COL774: Machine Learning
Assignment 1

Ankur Kumar (2025AIB2557)
Indian Institute of Technology Delhi

1. Linear Regression

1.1. Batch Gradient Descent

Refer code for the implementation of batch gradient descent. The following learning rate and stopping criterion were choosen :-

- **Learning rate:** We know that the mean squared loss (MSE) used in linear regression is $1/n||X^TX||_2$ Lipschitz smooth, where n is the number of data points and $||.||_2$ is the spectral norm. Also the X is the data matrix which in our case is a single dimensional vector, so the spectral norm is just the dot product of input data with itself. Now for convergence $\eta \leq 1/L$, where η is the learning rate. Based on this we used $\eta = 0.5$.

- **Stopping criterion:** For stopping we took the difference between the maximum and minimum of last 10 values of the loss function J, and stopped the iteration when it was lower than a tolerance threshold $\epsilon = 10^{-9}$. We know that for batch gradient descent the loss function is non increasing so we are basically comparing the loss fucntion values which are 10 iterations apart. Mathematically this can be expressed like this:

$$\max_{k-10 \leq i \leq k} J^{(i)} - \min_{k-10 \leq i \leq k} J^{(i)} < \epsilon$$

This criterion was satisfied after 28 iterations, which means we reached convergence (as per our criterion) only after 28 iterations. Figure 1 shows how loss function changes with iterations.

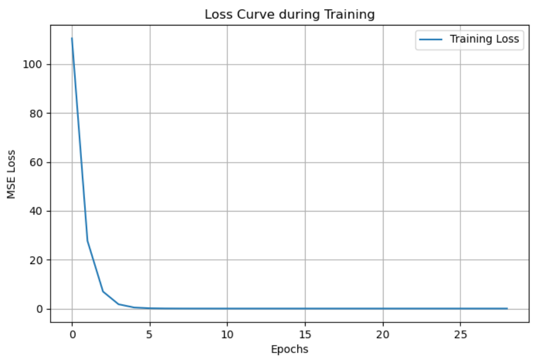


Figure 1: Training loss over the iterations.

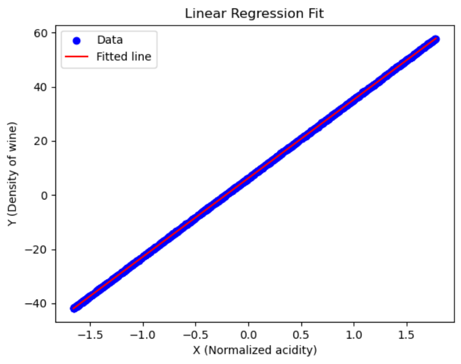


Figure 2: Training loss over the iterations.

The final parameters obtained are as follows:
w (the slope - the coefficient of x) : **29.064**
b (the y-intercepy - the constant) : **6.218**

1.2. Data & Hypothesis function

Figure 2 shows the Data (normalized x - acidity with y - wine density) along with the hypothesis function learned by linear regression. We can clearly see that data follows a linear trend and our hypothesis function perfectly captures it.

1.3. 3D plot for error function

Figure 3 presents different views of the loss function plotted over varying parameter values (different w & b). The plots are centered around the optimal parameters (w, b) obtained using gradient descent. The surface clearly resembles a paraboloid, illustrating that the loss function is convex.

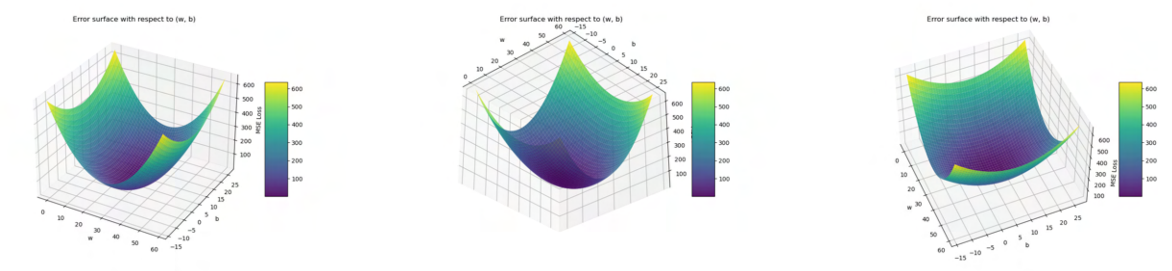


Figure 3: Loss function 3D surface views.

To visualize how gradient descent updates the parameter values, we plotted its successive iterations on the 3D loss function surface. The animation can be viewed by running the code, or by clicking here to see the GIF. A smaller learning rate, $\eta = 0.01$, was used to provide a clearer visualization of the gradient descent updates. See a static view in Figure 4

1.4. Error function contours

We visualized the error function by plotting its contours and overlaying the successive iterations of gradient descent. The animation can be viewed by running the code, or by clicking here to see the GIF. Here we again used a smaller learning rate, $\eta = 0.01$. See a static view in Figure 5

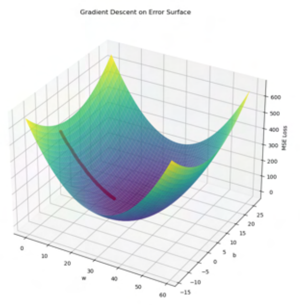


Figure 4: Gradient Descent on 3D loss surface.

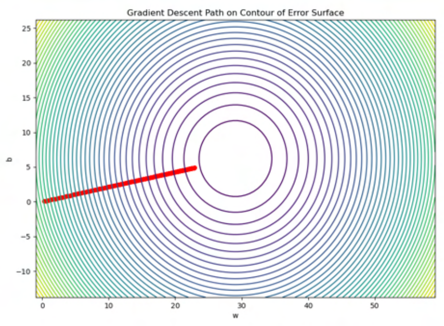


Figure 5: Gradient Descent on 2D contour.

1.5. Gradient descent with different learning rate values

We implemented gradient descent with different learning rate values (η). The animation illustrating how gradient descent updates the parameter values can be viewed by running the code, or by clicking here to see the GIF. Note that the GIF does not show convergence for all values of η. A static view can be seen in Figure 6

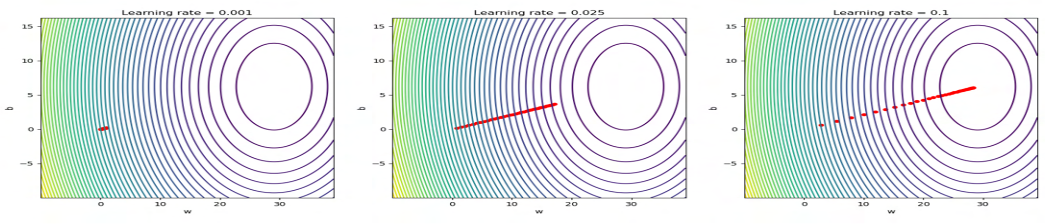


Figure 6: Gradient descent with different η.

From the plots, we observe that gradient descent converges more slowly when the learning rate is smaller. This is intuitive, since a smaller learning rate results in smaller updates on the loss surface. In all the cases considered, i.e., $\eta \in 0.001, 0.025, 0.1$, convergence is guaranteed because batch gradient descent converges for all $\eta \leq 0.5$, given that the Lipschitz constant L is approximately 0.5.

2. Sampling, Closed Form and Stochastic Gradient Descent

2.1. Sampling for data preparation

See code for implementation.

2.2. Stochastic gradient descent

See code for implementation.
Refer Table 1 to see values of parameters obtained by stochastic gradient descent with different batch size.
Stopping criterion: A good stopping criterion is crucial for gradient-based optimization. In batch gradient descent, the loss function is guaranteed to be non-increasing, which allows us to simply compare the last two loss values and stop the iterations once their difference falls below a tolerance threshold ϵ.

However, this approach does not work for stochastic gradient descent (SGD), since the loss is only estimated at each step and is not guaranteed to decrease monotonically. To handle this, we instead consider a moving window of size k containing the most recent loss values. We compute the average of the first half of the window (from 0 to $k/2$) and the average of the second half (from $k/2$ to k). The iterations are stopped when the difference between these two averages falls below a predefined tolerance threshold ϵ.
We use slightly different values of k for different batch sizes. In particular, we choose k to be inversely proportional to the batch size, since for smaller batch sizes the loss function is noisier and we require a larger window to smooth out the fluctuations, whereas for larger batch sizes the loss estimates are more stable and a smaller window is sufficient for detecting convergence. To see the exact k and ϵ used see Table 1.

Batch Size	**Stopping criterion**	$\boldsymbol{\theta_0}$	$\boldsymbol{\theta_1}$	$\boldsymbol{\theta_2}$	**Iterations**
1	$k = 200000,\ \epsilon = 10^{-6}$	3.028	1.020	2.004	235612
80	$k = 100000,\ \epsilon = 10^{-6}$	3.000	0.997	1.996	118599
8000	$k = 100,\ \epsilon = 10^{-6}$	1.943	1.229	1.923	3566
800000	$k = 2,\ \epsilon = 10^{-5}$	2.640	1.077	1.973	7537

Table 1: Stopping criteria and results for different batch sizes in SGD.

2.3. Convergence Speed and closed form solution

The *Iterations* column in Table 1 reports the number of times the loss function was evaluated in each run of SGD. Although smaller batch sizes require more iterations, each individual loss computation is faster and less expensive. In contrast, SGD with a batch size of 800,000 (which corresponds to batch gradient descent) takes the longest time to reach the convergence condition defined for it. It is also important to note that the total number of iterations is strongly influenced by the stopping criterion: for larger values of ϵ, batch gradient descent may converge in fewer iterations.
See Table 2 to see a quantitative comparison of parameters learned by different techniques.

Method / Batch Size	$\boldsymbol{\theta_0}$	$\boldsymbol{\theta_1}$	$\boldsymbol{\theta_2}$
True Parameters	3.000	1.000	2.000
Closed-form Solution	2.999	0.999	2.000
SGD (Batch = 1)	3.028	1.020	2.004
SGD (Batch = 80)	3.000	0.997	1.996
SGD (Batch = 8000)	1.943	1.229	1.923
SGD (Batch = 800000)	2.640	1.077	1.973

Table 2: Comparison of parameter estimates from SGD, closed-form solution, and true parameters.

Method	Batch Size	Training Loss	Testing Loss
SGD	1	1.0039	1.0045
SGD	80	0.9991	1.0002
SGD	8000	1.1582	1.1616
SGD	800000	1.0176	1.0193
Closed-form (Normal Eq.)	–	0.9992	1.0002

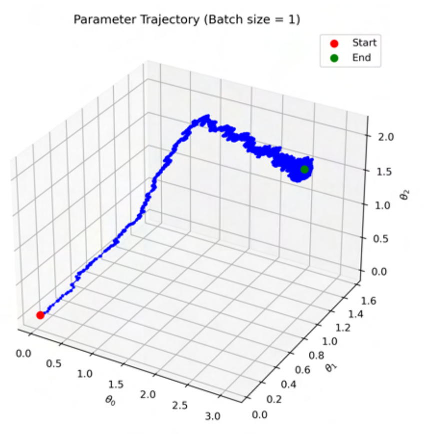
Table 3: Comparison of training and testing errors for different techniques

2.4. Train and Test Error

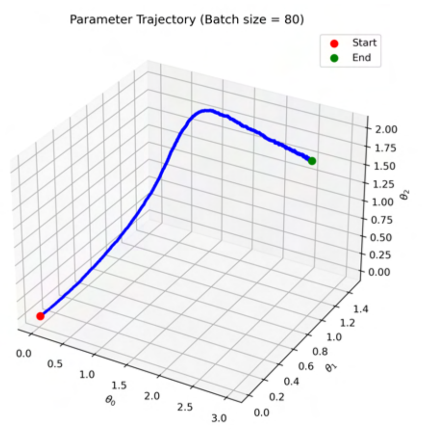
See Table 3 to see train and test error of different techniques used to evaluate the parameters.

We observe that the training and testing errors are nearly identical across all cases. This is expected since both the training and testing datasets were sampled from the same underlying distribution. However, the testing error is consistently (though only slightly) higher than the training error. This is because the optimization process minimizes the loss function with respect to the training data, causing the learned parameters to fit the training set marginally better and resulting in a lower training loss compared to the testing loss.

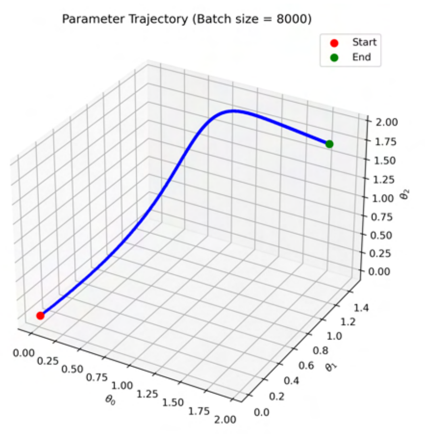
2.5. Parameter Plotting in 3D



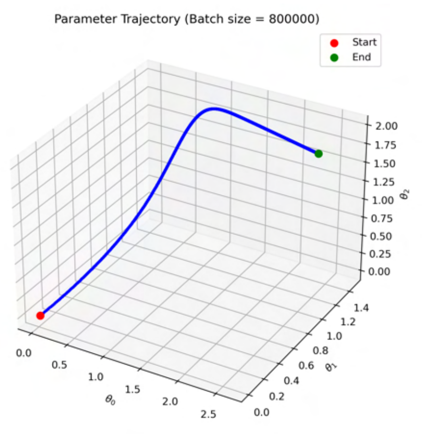
(a) SGD with batch size 1



(b) SGD with batch size 80



(c) SGD with batch size 8000



(d) Batch Gradient Descent

Figure 7: Parameter movement plotted in 3D

The plots align with our intuition: SGD with batch size 1 exhibits the most stochastic behavior in the parameter updates. As shown in Figure 7 (a), the trajectory fluctuates significantly, especially near the optimum. In contrast, the trajectories in the subsequent plots become progressively smoother and more stable as the batch size increases.

3. Logistic Regression

3.1. Fitting Logistic Regression to data

See the code for implementation. Figure 8 illustrates the evolution of the parameters across iterations, demonstrating how Newton's method converges to the optimum.

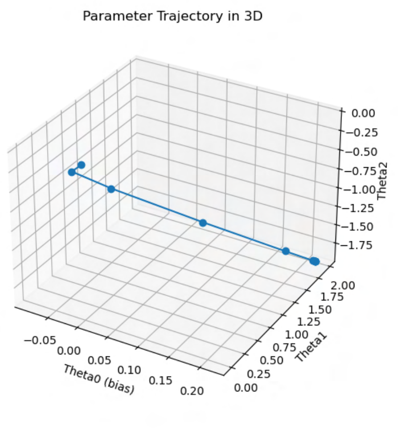


Figure 8: Training loss over the iterations.

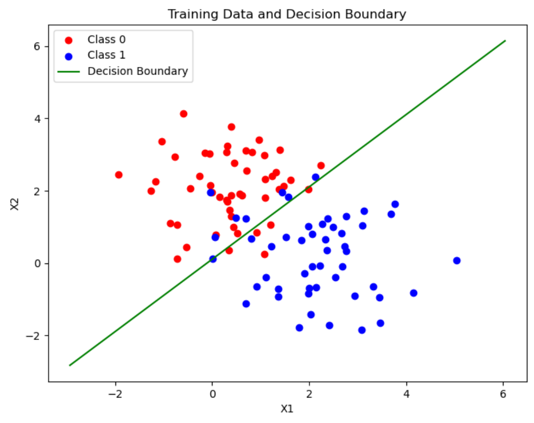


Figure 9: Training loss over the iterations.

The optimum values of the parameters are:

$$\boxed{\theta_0 = 0.2157 \quad \theta_1 = 1.9626 \quad \theta_2 = -1.9648}$$

Here θ_0 is the constant, θ_1 is the coefficient of x_1, and θ_2 is the coefficient of x_2.

3.2. Plotting the decision boundary

Figure 9 shows the data along with the decision boundary learned by the logistic regression model, which serves as the classifier to separate the two classes.

4. Gaussian Discriminant Analysis

4.1. Closed Form Solution

Gaussian Discriminant Analysis (GDA) is a generative classification algorithm that models the joint distribution of the features $x \in R^n$ and the class label $y \in \{0, 1\}$. The assumption is that the class-conditional distributions are multivariate Gaussian:

$$x \mid y = k \sim \mathcal{N}(\mu_k, \Sigma_k), \quad k \in \{0, 1\}$$

where μ_k is the mean vector for class k, and Σ_k is the covariance matrix for class k.
In the simplified case, we assume that both classes share the same covariance matrix, i.e., $\Sigma_0 = \Sigma_1 = \Sigma$. This assumption results in a linear decision boundary between the two classes.
The parameters μ_0, μ_1 are estimated as the sample means of the data points belonging to each class, and Σ is estimated as the average covariance across all data points:

$$\mu_k = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = k\}\, x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = k\}}, \qquad \Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

We implemented Gaussian Discriminant Analysis (GDA) to classify salmon into two categories: Alaska and Canada. Each fish is represented by two attributes, x_1 and x_2. The dataset was normalized prior to training. (See implementation in code for details.) Using maximum likelihood estimation, we obtained the class means and covariance matrices.

$$\mu_0 \text{ (Alaska)} = \begin{bmatrix} 96.6004 \\ 427.8804 \end{bmatrix}, \quad \mu_1 \text{ (Canada)} = \begin{bmatrix} 135.6804 \\ 364.8404 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 287.482 & -26.748 \\ -26.748 & 1123.25 \end{bmatrix}$$

4.2. Plotting Data for visualization

The training data points are shown in Figure 10, with Alaska and Canada represented by different markers.

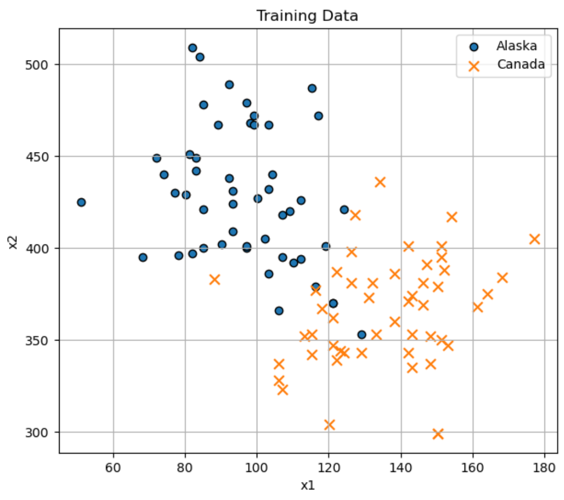


Figure 10: Training data points.

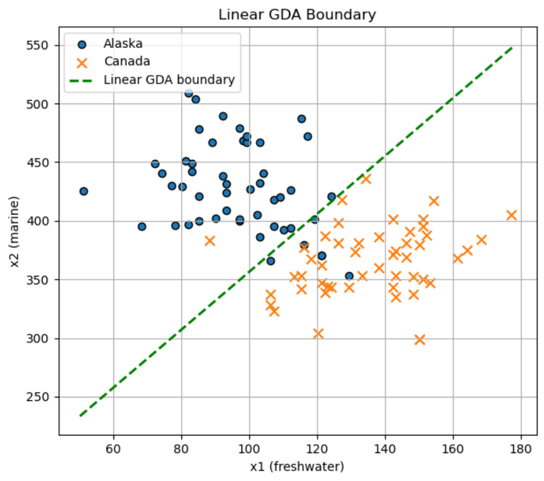


Figure 11: Linear decision boundary.

4.3. Linear Boundary

Assuming $\Sigma_0 = \Sigma_1 = \Sigma$, the decision boundary becomes linear:

$$w^T x + b = 0$$

where

$$w = \Sigma^{-1}(\mu_1 - \mu_0), \quad b = \tfrac{1}{2}\left(\mu_0^T \Sigma^{-1} \mu_0 - \mu_1^T \Sigma^{-1} \mu_1\right) + \ln\left(\frac{\phi}{1-\phi}\right)$$

and $\phi = P(y = 1)$. See the decision boundary in Figure 11

4.4. Covariance Matrices

In the general setting of Gaussian Discriminant Analysis (GDA), we allow each class to have its own covariance matrix. The maximum-likelihood estimate of the covariance matrix for class $k \in \{0, 1\}$ is given by:

$$\Sigma_k = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = k\}\,(x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T}{\sum_{i=1}^{m} 1\{y^{(i)} = k\}}$$

where μ_k is the mean vector of class k, and the indicator function $1\{y^{(i)} = k\}$ selects the points belonging to class k.

Using this formula, the estimated covariance matrices for the two classes are:

$$\Sigma_0 = \begin{bmatrix} 255.3956 & -184.3308 \\ -184.3308 & 1371.1044 \end{bmatrix}, \quad \Sigma_1 = \begin{bmatrix} 319.5684 & 130.8348 \\ 130.8348 & 875.3956 \end{bmatrix}$$

4.5. Quadratic Boundary

When separate covariances are allowed, the decision boundary becomes quadratic:

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

See the Quadratic decision boundary in Figure 12

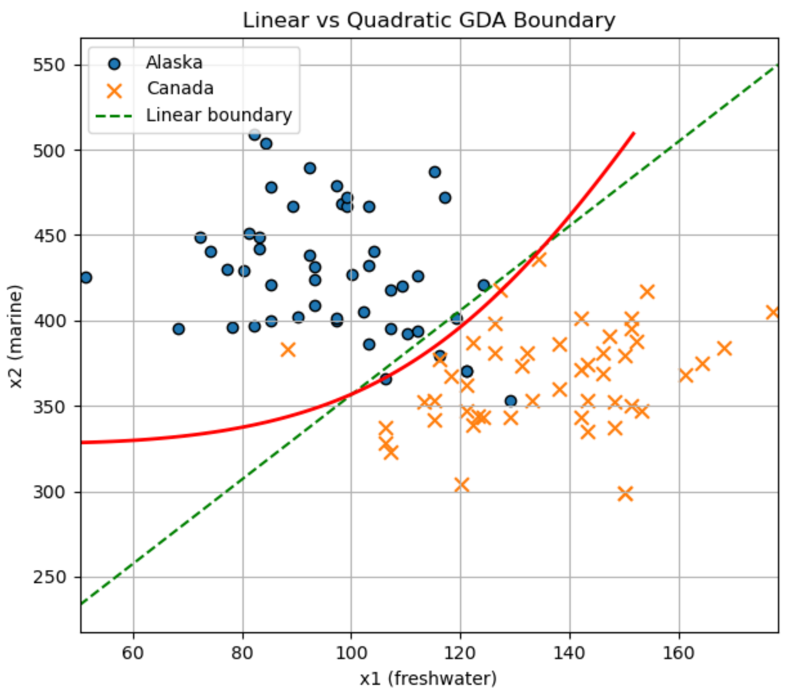


Figure 12: Quadratic decision boundary (red solid) compared with linear boundary (green dashed).

4.6. Analysis of the decision boundary

We can clearly see from Figure 12 that the shape of the decision boundary differs, and hence their classification power also differs. In LDA, we assume a stronger condition by imposing the same covariance matrix across classes. This restriction simplifies the model, resulting in a linear boundary, but also makes it less flexible compared to the more general quadratic boundary in QDA.
We observe that the Alaska samples (blue points) and Canada samples (orange points) are not linearly separable in the given feature space. The linear boundary misclassifies several points in the overlap region between the two classes. In contrast, the quadratic boundary adapts to the distribution more effectively by bending around the clusters, thereby reducing misclassification.