COL774 Assignment 1

Mridul Gupta (AIZ218322)

Friday 10 September 2021

1 Q1

1.1 Q1(a)

Run as

```
$ python one_a.py --data-x path/to/linearX.csv
--data-y path/to/linearY.csv
```

1. The first thing I did was look at the data and note how many lines it was. The input data was 1D and had 100 samples.
2. I wrote a simple code to load the data and plot. And scaled the data to zero centered and unit variance.
3. I wrote the piece of code to calculate the cost function and plotted the cost function, keeping one of the θ_i constant at 0 while varying the other. In this way I was able to find that the optimum parameter was $\hat{\theta} = [0, 1]$.
4. Then, I wrote the gradient calculation step of gradient descent and combined it with the data loading step.
5. After all the debugging was done, I rewrote the code using `dataloader` style syntax to make working easier. The inspiration was pytorch code I worked on before.
6. I initialized the parameters to $[0, 0]$.
7. I also experimented with different number of epochs, learning rates and convergence criteria. Early on, I was using the convergence of θ^t to an epsilon-ball $\mathbb{B}_\epsilon$ to determine convergence. But later, I settled to using the convergence of the cost function values over time as convergence criterion.

Learning Rate: 0.001
Stopping criteria: $\|J^t - J^{t-1}\|_2 < 10^{-6}$
Final parameter values: $\theta_0 = 0.96503633$ and $\theta_1 = 0.00130276$

1.2 Q1(b)

Run as

```
$ python one_b.py --data-x path/to/linearX.csv
--data-y path/to/linearY.csv
```

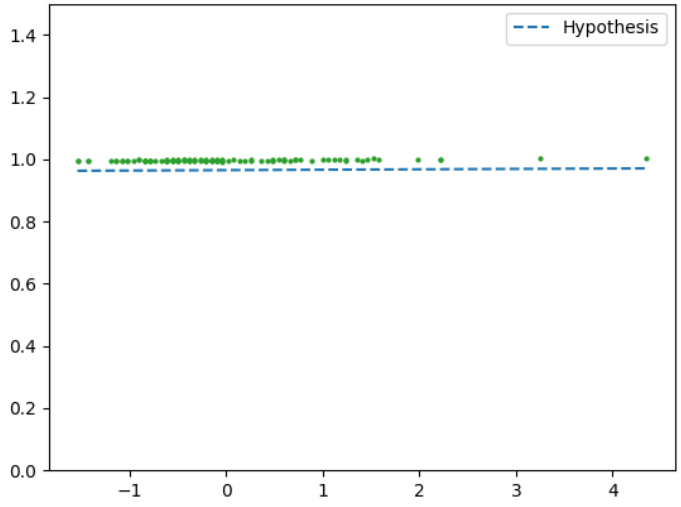


Figure 1: Data

1.3 Q1(c)

Run as

```
$ python one_c.py --data-x path/to/linearX.csv --data-y
path/to/linearY.csv [--no-rotate]
```

Note that the default animation rotates, this was done to save frames in a way that the three dimensionality is visible. This can be disabled using `--no-rotate` option.

The animation cannot be attached here, instead I'm attaching the first and the final epoch snapshots. See figure 2-3.

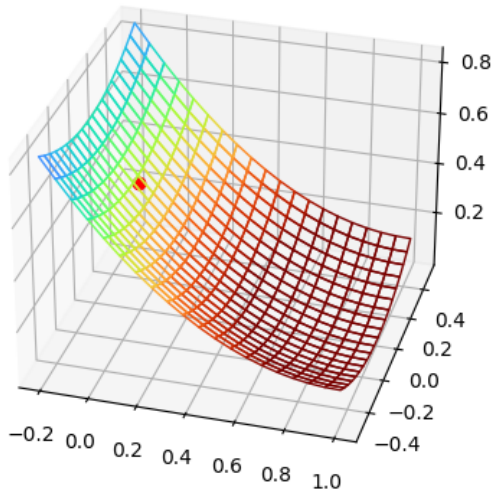


Figure 2: First epoch

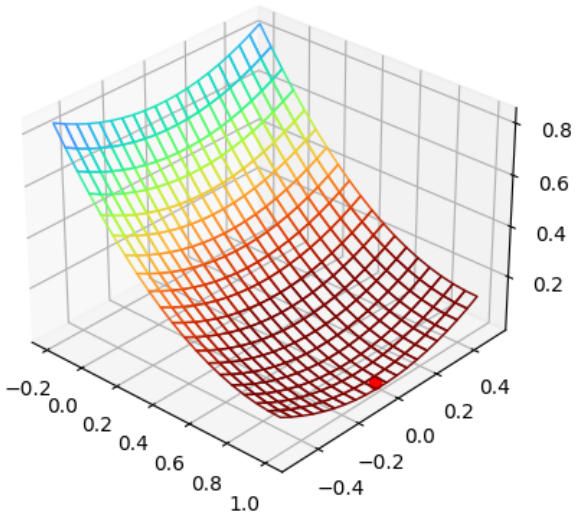


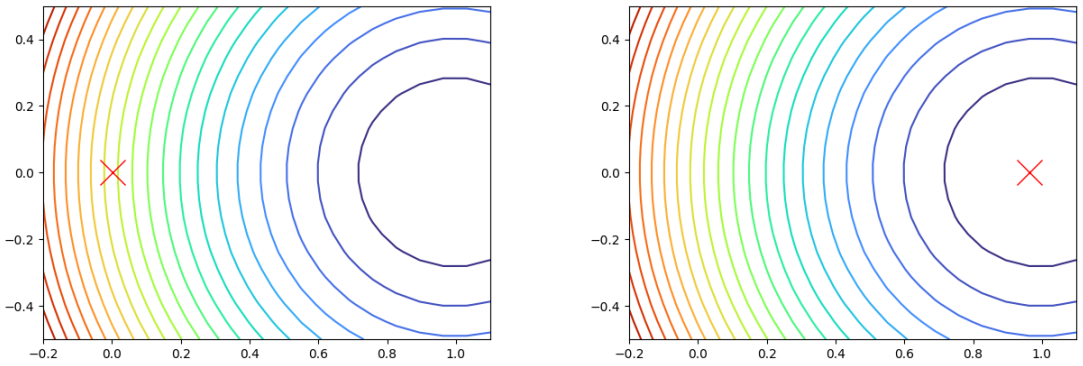
Figure 3: Last epoch

1.4 Q1(d)

Run as

```
$ python one_d.py --data-x path/to/linearX.csv
--data-y path/to/linearY.csv
```

Again, cannot attach animation here, but attaching animation in the zip file. See figure 4.



(a) First epoch

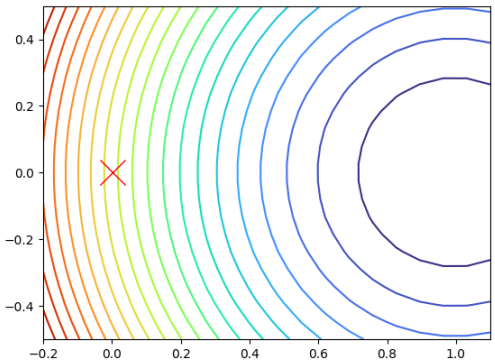
(b) Last epoch

Figure 4: Contour plot snapshots

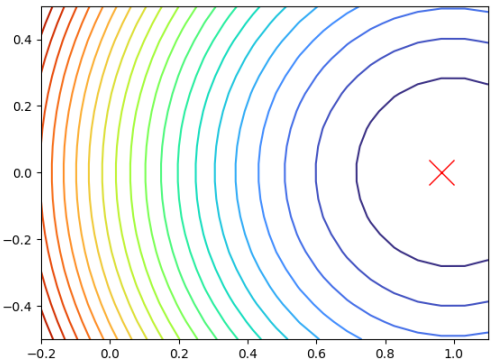
1.5 Q1(e)

I observe exactly what I'd hope to, the parameters move towards the optimum faster as the step size grows. See table. The direction of movement is almost the same, the changes in step size were small so that the algorithm never diverged.

step size	#epochs before convergence
0.001	3454
0.025	201
0.1	55

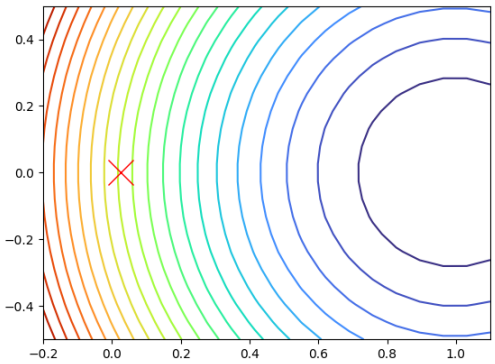


(a) First epoch 0.001

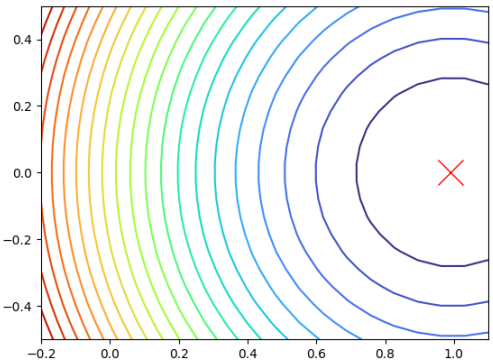


(b) Last epoch 0.001

Figure 5: Contour plot

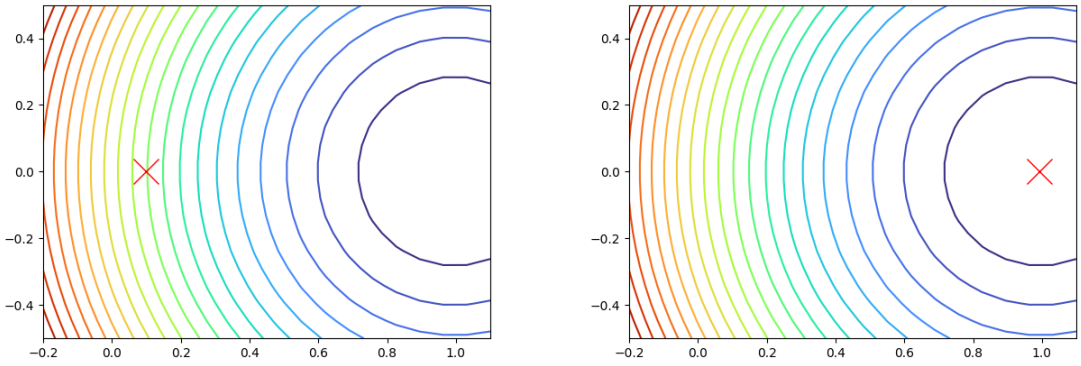


(a) First epoch 0.025



(b) Last epoch 0.025

Figure 6: Contour plot



(a) First epoch 0.1

(b) Last epoch 0.1

Figure 7: Contour plot

2 Q2

2.1 Q2(a)

Run as

```
$ python two_a.py
```

1. I wrote a snippet that takes the theta parameters, the size of the dataset to generate, the μ_1, σ_1^2, μ_2, σ_2^2 corresponding to the features x_1 and x_2 and also the variance of the noise σ_{noise}^2. And generates the dataset $\{x^{(i)}, y^{(i)}\}$ and returns it.

2. A little note, to vectorize the noise adding, the θ is appended with a 1 just as 1 is added to the x data to incorporate the intercept term in it.

2.2 Q2(b)

Run as

```
$ python two_b.py [--epochs [number-of-epochs]] [--batch-size
[batch-size]] [--eps [epsilon-used-in-convergence]]
```

Note, all the options are non-mandatory.

1. I wrote the `DataLoader` class, the `getdata` function, the `get_dataloader` function, the `dist` function, the `J` function as helper functions.

2. The `DataLoader` class is now augmented to support shuffling, and return mini-batches of data.

3. The main stochastic gradient code is in function named `two_b`. It's pretty straight forward. It consists of the initialization, the outer loop for each epoch and the inner loop to perform gradient descent on the minibatch. The batch size component is encapsulated in the `DataLoader`.

2.3 Q2(c)

Run as

```
$ python two_c.py --batch-size <batch-size> --test-file
path/to/q2test.csv [--no-skip-first]
```

The `--no-skip-first` argument relates to the csv file, whether the first row represent header or data.

Batch size	θ_0	θ_1	θ_2
1000000	2.87880453	1.0266891	1.99118959
10000	2.99767533	1.00118544	1.9995014
100	2.99932185	0.99948737	2.00046515
1	3.00208729	1.01619096	1.97847073
Original	3	1	2

Deltas	θ_0	θ_1	θ_2
1000000	0.12119547	0.0266891	0.00881041
10000	0.00232467	0.00118544	0.0004986
100	0.00067815	0.0051263	0.00046515
1	0.00208729	0.01619096	0.02152927

Percentage	θ_0	θ_1	θ_2
1000000	4.04	2.67	0.44
10000	0.08	0.11	0.02
100	0.02	0.51	0.02
1	0.07	1.62	1.08

Batch size	Euclidean norm of %ages seen as a vector
1000000	4.86
10000	0.14
100	0.51
1	1.95

Batch Size	MSE on test set
1000000	1.0261181852211296
10000	0.9830393147537965
100	0.9829447376744409
1	1.0225014963300452
Original θ	0.9829469214999982

Batch size	Epochs	Iterations
1000000	11500	11500
10000	240	24000
100 (did not converge)	20	200000
1	<1	83000

The speed of convergence was ordered like this for batch sizes: $10000 < 1 < 1000000 < 100^*$ (convergence was not reached for batch size 100.)

2.4 Q2(d)

Run as

```
$ python two_d.py --batch-size <batch-size> [--start-skip
[iteration number]] [--skip-frame [number of frames to
skip]] [--sleep [sleep duration]] [-s]
[--stop-at [stop iteration number]]
```

–**start-skip** Since the parameter movement starts slowing, skipping some frames makes the movement visible to eye, this argument tells which iteration to start skipping from

–**skip-frame** This tells how many frames to skip

–**sleep** The sleep duration between each frame plot

-s Save plots to directory named frames in the same folder as the code, to create animation with ffmpeg.

–**stop-at** Since the movements towards the end get really slow and imperceptible, this helps stop the animation early, also helpful while saving frames.

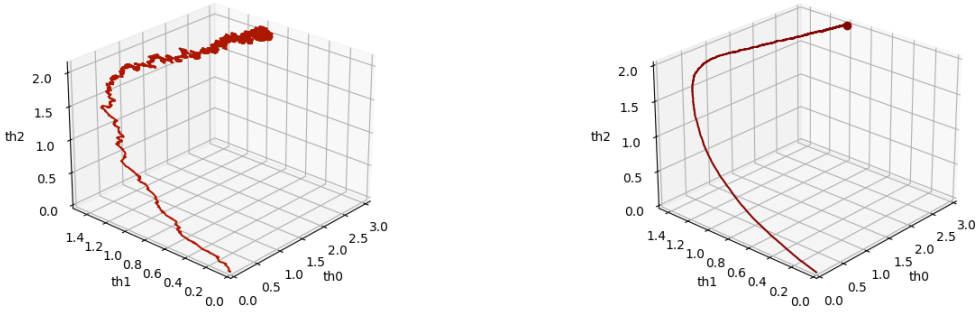
In the cost, we are basically performing sample mean of the squared errors. This is the number we want to minimize. And thus the gradient is also the sample mean of derivatives of squared error (in this case because the hypothesis does not create complex combinations of features).

The sample mean converges to the population mean as the sample size goes to ∞ according to weak law of large numbers.

Weak Law of Large Numbers: $\lim_{n\to\infty} P(|\bar{x_n} - \mu| > \varepsilon) = 0$ for all $\varepsilon > 0$.

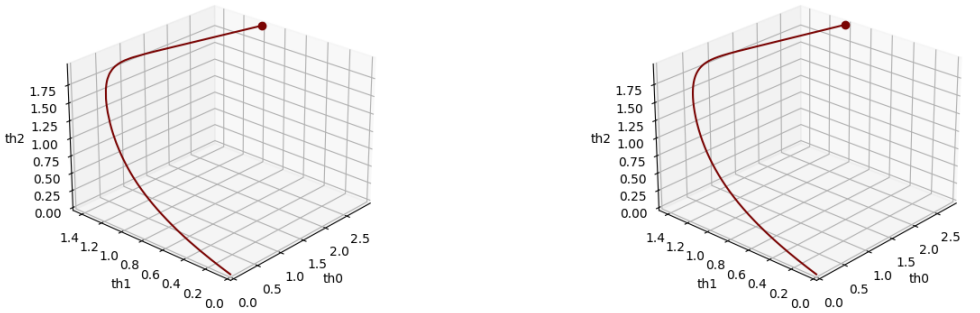
This means that the approximation to the gradient gets smoother as the batch size increases. Which is visible in the smoothness of the path the parameters take as batch size

increases. But this better approximation comes at the added cost of calculations. And as the batch size goes up, the number of updates to θ goes down per epoch, and more epochs are needed. Each epoch operates once on the whole data, thus the time taken is huge.



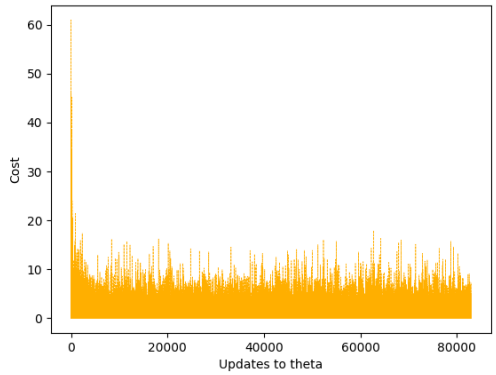
(a) Batch size = 1 (b) Batch size = 100

Figure 8: Parameter movement curve

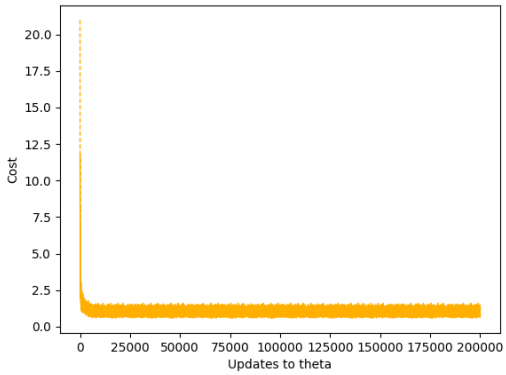


(a) Batch size = 10000 (b) Batch size = 1000000

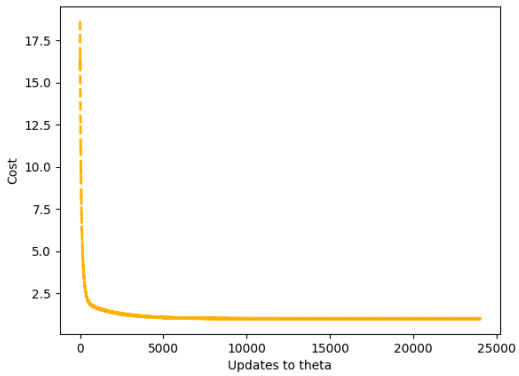
Figure 9: Parameter movement curve



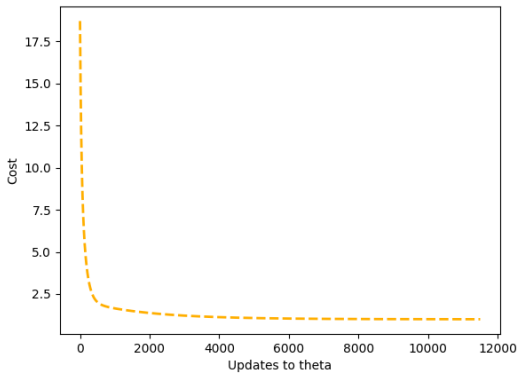
(a) Batch size: 1



(b) Batch size 100



(c) Batch size: 10000



(d) Batch size 1000000

Figure 10: Learning curves

3 Q3

3.1 Q3(a)

Run as

```
$ python three_a.py --data-x path/to/logisticX.csv --data-y
path/to/logisticY.csv
```

$$\ell(\theta; x, y) = \sum_{i=1}^{m} \left(y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right) \tag{1}$$

3.1.1 The Hessian of the log likelihood

First, as we know:

$$\nabla_\theta \ell(\theta; x, y) = \sum_{i=1}^{m} x^{(i)} \big(y^{(i)} - \hat{y}^{(i)}\big)$$

where $\hat{y}^{(i)} = \sigma(\theta^T x)$ and $\sigma(\cdot)$ is the sigmoid function. $\sigma(z) = \dfrac{1}{1+e^{-z}}$. The Hessian is the gradient of the gradient. It helped me to look at the equation component-wise.

$$\frac{\partial}{\partial \theta_j} \ell(\theta; x, y) = \sum_{i=1}^{m} x_j^{(i)} \big(y^{(i)} - \hat{y}^{(i)}\big)$$

I can now figure out the component of the Hessian that should be in the $j^{\text{th}}-$row and $k^{\text{th}}-$column (and vice versa, because the matrix is symmetric).

$$\begin{aligned}
\frac{\partial^2}{\partial \theta_j \partial \theta_k} \ell(\theta; x, y) &= \frac{\partial}{\partial \theta_k} \sum_{i=1}^{m} x_j^{(i)} \big(y^{(i)} - \sigma(\theta^T x^{(i)})\big) \\
&= -\sum_{i=1}^{m} x_j^{(i)} x_k^{(i)} \times \sigma'(\theta^T x^{(i)}) \\
&= -\sum_{i=1}^{m} x_j^{(i)} x_k^{(i)} \times \frac{e^{-\theta^T x^{(i)}}}{(1+e^{-\theta^T x^{(i)}})^2}
\end{aligned}$$

This gives the $(j,k)-$th position of the hessian. The contribution for the whole Hessian H, given one $x^{(i)}$ can be seen as the outer-product of the vector $x^{(i)}$ with itself.

Other things that I'd like to mention are:

1. The implementations `sigmoid`, `sigmoid_prime`, `J` of $\sigma(\cdot)$, $\sigma'(\cdot)$ and $\ell(\cdot)$ respectively were unstable. The reason was the exponential and logarithmic functions. I stabilized them with checking conditions to avoid overflow and underflow.

2. The Hessian also sometimes became non-invertible, a small constant was added to the diagonal elements of the Hessian to make the inverse stable.

3.1.2 The θ parameter values:

$$\begin{aligned}
\theta_0 &= 73.41152084 \\
\theta_1 &= 215.9852777 \\
\theta_2 &= -223.87490138
\end{aligned}$$

3.2 Q3(b)

Run as

```
$ python three_b.py --data-x path/to/logisticX.csv --data-y
path/to/logisticY.csv
```

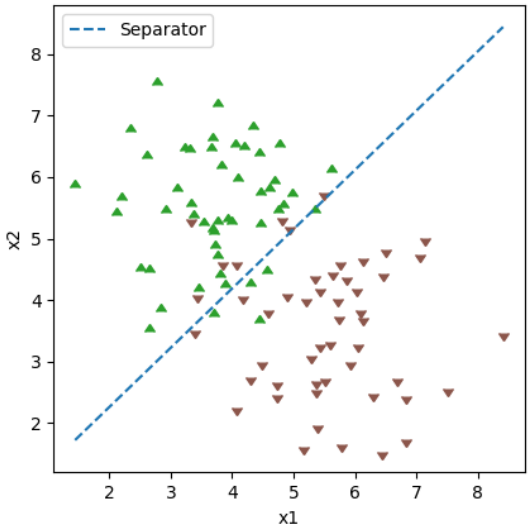


Figure 11: Decision Boundary for Logistic Regression

4 Q4

4.1 Q4(a)

Run as

```
$ python four_a.py --data-x path/to/q4x.dat --data-y
path/to/q4y.dat [-scale]
```

Note, the optional argument `-scale` normalizes the data, since it was mentioned to normalize data for Q4. Although, running without it gives the correct result as well. Values listed below are for runs with `-scale` enabled.

The values of the parameters found are:

$$\mu_0 = \begin{bmatrix} -0.75150837 \\ 0.68166023 \end{bmatrix}$$

$$\mu_1 = \begin{bmatrix} 0.75150837 \\ -0.68166023 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 0.42523517 & -0.02224756 \\ -0.02224756 & 0.52533933 \end{bmatrix}$$

4.2 Q4(b)

Run as

```
$ python four_b.py --data-x path/to/q4x.dat --data-y
path/to/q4y.dat [-scale]
```

The data plotted is shown in figure 12.

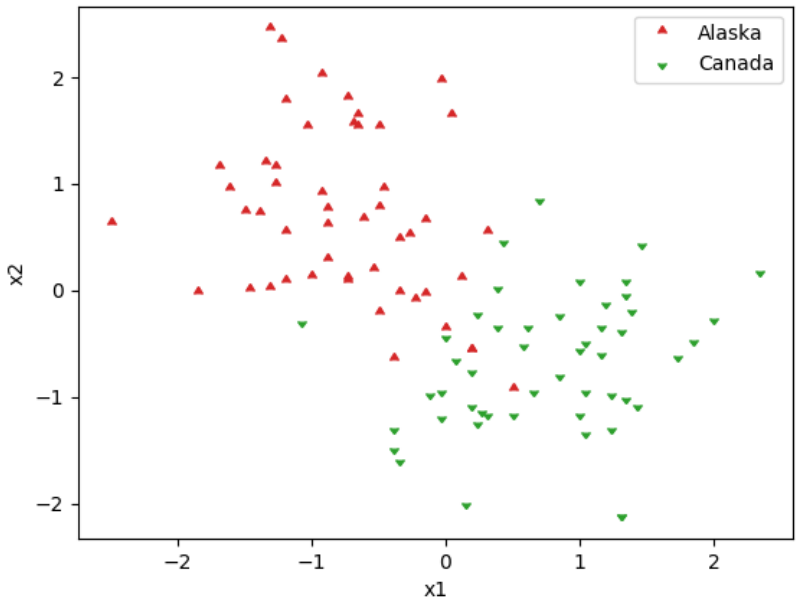


Figure 12: Data

4.3 Q4(c)

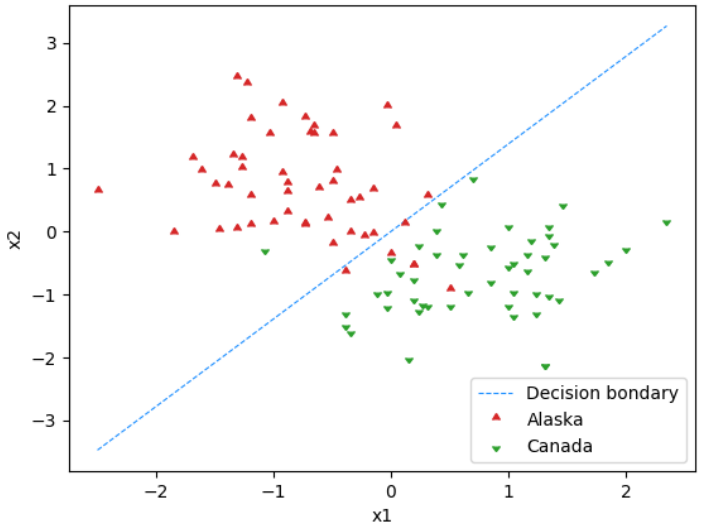


Figure 13: Linear Decision Boundary by GDA

Equation of the linear decision boundary is:

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

With the values of the parameters in place and with $x = [x_1 \; x_2]^T$:

$$0 = -6.43929354 \times 10^{-15} - 3.40633 x_1 + 2.45087 x_2 \tag{2}$$

4.4 Q4(d)

Run as

```
$ python four_d.py --data-x path/to/q4x.dat --data-y
path/to/q4y.dat [-scale]
```

The values of the parameters obtained are

$$\mu_0 = \begin{bmatrix} -0.75150837 \\ 0.68166023 \end{bmatrix}$$

$$\mu_1 = \begin{bmatrix} 0.75150837 \\ -0.68166023 \end{bmatrix}$$

$$\Sigma_0 = \begin{bmatrix} 0.37777389 & -0.15331651 \\ -0.15331651 & 0.6412598 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 0.47269646 & 0.10882139 \\ 0.10882139 & 0.40941887 \end{bmatrix}$$

4.5 Q4(e)

Run as

```
$ python four_e.py --data-x path/to/q4x.dat --data-y
path/to/q4y.dat [-scale]
```

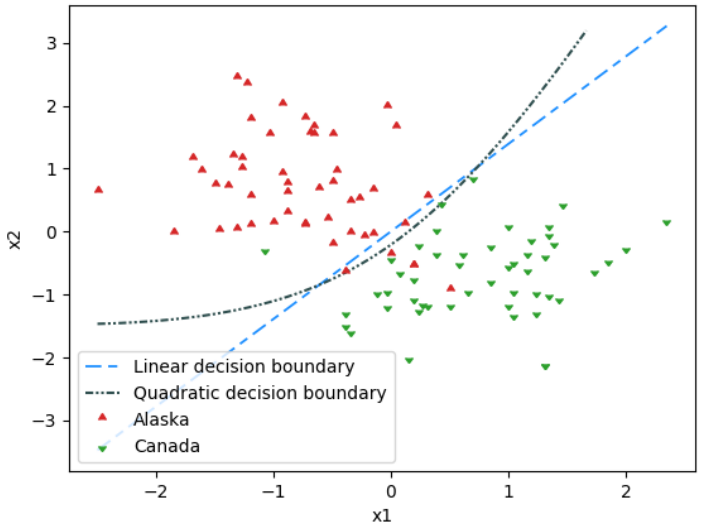


Figure 14: Quadratic and Linear Decision Boundary for GDA

The equation of the decision boundary with parameter values substituted is:

$$-0.8746x_2^2 + 0.67812x_1^2 + 2.5996x_1x_2 + 5.748x_2 - -7.654x_1 - 1.1695739 = 0$$

To get the equation of the decision boundary when $x \in \mathbb{R}^2$ I'll begin here:

$$P(y=1|x;\theta) = \frac{1}{1+\frac{P(x|y=0;\theta)P(y=0;\theta)}{P(x|y=1;\theta)P(y=1;\theta)}} \tag{3}$$

As in the lecture, we want $P(y=1|x;\theta) = 0.5$. And taking $A = \frac{P(x|y=0;\theta)P(y=0;\theta)}{P(x|y=1;\theta)P(y=1;\theta)}$, this implies $\log A = 0$.

$$A = \frac{1-\phi}{\phi} \times \frac{(2\pi)^{-\frac{n}{2}}|\Sigma_0|^{-0.5}\exp(-0.5(x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0))}{(2\pi)^{-\frac{n}{2}}|\Sigma_1|^{-0.5}\exp(-0.5(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1))}$$

$$\log A = \log\left[\frac{1-\phi}{\phi}\frac{|\Sigma_1|^{\frac{1}{2}}}{|\Sigma_0|^{\frac{1}{2}}}\right] - \frac{1}{2}\big((x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0) - (x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\big)$$

Taking the first log term there as k.

$$\log A = k - \frac{1}{2}\big(x^T\Sigma_0^{-1}x - 2\mu_0^T\Sigma_0^{-1}x + \mu_0^T\Sigma_0^{-1}\mu_0 - \big[x^T\Sigma_1^{-1}x - 2\mu_1^T\Sigma_1^{-1}x + \mu_1^T\Sigma_1^{-1}\mu_1\big]\big)$$

Adding the constants together

$$a = k - \frac{1}{2}\big[\mu_0^T\Sigma_0^{-1}\mu_0 - \mu_1^T\Sigma_1^{-1}\mu_1\big]$$

$$\log A = a - \frac{1}{2}\big[x^T\Sigma_0^{-1}x - x^T\Sigma_1^{-1}x - 2\big(\mu_0^T\Sigma_0^{-1}x - \mu_1^T\Sigma_1^{-1}x\big)\big]$$

This was all general, but now for the two-dimensional, data, we know that the covariance matrices are $\mathbb{R}^{2\times 2}$ and the mean vector is $\mathbb{R}^{2\times 1}$. Also, $\mu_i^T\Sigma_i^{-1} \in \mathbb{R}^{1\times 2}$ for $i \in \{0,1\}$. Using more notations:

$$\Sigma_0^{-1} - \Sigma_1^{-1} = \begin{bmatrix}\alpha & \beta\\ \gamma & \delta\end{bmatrix}$$

$$\mu_0^T\Sigma_0^{-1} - \mu_1^T\Sigma_1^{-1} = \begin{bmatrix}\varphi & \pi\end{bmatrix}$$

$$x = \begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

Let $b = 2a$, $\log A = 0$

$$0 = b - \alpha x_1^2 - (\gamma+\beta)x_1x_2 - \delta x_2^2 + 2\varphi x_1 + 2\pi x_2$$

At $x = t$, we can set

$$
\begin{aligned}
P &= -\delta \\
Q &= 2\pi - (\gamma + \beta)t \\
R &= 2\varphi t + b - \alpha t^2 \\
&\quad \text{such that} \\
0 &= Px_2^2 + Qx_2 + R \\
&\quad \text{and} \\
x_2 &= \frac{-Q \pm \sqrt{Q^2 - 4PR}}{2P}
\end{aligned}
$$

This is used in the program to calculate the x_2 coordinate for given x_1 coordinate to plot the graph. We get two values for x_2 for a give x_1, corresponding to points where the probability of sample being in each class is $\frac{1}{2}$. I only plot the one that would be visible in the graph based on the y-axis limits produced by the linear separator.

4.6 Q4(f)

The quadratic separator believes that as x_1 decreases, x_2 has to decrease much slower for the fish to still belong to class "Alaska". The linear separator is more biased towards a simpler decision boundary. The quadratic separator has more power to fit complicated data. This power is visible in the better fit when compared with the linear separator. In this situation I see three more points on the correct side of the curve in exchange of letting go of one that was being classified correctly before to the wrong side.