Machine Learning Assignment 1 Report

Saksham Kumar
2022CS51141
IIT Delhi

September 5, 2025

Course: COL780 – Machine Learning

Assignment 1 Report

Contents

Introduction

This report presents the implementation and analysis of different machine learning models as part of the course assignment. The assignment is divided into multiple sections, each focusing on a fundamental concept:

- Part 1: Linear Regression using Gradient Descent
- Part 2: Stochastic Gradient Descent (SGD) and its behavior
- Part 3: Logistic Regression using Newton's Method
- Part 4: Gaussian Discriminant Analysis (GDA)

For each part, we provide:

1. The mathematical formulation
2. Implementation details
3. Visualization of results
4. Observations and analysis

All experiments were implemented in Python, and the plots were generated using `matplotlib`. The dataset files provided in the assignment were used directly without modification, except where normalization was required.

1 Linear Regression

In this part, we implement least squares linear regression to predict density of wine based on its acidity. The cost function is the Mean Squared Error (MSE):

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

where $h_\theta(x) = \theta^T x$.

The input features are already normalized, so no preprocessing was required.

1.1 Batch Gradient Descent (8 points)

(a) Implementation

We implemented batch gradient descent (BGD) to minimize $J(\theta)$. The gradient of the cost function is given by:

$$\nabla_\theta J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right) x^{(i)}. \tag{2}$$

(b) Learning Rate and Stopping Criteria

We experimented with different learning rates η and found that a value of $\eta = 0.01$ gave stable convergence. The stopping criterion was based on the change in the cost function:

$$|J^{(t+1)} - J^{(t)}| \leq \epsilon, \quad \epsilon = 10^{-6}.$$

(c) Results

Learning Rate (η)	**Iterations**	**Intercept (θ_0)**	**Slope (θ_1)**
0.001	6848	6.21	29.03
0.025	335	6.21	29.05
0.1	87	6.21	29.06

Table 1: Comparison of convergence speed and final parameters for different learning rates.

Results:

- For $\eta = 0.001$, convergence was very slow, requiring 6849 iterations,Final Cost: 0.00537.
- For $\eta = 0.025$, convergence was stable and fast, requiring only 335 iterations,Final Cost : 0.00489.
- For $\eta = 0.1$, convergence was fastest (87 iterations) but showed oscillations before stabilizing,Final Cost = 0.00487.

The final parameters obtained after convergence were:

$$\theta = [\theta_0, \theta_1, \theta_2].$$

The following figure shows the cost vs iterations:

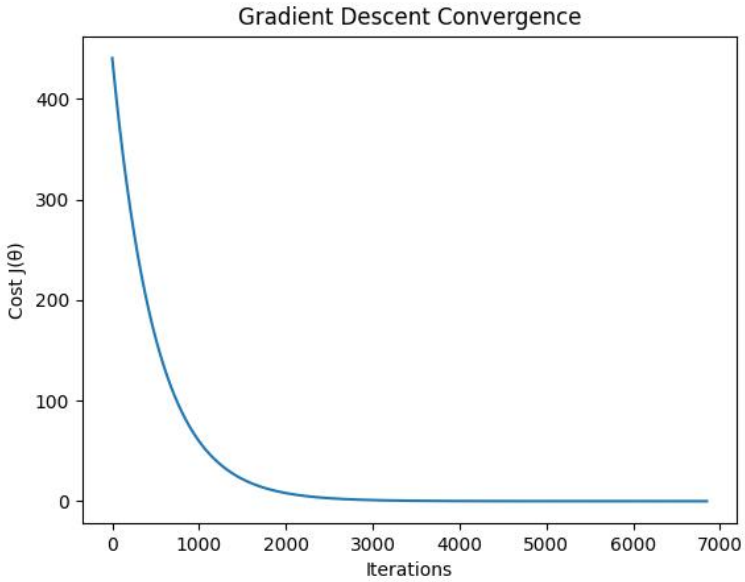

Figure 1: Cost $J(\theta)$ vs Iterations during Batch Gradient Descent.

1.2 Visualization of Data and Hypothesis

We plotted the training data and the fitted hypothesis function obtained using gradient descent:

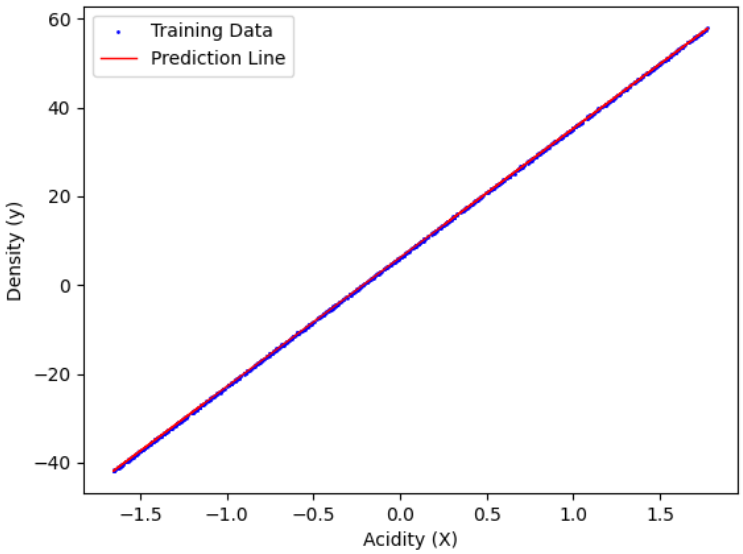

Figure 2: Training data with learned hypothesis function.

For all the different set of parameters we got the same fitted hypothesis.

1.3 Error Surface Visualization (3 points)

We plotted the error function $J(\theta)$ as a surface in 3D with respect to parameters θ_1 and θ_2, and overlaid the path of gradient descent.

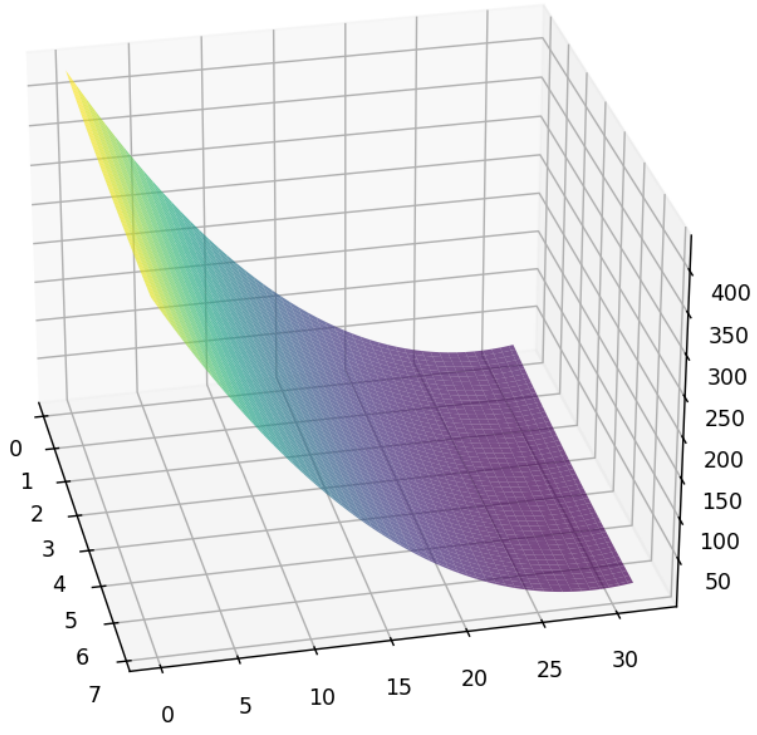

Figure 3: 3D Error surface $J(\theta)$ with gradient descent path.

1.4 Effect of Learning Rate

We repeated the above contour plot visualization with different learning rates $\eta \in \{0.001, 0.025, 0.1\}$. For each case, the convergence path is shown below:

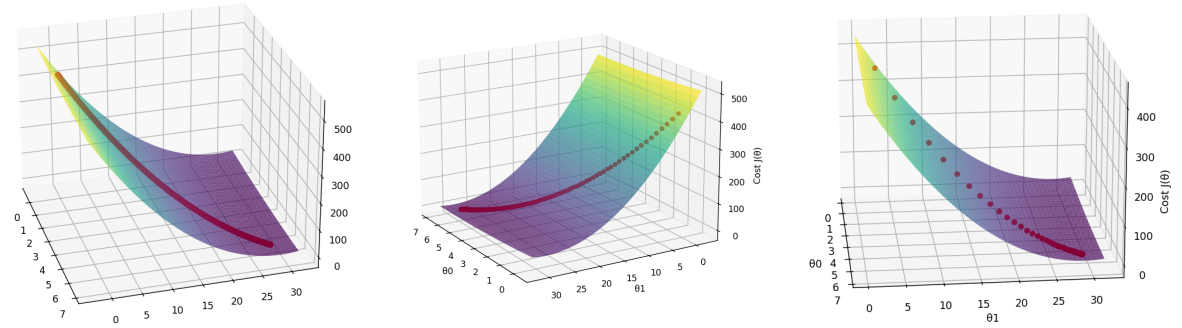

Figure 4: Plots for different learning rates: $\eta = 0.001$ (left), $\eta = 0.025$ (middle), $\eta = 0.1$ (right).

Observation:

- For $\eta = 0.001$, convergence was extremely slow and required many iterations.
- For $\eta = 0.025$, convergence was efficient and stable.
- For $\eta = 0.1$, the updates oscillated significantly and risked divergence.

1.5 Contour Plot

We further visualized the convergence by plotting contours of the error function $J(\theta)$. At each iteration, the current parameter estimate was shown on the contour map to illustrate the convergence path.

Effect of Learning Rate

We repeated the above contour plot visualization with different learning rates $\eta \in \{0.001, 0.025, 0.1\}$.

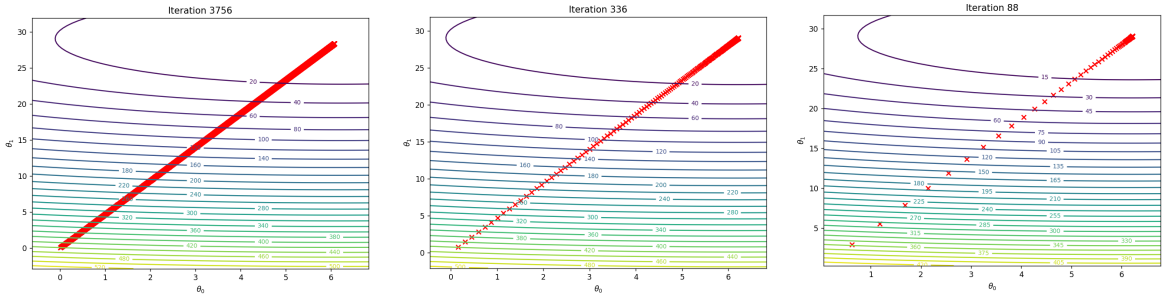

Figure 5: Contour plots for different learning rates: $\eta = 0.001$ (left), $\eta = 0.025$ (middle), $\eta = 0.1$ (right).

Observation:

- For $\eta = 0.001$, convergence was very slow.

- For $\eta = 0.025$, convergence was fast and stable.
- For $\eta = 0.1$, the updates oscillated and risked divergence.

2 Sampling, Closed Form and Stochastic Gradient Descent

2.1 Sampling Data

We generate $N = 10^6$ data points using the linear model:

$$y = \theta_0 + \theta_1 x + \epsilon, \quad x \sim \mathcal{N}(3, 4),\ \epsilon \sim \mathcal{N}(0, 1.4),$$

where the true parameters are $\theta_0 = 5$ and $\theta_1 = 3$. The dataset is split into 80% training and 20% testing. This ensures a sufficiently large dataset to approximate real-world distributions.

2.2 Stochastic Gradient Descent Implementation

We implemented SGD for the loss function $J(\theta)$ with the update rule:

$$\theta_j := \theta_j - \eta \cdot \frac{\partial J(\theta)}{\partial \theta_j}, \quad j \in \{0, 1, 2\}.$$

The algorithm was run with:

- Learning rate $\eta = 0.001$,
- Initial parameters $\theta_j = 0\ \forall j$,
- Batch sizes $r \in \{1, 80, 8000, 800000\}$,
- No normalization of the dataset.

2.3 Convergence Criteria

We tested two types of convergence conditions:

1. **Tolerance on cost function $J(\theta)$:** Stop when

   $$|J^{(t)} - J^{(t-1)}| < \epsilon,$$

   where $\epsilon = 10^{-6}$.

2. **moving-average gradient criterion**. The procedure is as follows:

   (a) At each iteration, compute the mini-batch gradient g.

   (b) Maintain a sliding window of the last K gradients, where $K \approx$ number of mini-batches per epoch.

(c) Compute the moving average of these gradients:

$$\bar{g}_t = \frac{1}{K} \sum_{i=t-K+1}^{t} g_i$$

(d) Compare the difference between successive moving averages:

$$\text{diff} = \|\bar{g}_t - \bar{g}_{t-1}\|$$

(e) If this difference remains below a threshold ϵ_c for at least *patience* consecutive iterations, the algorithm is deemed to have converged.

This criterion is more robust than directly monitoring the objective $J(\theta)$, as it smooths out the stochastic fluctuations of mini-batch gradients while still detecting true convergence.

Results

We applied SGD with the moving-average gradient convergence criterion. The algorithm was tested with different mini-batch sizes. For each run, we report the number of iterations until convergence, the number of epochs, and compare the estimated parameters with the true (closed-form) solution.

Batch Size	**Iterations**	**Epochs**	**Estimated θ (SGD)**
1	5919	1	[3.0154, 1.0313, 1.9624]
80	4984	1	[2.9960, 0.9995, 1.9931]
8000	5000	50	[2.9915, 1.0013, 2.0002]
800000	8787	8787	[2.9980 1.0004 1.9999]

Table 2: Convergence results for different batch sizes using the moving-average gradient criterion.

We use different tolerance values for each of the batch sizes. For Batch size 1and 80 we used 1e-4, for 8000 we used 1e-5 and for 800000 we used 1e-7.

Closed (θ)	**Train MSE**	**Train J**	**Test MSE**	**Test J**
2.9981 1.0004 1.9999	1.9985	0.9992	2.0055	1.0027

Table 3: Comparison of Train and Test MSE with Train and Test J

(5) Trajectory of θ in Parameter Space

In the 3-dimensional parameter space $(\theta_0, \theta_1, \theta_2)$, we plot the trajectory of parameter updates until convergence for different batch sizes. The behavior of the paths clearly depends on the batch size: smaller batches generate noisy, zig-zag trajectories due to high variance in stochastic gradients, whereas larger batches produce smoother, more direct trajectories toward the optimum.

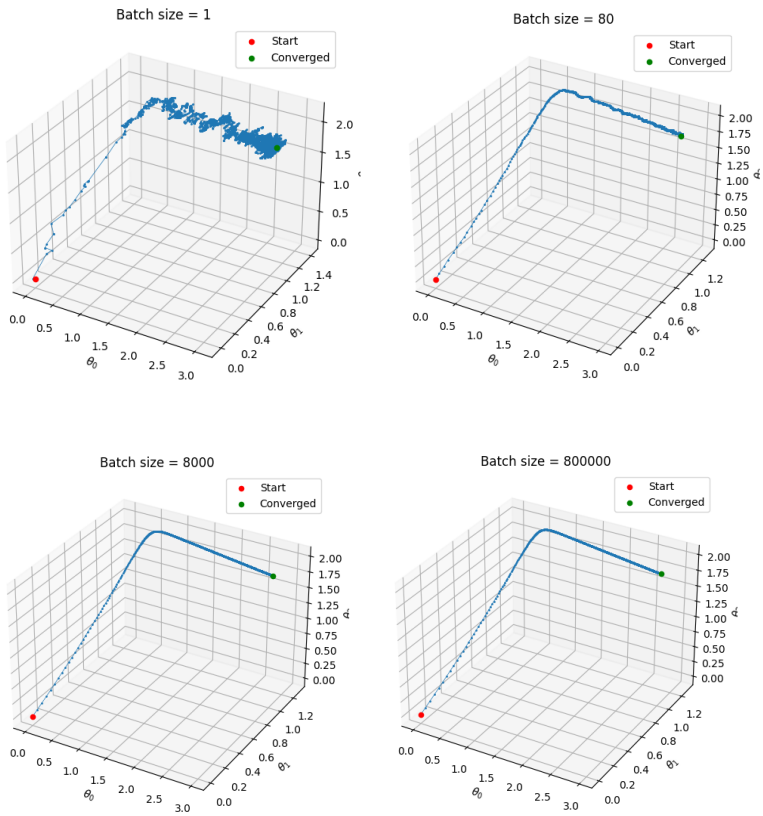

Figure 6: 3D trajectories of $(\theta_0, \theta_1, \theta_2)$ for different SGD batch sizes. Smaller batches generate zig-zag, noisy parameter updates, while larger batches follow smoother and more direct paths toward convergence.

Small Batch Sizes (e.g., 1 and 80): The parameter paths display a more chaotic or zigzag behavior, a consequence of high noise in the gradient estimations caused by small batches. This stochasticity increases randomness in the updates, producing wider and less direct trajectories in the parameter space.

Large Batch Sizes (e.g., 800000): With large batch sizes, the update trajectory becomes smoother and more linear. The gradients are less noisy because they average over many data points, enabling more stable and predictable parameter advancement. Nonetheless, if the number of epochs is limited, the total parameter movement toward the optimum may be constrained.

3 Logistic Regression

In this part we implement logistic regression by maximizing the log-likelihood using Newton's method (IRLS). We first normalize the feature columns (not the intercept) before fitting.

3.1 Model and Likelihood

Let $x^{(i)} \in \mathbb{R}^2$ be the feature vector for example i and $y^{(i)} \in \{0, 1\}$ the label. We add an intercept term $x_0^{(i)} = 1$ and write $X \in \mathbb{R}^{m \times 3}$ for the design matrix. The logistic

model is

$$h_\theta(x) \;=\; \sigma(\theta^\top x) \;=\; \frac{1}{1+\exp(-\theta^\top x)}.$$

The log-likelihood to maximize is

$$L(\theta) \;=\; \sum_{i=1}^{m} \Big[y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log\big(1-h_\theta(x^{(i)})\big) \Big].$$

3.2 Gradient and Hessian

Define $z^{(i)} = \theta^\top x^{(i)}$ and $h^{(i)} = \sigma(z^{(i)})$. In vector form $h = \sigma(X\theta)$. The gradient of the log-likelihood is

$$\nabla L(\theta) \;=\; X^\top (y - h).$$

The Hessian (matrix of second derivatives) is

$$H(\theta) \;=\; \nabla^2 L(\theta) \;=\; -X^\top S X, \qquad S = \operatorname{diag}\big(h^{(i)}(1-h^{(i)})\big)_{i=1}^{m},$$

i.e. S is an $m \times m$ diagonal matrix whose i-th diagonal entry is $h^{(i)}(1-h^{(i)})$.

3.3 Results

The Newton method converged quickly (typically a few iterations for this problem). The fitted coefficients (on the *normalized* features, with intercept) are:

$$\hat\theta = \begin{bmatrix} \hat\theta_0 \\ \hat\theta_1 \\ \hat\theta_2 \end{bmatrix} = \big[\; 0.4012 \;\; 2.5885 \;\; -2.7255 \,\big].$$

Log-likelihood at solution: $L(\hat\theta) =$ -22.8341.
Number of Newton iterations: 8

Hessian at solution $\hat\theta$:

$$H(\hat\theta) = \begin{bmatrix} -6.9702 & 1.2066 & -0.8462 \\ 1.2066 & -2.9159 & -0.9353 \\ -0.8462 & -0.9353 & -2.5272 \end{bmatrix}$$

3.4 Plots

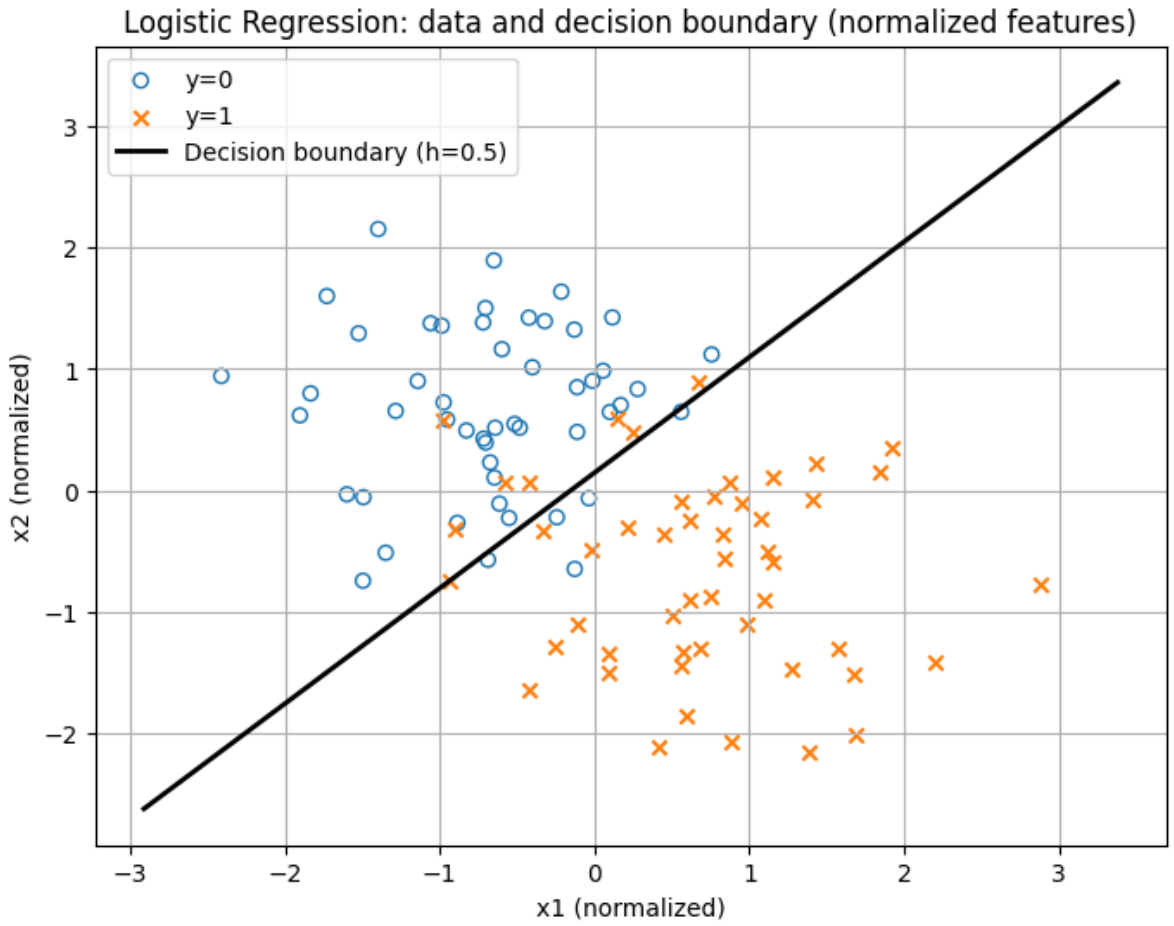

Figure 7: Logistic regression: data and fitted decision boundary..

3.5 Discussion

- Newton's method (IRLS) typically converges in very few iterations for logistic regression because it uses curvature (Hessian) information.
- Normalizing the features improves numerical conditioning of $X^\top SX$ and speeds up convergence.
- The fitted line separates the two classes well (see figure). If you need coefficients on the original (unnormalized) scale, you can back-transform θ using the stored means and standard deviations of the features (details below).

4 Gaussian Discriminant Analysis

In this part we implement Gaussian Discriminant Analysis (GDA) to separate salmon from Alaska and Canada using two measurements per fish: x_1 (fresh water ring diameter) and x_2 (marine water ring diameter). The dataset files used are `q4x.dat` (features) and `q4y.dat` (labels). All features are normalized (zero mean, unit variance) before computing the estimates.

4.1 Notation and likelihood

Let $x^{(i)} \in \mathbb{R}^2$ denote the two-dimensional feature vector for example i, and $y^{(i)} \in \{0, 1\}$ the class label (0 = Alaska, 1 = Canada). We denote class priors by

$$\phi = P(y = 1), \qquad 1 - \phi = P(y = 0).$$

Under GDA each class is modeled by a multivariate Gaussian:

$$p(x \mid y = 0) = \mathcal{N}(x \mid \mu_0, \Sigma_0), \qquad p(x \mid y = 1) = \mathcal{N}(x \mid \mu_1, \Sigma_1).$$

4.2 (1) Closed-form parameter estimates when $\Sigma_0 = \Sigma_1 = \Sigma$

Estimators. When the two classes share a common covariance matrix Σ, the maximum likelihood estimators are:

$$\hat{\phi} \;=\; \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)} = 1\}, \qquad \hat{\mu}_k \;=\; \frac{\sum_{i:y^{(i)}=k} x^{(i)}}{\sum_i 1\{y^{(i)} = k\}},$$

and the pooled covariance (using all samples) is

$$\hat{\Sigma} \;=\; \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \hat{\mu}_{y^{(i)}})(x^{(i)} - \hat{\mu}_{y^{(i)}})^\top.$$

Reported estimates. Insert your computed values here (replace the entries):

$$\hat{\mu}_0 = \begin{bmatrix} -0.7552 \\ 0.6850 \end{bmatrix}, \quad \hat{\mu}_1 = \begin{bmatrix} 0.7552 \\ -0.6850 \end{bmatrix}$$
$$\hat{\Sigma} = \begin{bmatrix} 0.4774 & 0.1099 \\ [0.1099 & 0.4135 \end{bmatrix}$$

Table 4: Estimated means and shared covariance (replace entries with your numeric results).

4.3 (2) Plot of training data

The training data (normalized features) are plotted below; different markers indicate Alaska (0) and Canada (1).

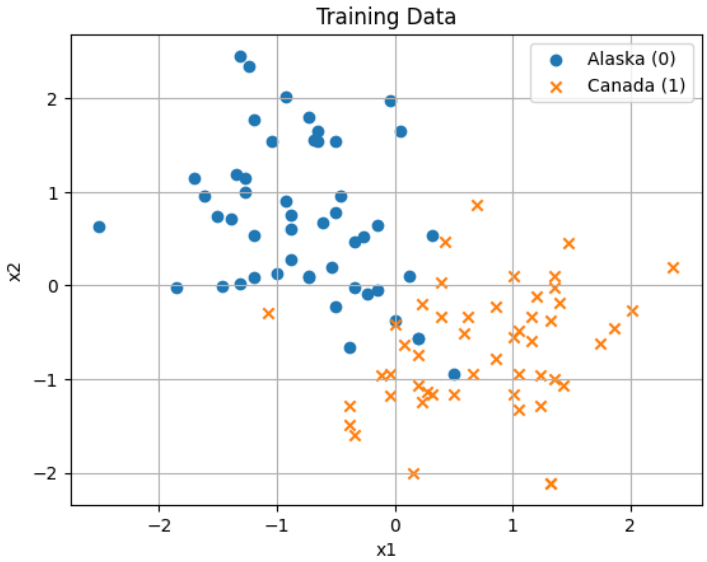

Figure 8: Training data: x_1 vs x_2 (normalized). Blue circles: Alaska (0); red crosses: Canada (1).

4.4 (3) Linear decision boundary when $\Sigma_0 = \Sigma_1$

Derivation. The log-posterior ratio (decision rule) is:

$$\log \frac{p(y=1 \mid x)}{p(y=0 \mid x)} = \log \frac{\phi}{1-\phi} + \frac{1}{2}\big((x-\mu_0)^\top \Sigma^{-1}(x-\mu_0) - (x-\mu_1)^\top \Sigma^{-1}(x-\mu_1)\big).$$

When you expand the quadratic terms and cancel identical $x^\top \Sigma^{-1} x$, the boundary becomes linear:

$$\underbrace{(\mu_1-\mu_0)^\top \Sigma^{-1}}_{\theta^\top} x + c = 0,$$

where

$$\theta = \Sigma^{-1}(\mu_1-\mu_0), \qquad c = \log \frac{\phi}{1-\phi} - \tfrac{1}{2}\big(\mu_1^\top \Sigma^{-1}\mu_1 - \mu_0^\top \Sigma^{-1}\mu_0\big).$$

Thus the decision boundary is a straight line in x.

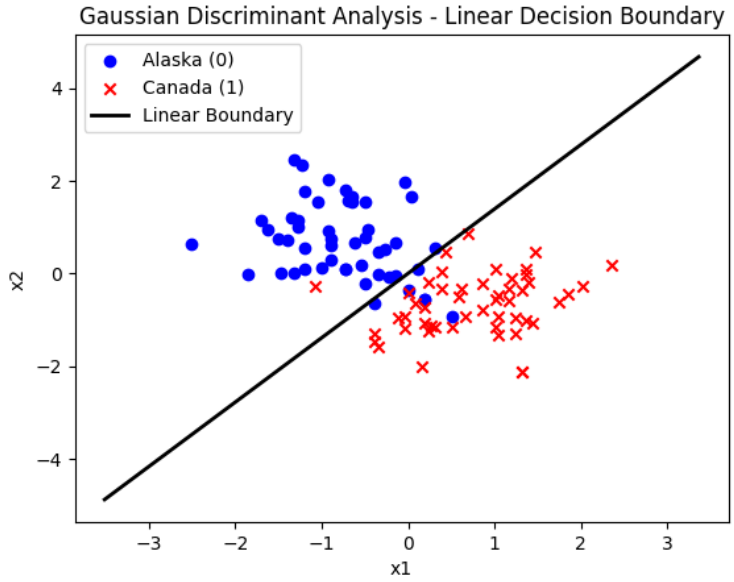

Figure 9: Training data with linear GDA decision boundary (shared covariance).

Plot with linear boundary.

4.5 (4) General GDA: separate covariances Σ_0, Σ_1

Estimators. Now allow each class its own covariance. The MLEs are:

$$\hat{\mu}_k = \frac{1}{m_k} \sum_{i:y^{(i)}=k} x^{(i)}, \qquad \hat{\Sigma}_k = \frac{1}{m_k} \sum_{i:y^{(i)}=k} (x^{(i)} - \hat{\mu}_k)(x^{(i)} - \hat{\mu}_k)^\top,$$

where $m_k = \sum_i 1\{y^{(i)} = k\}$.

$$\hat{\mu}_0 = \begin{bmatrix} -0.7552 \\ 0.6850 \end{bmatrix}, \quad \hat{\mu}_1 = \begin{bmatrix} 0.7552 \\ 0.6850 \end{bmatrix}$$
$$\hat{\Sigma}_0 = \begin{bmatrix} 0.4295 & -0.0224 \\ -0.0224 & 0.5306 \end{bmatrix}, \quad \hat{\Sigma}_1 = \begin{bmatrix} 0.4774 & 0.1099 \\ 0.1099 & 0.4135 \end{bmatrix}$$

Table 5: Estimated means and class-conditional covariances (replace entries with your numeric results).

Reported estimates.

4.6 (5) Quadratic decision boundary

Derivation. With distinct covariances, equate log class-conditional densities plus priors:

$$\log \frac{\phi}{1-\phi} + \log \mathcal{N}(x; \mu_1, \Sigma_1) - \log \mathcal{N}(x; \mu_0, \Sigma_0) = 0.$$

Expanding yields a quadratic form in x:

$$x^\top Ax + b^\top x + d = 0,$$

where

$$A = \tfrac{1}{2}(\Sigma_0^{-1} - \Sigma_1^{-1}), \qquad b = \Sigma_1^{-1}\mu_1 - \Sigma_0^{-1}\mu_0,$$

and

$$d = \tfrac{1}{2}(\mu_0^\top \Sigma_0^{-1}\mu_0 - \mu_1^\top \Sigma_1^{-1}\mu_1) + \frac{1}{2}\log\frac{|\Sigma_0|}{|\Sigma_1|} + \log\frac{1-\phi}{\phi}.$$

Thus the separating curve is quadratic (conic section) and can be plotted by evaluating the left-hand side on a grid and contouring the zero level.

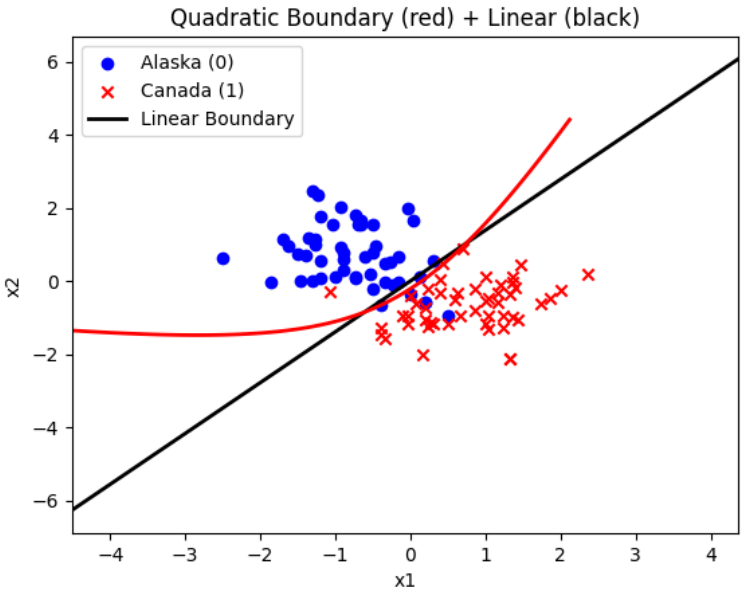

Figure 10: Training data with linear boundary (black) and quadratic boundary (red) obtained from class-specific covariances.

Plot with quadratic boundary.

4.7 (6) Analysis and observations

- **Linear vs quadratic:** If $\Sigma_0 \approx \Sigma_1$ the linear GDA boundary is sufficient. When the class covariances differ significantly the quadratic boundary can better separate classes (it adapts to different spreads and orientations).
- **Interpretation of covariances:** The eigenvectors of Σ_k show principal directions of spread for class k; the eigenvalues express variance along those axes. If one class is elongated in a direction where the other is compact, quadratic boundary bends accordingly.
- **Numerical stability:** Normalization of features improves conditioning of covariance estimates and stabilizes matrix inversions. If Σ_k is near singular, regularize by adding λI.

- **Empirical fit:** (Insert your short empirical comment here — e.g., "The quadratic boundary slightly improves class separation near the upper-left region; however the linear model already achieves reasonably low misclassification error.")