Machine Learning for Electrical and Electronics Engineering

Homework 1

Berke Aziz Yılmaz 040200381

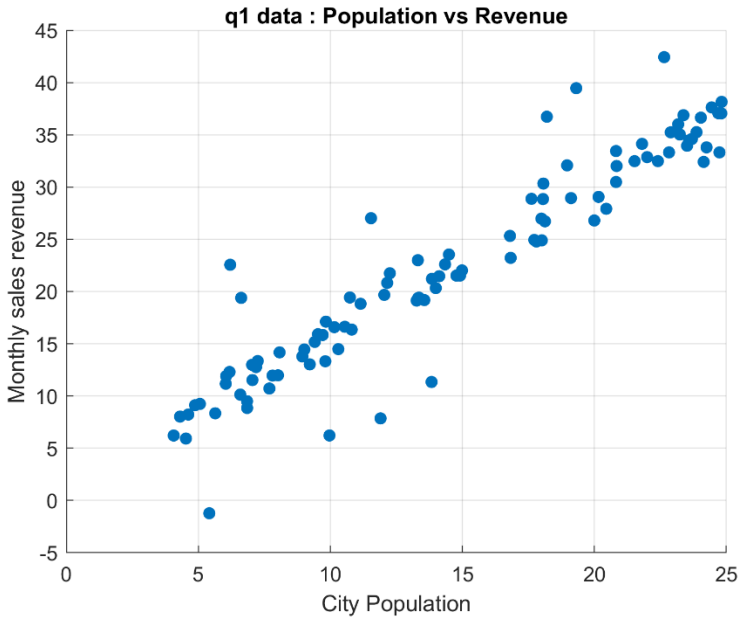
Q1

1. Linear Regression with One Variable Using L1 Loss

In this part, the dataset q1data.txt is used. The input variable x represents the city population, and the output variable y represents the monthly sales revenue of an electronics store in that city. The aim is to learn a linear model that predicts the monthly sales revenue from the population using the L1 loss (mean absolute error).

1.1 Visualizing the Data

A scatter plot of the training data is shown in Figure 1.1. Each point corresponds to a city, with its population on the horizontal axis and the monthly sales revenue on the vertical axis.From the plot we can see a clear positive trend: cities with larger populations tend to have higher monthly sales revenues. The relationship looks approximately linear, although there is some spread around the main trend line.



(Figure 1.1: Scatter plot of city population vs. monthly sales revenue.)

1.2 Model and L1 Cost Function

The relationship between population and revenue is modeled by a linear hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x \quad \textbf{(1.1)}$$

The parameters are learned by minimizing the mean absolute error (L1 loss):

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \left|h_\theta\left(x^{(i)}\right) - y^{(i)}\right| \quad \textbf{(1.2)}$$

Here, m is the number of training examples and $\left(x^{(i)}, y^{(i)}\right)$ denotes the i-th data point.

1.3 Gradient Descent Implementation with L1 Loss

To minimize the L1 cost function, batch gradient descent with a subgradient of the absolute value is used. For each training example we define $x_0^{(i)} = 1,\ x_1^{(i)} = x^{(i)}$

The update rule for each parameter θ_j for (j =0,1) is:

$$\sum_{i=1}^{m} sign\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}, j = 0,1 \quad \textbf{(1.3)}$$

The sign function used in the subgradient is defined as:

$$sign(z) = -1 \ \ if\ z < 0$$
$$sign(z) = 0 \ \ if\ z = 0$$
$$sign(z) = 1 \ \ if\ z > 0 \quad \textbf{(1.4)}$$

The algorithm is implemented in MATLAB in a custom function gradientDescentL1.m. At each iteration the hypothesis values are computed for all training examples, the subgradient is calculated using the formula above, and the parameters are updated simultaneously.

1.4 Effect of the Learning Rate

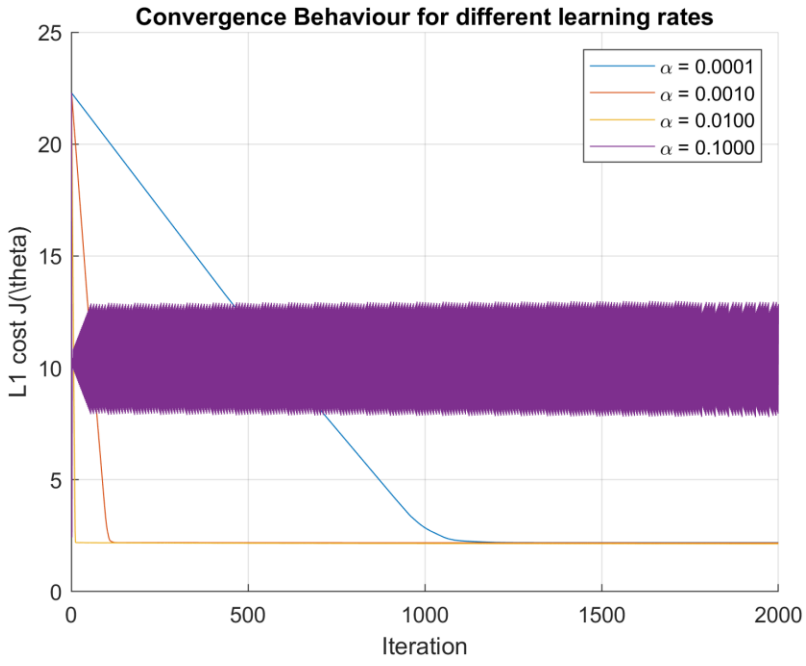
To study the effect of the learning rate, four different values were tested:

$$\alpha \in \{0.0001,\ 0.001,\ 0.01,\ 0.1\} \quad \textbf{(1.5)}$$

For each learning rate, gradient descent was run for 2000 iterations and the cost value J(θ) was recorded at every step. The resulting curves are shown in Figure 1.2.

For alpha = 0.0001, the cost decreases almost monotonically but very slowly, so convergence is too slow. For alpha = 0.001 and alpha = 0.01, the cost drops quickly and stabilizes at a low value. Among these, alpha = 0.01 reaches a slightly lower final cost and converges faster. For alpha = 0.1, the cost function oscillates around a higher value and does not converge, which shows that this learning rate is too large and causes the updates to overshoot the minimum.

Based on these observations, alpha = 0.01 was chosen as the learning rate for the final model.



(Figure 1.2: L1 cost J(theta) versus iteration for different learning rates.)

1.5 Final Parameters and Linear Fit

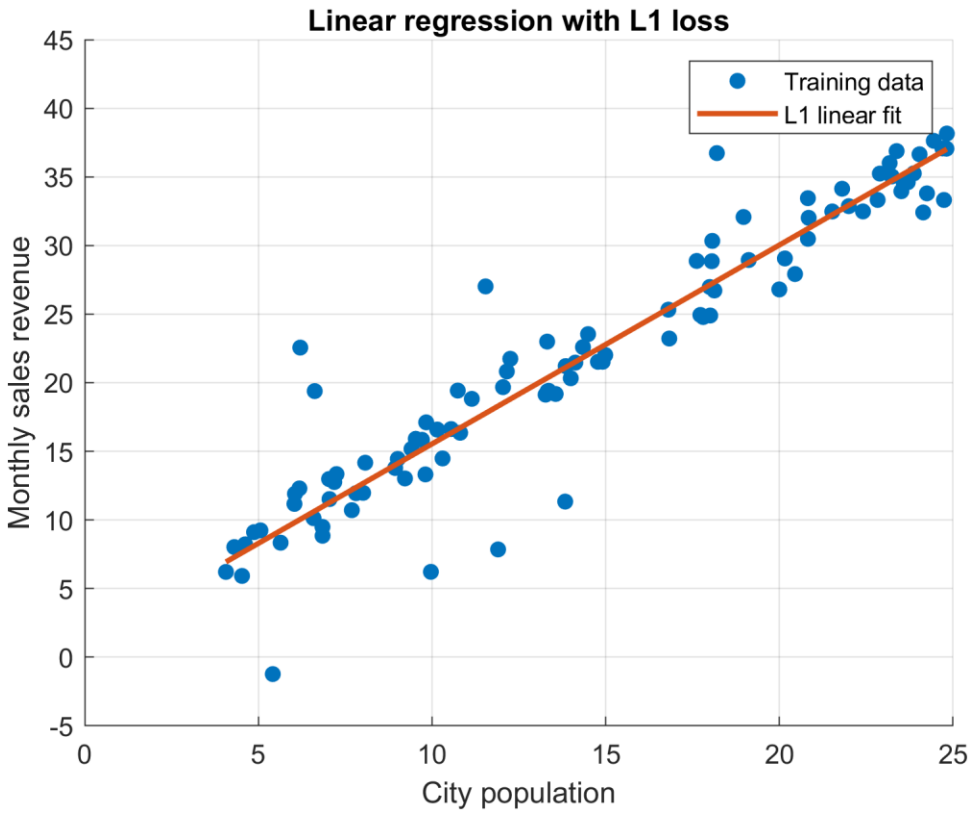
Using the learning rate and 2000 iterations, gradient descent converged to the following parameter values:

$$\theta_0 = 1.0506, \qquad \theta_1 = 1.4492 \tag{1.6}$$

Therefore, the learned linear regression model is:

$$h_\theta(x) = 1.0506 + 1.4492\,x \tag{1.7}$$

The regression line given by this model is plotted together with the training data in Figure 1.3**.** The line follows the general trend of the points closely and explains the increasing behavior of the revenue as a function of population.



(Figure 1.3: Learned linear regression line (L1 loss) over the training data.)

1.6 Prediction of Monthly Sales Revenue

In the dataset, the population feature is stored in units of thousands of people. Thus:

a city with 20,000 inhabitants corresponds to x = 20,

a city with 60,000 inhabitants corresponds to x = 60.

Using the learned model, the predicted revenues are:

$$h_\theta(20) = 1.0506 + 1.4492 \cdot 20 \approx 30.03 \quad \textbf{(1.8)}$$

$$h_\theta(60) = 1.0506 + 1.4492 \cdot 60 \approx 88.00 \quad \textbf{(1.9)}$$

So, the model predicts a monthly sales revenue of approximately 30.03 (in the same units as the original dataset) for a city with population 20,000, and approximately 88.00 for a city with population 60,000.

1.7 Visualization of the L1 Cost Function

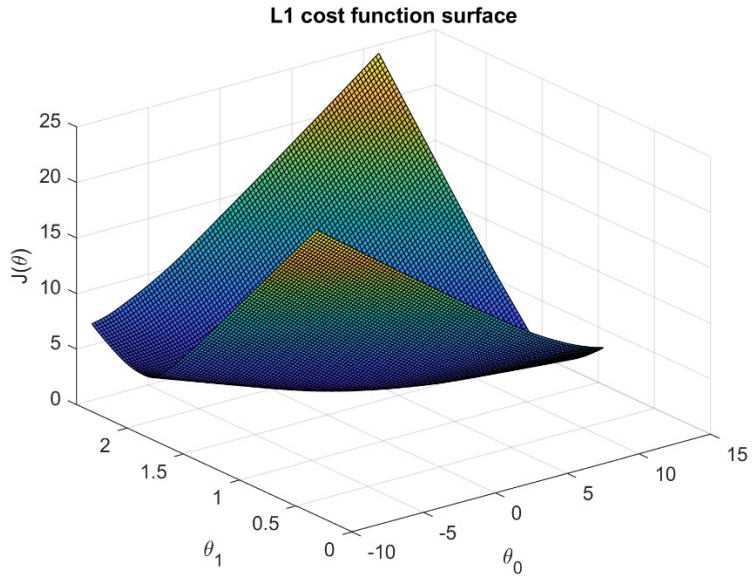
To better understand the optimization landscape, the L1 cost function was evaluated on a grid of theta_0 and theta_1 values around the optimum, and two plots were produced:

a 3D surface plot (Figure 1.4),

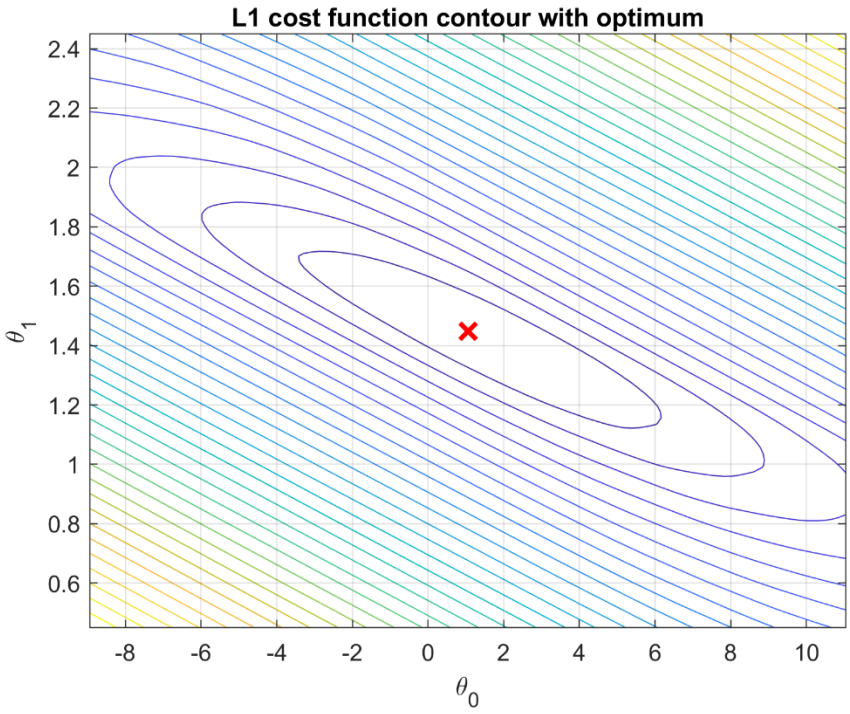
a contour plot (Figure 1.5).

In the surface plot, the cost function forms a valley-shaped region where the values are small. Because of the absolute value in the loss, the surface is not perfectly smooth and has sharper ridges compared to an L2 loss surface.

In the contour plot, lines of equal cost are displayed in the (θ_0, θ_1) plane. The point corresponding to the parameters found by gradient descent, (θ_0, θ_1) = (1.0506, 1.4492), is marked on the plot and lies near the centre of the low-cost region. This confirms that gradient descent with the chosen learning rate has converged to a good solution.



(Figure 1.4: Surface plot of the L1 cost function J(theta_0, theta_1)).



(Figure 1.5: Contour plot of the L1 cost function with the optimal parameters marked.)

Q2

2.1 - Linear Regression with Multiple Variables

In this part, I implemented a multivariate linear regression model to predict the unit area price of a house using three features from the dataset q2data.txt:

house age (years)

distance to the nearest MRT station (meters)

number of convenience stores in the neighborhood

The target variable is the house price per unit area.

The hypothesis function is a linear model with a bias term:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \quad \textbf{(2.1)}$$

Where X_1,X_2 and X_3 correspond to age, distance, and number of stores respectively.

2.2 Data Preprocessing – Feature Normalization

Before running gradient descent, I normalized all three input features using mean–standard deviation normalization.
For each feature X_j(age, distance, stores), I computed the mean and standard deviation and transformed each value as:

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \quad \textbf{(2.2)}$$

$$\sigma_j = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(x_j^{(i)} - \mu_j\right)^2} \quad \textbf{(2.3)}$$

$$x_{j,\text{norm}}^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j} \quad \textbf{(2.4)}$$

The bias column is not normalized (it is always 1).

Reason for normalization.

The raw feature scales in this dataset are very different: age is on the order of tens of years, distance is on the order of thousands of meters, and the number of stores is a small integer. Without normalization, the cost function contours become highly elongated and gradient descent has to use a very small learning rate to remain stable. By normalizing the features to similar scales (approximately zero mean and unit variance), the cost function becomes more "isotropic", which leads to:

faster convergence of gradient descent,

easier selection of a single learning rate α that works well for all parameters.

The same μ and σ values are stored and later reused when predicting the price for a new house.

2.3 Gradient Descent Implementation with L2 Loss

2.3.1 Cost function and gradient

For multivariate linear regression with L2 loss, the cost function is

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \quad \textbf{(2.5)}$$

In matrix form this is

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y) \quad \textbf{(2.6)}$$

where X is the design matrix that contains a column of ones (bias) and the three normalized features, θ is a 4×14 parameter vector, and y is the vector of target prices.

The gradient of the cost function with respect to θ is

$$\nabla_\theta J(\theta) = \frac{1}{m}X^T(X\theta - y) \quad \textbf{(2.7)}$$

The batch gradient descent update rule is then

$$\theta = \theta - \alpha\frac{1}{m}X^T(X\theta - y) \quad \textbf{(2.8)}$$

2.3.2 Implementation details

I implemented a function gradientDescentMulti(X, y, theta, alpha, num_iters) that:

takes the normalized design matrix X (with the bias column),iteratively updates θ using the gradient descent rule above, stores the history of J(θ) for each iteration in J_hist.

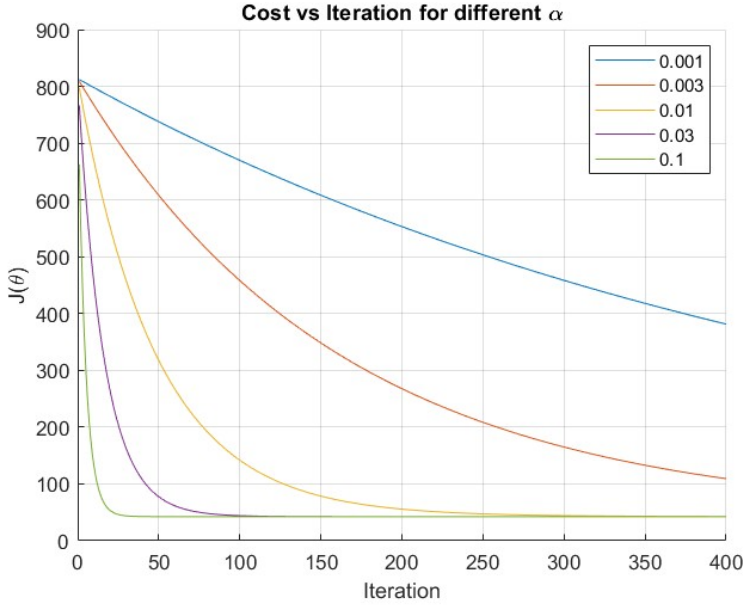
The initial parameter vector was set to θ=0 (4×1 zeros).
The number of iterations was chosen as 400 for all learning rates.

2.3.3 Effect of the learning rate α

To study the effect of the learning rate, I ran gradient descent with the following values:

α=0.001, α=0.003, α=0.01 α=0.03, α=0.10 **(2.9)**

For each α, I plotted the value of J(θ) versus the iteration number.



The resulting graph is shown in Figure 2.1: "Cost vs Iteration for different α".

Observations from Figure 2.1:

For α=0.001, the cost decreases very slowly and has not converged after 400 iterations.For =0.003 and =0.01, convergence is faster but still relatively slow compared to larger α. For =0.03, the cost drops quickly and flattens out at a low value within the first 50–100 iterations. For α=0.10, the cost decreases the fastest and reaches a very low value in only a few dozen iterations, while remaining stable (no divergence or oscillation was observed).

2.3.4 Selecting the best learning rate and final parameters

To select the best learning rate, I compared the final cost value after 400 iterations for each α.

$$J_{\text{final}}(\alpha) = J^{(\text{iter}=400)}(\theta;\alpha) \quad \textbf{(2.10)}$$

and chose the learning rate that yields the smallest J_{final}. According to this criterion, the best learning rate was:

$$\alpha = 0.1 \quad \textbf{(2.11)}$$

Using $\alpha = 0.1$ and the normalized features, the learned parameter vector is:

$$\theta_{GD} = [\,37.9802, -2.8807, -6.7891, 3.8217\,] \quad \textbf{(2.12)}$$

2.3.5 Prediction with gradient descent (normalized features)

For a new house with

$$x_1 = 20 \text{ years}, \quad x_2 = 2500 \text{ meters}, \quad x_3 = 5 \quad \textbf{(2.13)}$$

the same normalization parameters μ and σ computed from the training set are used.

$$x_j^{(\text{norm})} = \frac{x_j - \mu_j}{\sigma_j}, \quad j = 1,2,3 \quad \textbf{(2.14)}$$

Then the input vector (including the bias term) becomes.

$$\tilde{x} = \left[1, x_1^{(norm)}, x_2^{(norm)}, x_3^{(norm)}\right] \quad \textbf{(2.15)}$$

The predicted unit area price is:

$$\widehat{y_{GD}} = \tilde{x}\,\theta_{GD} = 30.9596 \quad \textbf{(2.16)}$$

2.4 Normal Equation (closed-form solution)

2.4.1 Derivation

For linear regression with the squared-error cost,

$$J(\theta) = \frac{1}{2m} |X\theta - y|_2^2 \quad \textbf{(2.17)}$$

setting the gradient to zero gives:

$$\nabla_\theta J(\theta) = \frac{1}{m} X^T(X\theta - y) = 0 \;\; -> \;\; X^T X\theta = X^T y \quad \textbf{(2.18)}$$

If $X^T X$ is invertible, the optimal solution is:

$$\theta_{NE} = (X^T X)^{-1} X^T y \quad \textbf{(2.19)}$$

2.4.2 Result (raw features, no normalization)

Using the raw (non-normalized) features with a bias term, the normal equation yields:

$$\theta_{NE} = 42.9773,\ -0.2529,\ -0.0054,\ 1.2974 \quad \textbf{(2.20)}$$

For the same house (20, 2500, 5), the predicted unit area price is:

$$\widehat{y_{NE}} = 30.9596 \quad \textbf{(2.21)}$$

2.5 Discussion and comparison

Gradient descent was performed on normalized features to ensure stable and fast convergence across parameters. The normal equation provides a direct closed-form solution without requiring normalization (although normalization can still be beneficial for numerical conditioning in some problems).

In this dataset, the prediction obtained via gradient descent and the normal equation are identical up to four decimal places:

$$\widehat{y_{GD}} = \widehat{y_{NE}} = 30.9596 \quad \textbf{(2.22)}$$

which indicates that gradient descent successfully converged to the optimal solution.

From the sign of the learned parameters, increasing house age and MRT distance tends to decrease the predicted unit area price (negative coefficients), while an increase in the number of nearby convenience stores tends to increase the predicted price (positive coefficient). (The magnitude of coefficients between GD and normal equation is not directly comparable because GD was trained on normalized features.)

Q3

3.1 Dataset

In Question 3, I used the dataset q3data.txt, which contains m=200 samples. Each sample has five input features and a binary label y ∈ {0,1} indicating survival.

The feature vector is defined as $(x_1, x_2, x_3, x_4, x_5) = (\text{Age, Gender, Speed, Helmet, Seatbelt})$, where Gender is coded as 1 for male, and Helmet/Seatbelt are binary indicators.

3.2 Logistic regression model

The logistic regression hypothesis uses a sigmoid applied to a linear score.

$$h_\theta(x) = \sigma(\theta^T x) \quad \textbf{(3.1)}$$

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad \textbf{(3.2)}$$

A bias term is included, so each input is augmented as:

$$x = [1, x_1, x_2, x_3, x_4, x_5] \quad \textbf{(3.3)}$$

3.3 Cost function and gradient (unregularized)

The unregularized logistic regression cost function is:

The unregularized logistic regression cost function is:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right) \quad \textbf{(3.4)}$$

The gradient of the cost is:

$$\nabla_\theta J(\theta) = \frac{1}{m}X^T(h-y) \quad \textbf{(3.5)}$$

where X is the design matrix (including the bias column), $h = \sigma(X\theta)$, and y is the vector of labels.

3.4 Optimization using fminunc

I minimized J(θ) using MATLAB fminunc with the quasi-Newton algorithm. The analytic gradient was provided to the optimizer. The optimization terminated successfully when the gradient norm became smaller than the optimality tolerance ("Local minimum found").

3.5 Results using all five features

Using all five features, the learned parameter vector is:

$$\theta_{all} = [-1.1550, 0.0167\ ,0.5687, 0.0024, -0.2639,0.3129] \quad \textbf{(3.6)}$$

3.6 Probability prediction for the given person

For the person described in the homework (age =50, male =1, speed =120, helmet =1, seatbelt =1), the input vector is:

$$x_{new} = [1,50,1,120,1,1] \quad \textbf{(3.7)}$$

The predicted survival probability is:

$$p(\,y = 1 \mid x_{new}\,) = \sigma(x_{new}\theta_{all}) = 0.6431 \quad \textbf{(3.8)}$$

Using a classification threshold of 0.50.50.5, the training accuracy is:

Accuracy=%56.50

3.7 Decision boundary using Age and Speed

To visualize a decision boundary in 2D, I trained a second logistic regression model using only Age and Speed. In this case, the augmented feature vector is:

$$x = [1, Age, Speed] \quad \textbf{(3.9)}$$

The learned parameters are:

$$\theta_{Age,Speed} = [-0.7545\ , 0.0145, 0.0021] \quad \textbf{(3.10)}$$

The decision boundary corresponds to $h_\theta(x) = 0.5$, which is equivalent to:

$$\theta^T x = 0 \quad \textbf{(3.11)}$$

Thus, the boundary line satisfies:

$$\theta_0 + \theta_1 Age + \theta_2 Speed = 0 \quad \textbf{(3.12)}$$

Solving for Speed gives:

$$Speed = -(\theta_0 + \theta_1 Age)/\theta_2 \quad \textbf{(3.13)}$$

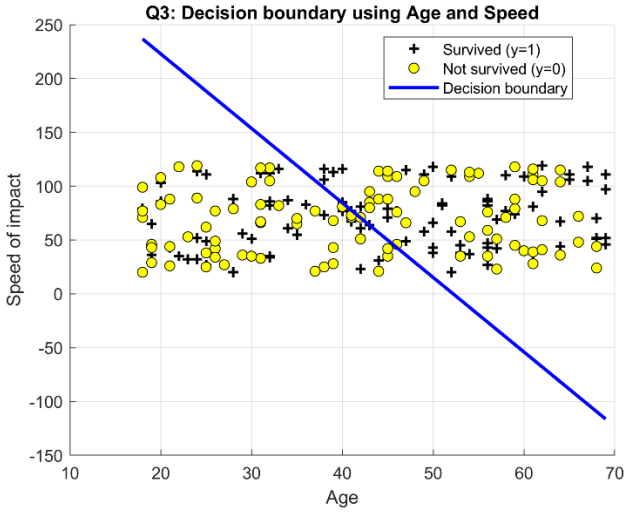


Figure 3.1. Decision boundary of logistic regression using only Age and Speed. Points correspond to survived (y=1) and not survived (y=0).

3.8 Discussion

The Age–Speed decision boundary may not separate the classes well because the original problem is higher-dimensional, and the classes overlap when projected onto two features. The model trained with all five features is more appropriate for probability prediction (e.g., p=0.6431 for the specified person), while the Age–Speed boundary is mainly used for visualization.

Q4

4.1 Dataset and visualization

In Question 4, the dataset contains two input features and a binary label y ∈ {0,1} The two features are Moisture x1 and Nutrient (x2). The positive class y=1 (grew) and the negative class y=0 (failed) are shown in Figure 4.1. The scatter plot suggests that the classes are not linearly separable in the original (x1 , x2) space, so a nonlinear decision boundary is required.

Figure 4.1 is the raw data plot.

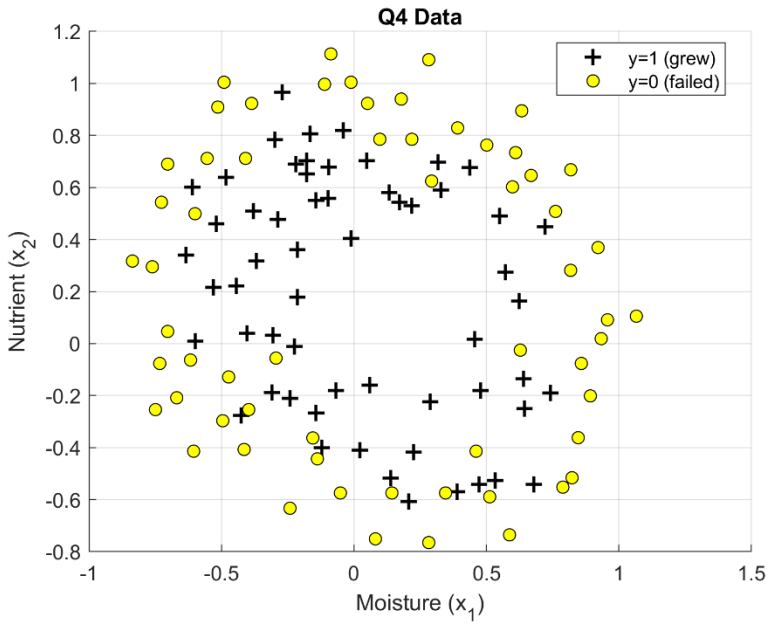


Figure 4.1. Q4 dataset in the (x1, x2) plane. "+" indicates y=1 (grew) and "o" indicates y=0 (failed).

4.2 Polynomial feature mapping (degree = 5)

To model a nonlinear boundary, I mapped the original two features into polynomial features up to degree 5. This creates a higher-dimensional feature vector that allows logistic regression to represent nonlinear boundaries in the original input space.

$$\Phi(x_1, x_2) = [1, x_1, x_2, x_1^2, x_1x_2, x_2^2, \dots] \quad \textbf{(4.1)}$$

For degree d=5, the number of mapped features (including the bias term) is:

$$n = \frac{(d+1)(d+2)}{2} \quad \textbf{(4.2)}$$

So the design matrix becomes

$$X \in R^{m \times 21} \quad \textbf{(4.3)}$$

4.3 Regularized logistic regression model

The hypothesis remains the logistic sigmoid applied to a linear score in the mapped feature space:

$$h_\theta(x) = \sigma(\theta^T x) \quad \textbf{(4.4)}$$

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad \textbf{(4.5)}$$

Here $x = \Phi(x_1, x_2)$ is the mapped feature vector (dimension 21).

4.4 Cost function and gradient with L2 regularization

I used L2-regularized logistic regression to control overfitting by penalizing large parameter values. The bias term θ_0 is not regularized.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right) + \frac{\lambda}{2m}\sum_{j=1}^{n-1}\theta_j^2 \quad \textbf{(4.6)}$$

The gradient is:

$$\nabla_\theta J(\theta) = \frac{1}{m}X^T(h-y) + \frac{\lambda}{m}[0, \theta_1, \theta_2, \ldots, \theta_{n-1}] \quad \textbf{(4.7)}$$

4.5 Optimization method

The parameters θ were estimated by minimizing J(θ) using MATLAB fminunc (quasi-Newton). The analytic gradient was supplied, so the optimizer converged when the gradient norm dropped below the optimality tolerance ("Local minimum found").

4.6 Decision boundary

The decision boundary corresponds to $h_\theta(x) = 0.5$, which is equivalent to $\theta^T x = 0$ in the mapped feature space.

$$\theta^T \Phi(x_1, x_2) = 0 \quad \textbf{(4.8)}$$

In practice, I evaluated $\theta^T \Phi(x_1, x_2)$ on a dense grid over (x_1, x_2) and plotted the contour where it equals zero. This contour is shown as the boundary curve in Figures 4.2–4.5 for different λ.

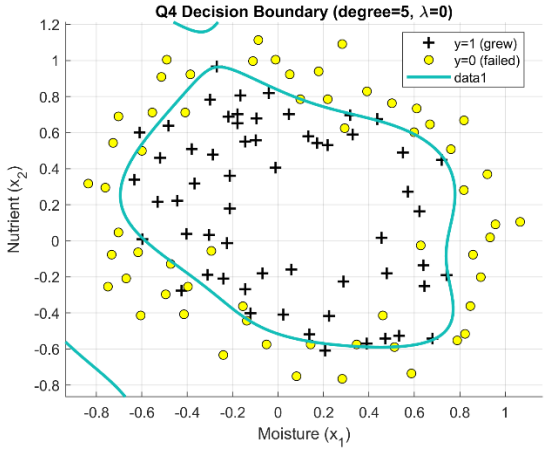


Figure 4.2 Decision boundary for degree 5 polynomial features with λ=0.

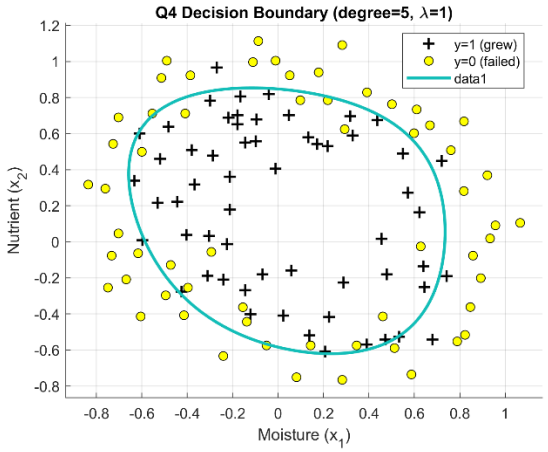


Figure 4.3 Decision boundary for degree 5 polynomial features with λ=1.

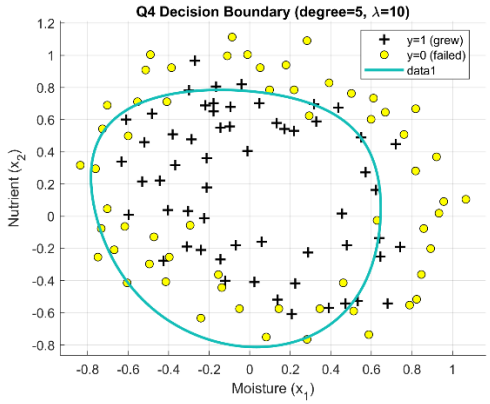


Figure 4.4 Decision boundary for degree 5 polynomial features with λ=10.

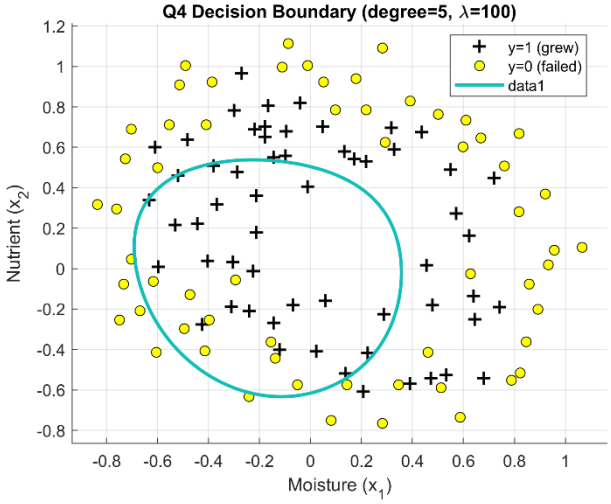


Figure 4.5 Decision boundary for degree 5 polynomial features with λ=100.

4.7 Effect of the regularization parameter λ

To study the effect of regularization, I trained the model with degree 5 features for $\lambda \in \{0,1,10,100\}$. The resulting decision boundaries and training accuracies are shown in Figures 4.2–4.5 and summarized below.

$\lambda=0$, the final cost and training accuracy were: $J(\theta)=0.302826$, Accuracy =%87.29. The boundary in Figure 4.2 is highly flexible and follows the training data closely. The learned parameters also have large magnitudes, indicating a complex model with higher risk of overfitting.

For $\lambda=1$, the final cost and training accuracy were: $J(\theta)=0.541111$, Accuracy =%83.05. Compared to $\lambda=0$, Figure 4.3 shows a smoother and more stable boundary. Regularization shrinks the parameter magnitudes and reduces unnecessary oscillations while still capturing the main nonlinear separation.

For λ=10, the final cost and training accuracy were: J(θ)=0.655233, Accuracy =%70.34. In Figure 4.4, the boundary becomes noticeably simpler (more "rounded"). This indicates stronger regularization, which increases bias and reduces variance, leading to underfitting relative to the smaller-λ cases.

For λ=100, the final cost and training accuracy were: J(θ)=0.687803, Accuracy =64.41%.The boundary in Figure 4.5 is very smooth and close to a simple shape, meaning the model is heavily constrained. This is the clearest underfitting case: the model cannot represent the nonlinear structure needed to separate the classes well.

Overall, increasing λ reduces model complexity by shrinking the coefficients, producing smoother boundaries but lowering training accuracy. In this experiment, λ=1 provides a good trade-off between boundary smoothness and classification performance compared to λ=0 (more complex) and λ≥10 (too constrained).

Q5

5.1 Dataset and visualization

The dataset has 4 input features and a 3-class label. The classes are Adelie, Chinstrap, and Gentoo. The features are:

$$x = [x_1, x_2, x_3, x_4] \quad \textbf{(5.1)}$$

$$x_1 = BillLength(mm), x_2 = BillDepth(mm), x_3 = FlipperLength(mm), x_4 = BodyMass(g)$$

I removed rows with missing values. After cleaning, the dataset has:

$$m = 342$$

To see the data, I plotted Bill Length vs Body Mass. The classes overlap in this 2D plot, so separation is not perfect in 2D.

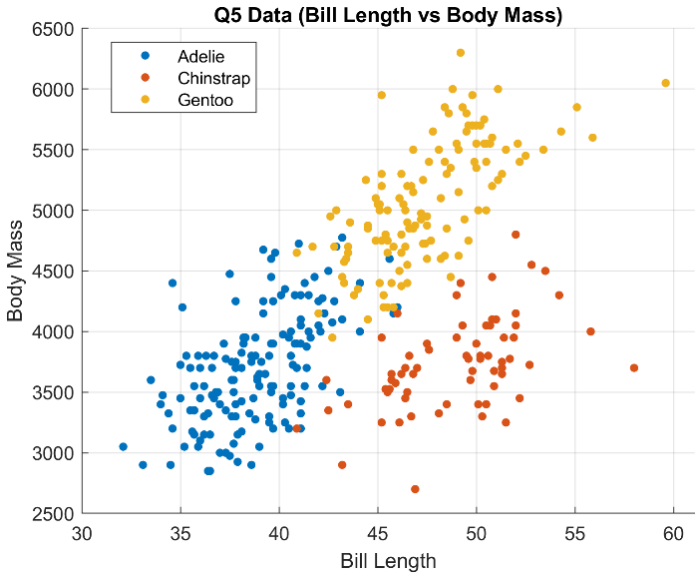


Figure 5.1: Raw data plot for Bill Length vs Body Mass.

5.2 Gaussian class model and priors

For both methods, I assume each class follows a Gaussian distribution:

$$p(x \mid k) = N(x; \mu_k, \Sigma_k) \quad \textbf{(5.2)}$$

The class prior is estimated from data:

$$\pi_k = \frac{N_k}{m} \quad \textbf{(5.3)}$$

where N_k is the number of samples in class k.

5.3 LDA (Linear Discriminant Analysis)

In LDA, all classes share the same covariance matrix:

$$\Sigma_k = \Sigma \quad \text{for all k} \quad \textbf{(5.4)}$$

The LDA score (discriminant) for class k is

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \ln(\pi_k) \quad \textbf{(5.5)}$$

Prediction rule:

$$\hat{y} = \arg\max_k \delta_k(x) \quad \textbf{(5.6)}$$

x, the decision boundary is linear.

5.4 QDA (Quadratic Discriminant Analysis)

In QDA, each class has its own covariance matrix:

Σ_k can be different for each k

The QDA score (discriminant) for class k is

$$\delta_k(x) = -\frac{1}{2}\ln|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \ln(\pi_k) \quad \textbf{(5.7)}$$

Prediction rule:

$$\hat{y} = \arg\max_k \delta_k(x) \quad \textbf{(5.8)}$$

Because of the quadratic term, the decision boundary can be curved.

5.5 Parameter estimation

For each class k, the mean is:

$$\mu_k = \frac{1}{N_k}\sum_{i:y^{(i)}=k} x^{(i)} \quad \textbf{(5.9)}$$

For LDA, the pooled covariance matrix is:

$$\Sigma = \frac{1}{m-K}\sum_{k=1}^{K}\sum_{i:y^{(i)}=k}\left(x^{(i)}-\mu_k\right)\left(x^{(i)}-\mu_k\right)^T \quad \textbf{(5.10)}$$

For QDA, the class covariance matrix is:

$$\Sigma_k = \frac{1}{N_k}\sum_{i:y^{(i)}=k}\left(x^{(i)}-\mu_k\right)\left(x^{(i)}-\mu_k\right)^T \quad \textbf{(5.11)}$$

(Here k =3)

In implementation, a very small diagonal term can be added to avoid numerical problems.

5.6 Decision boundary plots

To draw decision boundaries, I used only two features for the plot:

$z = [BillLength, BodyMass]$

I created a dense grid in this 2D space. For each grid point, I computed $\delta_k(z)$ and selected the class with max score. The boundary is where the predicted class changes.

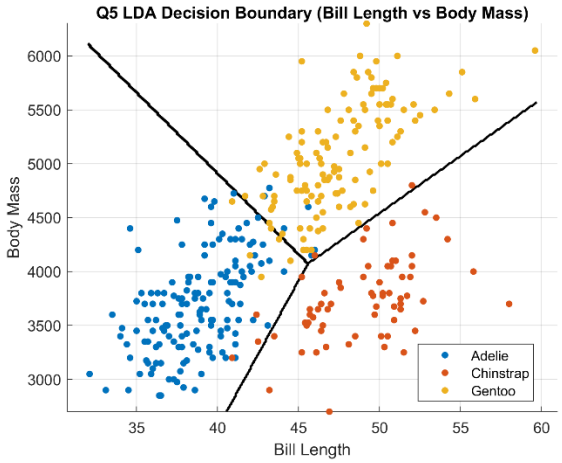


Figure 5.2: LDA decision boundary for Bill Length vs Body Mass.

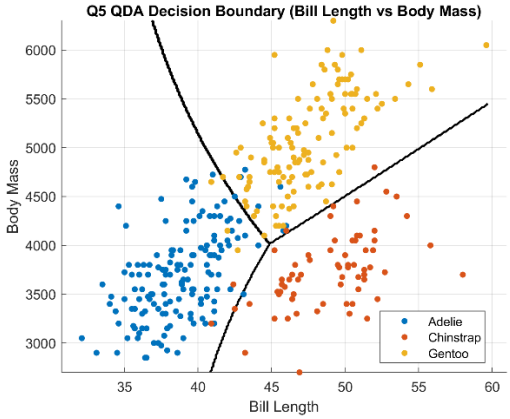


Figure 5.3: QDA decision boundary for Bill Length vs Body Mass.

5.7 Results (confusion matrix and accuracy)

The classification performance is summarized with confusion matrices and training accuracy. The accuracy is computed as:

$$Accuracy = \frac{\sum diag(C)}{\sum C} \quad \textbf{(5.12)}$$

For custom LDA, the confusion matrix (rows = true class, columns = predicted class) is:

$$C_{LDA} = [145,0,6; 4,62,2; 6,0,117] \quad \textbf{(5.13)}$$

This corresponds to:

$$Accuracy_{LDA} = (145 + 62 + 117)/342 = 0.9474 \quad \textbf{(5.14)}$$

The LDA model classifies most samples correctly. The main errors occur between Adelie and Gentoo. For example, 6 Adelie samples are predicted as Gentoo and 6 Gentoo samples are predicted as Adelie. This is expected because these two classes overlap more in some feature regions. Chinstrap is also classified well, but a small number of Chinstrap samples are predicted as Adelie (4 samples) or Gentoo (2 samples). Overall, LDA provides a strong baseline with high accuracy using the four features.

5.8 MATLAB fitcdiscr comparison

I also used MATLAB fitcdiscr to check the results. The accuracies match the custom code:

fitcdiscr (linear): 94.74%

fitcdiscr (quadratic): 95.03%

This shows the custom LDA/QDA implementation is consistent with MATLAB.

Q6

6. k-Nearest Neighbors (k-NN) Classification

In this task, the goal is to classify penguin samples into three classes (Adelie, Chinstrap, Gentoo) using the k-Nearest Neighbors algorithm and evaluate performance with 5-fold cross-validation, then compare a custom implementation against MATLAB's fitcknn.

6.1 Problem Definition (k-NN Rule)

k-NN predicts the class of a test sample x by looking at the labels of its k nearest training samples and selecting the majority class. For a g-class classification problem, the prediction rule is given as:

$$\hat{h}(x) = \arg\max_{l \in \{1,\dots,g\}} \sum_{i:x^{(i)} \in N_k(x)} I\left(y^{(i)} = l\right) \quad \textbf{(6.1)}$$

Here, $N_k(x)$ is the set of k nearest neighbors of x, and I(·) is the indicator function.

The distance between a test sample and training samples is computed using Euclidean distance:

$$d\left(x_{test}^{(i)}, x_{train}\right) = \sqrt{\sum_{j=1}^{p} \left(x_{train,j} - x_{test,j}^{(i)}\right)^2} \quad \textbf{(6.2)}$$

where p is the number of features.

6.2 Implementation Setup (5-Fold Cross-Validation)

The dataset is split into 5 folds using cvpartition. For each fold, 4 folds are used for training and the remaining fold is used for testing. This is repeated until each fold has been used once as the test set. In every fold, predictions are generated for the test partition, a confusion matrix is computed, and the evaluation metrics (Accuracy, Precision, Recall, F1-score) are calculated; finally, the average of these metrics over 5 folds is reported.

Feature standardization is applied using statistics computed from the training portion of each fold (training mean and training standard deviation), and the same transformation is applied to that fold's test portion. This avoids information leakage and is appropriate for distance-based methods.

6.3 Models Compared

Two models are evaluated under the exact same fold splits. The first model is a custom k-NN implementation with k=3. For each test sample, distances to all training samples are computed, the nearest three neighbors are selected, and the predicted class is chosen by majority vote. The second model is MATLAB's fitcknn with NumNeighbors=3. Since the same preprocessing and folds are used, this comparison isolates the difference between a manual implementation and the built-in function.

In the obtained results, the custom implementation and fitcknn produced identical confusion matrices and identical metrics in every fold, which shows that the custom algorithm matches the built-in behavior under this setup.

6.4 Fold-wise Results (Confusion Matrices and Metrics)

The confusion matrices below are reported as "rows = true class, columns = predicted class" in the order [Adelie,Chinstrap,Gentoo].

Fold 1 confusion matrix is

$$C_1 = [29,0,0; 2,13,0; 0,0,24] \quad \textbf{(6.3)}$$

With $Accuracy = 0.9706, Precision_{macro} = 0.9785, Recall_{macro} = 0.9556, F1_{macro} = 0.9651$.The only error in this fold is that 2 Chinstrap samples are predicted as Adelie.

Fold 2 confusion matrix is

$$C_2 = [33,0,0; 0,14,0; 0,0,22] \quad \textbf{(6.4)}$$

and all samples are classified correctly, giving Accuracy=1.0000 and macro Precision/Recall/F1 all equal to 1.0000.

Fold 3 confusion matrix is

$$C_3 = [28,0,0; 0,16,0; 0,0,25] \quad \textbf{(6.5)}$$

and again the fold is perfectly classified with all metrics equal to 1.0000.

Fold 4 confusion matrix is

$$C_4 = [29,0,0; 0,12,0; 0,0,27] \quad \textbf{(6.6)}$$

which is also perfect classification with all metrics equal to 1.0000.

Fold 5 confusion matrix is

$$C_5 = [31,1,0; 1,10,0; 0,0,25] \quad \textbf{(6.7)}$$

with $\text{Accuracy} = 0.9706, \text{Precision\_macro} = 0.9593, \text{Recall\_macro} = 0.9593, \text{F1\_macro} = 0.9593$. The errors here are symmetric confusion between Adelie and Chinstrap (one sample each way), while Gentoo remains perfectly classified.

Across all folds, Gentoo has zero mistakes, and the only confusions occur between Adelie and Chinstrap in folds 1 and 5, indicating these two classes are closer in the chosen feature space.

6.5 Average Results Over 5 Folds and Comparison

Averaging over all five folds, the custom k-NN model achieves Accuracy =0.9882 , Precision_macro =0.9876 , Recall_macro =0.9830 and F1_macro =0.9849. MATLAB fitcknn achieves the exact same averages: Accuracy =0.9882, Precision_macro =0.9876, Recall_macro =0.9830 and F1_macro =0.9849. This one-to-one match confirms the correctness of the custom implementation and shows that, for this dataset and k= 3, the built-in method and the manual method behave identically.

6.6 Interpretation

The results are near-perfect, with three folds reaching 100% accuracy and the remaining two folds showing small errors only between Adelie and Chinstrap. Since k-NN relies directly on distances, this pattern suggests that Gentoo samples are well separated from the other two classes in the feature space, while Adelie and Chinstrap have partial overlap or are closer to each other for a small number of samples.