COL 774: Machine Learning
Assignment 1

Name: Suraj Reddy
Entry Number: 2025AIY7586

Problem 1: (20 points) Linear Regression

1. Given: A dataset with 1000 samples with 1 feature to predict density of wine based on its acidity. Least squares linear regression model is to be implemented to predict the density of wine. Loss function is given as:

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

where $h_\theta(x) = \theta_0 + \theta_1 x$.

(a) Implement batch gradient descent method for optimizing $J(\theta)$.

$$\theta_j := \theta_j - \alpha\frac{\partial J(\theta)}{\partial \theta_j} \tag{2}$$

where α is the learning rate.
Batch gradient descent updates θ_j after computing the gradients using all training examples.

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} \tag{3}$$

Final update rule:

$$\theta_j := \theta_j + \frac{\alpha}{m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} \tag{4}$$

(b) **Theoretical Learning Rate Using Lipschitz Constant:**
The gradient of the loss function is Lipschitz continuous with Lipschitz constant L if:

$$||\nabla J(\theta_1) - \nabla J(\theta_2)|| \leq L||\theta_1 - \theta_2||, \forall \theta_1, \theta_2 \tag{5}$$

For linear regression, the Lipschitz constant can be computed as:

$$L = \frac{1}{m}\lambda_{max}(X^T X) \tag{6}$$

where λ_{max} is the maximum eigenvalue of $X^T X$ and X is the design matrix.
The theoretical learning rate α can be chosen as:

$$\alpha = \frac{1}{L} \tag{7}$$

For the given dataset L is computed to be 1.00066194, hence the theoretical learning rate α is 0.9993383.
Stopping Criteria(as a function of the change in the value of $J(\theta)$):

The stopping criteria for the gradient descent algorithm can be defined based on the change in the value of the loss function $J(\theta)$ between successive iterations. That is to stop the iterations when the absolute change in $J(\theta)$ is less than a predefined threshold ϵ. Mathematically, this can be expressed as:

$$|J(\theta^{(k)}) - J(\theta^{(k-1)})| < \epsilon \tag{8}$$

where k is the iteration number and ϵ is a small positive constant. Here ϵ is set to $1e-6$.

(c) **Results:**

1. Initial parameters are set to 0. Epsilon value ϵ set to $1e-6$. For the theoretical learning rate of 0.9993383, the algorithm converged in 4 iterations. The final parameters obtained are $\theta_0 = 6.2186$ and $\theta_1 = 29.0647$. The final value of the loss function $J(\theta)$ is 0.0097.

```
Iteration 0: Parameters=[0, 0], LMS Loss=882.7602972896897
Iteration 1: Parameters=[6.247608812335031, 29.015027967716836], LMS Loss=0.013063255688556036
Iteration 2: Parameters=[6.218751953526112, 29.064627052198087], LMS Loss=0.009759669783485005
Iteration 3: Parameters=[6.218676471278382, 29.064756790864617], LMS Loss=0.009759647179909422
```

Figure 1: Batch Gradient Descent with Theoretical Learning Rate using Lipschitz Constant

2. Initial parameters are set to 0. Epsilon value ϵ set to $1e-6$. For the learning rate of 0.01, the algorithm converged in 833 iterations. The final parameters obtained are $\theta_0 = 6.2173$ and $\theta_1 = 29.0579$. The final value of the loss function $J(\theta)$ is 0.0098. The figure 2 shows the training results.

2. Plot: Plot showing the linear regression fit for learning rate 0.01. Data points are shown in blue and the regression line is shown in orange. The figure 3 shows the training results.

```
Iteration 100: parameters=[3.954672308479635, 18.41481650808863], loss=120.8680826088942
Iteration 200: parameters=[5.394459913870412, 25.16238788039263], loss=16.227684597175738
Iteration 300: parameters=[5.9186288473462465, 27.634842134106908], loss=2.1860508925309485
Iteration 400: parameters=[6.109451030421624, 28.540803606199944], loss=0.3017992492738601
Iteration 500: parameters=[6.178916822310882, 28.872768285906353], loss=0.04894912563540712
Iteration 600: parameters=[6.204203832272808, 28.994407810064494], loss=0.015018611455225045
Iteration 700: parameters=[6.213408500347885, 29.038979410773344], loss=0.010465369668159244
Iteration 800: parameters=[6.216758947338596, 29.05531152594266], loss=0.009854351696111178
Iteration 833: parameters=[6.217302721422789, 29.057975329517728], loss=0.009808458940084898
```

Figure 2: Batch Gradient Descent with Learning Rate 0.01

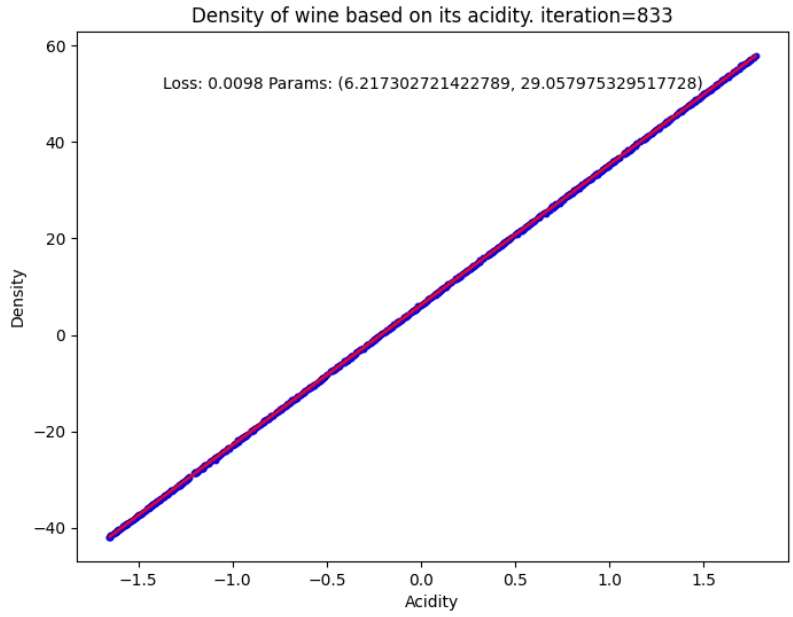


Figure 3: Linear Regression Fit for Learning Rate 0.01.

3. 3D Plot: 3-dimensional mesh showing the error function $J(\theta)$ on z axis and the parameters in the xy plane.

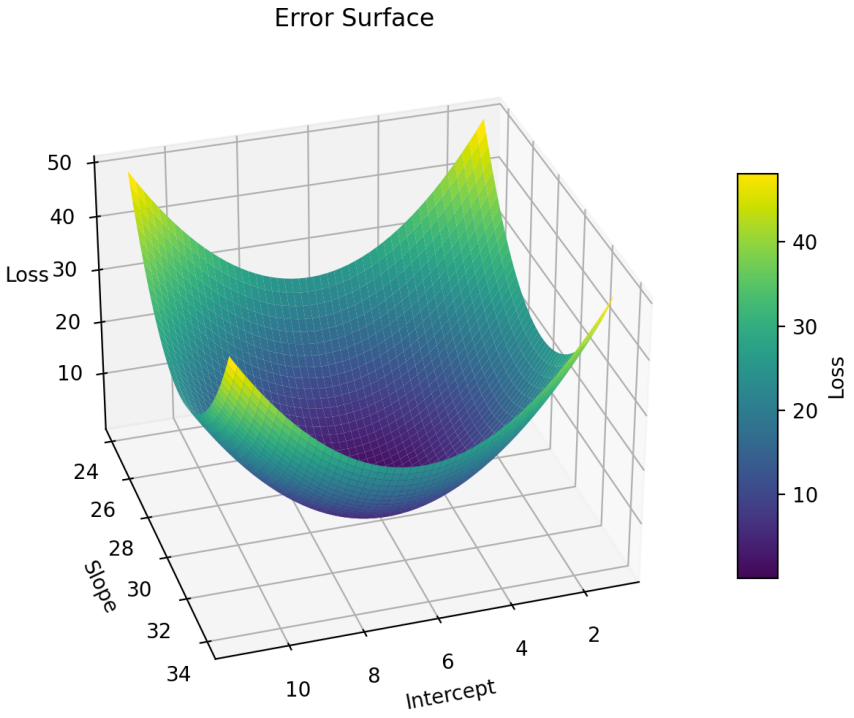


Figure 4: 3D Plot of Error Function $J(\theta)$

4. Contours: Contour plot showing the error function $J(\theta)$ with the path taken by gradient descent for learning rate 0.01 shown in orange.

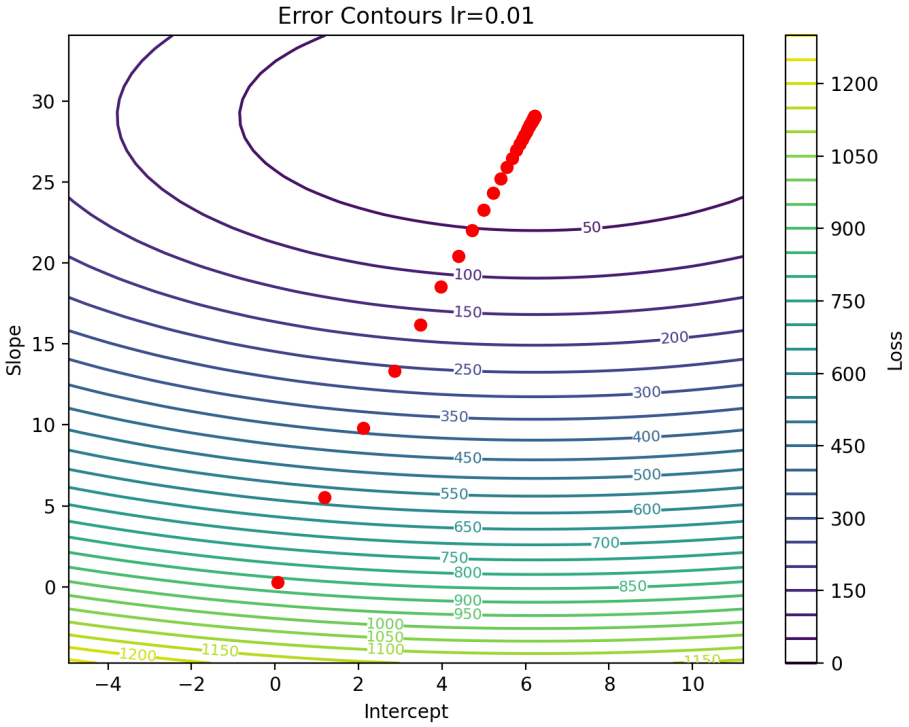


Figure 5: Contour Plot of Error Function $J(\theta)$ with Gradient Descent Path

5. Contours with different learning rates: Contour plots showing the error function $J(\theta)$ with the path taken by gradient descent for different learning rates.
Observations:

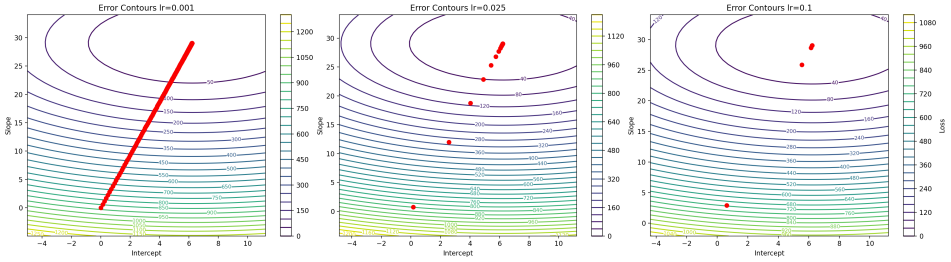


Figure 6: Contour Plot for Learning Rate 0.001

Figure 7: Contour Plot for Learning Rate 0.025

Figure 8: Contour Plot for Learning Rate 0.1

- The theoretical learning rate computed using the Lipschitz constant resulted in very fast convergence.
- A smaller learning rate of 0.01 required significantly more iterations to converge.
- The contour plots illustrate how different learning rates affect the path taken by gradient descent. A very small learning rate (0.001) results in slow progress towards the minimum, while a moderate learning rate (0.025) shows a more direct path.
- A larger learning rate (0.1) can cause oscillations around the minimum, indicating that it may be too high for stable convergence.
- Overall, the choice of learning rate is crucial for the efficiency and stability of the gradient descent algorithm.

Problem 2: (24 points) Sampling, Closed Form and Stochastic Gradient Descent

1. Data Sampling: Sample 1 million data points. True parameters are θ=[3,1,2]. x1 N(3,4), x2 N(-1,4), noise ϵ N(0,2). The data is generated according to the following equation:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \epsilon \tag{9}$$

80% of the data (800,000 data points) as train set, and the remaning 20% of the data (200,000 data points) as the test set.
2. (a) Stochastic Gradient Descent: Implement stochastic gradient descent to learn the parameters θ. Initial parameters are set to 0. Epsilon value ϵ set to $1e-6$. Learning rate is set to 0.001. The algorithm is run for 2 epochs. With batch size of 1. The loss function is the mean squared error. Final parameters

```
Epoch 3/5000, Batch 800000/800000, parameters=[2.9809382  0.98786295 2.01588326], loss=0.05673097435549429
Convergence reached.
Epoch 3/5000 completed. Parameters: [2.9809382  0.98786295 2.01588326], Loss: 1.0094496993044706
Final parameters: [2.9809382  0.98786295 2.01588326], Final Loss: 1.0094496993044706
Total execution time: 89.4594 seconds
```

Figure 9: Results of Stochastic Gradient Descent

learned are θ=[3.0556,1.0295,2.0285].
2. (b) Mini-Batch Gradient Descent: Implementing gradient descent for different batch sizes. Initial parameters are set to 0. Epsilon value ϵ set to $1e-6$. Learning rate is set to 0.001. The Batch sizes r = [1,80,8000,800000].

- **For batch size of 1:** Number of batches = 800000.
  Convergence criteria is change in loss with threshold ϵ=$1e-6$.
  Final parameters learned are θ=[2.9809,0.9876,2.0158].

```
Epoch 3/5000, Batch 800000/800000, parameters=[2.9809382  0.98786295 2.01588326], loss=0.05673097435549429
Convergence reached.
Epoch 3/5000 completed. Parameters: [2.9809382  0.98786295 2.01588326], Loss: 1.0094496993044706
Final parameters: [2.9809382  0.98786295 2.01588326], Final Loss: 1.0094496993044706
Total execution time: 89.4594 seconds
```

Figure 10: Learned parameters,time taken ,final loss and number of epochs taken for batch size of 1.

- **For batch size of 80:** Number of batches = 10000.
  Convergence criteria is change in loss with threshold ϵ=$1e-6$.
  Final parameters learned are θ=[2.9982,0.9963,2.0029].

- **For batch size of 8000:** Number of batches = 100.
  Convergence criteria is change in loss with threshold ϵ=$1e-6$.
  Final parameters learned are θ=[2.9906,1.0018,2.0006].

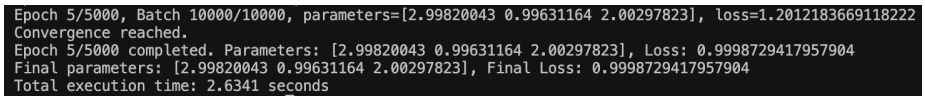


Figure 11: Learned parameters,time taken ,final loss and number of epochs taken for batch size of 80.

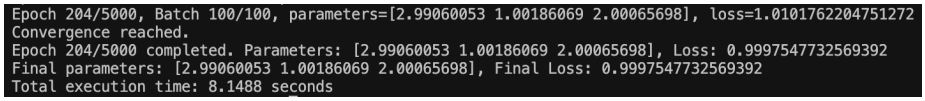


Figure 12: Learned parameters,time taken ,final loss and number of epochs taken for batch size of 8000.

- **For batch size of 800000:** Number of batches = 1.
  Convergence criteria is change in loss with threshold ϵ=$1e-6$.
  Final parameters learned are θ=[2.8882, 1.0245, 1.9929].

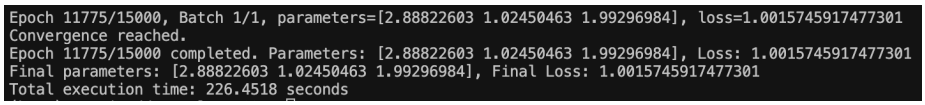


Figure 13: Learned parameters,time taken ,final loss and number of epochs taken for batch size of 800000.

3. (a) Convergence for different batch sizes: Algorithm with different batch sizes converge to different parameter values. The time taken for convergence varies with batch size. The results are summarized in Table 1.

Batch Size	Epochs	Time Taken (s)
1	3	89.4594
80	5	2.6314
8000	204	8.1488
800000	11775	226.4518

Table 1: Convergence comparison for different batch sizes: Number of epochs and time taken.

3. (b) Closed Form Solution: The closed form solution for linear regression is given by:

$$\theta = (X^T X)^{-1} X^T y \tag{10}$$

Where X is the feature matrix and y is the target vector. Number of data points = 800000, Number of features = 3. The labels are generated using the true parameters θ=[3,1,2].

- When noise is not added to the labels:

$$y = 3 * x_0 + 1 * x_1 + 2 * x_2 \tag{11}$$

The learned parameters are θ=[3.0000,1.0000,2.0000].

- When noise is added to the labels:

$$y = 3 * x_0 + 1 * x_1 + 2 * x_2 + \epsilon \tag{12}$$

The learned parameters are θ=[3.00344547 0.99932589 2.00115889].

3. (c) Comparison of Parameters: The true parameters are θ=[3,1,2]. The parameters learned using closed form solution and stochastic gradient descent are very close to the true parameters.

- True Parameters: [3, 1, 2]
- Closed Form Solution: [3.0000, 1.0000, 2.0000]
- Stochastic Gradient Descent: [2.9809, 0.9876, 2.0158]

4. Mean Squared Error on Test Set: The Mean Squared Error (MSE) is calculated using the formula:

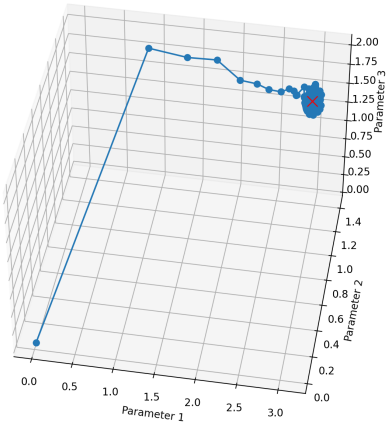
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{13}$$

Where y_i is the true value and $\hat{y}_i$ is the predicted value. The test set contains 200000 data points. The train set contains 800000 data points. The batch size of 8000 gives the lowest MSE on the train set. The parameters learned using batch size of 8000 are θ=[2.9906,1.0018,2.0006] with a training loss of 0.9997 till convergence. The MSE on the test set using these parameters is Test Loss: 0.4994.

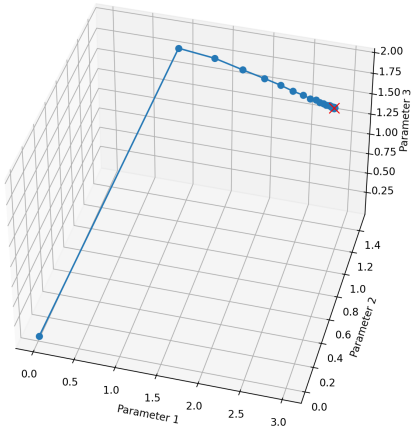
5. Trajectory of Parameters: The trajectory of parameters in 3D space for different batch sizes.

Observation:

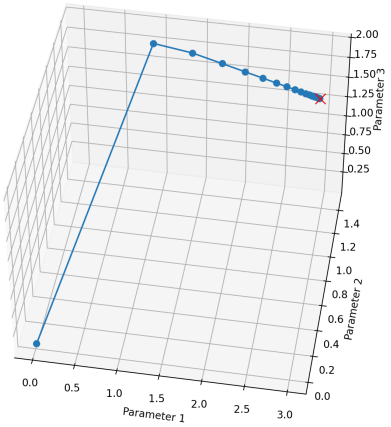
The movement of parameters becomes smoother as the batch size increases. This is because larger batch sizes provide a better estimate of the gradient, reducing the noise in the updates. Smaller batch sizes lead to more erratic movements due to the high variance in the gradient estimates. This behavior is intuitive as larger batches average out the noise, leading to more stable and consistent updates towards convergence.



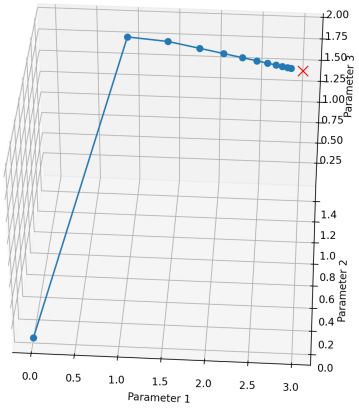
(a) Batch size = 1.



(b) Batch size = 80.



(c) Batch size = 8000.



(d) Batch size = 800000.

Figure 14: Trajectory of parameters for different batch sizes.

Problem 3: (15 points) Logistic Regression

Newton's Method for Logistic Regression

We want to maximize the log-likelihood function:

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

where

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}.$$

The gradient of the log-likelihood is:

$$\nabla_\theta L(\theta) = X^T\big(y - h_\theta\big),$$

The Hessian (matrix of second derivatives) is:

$$H = -X^T R X,$$

where

$$R = \operatorname{diag}\Big(h_\theta(x^{(i)})\big(1 - h_\theta(x^{(i)})\big)\Big)_{i=1}^{m}$$

Newton's Method Update Rule Newton's method updates the parameter vector as:

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1}\nabla_\theta L(\theta^{(t)}).$$

Substituting the expressions for gradient and Hessian:

$$\theta^{(t+1)} = \theta^{(t)} + \big(X^T R X\big)^{-1} X^T\big(y - h_\theta\big).$$

Training results:
Parameters learned = [0.46722676 2.55770122 -2.78143761] plot shown in image 15

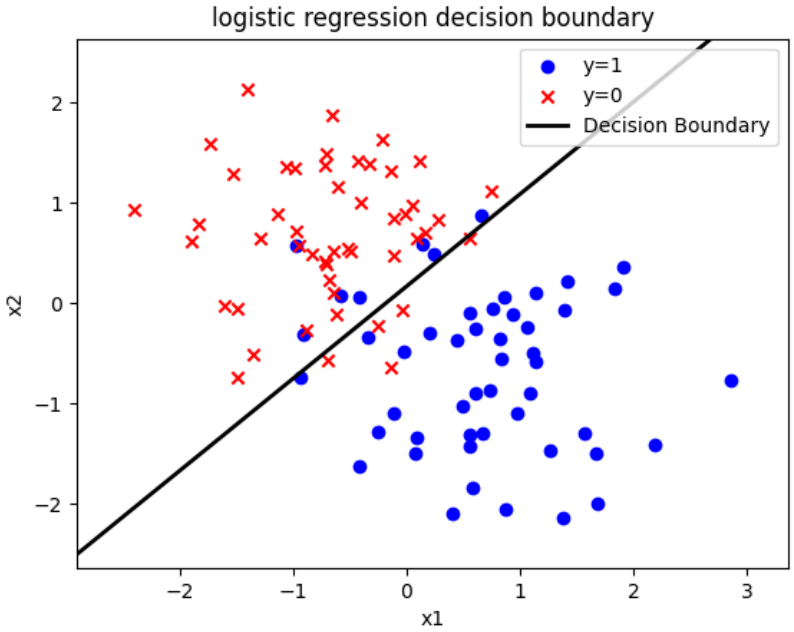


Figure 15: Results of Newtons method for logistic regression

(25 points) Gaussian Discriminant Analysis

1. GDA using the closed form equations

Assumption: both classes have same covarience matrix.
results:

$$\text{Alaska mean: } \mu_{\text{Alaska}} = \begin{bmatrix} -0.7553 \\ 0.6851 \end{bmatrix}$$

$$\text{Canada mean: } \mu_{\text{Canada}} = \begin{bmatrix} 0.7553 \\ -0.6851 \end{bmatrix}$$

$$\text{Shared covariance matrix: } \Sigma = \begin{bmatrix} 0.4295 & -0.0225 \\ -0.0225 & 0.5306 \end{bmatrix}$$

2. Plotting the training data corresponding to the two coordinates of the input features

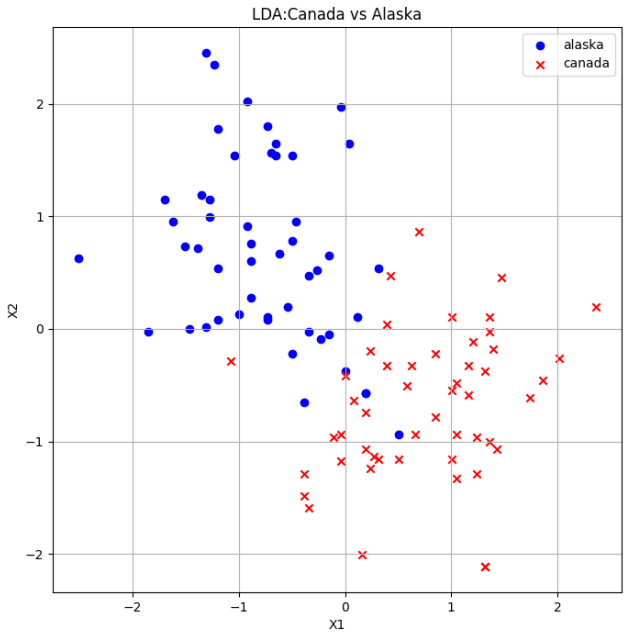


Figure 16: Plotting the training data

3. GDA boundary equations

$$\theta = \Sigma^{-1}(\mu_1 - \mu_0), \qquad \theta_0 = -\tfrac{1}{2}\mu_1^\top \Sigma^{-1}\mu_1 + \tfrac{1}{2}\mu_0^\top \Sigma^{-1}\mu_0 + \log\frac{\phi}{1-\phi},$$

$$\text{Decision boundary:} \quad \theta^\top x + \theta_0 = 0.$$

$$\text{In 2D:} \quad x_2 = -\frac{\theta_1}{\theta_2}x_1 - \frac{\theta_0}{\theta_2}.$$

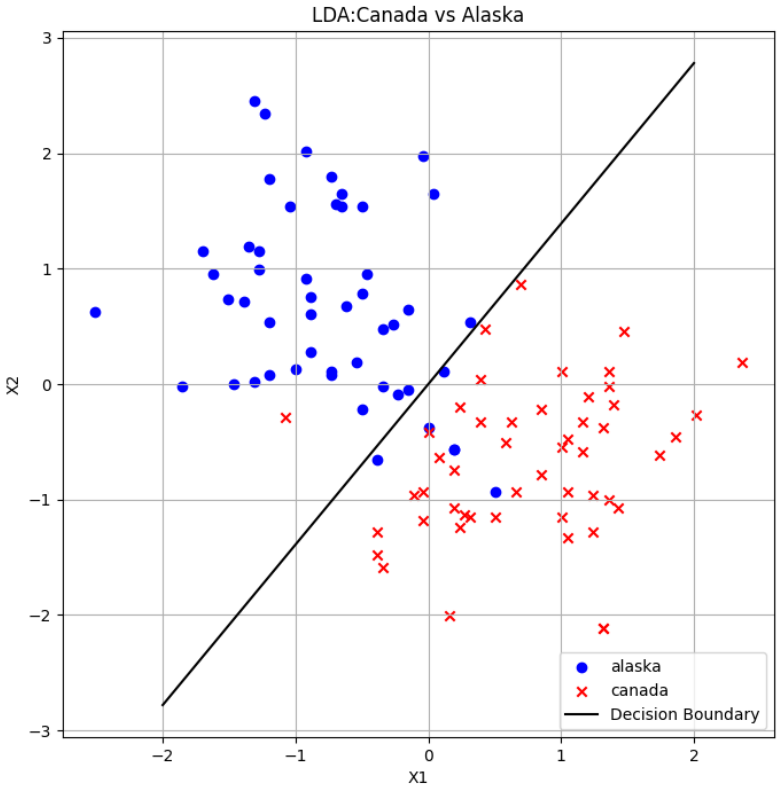


Figure 17: Results of Newtons method for logistic regression with linear decision boundary

4. GDA for classes with own covariance matrix

Results:

$$\text{Learned parameters: } \theta = \begin{bmatrix} -3.38925452 \\ 2.43858399 \end{bmatrix}$$

$$\text{Canada covariance matrix: } \Sigma_{\text{Canada}} = \begin{bmatrix} 0.47747117 & 0.1099206 \\ 0.1099206 & 0.41355441 \end{bmatrix}$$

$$\text{Alaska covariance matrix: } \Sigma_{\text{Alaska}} = \begin{bmatrix} 0.38158978 & -0.15486516 \\ -0.15486516 & 0.64773717 \end{bmatrix}$$

5. Quadratic boundary

The equation for the quadratic boundary separating the two regions in terms of the parameters are as follows:

$$\text{Boundary:} \quad \log \frac{p(x \mid y=1)\,\phi}{p(x \mid y=0)\,(1-\phi)} = 0,$$

where $p(x \mid y=k) = \mathcal{N}(x; \mu_k, \Sigma_k)$. Equivalently,

$$(x-\mu_1)^\top \Sigma_1^{-1}(x-\mu_1) - (x-\mu_0)^\top \Sigma_0^{-1}(x-\mu_0) + \log \frac{|\Sigma_1|}{|\Sigma_0|} - 2\log \frac{\phi}{1-\phi} = 0.$$

$$\text{Decision boundary (GDA, } \Sigma_0 \neq \Sigma_1\text{):} \qquad x^\top A x + b^\top x + c = 0,$$

where

$$A = \Sigma_1^{-1} - \Sigma_0^{-1}, \qquad b = -2\big(\Sigma_1^{-1}\mu_1 - \Sigma_0^{-1}\mu_0\big),$$
$$c = \mu_1^\top \Sigma_1^{-1}\mu_1 - \mu_0^\top \Sigma_0^{-1}\mu_0 + \log \frac{|\Sigma_1|}{|\Sigma_0|} - 2\log \frac{\phi}{1-\phi}.$$

Explanations:

- A is the quadratic coefficient matrix. If $A \neq 0$ the boundary is genuinely *quadratic* (conic section).
- b is the linear coefficient vector; it depends on the difference of precision-weighted means $\Sigma_k^{-1}\mu_k$.
- c is the constant term. The term $\log \frac{|\Sigma_1|}{|\Sigma_0|}$ accounts for the different volume/shape of the two Gaussians, while $-2\log \frac{\phi}{1-\phi}$ shifts the boundary according to the class prior $\phi = \Pr(y=1)$.

- In the *special case* $\Sigma_0 = \Sigma_1 = \Sigma$ we have $A = 0$, the quadratic term cancels, and the boundary reduces to the linear form

$$(\Sigma^{-1}(\mu_1 - \mu_0))^\top x + \theta_0 = 0,$$

  i.e. the familiar linear GDA separator.

- To plot the boundary in 2D, evaluate the scalar function

$$f(x) = x^\top A x + b^\top x + c$$

  on a grid and draw the contour $f(x) = 0$ (or solve the quadratic equation for x_2 when possible).

$$x^\top(\Sigma_1^{-1} - \Sigma_0^{-1})x \; - \; 2(\Sigma_1^{-1}\mu_1 - \Sigma_0^{-1}\mu_0)^\top x \; + \; \mu_1^\top\Sigma_1^{-1}\mu_1 - \mu_0^\top\Sigma_0^{-1}\mu_0 + \log\frac{|\Sigma_1|}{|\Sigma_0|} - 2\log\frac{\phi}{1-\phi} = 0$$

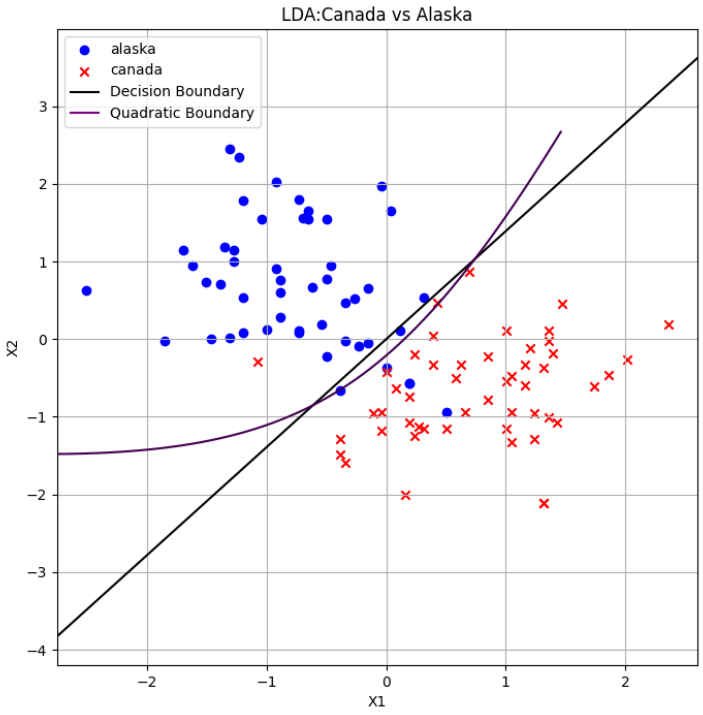


Figure 18: Results of Newtons method for logistic regression for linear and quadratic decision boundaries.

Analysis of the linear as well as the quadratic boundaries obtained by GDA

Linear Boundary (LDA)

- Forms a straight hyperplane that bisects the space between class means.
- Assumes both classes share the same covariance structure (homoscedastic).
- The boundary is perpendicular to the line connecting the two class means.
- Mathematically:

$$w^\top (x - \text{midpoint}) = 0, \qquad w = \Sigma^{-1}(\mu_1 - \mu_2).$$

Quadratic Boundary (QDA)

- Forms a curved boundary (ellipse, parabola, or hyperbola).
- Accounts for different covariance matrices per class (heteroscedastic).
- Can adapt to classes with different shapes, orientations, and spreads.
- Mathematically:

$$(x - \mu_1)^\top \Sigma_1^{-1}(x - \mu_1) - (x - \mu_2)^\top \Sigma_2^{-1}(x - \mu_2) + \log \frac{|\Sigma_1|}{|\Sigma_2|} = 0.$$

Linear Boundary (LDA) is preferred when:

- Classes have similar covariance structures.
- Limited training data (fewer parameters to estimate).
- Classes are roughly spherical with similar spreads.
- A simple, interpretable model is desired.

Quadratic Boundary (QDA) is preferred when:

- Classes have significantly different covariance matrices.
- One class is more spread out than the other.
- Classes have different orientations in feature space.
- Sufficient training data is available.

The choice between linear and quadratic boundaries fundamentally depends on whether the equal covariance assumption of LDA holds for your specific dataset. The quadratic method is more general but requires more data to estimate reliably.