COL774 ML Assignment 1

Parth Thakur   2021CS50615

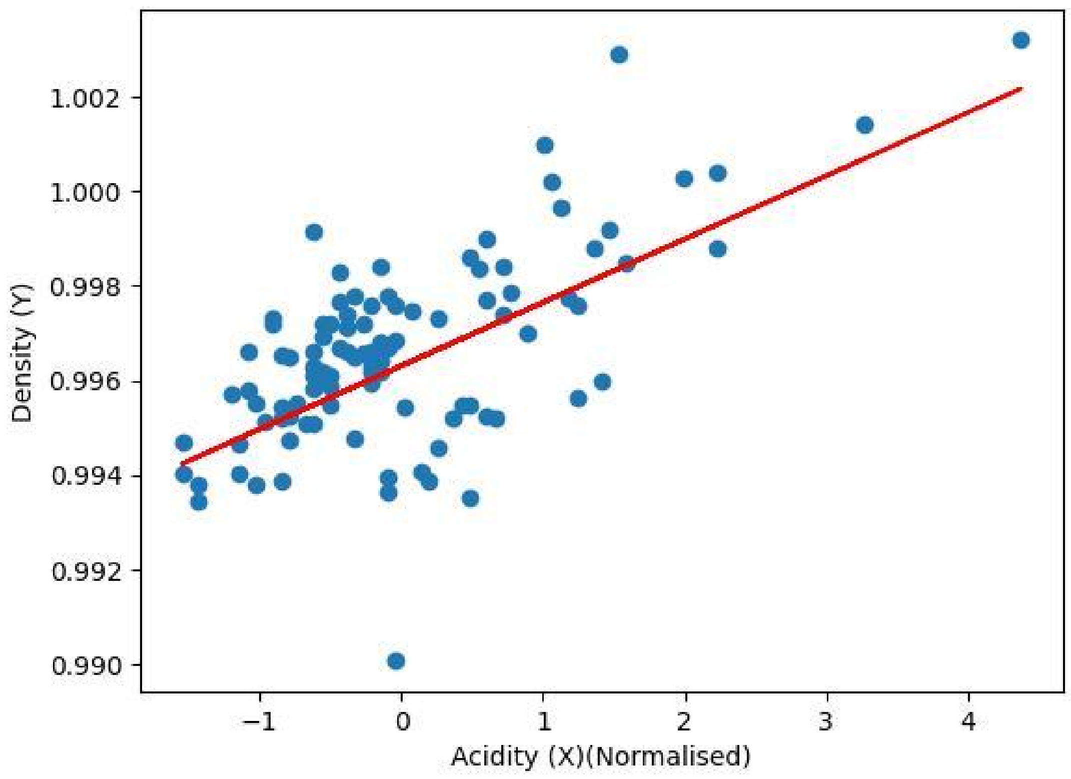
Question 1, Linear Regression

Please see q1.ipynb
Standard batch Gradient descent is implemented in the function gradient_descent()

(a)

- Learning Rate
  - The learning rate was chosen to be 0.01
- Stopping Criteria
  - The gradient descent is set to stop when the change in loss function becomes lesser than 10^-9
- Final Set of Parameters
  - Slope:
    - 0.0013397770034965105
  - Intercept:
    - 0.9963085046961379
  - Loss Function:
    - 1.2443213939662906e-06
  - Number of Iterations required for convergence:
    - 803

(b)



(c)

Please see "(c) 3D_trajectory_line_mesh.gif"

(d)

Please see "(d) 2D_trajectory_point_contour.gif"

(e)

Please see "(e) 2D Contour 0.1.gif

(e) 2D Contour 0.001.gif
(e) 2D Contour 0.025.gif”

Observation:

The convergence speed is much faster when the learning rate is large as the point approaches the minima quickly.

Question 2, Sampling and Stochastic Gradient Descent

Stochastic gradient descent is implemented in the function stochastic_gradient_descent(). The function evaluates for various batch sizes specified in the parameters.

(a)

1000000 values are generated using np.random.normal() using a random seed of 42.

(b)

Batch Size r	θ1	θ2	θ3
1	2.96104208	0.9928375	2.02408767
100	3.00006735	1.00003645	1.99948716
10,000	3.00346594	0.9998375	2.00067572
1,000,000	2.99137043	1.00196525	2.00027757

(c)

Original Test error (using original parameters [3,1,2]) = 1.965893843

Batch Size r	Test Error
1	2.029786951
100	1.966056623
10,000	1.965740705
1,000,000	1.966117036

Even with a constant learning rate and identical stopping criteria across all scenarios, different batch sizes led to different converged parameters. The learned parameters were remarkably close to the original values of [3,1,2] used to generate the data. The deviations were generally less than 1%, with two exceptions:
Θ3, in the case of a batch size of 1, had approximately a 2.5% error, and
Θ2 for a batch size of 1,000,000 exhibited around a 4% error.

In terms of convergence speed, a distinct pattern emerged. The time required for convergence was highest for a batch size of 1, decreased sharply for intermediate batch sizes, and gradually increased again for larger batch sizes. Several factors contribute to this trend:

Small Batch Size (1): The time to converge was elevated because the algorithm didn't take advantage of vectorization. Furthermore, the number of epochs needed was comparable to that of a batch size of 100, which did benefit from vectorization, leading to slower performance.

Intermediate Batch Size (100): This size optimized both computational efficiency and speed, as it fits nicely into CPU memory and exploited vectorization. Convergence was achieved in relatively few epochs, making it the fastest among the tested sizes.

Large Batch Size (10,000): The time required increased due to a substantial rise in the number of epochs needed for convergence.

Full Batch Size (1,000,000): This is essentially Batch Gradient Descent and requires the maximum number of epochs (10,000, as per the stopping criterion), making it relatively slow.

Regarding the test error on the new dataset, an intermediate batch size yielded the closest approximation to the original hypothesis. A batch size of 1 led to higher error rates due to the high variance in parameter updates, while the largest batch size resulted in higher error due to incomplete convergence (as no early stopping was implemented).

(d)

Please see "3D_trajectory_linebatch size_1.gif
3D_trajectory_linebatch size_100.gif
3D_trajectory_linebatch size_10000.gif
3D_trajectory_linebatch size_1000000.gif
"

Upon visualizing the trajectories of
θ across multiple iterations of Stochastic Gradient Descent (SGD) with varying batch sizes, distinct patterns emerge:

When the batch size is 1,000,000, which essentially mirrors a standard Batch Gradient Descent algorithm, the path initially diverges before ultimately converging to the optimal point. This trajectory appears to be smooth and well-defined.

For intermediate batch sizes of 100 and 10,000, the trajectories roughly emulate the path taken by the full batch size of 1,000,000. However, upon closer inspection, it's evident that these paths are more oscillatory and less smooth compared to the full batch gradient descent.

With a batch size of just 1, the trajectory is considerably more erratic. Despite this volatility, the overall direction of movement aligns generally well with the trajectories observed for larger batch sizes.

These observations align with our theoretical expectations. Smaller batch sizes introduce greater variance in the data points and, consequently, the computed gradients. This is why we observe a more 'rugged' or 'jagged' path for smaller batch sizes. Despite this, the general trend or direction of these smaller batches tends to agree with that of the larger batch sizes, affirming the efficacy of SGD even with limited data in each iteration.

Question 3, Logistic Regression

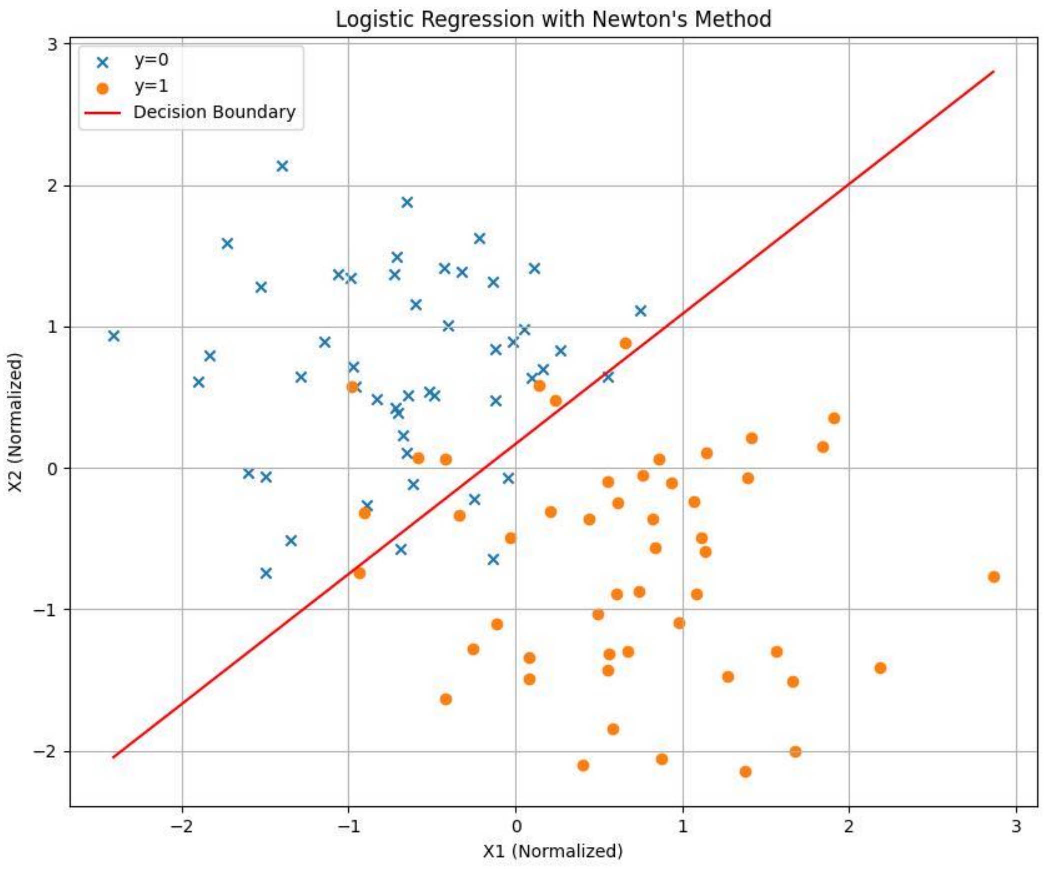
Logistic regression is implemented using Newton's method for approximating the root of a polynomial in the functions newton_method().

(a)

Final parameters:

Theta 0	0.55952206
Theta 1	1.93247687
Theta 2	-1.99558285

(b)



Question 4, Gaussian Discriminant Analysis

The calculations are done based on the standard formulae for GDA discussed in the class. SOme of them have been mentioned below.

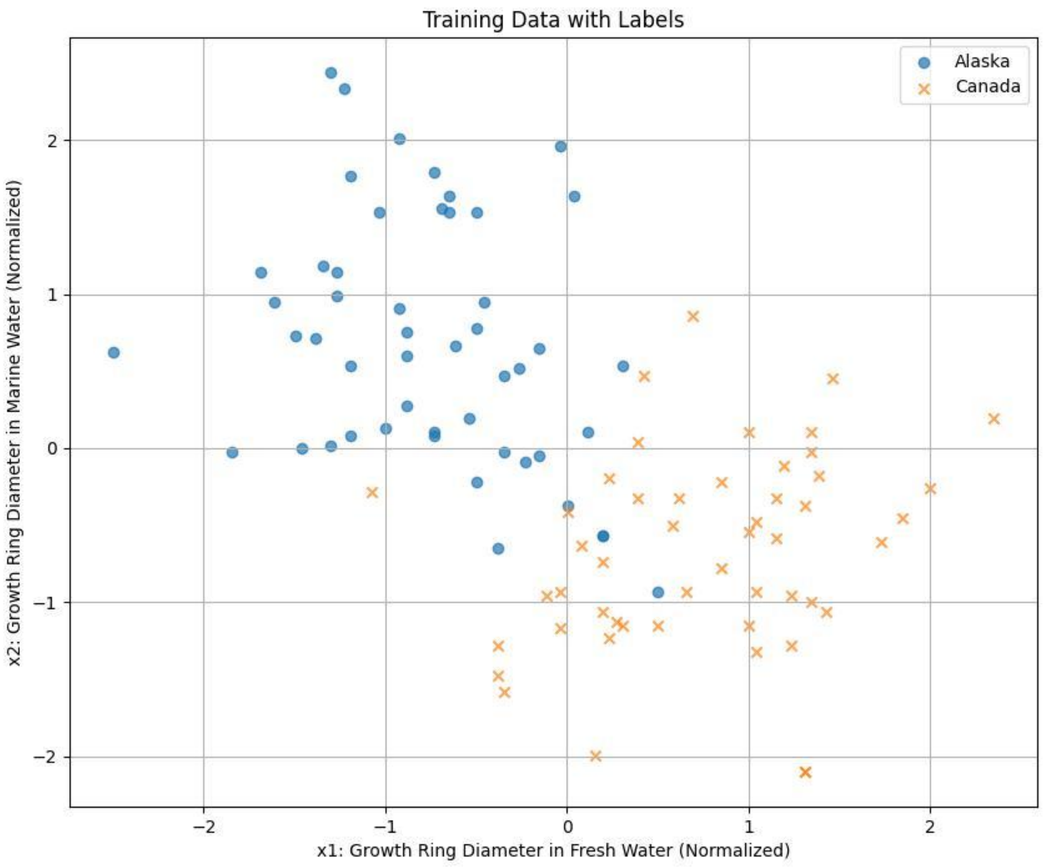
(a)

```
µ0 = ( −0.752
     0.682 )
```

$$\mu_1 = \begin{pmatrix} 0.752 \\ -0.682 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 0.425 & -0.022 \\ -0.022 & 0.525 \end{pmatrix}$$

(b)



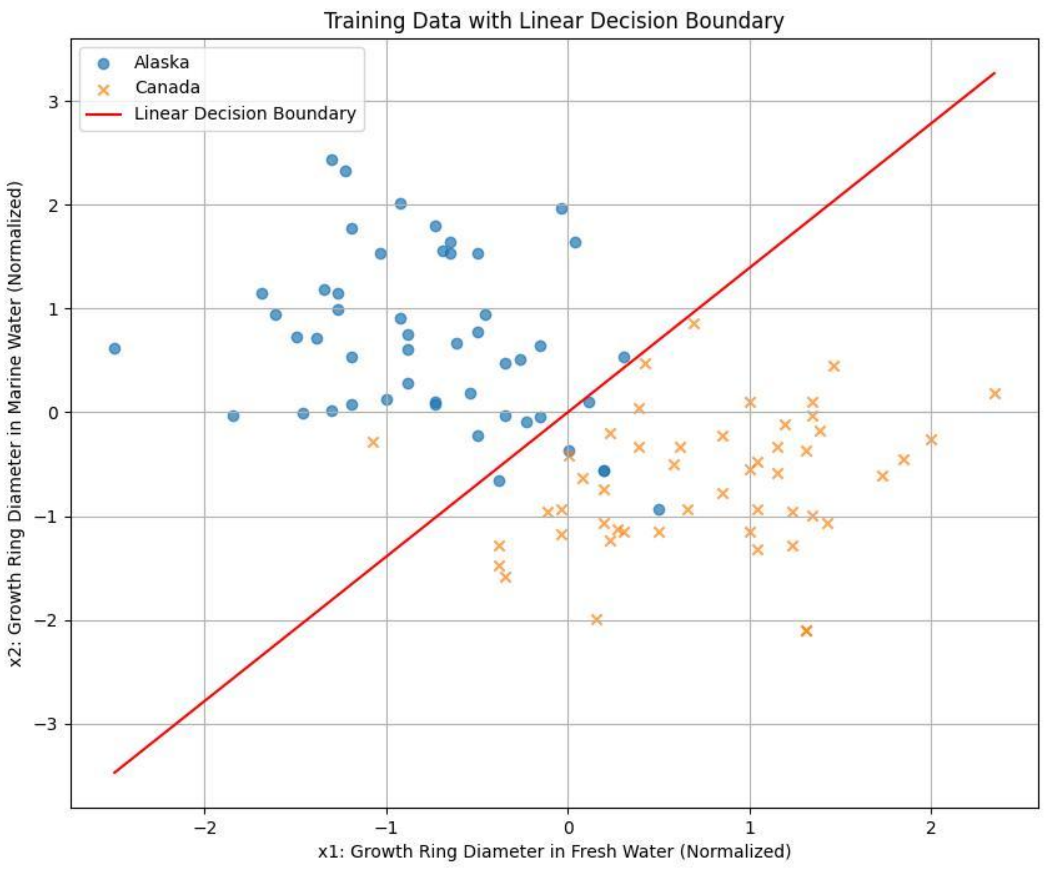
(c)

$$\theta^T x = 0$$

Where θ is a vector computed as:

$$\theta = \Sigma^{-1}(\mu_1 - \mu_0)$$

The vector θ for the decision boundary is calculated as [3.41,-2.45]. Plotting the line:



(d)

$$\Sigma_0 = \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}\{y^{(i)} = 0\}(x^{(i)} - \mu_0)(x^{(i)} - \mu_0)^T$$

$$\Sigma_1 = \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}\{y^{(i)} = 1\}(x^{(i)} - \mu_1)(x^{(i)} - \mu_1)^T$$

Calculating these, we get

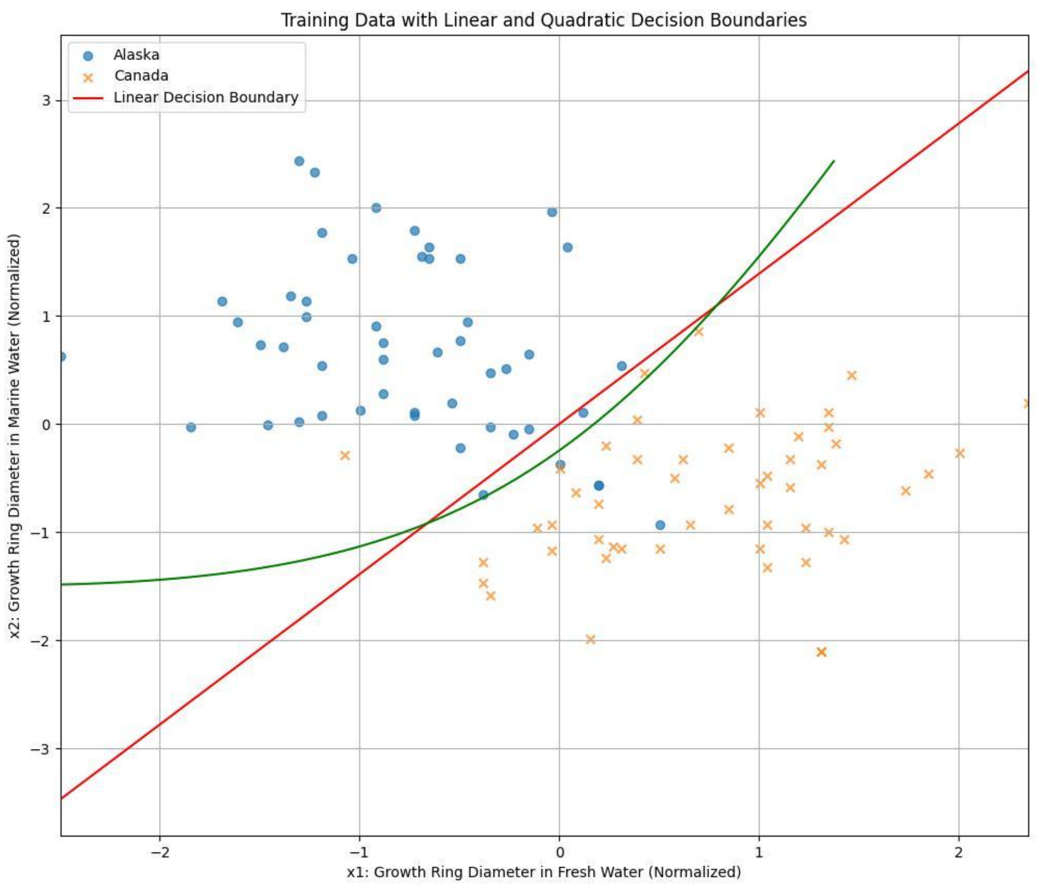
```
Σ0 = ( 0.378   -0.153
        -0.153      0.641 )

Σ1 = ( 0.473   0.109
        0.109 0.409 )
```

(e)

The equation defines the decision boundary:

$$(x - \mu_0)^T \Sigma_0^{-1} (x - \mu_0) = (x - \mu_1)^T \Sigma_1^{-1} (x - \mu_1)$$



(f)

Upon observing the linear and quadratic boundaries:

Linear Boundary: The linear decision boundary separates the two classes but isn't perfectly tailored to the data. In a real-world scenario, this might result in some misclassifications.

Quadratic Boundary: The quadratic decision boundary provides a more nuanced separation between the two classes. It fits the natural shape of the class distributions, potentially resulting in fewer misclassifications compared to the linear boundary.

In summary, while the linear boundary is easier to compute and interpret, the quadratic boundary may offer higher classification accuracy, especially when the covariance matrices of the two classes are significantly different.