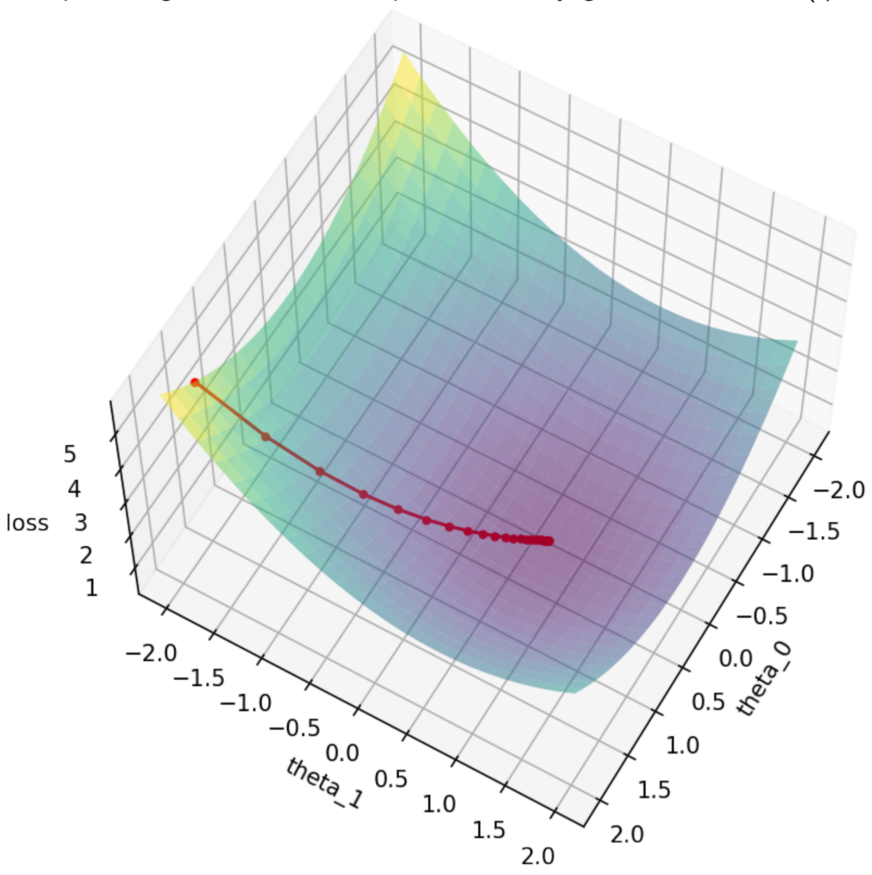
(a) The parameters we used were:

- Learning rate $\eta = 0.01$
- Convergence limit $\epsilon = 0.0001$
- Final set of parameters: $(\theta_0, \theta_1) = (0.00561383, 0.6451277)$

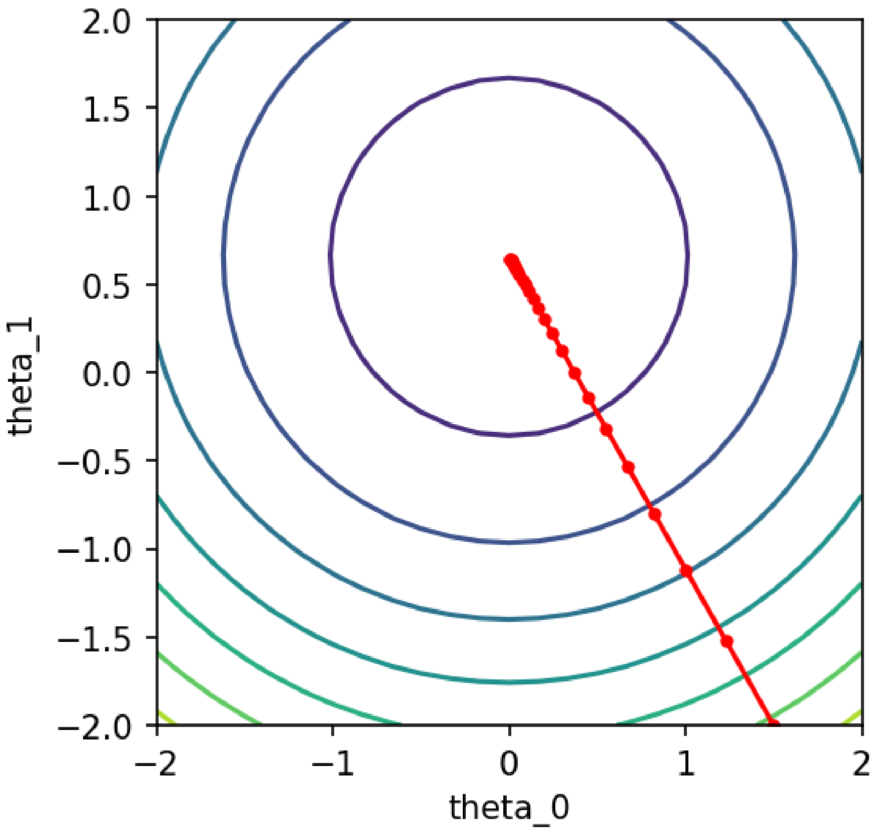
(b)



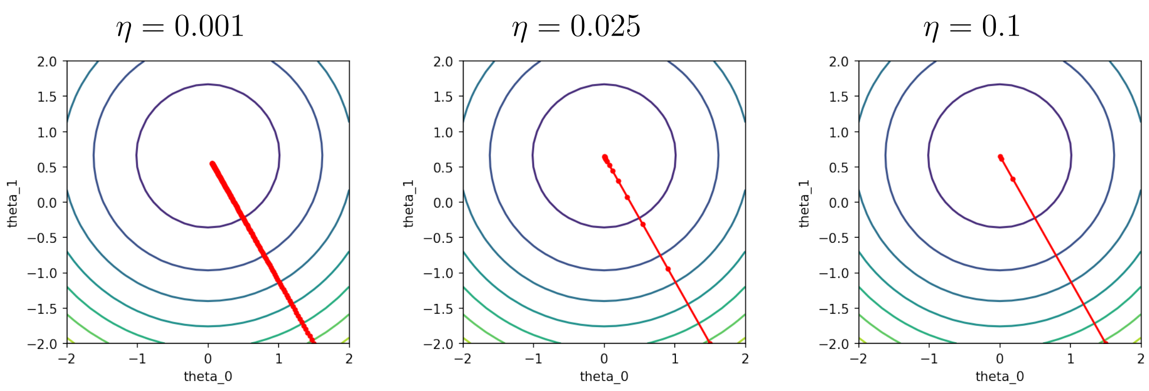
(c) 3D mesh plot of gradient, and the path taken by gradient descent ($\eta = 0.01$):



(d) 2D contour plot of gradient, and the path taken by gradient descent ($\eta = 0.01$):



(e) Paths taken by Gradient Descent for varying values of $\eta$:

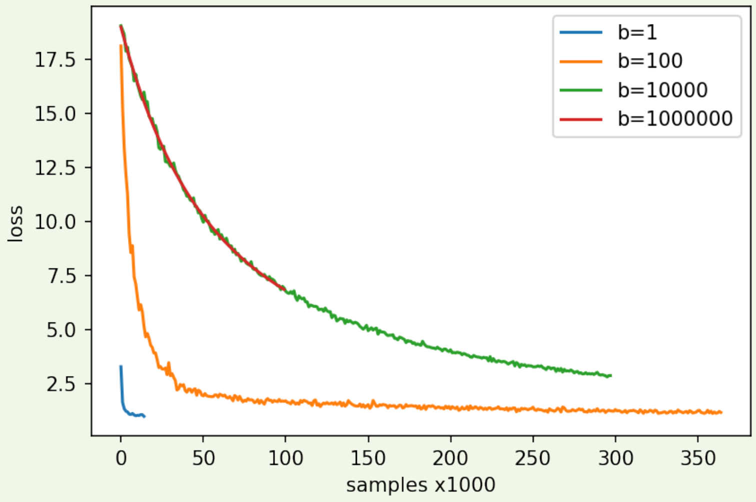


We notice that with a smaller step size, fewer steps are required to reach convergence; this is beneficial in this case, but can also lead to gradient descent overshooting/not converging in some cases.

## Stochastic Gradient Descent

On Convergence of SGD

From the video shared, it was suggested that to examine the convergence, loss functions averaged over atleast a 1000 examples be plotted during Stochastic Gradient Descent. A plot over different batch sizes gives the following:



We see that for $b = 10^4$ and $b = 10^6$, the algorithm does not converge as the learning rate is too low, and the cutoff condition stops it early. As a result, we also include an absolute loss cutoff for learning: stop when the averaged loss over 1000 examples is less than 1.
This example shows how to find such a window, given that we don't know the parameters that SGD will converge to: we manually examine the graphs and see what values our converging loss functions are reaching. The graph for $b = 1$ and $b = 100$ reaches values marginally under 1 (0.97), hence we choose 1 as our cutoff.

Note

We choose to keep the maximum number of epochs at 100, as for $b = 10^6$, the slow algorithm was not converging on my machine even after running it for ≈5 minutes.

1.

(b) With the following convergence conditions:

- number of epochs $t \leq 100$
- relative difference averaged over (atleast) past 1000 examples $\mathcal{J}^{t+1}_{\geq 1000} - \mathcal{J}^{t}_{\geq 1000} < r_d$
- absolute loss function averaged over (atleast) past 1000 examples $\mathcal{J}_{\geq 1000} < 1$

We obtain the following convergence(s):

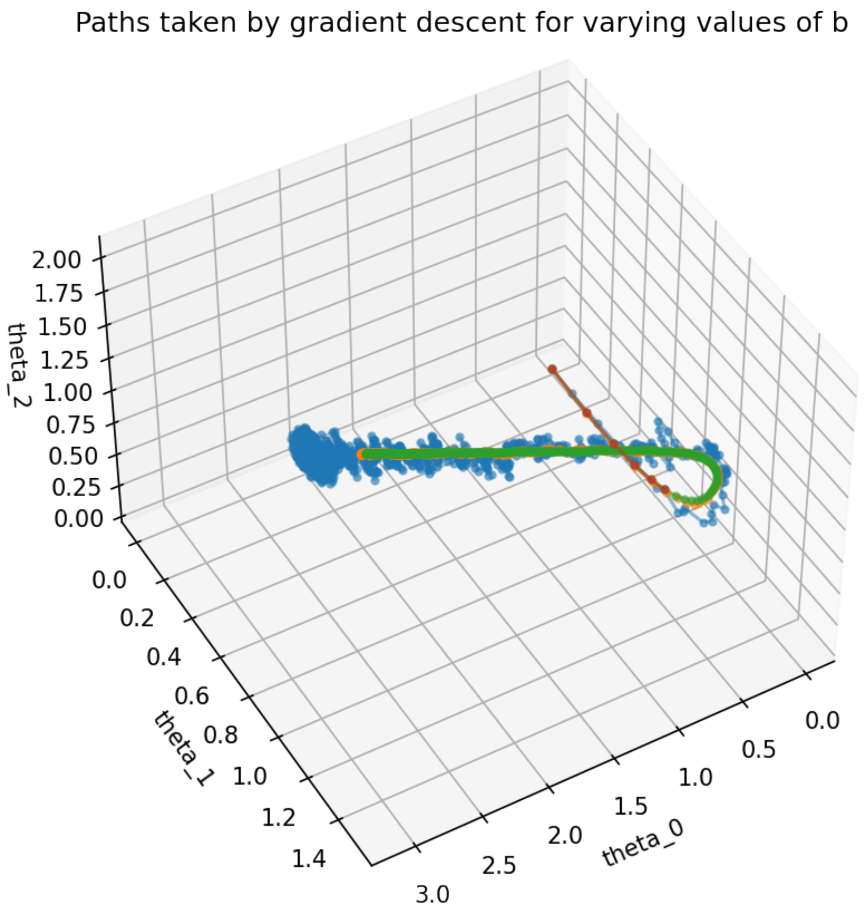
| $b$ | $r_d$ | $\theta$ | Time | Batches |
|---|---|---|---|---|
| 1 | 0.01 | $(3.03, 1.03, 2.03)$ | 2.46 s | 88000 |
| 100 | 0.001 | $(2.72, 1.06, 1.98)$ | 0.24 s | 8520 |
| 10000 | 0.001 | $(2.69, 1.07, 1.98)$ | 10.80 s | 8122 |
| 1000000 | 0.001 | $(0.25, 0.91, 0.46)$ | 14.33 s | 100 |

(c) All the training examples converge close to the actual parameters, except $b = 10^6$, as it hits the epoch limit. $b = 100$ converges the fastest. $r_d$ for 1 was taken to be $0.01$ as for $b = 1$, we get strong oscillations around the local minima (as will be visible in fig. (d)). Other parameters don't converge exactly to the local minima as their averaged loss functions reach below 1 sooner, and as they move slowly, the second condition of relative difference is also satisfied once they get very close to the minima.

For the test dataset with 10,000 examples, the loss values are as follows:

| $b$ | $\mathcal{J}(\theta)$ |
|---|---|
| 1 | 1.05 |
| 100 | 1.21 |
| 10000 | 1.25 |
| 1000000 | 121.0 |

(d)



Notice that for $b = 1$, the algorithm does not converge to the minima, but instead oscillates in a ball around the minima. This is fixed by taking a variable learning rate (as seen in the video). The other movements are slower, and the red line stops early because it's cut off by the epoch time.

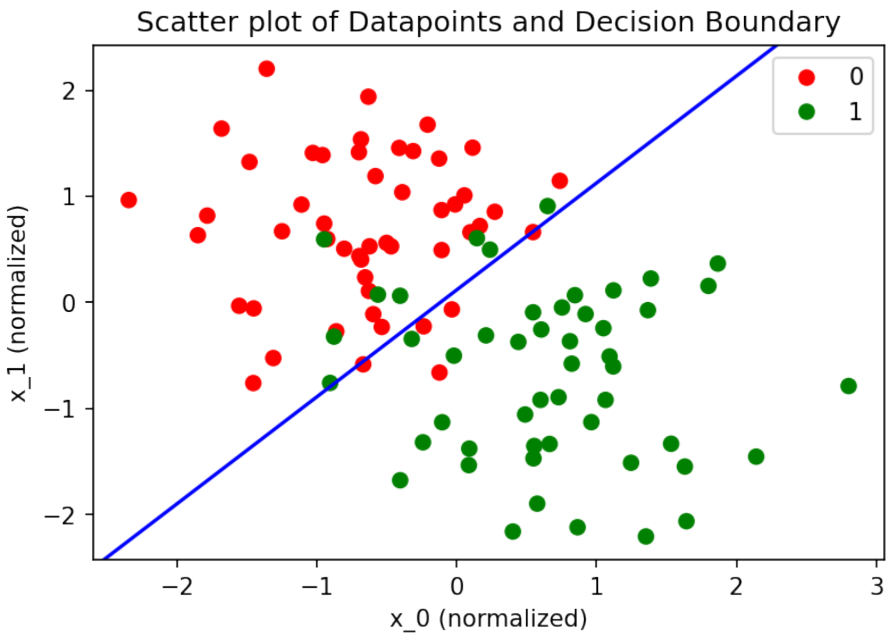
## Logistic Regression

(a) We obtained the hessian as

$$H(\theta) = \left[ \sum_{i=1}^{m} -\sigma(\theta^T x^{(i)})(1 - \sigma(\theta^T x^{(i)}))x_j^{(i)} x_k^{(i)} \right]_{j,k}^{n}$$

Optimizing this via newton method for 1000 iterations, we obtained $\hat{\theta} = (0.26, 2.25, -2.23)$

(b)

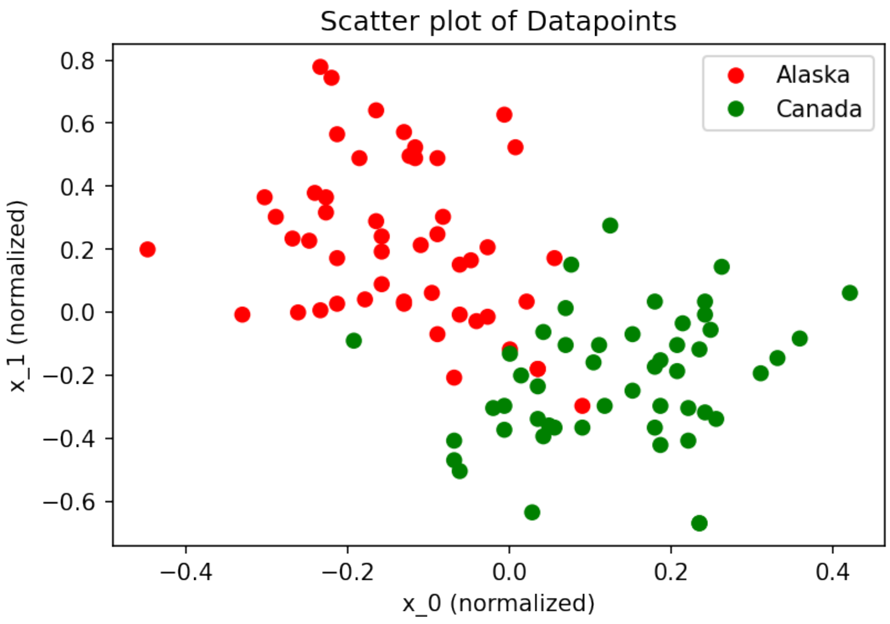


## Gaussian Discriminant Analysis

(a) Running GDA on the normalized data gives us

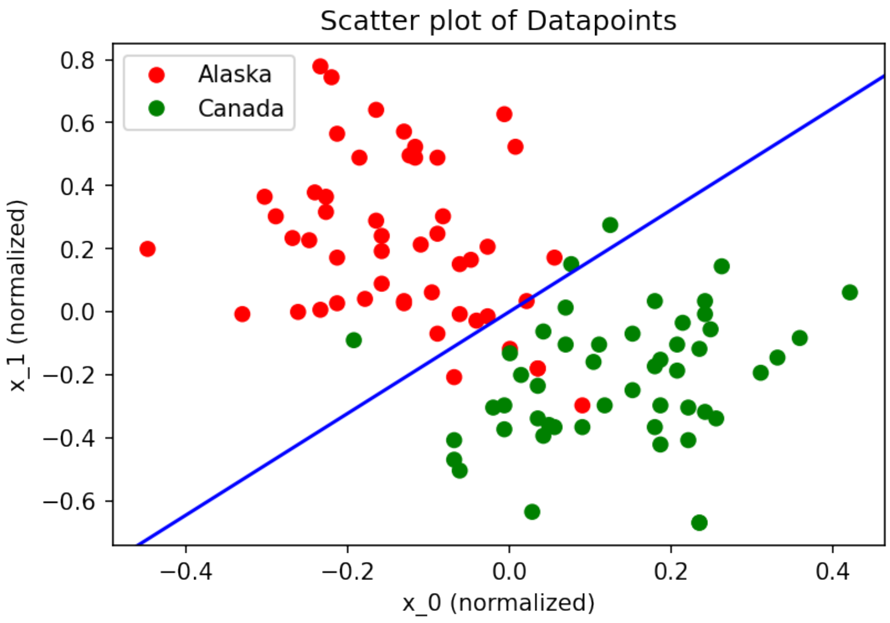
$$\begin{aligned}\phi &= 0.5\\ \mu_0 &= (-0.134, 0.217)\\ \mu_1 &= (0.134, -0.217)\\ \Sigma &= \begin{bmatrix} 0.014 & -0.001 \\ -0.001 & 0.053 \end{bmatrix}\end{aligned}$$

(b)



(c) For the same covariance matrix, the separator is a normal to the line joining the two means, passing through the point $\mu_0(1-\phi)+\mu_1\phi$

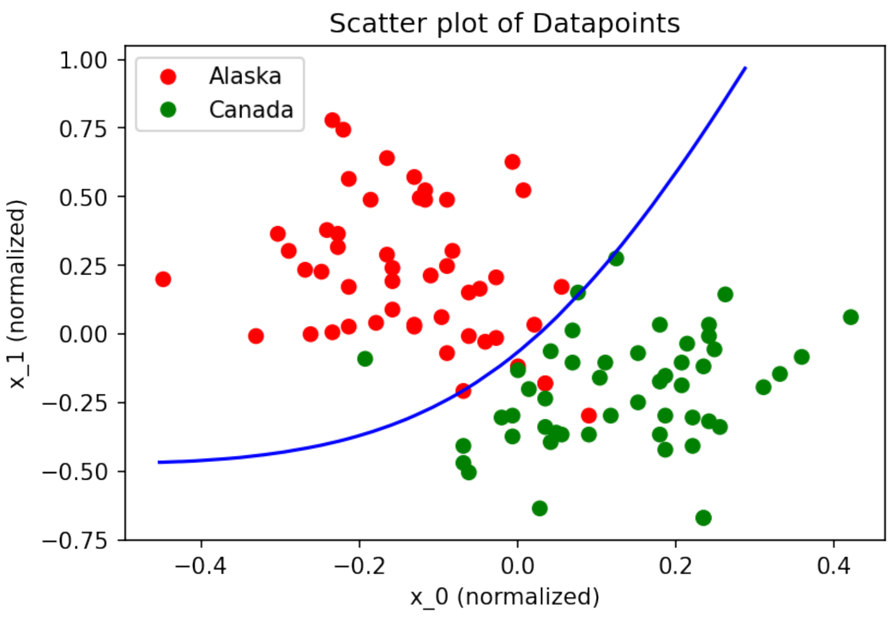
In this case, since the number of datapoints in both classes are equal ($\phi = 0.5$) the separator is simply the perpendicular bisector.



(d) Running GDA for two separate covariances on the normalized data gives us

$$\begin{aligned}
\phi &= 0.5 \\
\mu_0 &= (-0.134, 0.217) \\
\mu_1 &= (0.134, -0.217) \\
\Sigma_0 &= \begin{bmatrix} 0.012 & -0.009 \\ -0.009 & 0.065 \end{bmatrix} \\
\Sigma_1 &= \begin{bmatrix} 0.015 & 0.006 \\ 0.006 & 0.042 \end{bmatrix}
\end{aligned}$$

(e) The quadratic boundary obtained is as follows:



(f) The quadratic decision boundary is better at classification than the linear one: there are only 4 misclassifications in the quadratic classifier compared to 7 in the linear.

The quadratic decision boundary in this case is a **hyperbola**, as from Chan Pg. 24, we see that if $\Sigma_0^{-1} - \Sigma_1^{-1}$ has one negative eigenvalue and one positive eigenvalue, the separator will be hyperbolic. In this case, the eigenvalues come out to be $34.6$ and $-20.4$, implying that the decision boundary is a hyperbola.