COL 774: Machine Learning
Assignment 1

Anup Lal Nayak, 2022CS51827

1 Linear Regression

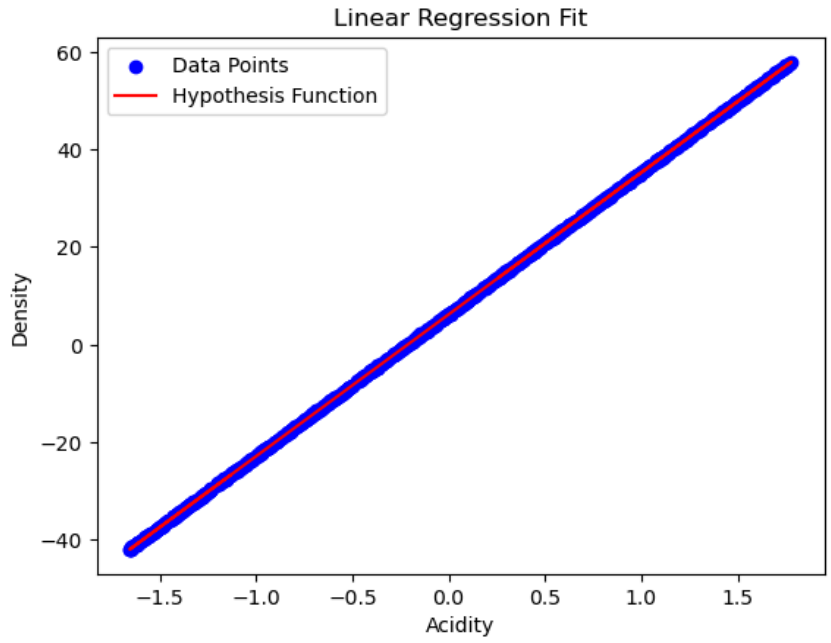
1a Batch Gradient Descent

Learning Rate = 0.01
Stopping Criteria = abs(J_prev - J_new) < 10e-15

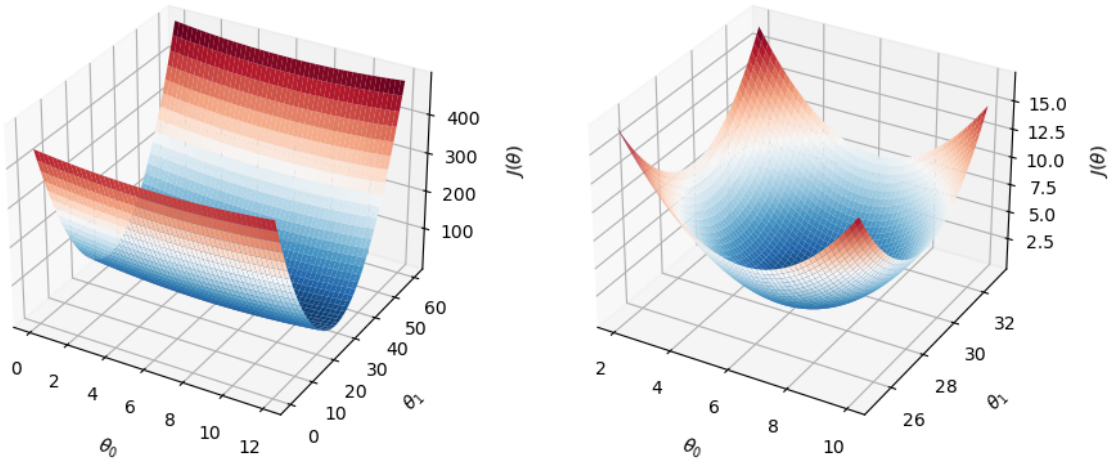
Final Parameters: [6.21867795 29.06475485]

Side note on learning rate:
The data provided to us for the analysis is almost perfectly linear, the following trend was observed for learning rates -
Too small a learning rate (e.g. η=0.001, η=0.0001) → Slow convergence.
Too large a learning rate (e.g. η=1 or more) → Might overshoot and diverge.
A value around **0.01 to 0.1** is typically optimal.

1b Plot of data and hypothesis function

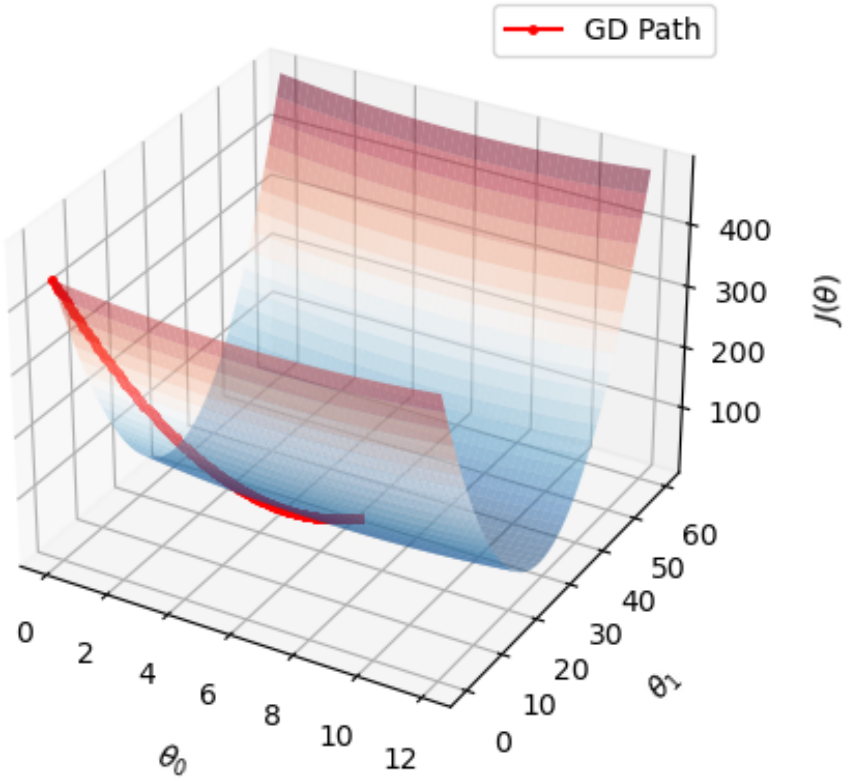


1c 3D Plot of Error Function

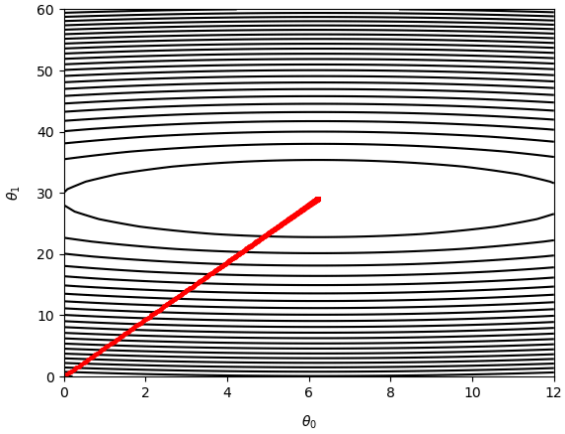


Note : The first plot is over the full range of theta values, while the second plot focuses near the optimal theta values which minimize J(theta).

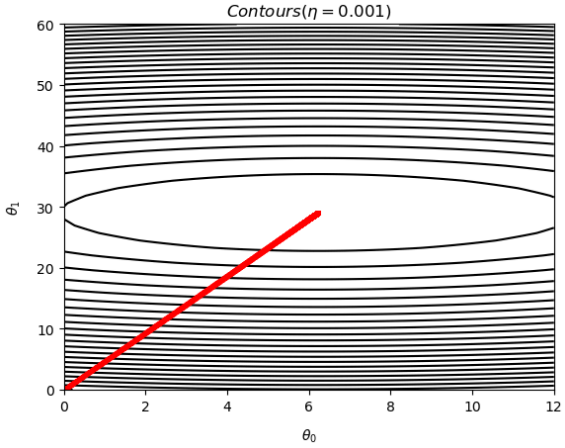
Plot with Error Values Traced Out

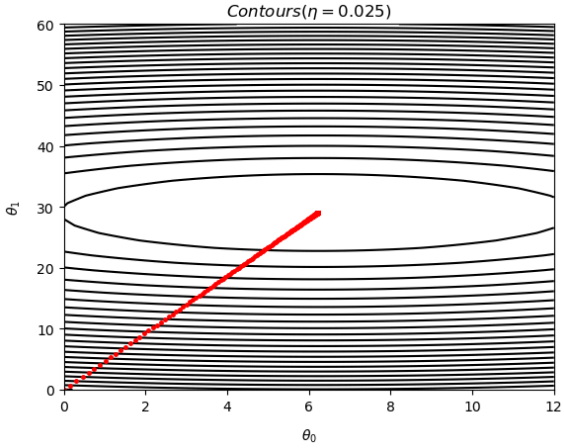


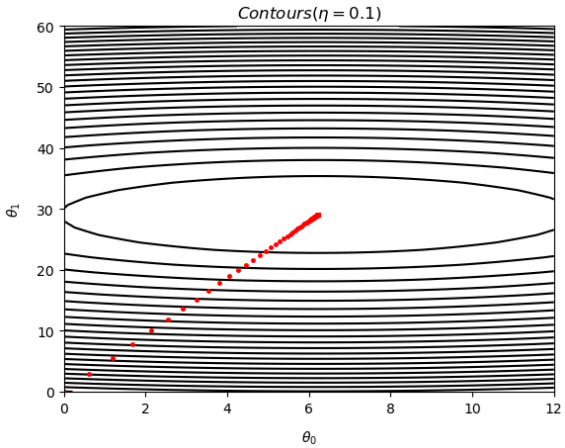
1d Contours of the error function at each iteration of GD



1e Contours for different values of η







For different learning rates η, the contour plot observations are:

- η=0.001: The gradient descent updates are very small, leading to slow but steady convergence without oscillations.
- η=0.025: A balanced learning rate, providing efficient convergence with minimal oscillations, reaching the minimum smoothly.
- η=0.1 Faster convergence but with slight oscillations around the minimum before stabilizing. Still efficient for well-scaled data. The total number of points is also less as less iterations are used.

As η increases, the trajectory of gradient descent changes: **small η** results in slow, smooth movement along contour lines, **moderate η** leads to efficient spiral-like convergence, and **large η** causes oscillations or divergence. The ideal η balances speed and stability, ensuring quick but controlled updates.

2 Sampling, Closed Form and Stochastic Gradient Descent

2a Sampling and Split

Split data after sampling has been saved in files:

generated_test_x.csv, generated_test_y.csv
generated_train_x.csv, generated_train_y.csv

**these have not been uploaded in the submission as they are very big*

2b Stochastic Gradient Descent

Convergence Criteria:

$$|\underset{\text{epoch } t+1}{average}(J(\theta^{t+1})) - \underset{\text{epoch } t}{average}(J(\theta^{t}))| \leq \varepsilon,$$

Batch Size	**Theta Values**	**Epochs**	**Time Taken**
1	[2.96925651 1.01119143 2.05958038]	28	179.5020 seconds
80	[2.99848874 0.99830698 1.99831796]	7	0.8997 seconds
8000	[2.98868344 1.00148966 1.99979509]	91	5.7025 seconds
800000	[2.88401131 1.01387318 1.99569734]	5321	385.6835 seconds

2c Closed Form Solution

Mini-batch gradient descent (batch size = 80 or 8000) is the most efficient in terms of both time and stability.
SGD (batch size = 1) is fast but noisy—good for large datasets where full-batch updates are infeasible.
Full-batch gradient descent (batch size = 800000) is the slowest but gives stable convergence.

The learned values of $\theta=[\theta 0,\theta 1,\theta 2]$ are **approximately the same** for all batch sizes. This indicates that the stochastic gradient descent (SGD) algorithm, regardless of batch size, is converging to the true underlying parameters. This is because we are using the same convergence condition instead of stopping after a certain number of epochs.

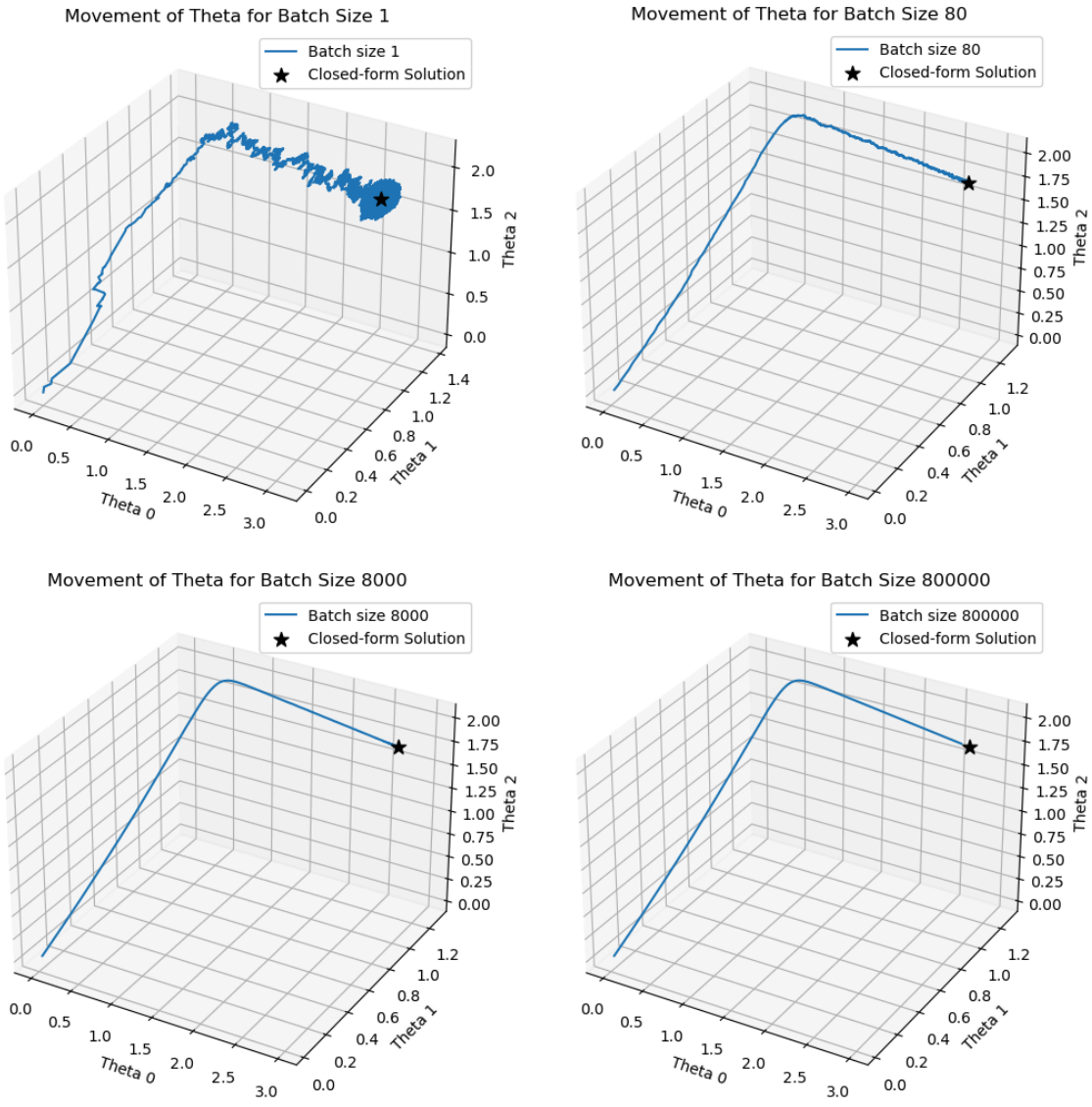
Closed-form solution for θ: [2.9996992 1.00019701 2.00022886]

Closed-form solution is the most accurate.
Mini-batch SGD (batch size = 80 or 8000) is the most efficient trade-off between speed and accuracy.
Very small batch sizes (batch size = 1) introduce more randomness, while full-batch (batch size = 800000) is slow and less flexible.

2d Train and Test Error

Batch Size	**Training Error**	**Test Error**
1	2.0070291471352255	2.0046009122584327
80	1.9971027641545818	1.995286882941997
8000	1.9971002144337986	1.9953522095340692
800000	2.0052528907507554	2.0029240110837234
Closed Form	1.9970153510682878	1.9953332652342235

2e Movement of Theta as parameters are updated



Batch Size = 1 (Top-Left Plot)

- The movement is highly **noisy and erratic**, with many small fluctuations before settling.
- This is due to the high variance in the gradient updates because each step is based on a single data point.
- Although it eventually converges, the randomness makes the trajectory much less smooth.

Batch Size = 80 (Top-Right Plot)

- The trajectory is much smoother than batch size 1 but still exhibits minor fluctuations.

- The updates are more stable, and it converges faster than the batch size 1 case.
- The path is relatively direct toward the closed-form solution.

Batch Size = 8000 (Bottom-Left Plot)

- The movement is **even smoother**, resembling a direct curve toward the final parameter values.
- The path is well-structured, showing that large batch sizes reduce noise significantly.
- It converges quickly compared to smaller batch sizes.

Batch Size = 800000 (Bottom-Right Plot)

- This is essentially **full-batch gradient descent**, and its path is the smoothest.
- The updates are very small, and the movement appears like a gradual curve toward convergence.
- However, the convergence is **very slow** because the algorithm updates only after computing gradients on the entire dataset.

Small batch sizes introduce noise, making the updates highly variable. This can help escape local minima but slows down convergence.

Moderate batch sizes (80, 8000) balance speed and stability, providing an efficient and relatively smooth trajectory.

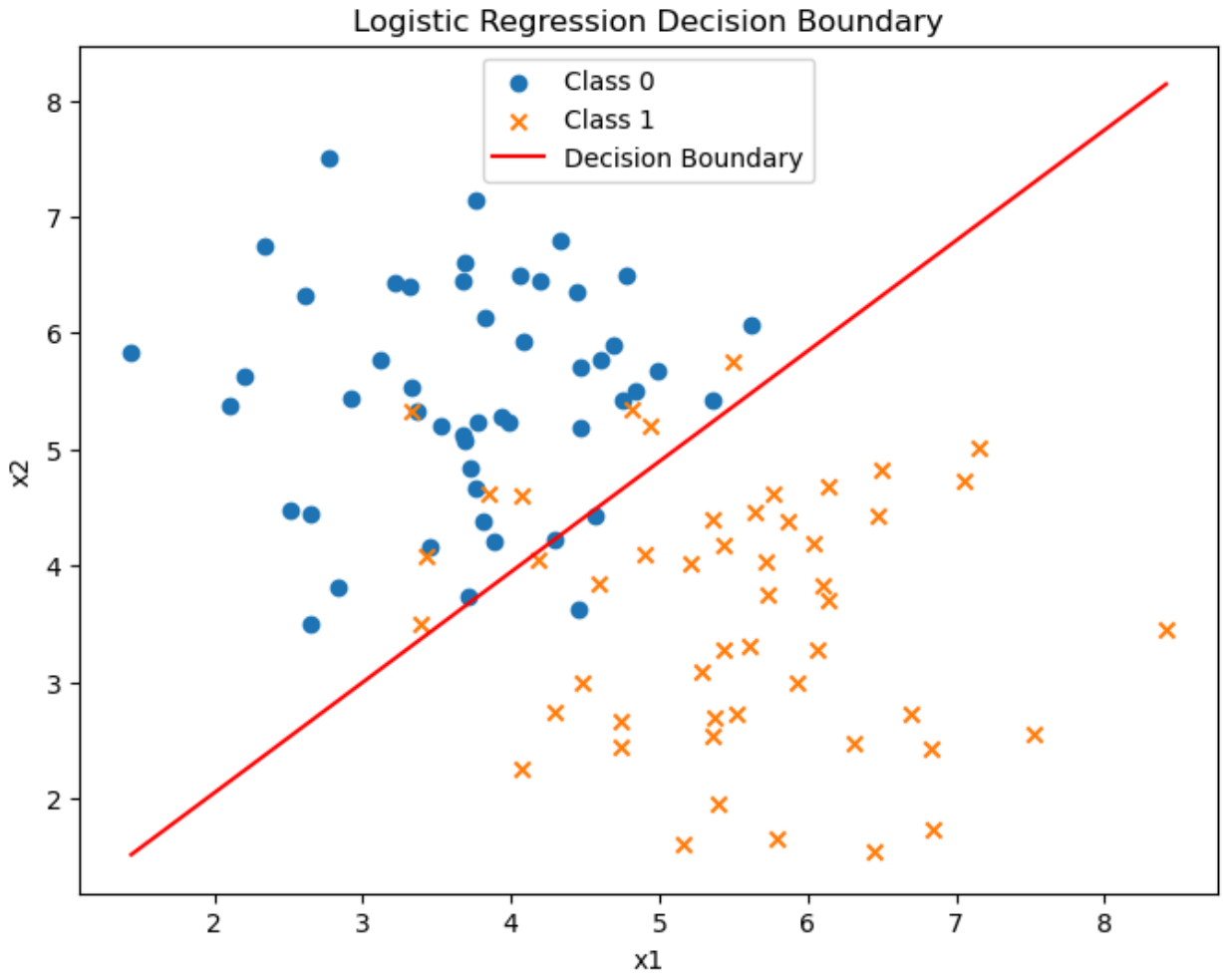
Full-batch gradient descent (800000) is the slowest but follows a very stable path, as each update is averaged over all data points.

3 Logistic Regression

3a Newton's Method

Final theta values: [0.40125316 2.5885477 -2.72558849]

3b Plot Values



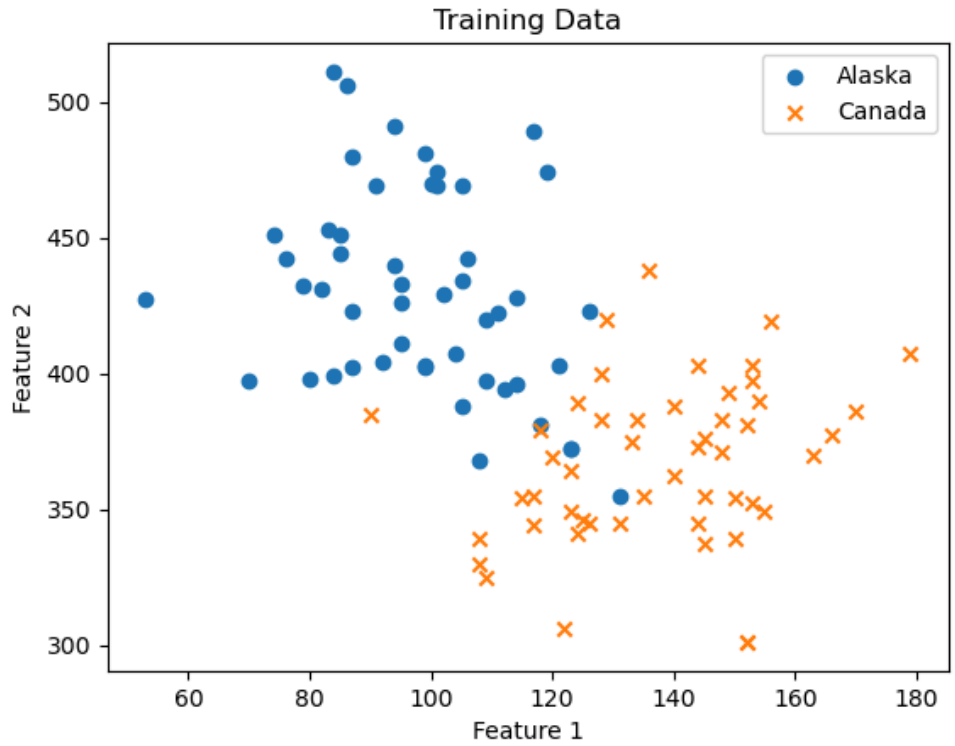
4 Gaussian Discriminant Analysis

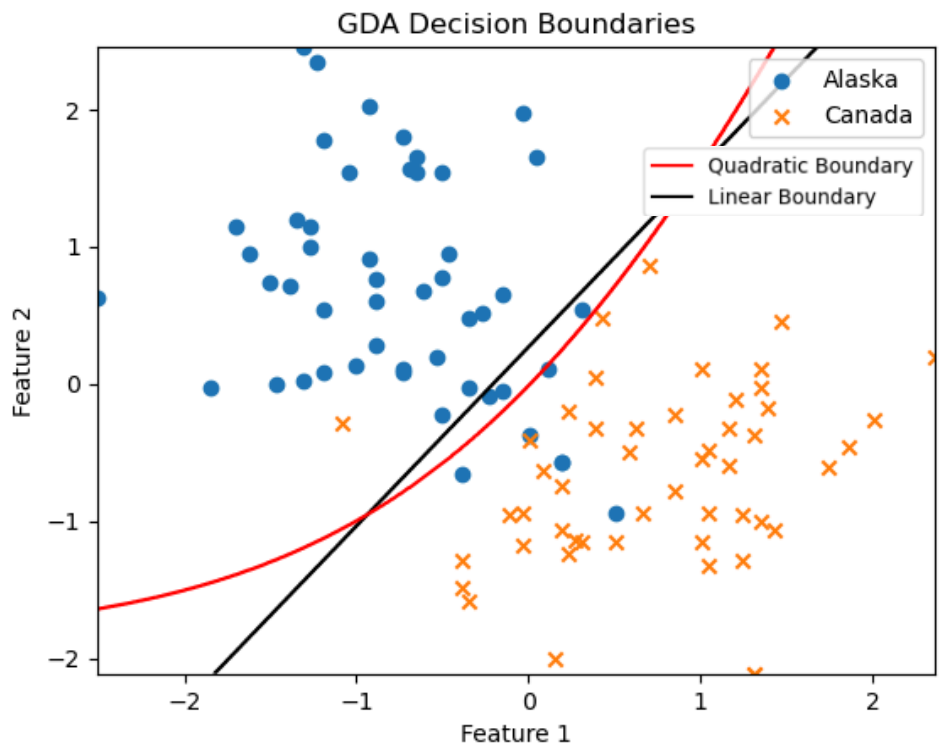
4a Linear GDA

Alaska -> 0, Canada -> 1

phi: 0.5
mu_0: [-0.75529433 0.68509431]
mu_1: [0.75529433 -0.68509431]
sigma: [[0.42953048 -0.02247228]
 [-0.02247228 0.53064579]]

4b Plot of training data





4c Equation of linear boundary

For a binary classification problem ($y \in \{0,1\}$), assume the feature vector $x \in \mathbb{R}^d$ follows a multivariate normal distribution for each class:

$p(x \mid y=0) = \mathcal{N}(\mu_0, \Sigma)$

$p(x \mid y=1) = \mathcal{N}(\mu_1, \Sigma)$

where:

- μ_0 and μ_1 are the mean vectors for each class.

- Σ is the shared covariance matrix for both classes.

The prior probabilities of the classes are:

$p(y=1) = \varphi, \quad p(y=0) = 1 - \varphi$

Using Bayes' rule, we obtain the posterior probability:

$p(y=1 \mid x) = [p(x \mid y=1)\, p(y=1)] / [p(x \mid y=1)\, p(y=1) + p(x \mid y=0)\, p(y=0)]$

Taking the log-odds of $p(y=1 \mid x)$ vs. $p(y=0 \mid x)$:

$\log [p(y=1 \mid x) / p(y=0 \mid x)] = \log [p(x \mid y=1)\, p(y=1) / p(x \mid y=0)\, p(y=0)]$

Substituting the Gaussian densities:

$\log \{ \exp[-1/2\, (x - \mu_1)^T \Sigma^{-1} (x - \mu_1)]\, \varphi \} / \{ \exp[-1/2\, (x - \mu_0)^T \Sigma^{-1} (x - \mu_0)]\, (1 - \varphi) \}$

Simplifying, we obtain the linear decision boundary equation:

$x^T \Sigma^{-1} (\mu_1 - \mu_0) - 1/2\, (\mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0) + \log [\varphi / (1 - \varphi)] = 0$

This equation represents a linear separator in GDA when the covariance matrices are the same for both classes. The decision boundary is a hyperplane in $\mathbb{R}$^d, where:

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

$b = -1/2\, (\mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0) + \log [\varphi / (1 - \varphi)]$

Thus, the decision rule is:

$w^T x + b = 0$

4d Quadratic GDA

phi: 0.5
mu_0: [-0.75529433 0.68509431]
mu_1: [0.75529433 -0.68509431]
sigma: [[0.42953048 -0.02247228]
 [-0.02247228 0.53064579]]

4e Equation of quadratic boundary

Using Bayes' rule, we obtain the posterior probability:

$p(y=1 \mid x) = [p(x \mid y=1)\, p(y=1)] / [p(x \mid y=1)\, p(y=1) + p(x \mid y=0)\, p(y=0)]$

Taking the log-odds of $p(y=1 \mid x)$ vs. $p(y=0 \mid x)$:

$\log [p(y=1 \mid x) / p(y=0 \mid x)] = \log [p(x \mid y=1)\, p(y=1) / p(x \mid y=0)\, p(y=0)]$

Substituting the Gaussian densities and simplifying, we obtain the quadratic decision boundary equation:

$x^T \Sigma_1^{-1} x - x^T \Sigma_0^{-1} x + 2(\mu_1^T \Sigma_1^{-1} - \mu_0^T \Sigma_0^{-1})\, x - (\mu_1^T \Sigma_1^{-1} \mu_1 - \mu_0^T \Sigma_0^{-1} \mu_0) + \log [\det(\Sigma_1) / \det(\Sigma_0)] + 2 \log [\varphi / (1 - \varphi)] = 0$

This equation represents a quadratic separator in GDA when the covariance matrices differ across classes. The decision boundary is no longer a hyperplane but a quadratic surface in $\mathbb{R}$^d.

4f Comments

Linear Boundary (Black Line):

- The linear decision boundary assumes that both classes share the same covariance matrix.
- It results in a straight-line separator between the two classes.
- While it works well for linearly separable data, it may not be optimal when the class distributions have different covariance structures.

Quadratic Boundary (Red Curve):

- The quadratic boundary appears to better fit the data distribution.
- This happens because the covariance matrices of the two classes are different, leading to a curved decision boundary.
- The quadratic separator allows for a more flexible partitioning of the feature space.

The quadratic boundary is more expressive and can capture the underlying data distribution better than the linear one when class covariances differ. However, if the distributions have similar covariances, the linear boundary is often sufficient and computationally simpler.