In the Name of God

Navid Zare

40435071

Statistical Pattern Recognition
Linear and Logistic Regression and Multiclass Classification

1. Linear Regression

1.1 Introduction and Theoretical Background

Linear regression is a fundamental supervised learning algorithm that models the relationship between a dependent variable (target) and one or more independent variables (features) by fitting a linear equation to observed data. In this assignment, we implement simple linear regression to predict automobile fuel efficiency (MPG) based on vehicle weight.

1.1.1 Mathematical Formulation

The simple linear regression model assumes a linear relationship between input x and output y:

$$\hat{y} = \theta_0 + \theta_1 x$$

Where:

- $\hat{y}$: is the predicted value of the dependent variable
- θ_0: is the intercept or bias term
- θ_1: is the slope coefficient representing the change in y per unit change in x
- x: is the independent variable (feature)

1.1.2 Cost Function (Mean Squared Error)

The model quality is measured using the Mean Squared Error (MSE) cost function:

$$J(\theta_0, \theta_1) = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2$$

Where n is the number of training samples. The factor of 1/2 simplifies the gradient computation. The objective is to find parameters θ_0 and θ_1 that minimize this cost function.

1.1.3 Gradient Descent

Gradient descent iteratively updates parameters in the direction of steepest descent:

$$\theta_j := \theta_j - \alpha \times \frac{\partial J}{\partial \theta_j}$$

The partial derivatives are:

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i)$$

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i) \times x_i$$

1.1.4 Closed-Form Solution (Normal Equation)

For linear regression, an analytical solution exists:

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

This provides the optimal parameters directly without iteration, though it requires matrix inversion which becomes computationally expensive for large feature sets.

1.2 Dataset Overview: Auto MPG

The Auto MPG dataset contains technical specifications of automobiles from the 1970s and early 1980s, collected by the StatLib library at Carnegie Mellon University.

1.2.1 Dataset Statistics

Metric	Value	Description
Total Samples	392	Number of vehicle records
Features	4	weight, horsepower, displacement, mpg
MPG Mean	23.45	Average fuel efficiency
MPG Std Dev	7.80	Variability in fuel efficiency

Table 1.1: Auto MPG Dataset Summary Statistics

1.2.2 Sample Data Records

Weight (lbs)	Horsepower	Displacement	MPG
1835.0	46.0	97.0	26.0
2830.0	103.0	131.0	20.3
2108.0	75.0	90.0	24.0
3730.0	110.0	258.0	15.0
3433.0	150.0	304.0	16.0

Table 1.2: Sample Records from Auto MPG Dataset

1.3 Exploratory Data Analysis

1.3.1 Correlation Analysis

The correlation coefficient between vehicle weight and MPG:

$$r(\text{weight}, \text{mpg}) = -0.8322$$

Interpretation of the Correlation Coefficient:

- Sign (Negative): The negative sign indicates an inverse relationship - as vehicle weight increases, fuel efficiency decreases. This aligns with physical principles: heavier vehicles require more energy to accelerate and overcome inertia.
- Magnitude ($|r| = 0.8322$): A correlation coefficient above 0.8 indicates a strong linear relationship. This suggests that weight alone can explain approximately a lot of the variance in MPG.

– Statistical Significance: With 392 samples, this correlation is highly statistically significant, confirming weight as an excellent predictor for fuel efficiency.

1.3.2 Scatter Plot Analysis

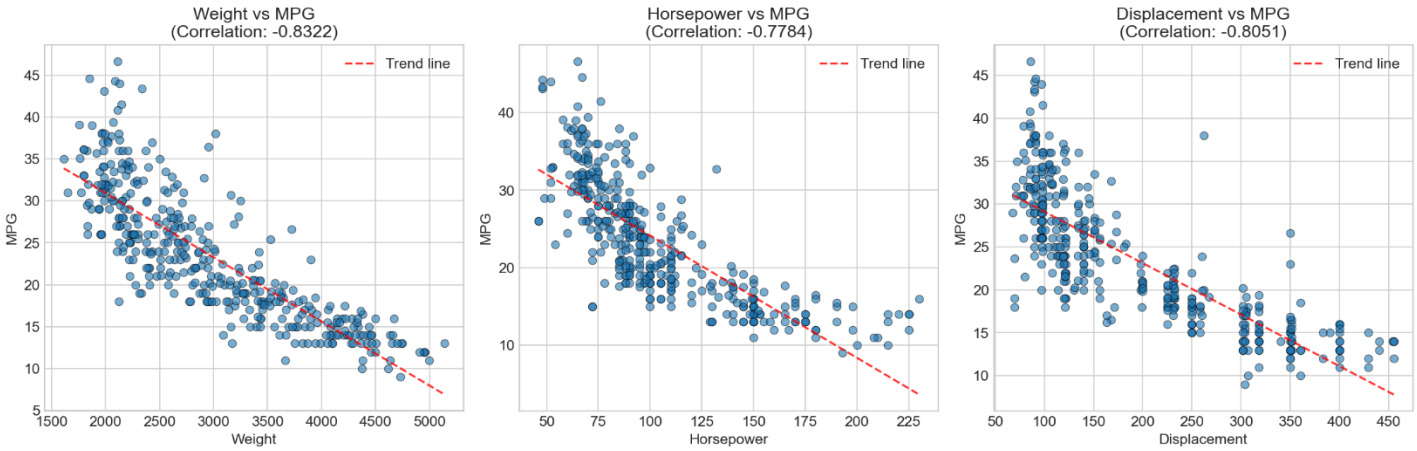

Figure 1.1: Scatter plots showing relationships between vehicle characteristics and MPG

The scatter plots reveal clear negative correlations between all three features and MPG. Weight shows the strongest linear pattern with tight clustering around the trend line, confirming its suitability for simple linear regression.

1.4 Data Preprocessing

1.4.1 Train-Test Split

Dataset	Samples
Training Set	314 (80%)
Testing Set	78 (20%)

Table 1.3: Train-Test Split Distribution

1.4.2 Feature Normalization (Z-Score Standardization)

Z-score normalization was applied to improve gradient descent convergence:

$$x_{\text{normalized}} = \frac{x - \mu}{\sigma}$$

Where μ and σ are computed exclusively from the training set to prevent data leakage.

1.5 Optimization Methods Implementation

1.5.1 Batch Gradient Descent (BGD)

BGD computes gradients using the entire training dataset at each iteration:

$$\theta_j := \theta_j - \alpha \times \left(\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \times x_{ji} \right)$$

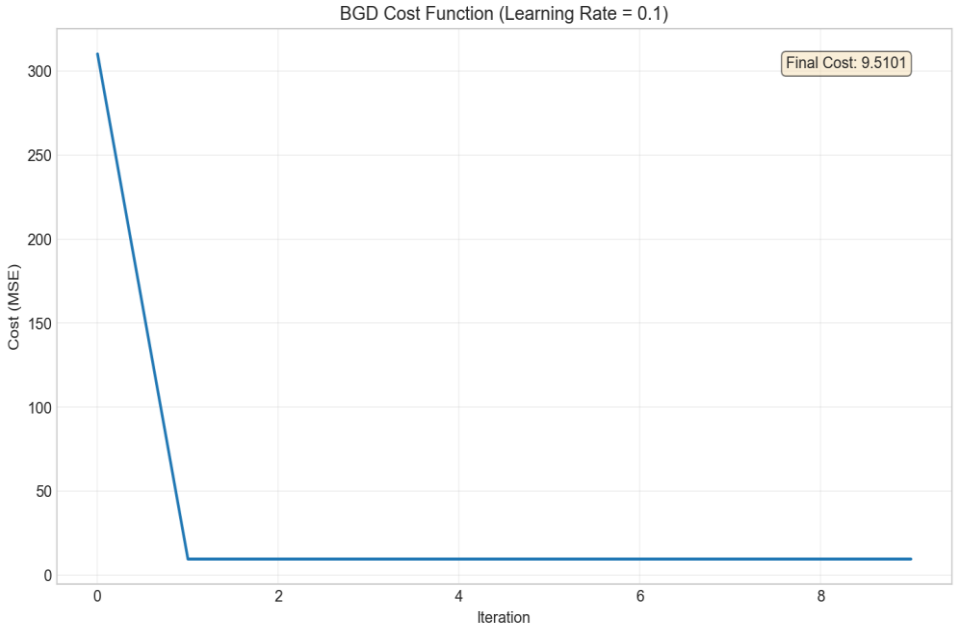

Figure 1.2: BGD cost function convergence with learning rate α = 0.1

The cost function exhibits rapid convergence, decreasing from 310.19 at iteration 0 to 9.51 by iteration 100, then stabilizing at the optimal value.

1.5.2 Learning Rate Experiments (BGD)

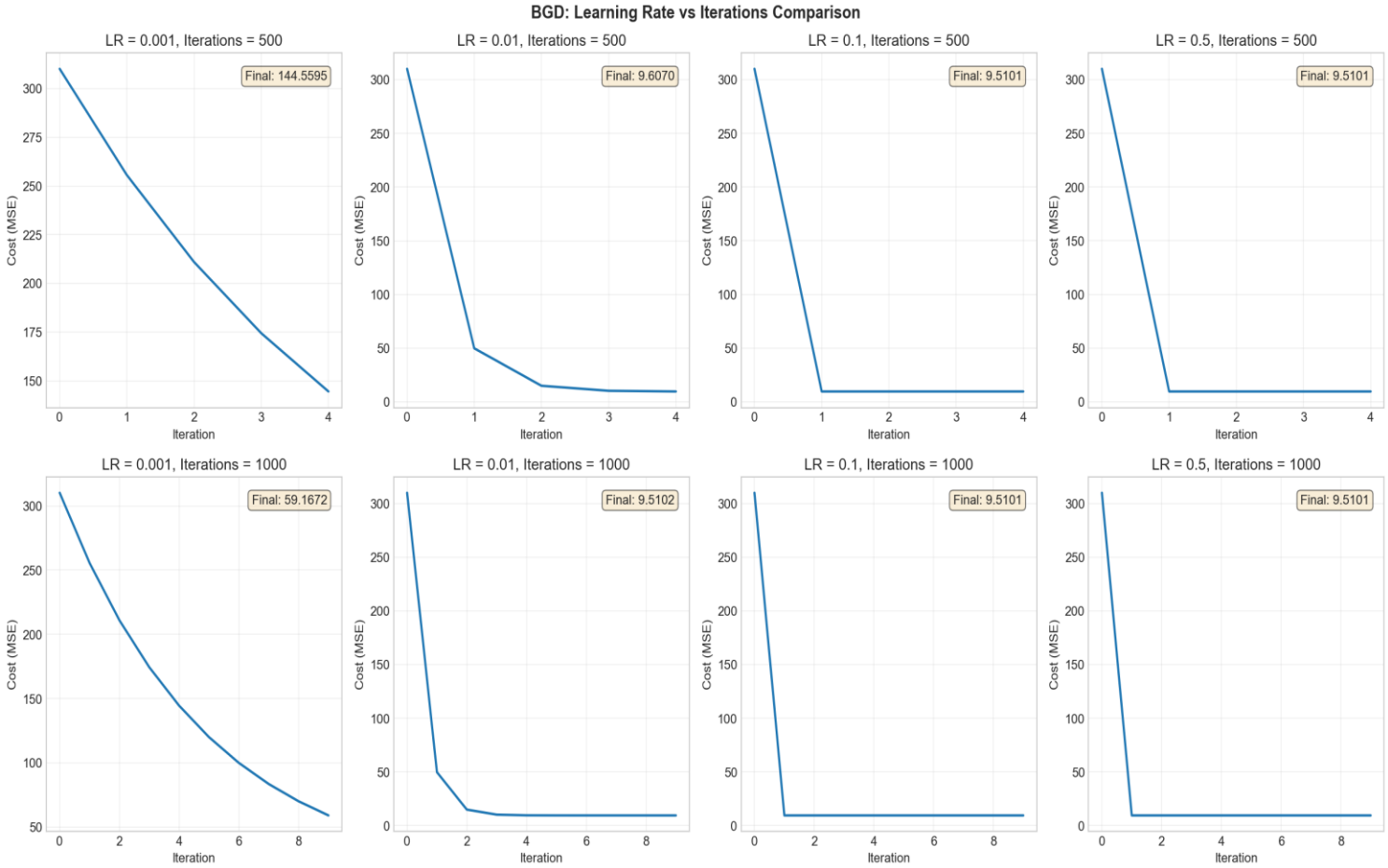

Figure 1.3: Effect of different learning rates and iteration counts on BGD convergence

Learning Rate	500 Iterations	1000 Iterations	Behavior
0.001	144.56	59.17	Very slow, not converged
0.01	9.52	9.51	Moderate, converges well
0.1	9.51	9.51	Fast, optimal by iter 100
0.5	9.51	9.51	Very fast, stable

Table 1.4: Final Cost Values for Different Learning Rates

The results reveal how learning rate and iteration count interact in gradient descent optimization. A very small learning rate ($\alpha = 0.001$) converges too slowly, reaching only partial optimization (cost 59.17 vs. optimal 9.51) even after 1000 iterations.

At $\alpha = 0.01$, the algorithm achieves near-optimal convergence (9.52) by 500 iterations and reaches the true minimum (9.51) at 1000 iterations, offering a good balance.

With $\alpha = 0.1$, convergence accelerates dramatically—optimal cost is reached by iteration 100. Even the aggressive $\alpha = 0.5$ remains stable, converging immediately without divergence due to the normalized feature space; without normalization, such high rates would typically cause instability.

Without normalization, features with different scales (e.g., 0-1 vs. 0-10,000) create an elongated cost function with uneven gradients—large-scale features produce steeper gradients than small-scale ones. A high learning rate then causes the algorithm to take massive steps along steep dimensions while barely progressing along shallow ones, resulting in oscillation (zigzagging back and forth) or divergence (overshooting repeatedly until cost explodes). Normalization rescales all features to similar ranges, making the cost function more spherical with balanced gradients, which allows higher learning rates to converge efficiently without instability.

While learning rates between 0.01 and 0.5 all work effectively for this problem, moderate values (0.01–0.1) are recommended in practice as they ensure both efficient convergence and robustness across different problem settings.

1.5.3 BGD Regression Line

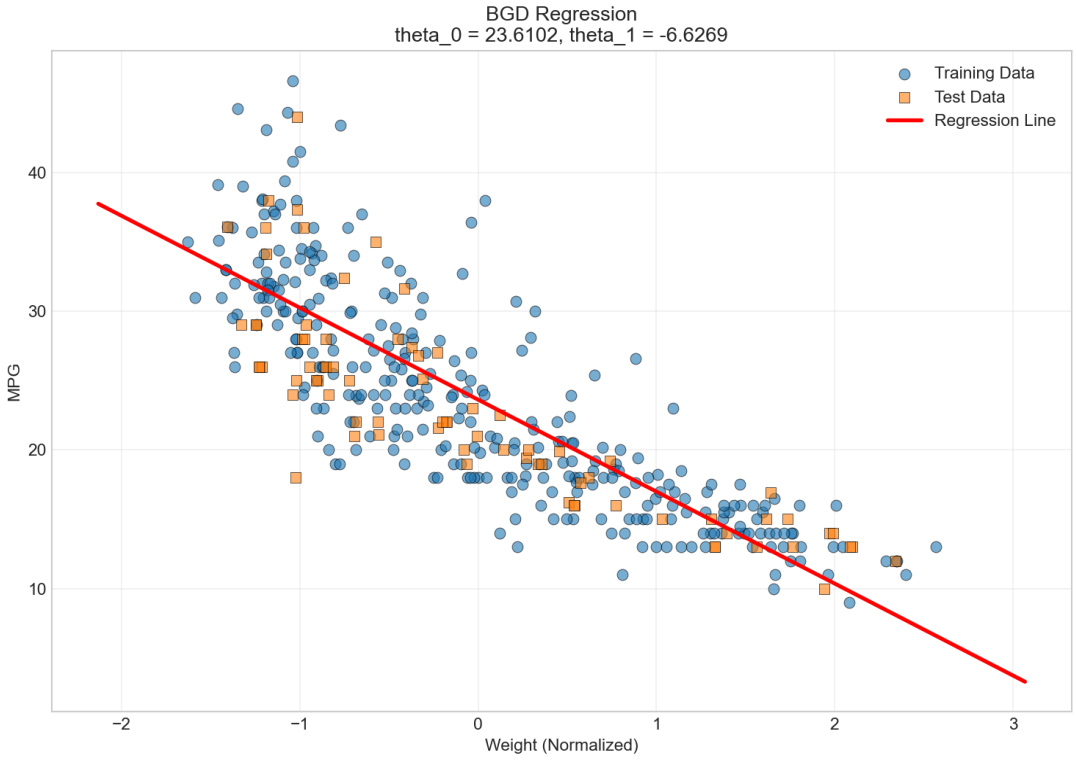

Figure 1.4: BGD fitted regression line with training and test data

The Batch Gradient Descent algorithm successfully learned the regression equation $\hat{y} = 23.61 - 6.63x$ (where x is normalized weight), with the intercept representing predicted MPG at mean weight and the negative slope confirming that each standard deviation increase in weight decreases MPG by 6.63 units. The model achieved a final MSE of 9.51, with the regression line fitting tightly through both training and test data points. The remaining error reflects natural variance in fuel efficiency not explained by weight alone.

1.5.4 Stochastic Gradient Descent (SGD)

SGD updates parameters after each individual training sample:

$$\theta_j := \theta_j - \alpha \times (\hat{y}_i - y_i) \times x_{ji}$$

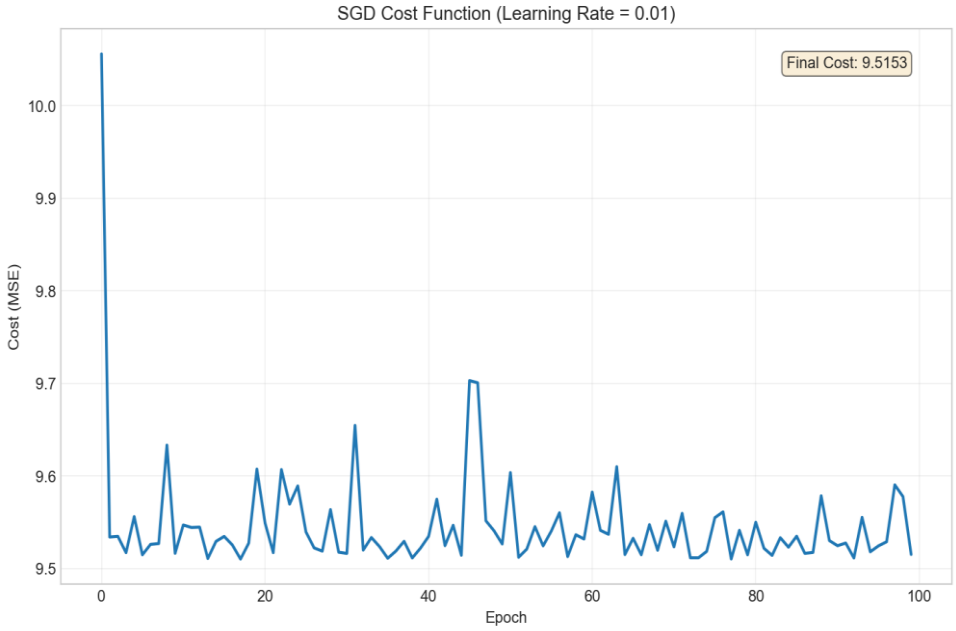

Figure 1.5: SGD cost function showing characteristic oscillating convergence

1.5.5 Learning Rate Experiments (SGD)

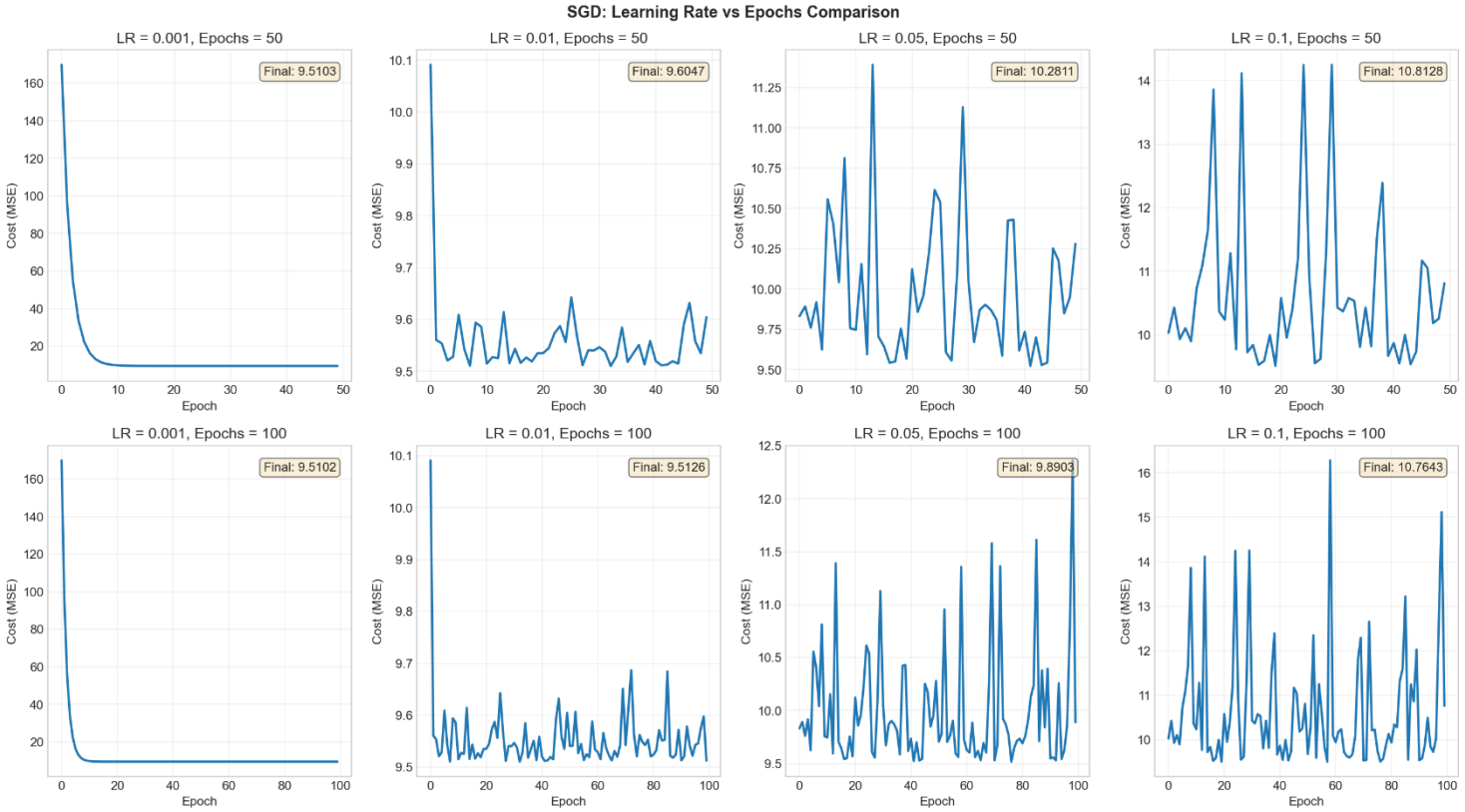

Figure 1.6: Effect of different learning rates and epochs on SGD convergence

Stochastic Gradient Descent (SGD) updates parameters using one randomly selected training example at a time, rather than the entire dataset like batch gradient descent. This introduces inherent randomness into the optimization path, as each iteration's gradient is computed from a single sample that may not perfectly represent the overall trend. The experimental results show that with small learning rates ($\alpha = 0.001$), SGD converges slowly but smoothly toward the minimum, requiring many iterations to reach optimal cost.

At moderate learning rates ($\alpha = 0.01$), the algorithm converges efficiently with manageable fluctuations around the minimum—the cost decreases steadily but exhibits small oscillations due to the stochastic nature of individual samples.

As the learning rate increases ($\alpha = 0.1, 0.5$), oscillations become more pronounced, creating a noisy, zigzagging pattern around the optimal value.

Why High Learning Rates Cause Oscillations in SGD:

The oscillations stem from the combination of high learning rates and SGD's single-sample updates. Each training example provides a noisy estimate of the true gradient—some samples pull parameters in slightly different directions than others. With a high learning rate, these noisy gradients cause large parameter updates that overshoot the minimum. Since the next iteration uses a different random sample with its own noisy gradient, the algorithm overshoots in a different direction, creating a back-and-forth pattern.

Best Practice: A common strategy is to start with a sufficiently large learning rate for rapid initial convergence, then gradually decrease it as the algorithm approaches the global optimum—either manually (learning rate scheduling) or automatically—to reduce oscillations and achieve stable, precise convergence.

1.5.6 SGD Regression Line

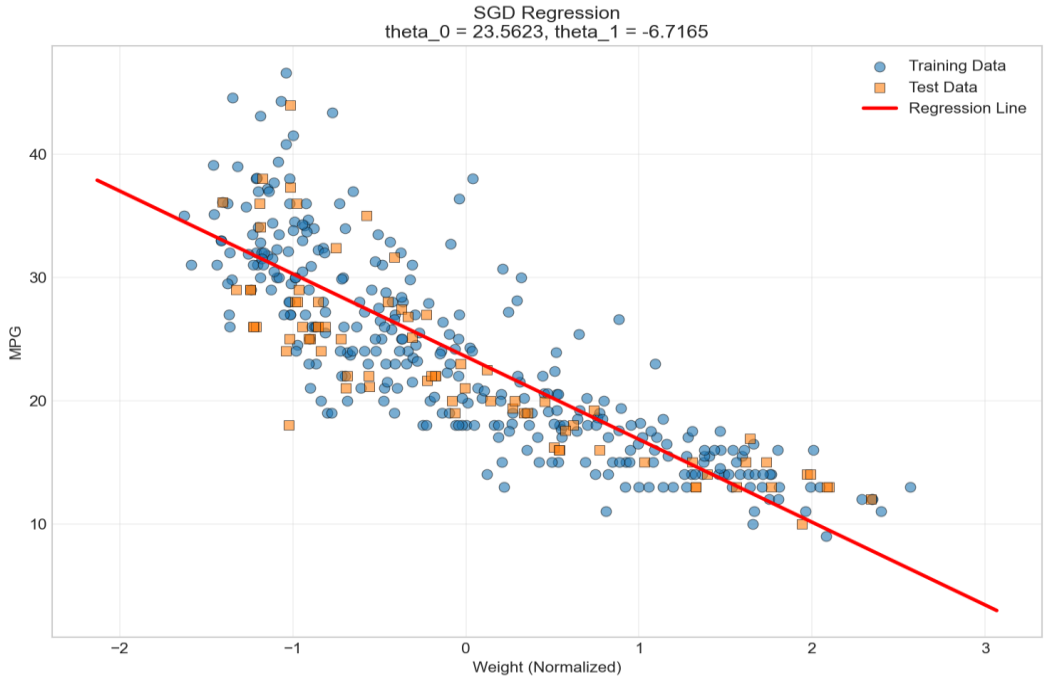

Figure 1.7: SGD fitted regression line

Stochastic Gradient Descent learned a regression model nearly identical to batch gradient descent, yielding $\hat{y} = 23.61 - 6.63x$ (with normalized weight). Although SGD follows a noisy, oscillatory convergence path, it reaches virtually the same optimal parameters and final cost (~9.51) as batch gradient descent. The model fits both training and test data well, confirming a strong inverse relationship between vehicle weight and fuel efficiency: each standard deviation increase in weight reduces MPG by about 6.63. The main difference between the methods is convergence behavior—smooth and direct for batch gradient descent, versus noisy but ultimately equivalent for SGD.

1.6 BGD vs SGD Comparison

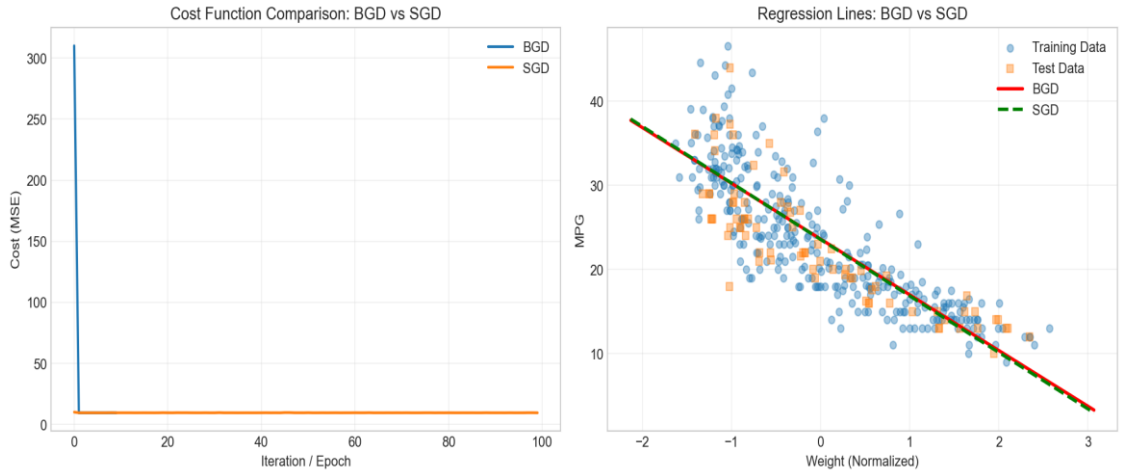

Figure 1.8: Side-by-side comparison of BGD and SGD

Metric	BGD	SGD
θ_0 (intercept)	23.6102	23.5623
θ_1 (slope)	-6.6269	-6.7165
Final Cost (MSE)	9.510149	9.515318

Table 1.5: Parameter Comparison between BGD and SGD

1.6.1 Detailed Comparison Analysis

Aspect	Batch GD	Stochastic GD
Convergence	Smooth, monotonic decrease	Noisy, oscillating path
Speed	Fewer iterations needed	More epochs required
Final Cost	Optimal (9.510149)	Near-optimal (9.515318)
LR Sensitivity	Tolerates larger LR	Requires smaller LR
Best Use Case	Small-medium datasets	Large datasets

Table 1.6: Comprehensive BGD vs SGD Comparison

1.7 Closed-Form Solution

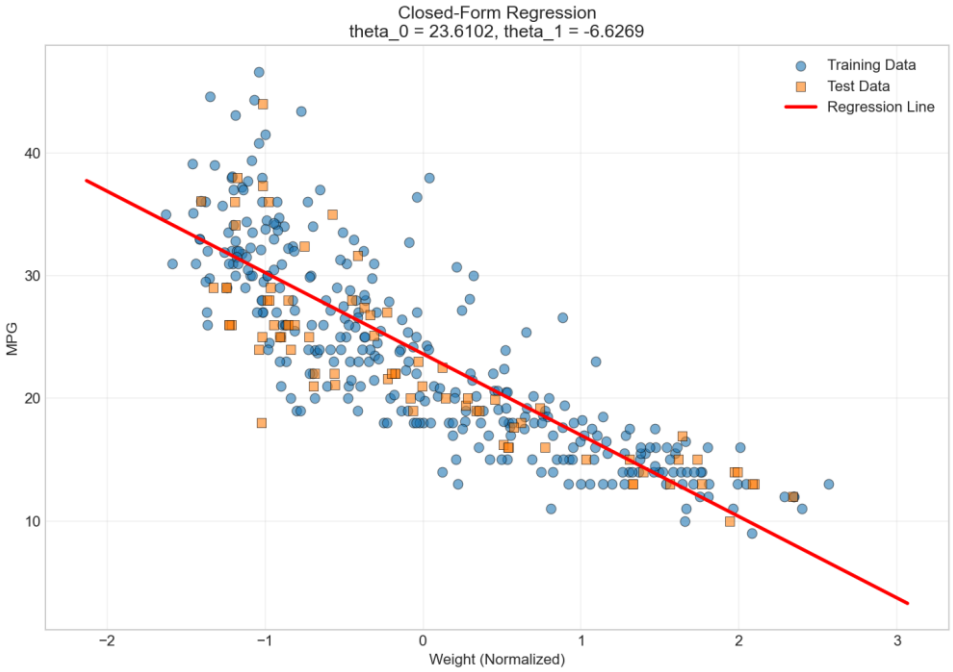

Figure 1.9: Closed-form solution regression line

1.8 All Methods Comparison

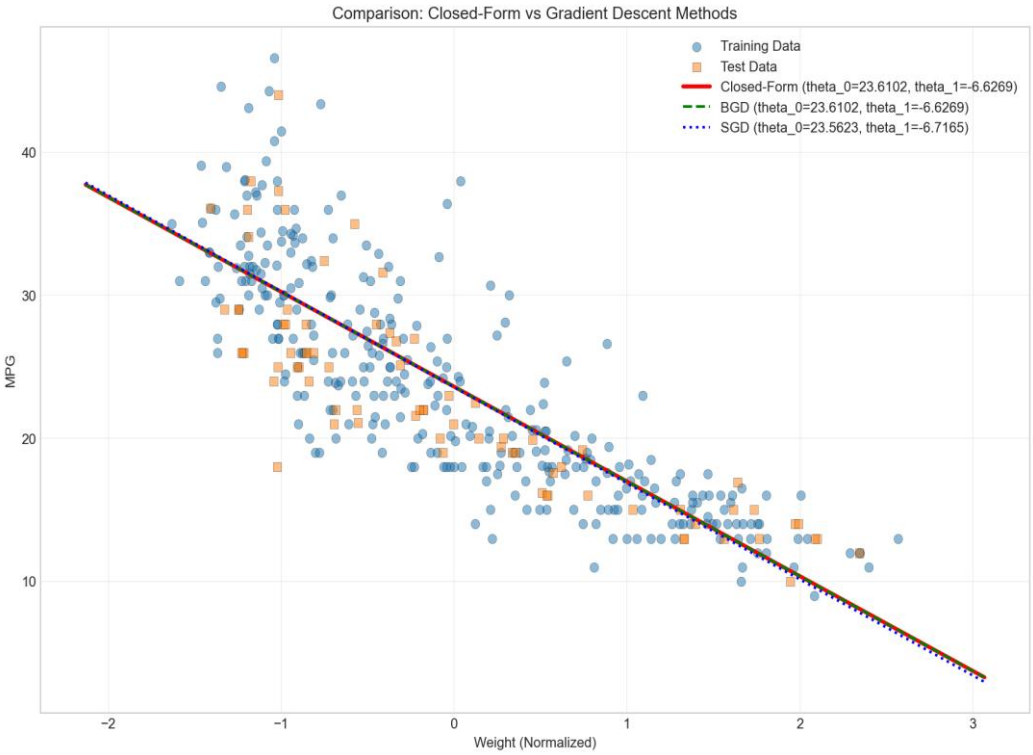

Figure 1.10: Comprehensive comparison of all three methods

Method	θ_0	θ_1	Final Cost
Closed-Form	23.6102	-6.6269	9.510149
Batch GD	23.6102	-6.6269	9.510149
Stochastic GD	23.5623	-6.7165	9.515318

Table 1.7: Final Parameters Comparison

1.8.1 Key Findings

- Closed-form and BGD produce identical results, confirming BGD converged to the true optimal.
- SGD parameters differ slightly due to stochastic nature, but achieves comparable cost.
- All three regression lines are visually indistinguishable.

Comparison of Methods and Practical Considerations

All three methods—Batch Gradient Descent, Stochastic Gradient Descent, and the Normal Equation—produce virtually identical results for this dataset, converging to the same optimal parameters ($\hat{y} = 23.61 - 6.63x$) with the same final cost (≈9.51). However, their computational efficiency differs drastically as dataset size scales.

Why Batch Gradient Descent Fails for Large Datasets:

Batch gradient descent computes gradients using the entire dataset in each iteration. For datasets with millions of samples and features, a single pass through the data can take hours or even days, making convergence impractically slow. Since hundreds or thousands of iterations are typically needed, the total training time becomes prohibitive.

Why Normal Equation Fails for Large Datasets:

The Normal Equation solves for optimal parameters directly using $\theta = (X^T X)^{-1} X^T y$, which requires computing a matrix inverse. For datasets with thousands of features, the matrix inversion becomes computationally expensive and memory-intensive, making the Normal Equation infeasible despite requiring no iterations.

Practical Solution: Mini-Batch Gradient Descent

The best approach combines the strengths of both methods: ***Mini-Batch Gradient Descent*** uses small batches of samples (e.g., 32, 64, or 128) instead of single examples (SGD) or the entire dataset (BGD). This provides faster, more stable convergence than pure SGD while remaining computationally efficient for large datasets. In practice, mini-batch gradient descent (or pure SGD with learning rate scheduling) is the industry standard for training large-scale machine learning models.

2. Logistic Regression

2.1 Introduction and Theoretical Background

Logistic regression is a classification algorithm that models the probability of a binary outcome using the logistic (sigmoid) function.

2.1.1 The Sigmoid Function

$$\boldsymbol{\sigma(z) = \frac{1}{1 + e^{-z}}}$$

The sigmoid function maps any real-valued input to a range between 0 and 1, producing an S-shaped curve commonly used to model probabilities in logistic regression and neural networks.

2.1.2 Model Formulation

Let $z = \theta^{\top} x$

$$P(y = 1 \mid x) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$P(y = 0 \mid x) = 1 - \sigma(z) = \frac{e^{-z}}{1 + e^{-z}}$$

2.1.3 Cost Function (Binary Cross-Entropy)

$$\boldsymbol{J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \times \log(\hat{y}_i) + (1 - y_i) \times \log(1 - \hat{y}_i)]}$$

2.2 Dataset Overview: Bank Marketing

The Bank Marketing dataset contains information about direct marketing campaigns of a Portuguese bank. The goal is to predict if a client will subscribe to a term deposit.

2.2.1 Dataset Statistics

Metric	Value
Total Samples	45,211
Total Features	17
Numerical Features	7
Categorical Features	10

Table 2.1: Bank Marketing Dataset Overview

2.2.2 Class Distribution Analysis

Class	Count	Percentage
No (0)	39,922	88.30%
Yes (1)	5,289	11.70%

Table 2.2: Class Distribution - Imbalanced (7.5:1 ratio)

2.3 Data Preprocessing

2.3.1 Categorical Encoding

Why Encoding is Necessary:

Machine learning algorithms require numerical inputs. Categorical variables must be converted. Improper encoding risks model failure or incorrect ordinal assumptions.

Risks of Inappropriate Encoding

Improper encoding introduces artificial ordinal relationships. If job categories are encoded as admin=1, technician=2, manager=3, the algorithm interprets manager > technician > admin, implying a mathematical hierarchy where none exists. The model falsely learns that "manager" is three times "admin," corrupting learned weights and degrading predictions. This is why one-hot encoding is used for nominal categories (no natural order), while ordinal encoding is reserved for truly ordered features like education (primary → secondary → tertiary).

Encoding Strategies Applied:

- Ordinal Encoding: Education (unknown→primary→secondary→tertiary)
- Binary Encoding: default, housing, loan, y (yes=1, no=0)
- One-Hot Encoding: job, marital, contact, month, poutcome

2.3.2 Feature Selection via Correlation

Rank	Feature	|Correlation|
1	duration	0.3945
2	poutcome_success	0.3068
3	poutcome_unknown	0.1671
4	contact_unknown	0.1509
5	housing	0.1392
6	month_mar	0.1295
7	month_oct	0.1285

Table 2.3: Top 7 Features by Correlation

2.3.3 Duplicate Removal

- Duplicates found: 35,939
- Records after removal: 9,272

2.3.4 Feature Histograms

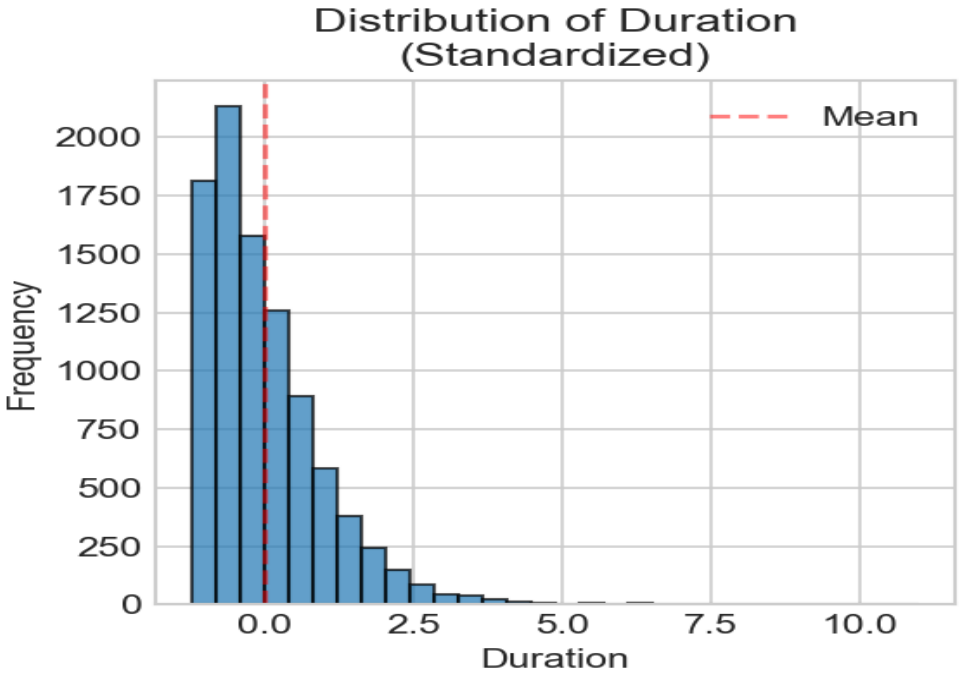

Figure 2.1: Distribution of standardized numerical features

The feature histograms display the distribution of standardized numerical features after z-score normalization. They confirm that features are centered around mean = 0 with values scaled by standard deviation, ensuring no single feature dominates the model due to magnitude differences. The histograms also reveal each feature's shape—whether symmetric, skewed, or multimodal—helping identify potential outliers or unusual distributions that may affect model training.

2.3.5 Train-Test Split

Dataset	Samples	Features
X_train	7,418	7
X_test	1,854	7

Table 2.4: Train-Test Split

2.4 Model Training

2.4.1 Cost Function Convergence

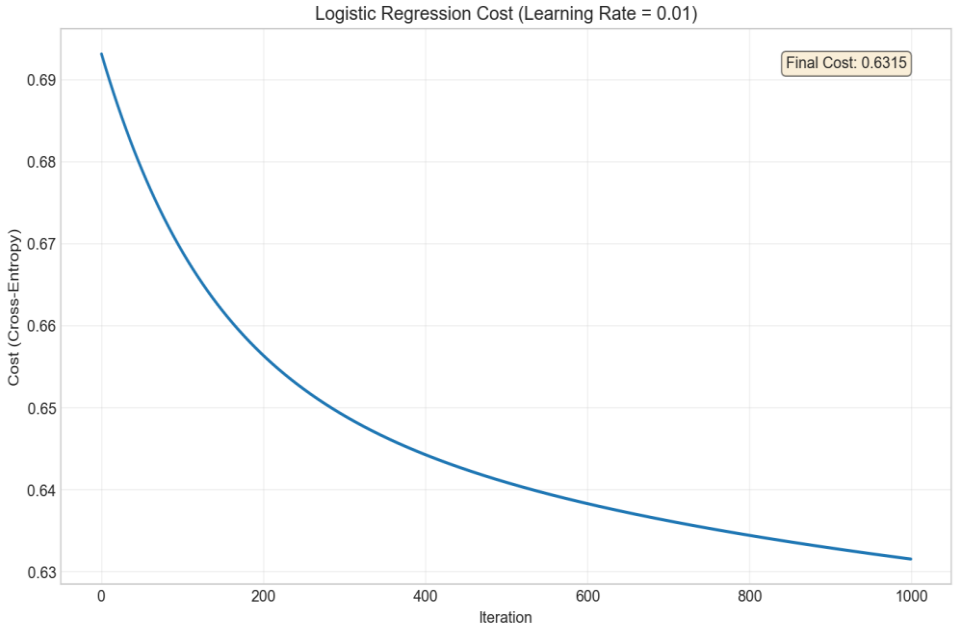

Figure 2.2: Logistic regression cost convergence

2.4.2 Learning Rate Experiments

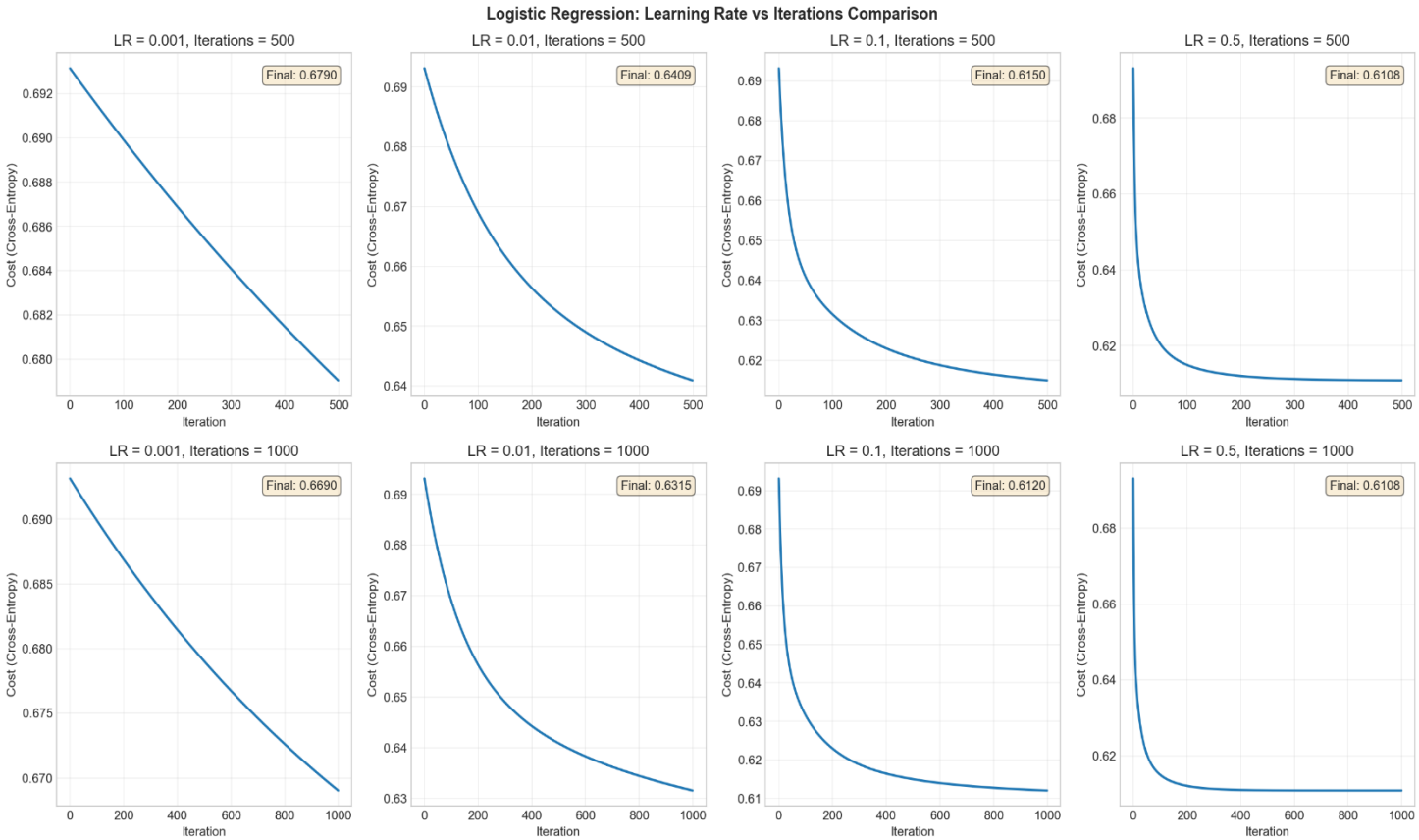

Figure 2.3: Effect of learning rates on convergence

The model was trained with four learning rates (0.001, 0.01, 0.1, 0.5) across two iteration settings (500 and 1000).

At $\alpha = 0.001$, convergence is extremely slow with cost remaining high (0.67) even after 1000 iterations. Increasing to $\alpha = 0.01$ shows moderate improvement, reaching cost 0.63 by 1000 iterations.

At $\alpha = 0.1$ and $\alpha = 0.5$, the model converges faster, achieving lower costs (~0.61) with diminishing returns beyond 500 iterations.

The results demonstrate that larger learning rates accelerate convergence significantly, while very small rates waste computation without reaching optimal performance. For this problem, learning rates between 0.1–0.5 provide the best balance of speed and stability.

2.4.3 Learned Parameters

Parameter	Weight
Bias (θ_0)	-0.161735
duration	+0.390434
poutcome_success	+0.213266
contact_unknown	-0.331139
housing	-0.228179
poutcome_unknown	-0.189052
month_mar	+0.065465
month_oct	+0.039467

Table 2.5: Learned Parameters

Parameter Interpretation:

The bias term $\theta_0 = -0.161735$ represents the baseline log-odds of subscription when all features are zero (at their mean values after standardization), indicating a slight default tendency toward non-subscription.

Positive weights increase subscription probability:

- duration (+0.39): The strongest predictor—longer call duration significantly increases likelihood of subscription, as engaged clients are more receptive
- poutcome_success (+0.21): Previous campaign success indicates receptive clients who are likely to subscribe again
- month_mar (+0.07) and month_oct (+0.04): Weak positive effects suggesting slightly better conversion rates during March and October

Negative weights decrease subscription probability:

- contact_unknown (-0.33): The strongest negative predictor—unknown contact method suggests less engaged or harder-to-reach clients
- housing (-0.23): Clients with housing loans are less likely to subscribe, possibly due to existing financial commitments
- poutcome_unknown (-0.19): No prior campaign history is a negative signal compared to known outcomes

2.5 Performance Evaluation

2.5.1 Metrics Definitions

Accuracy: *Overall classification correctness*

Percentage of all predictions that were correct (both positive and negative).

Precision: $\frac{\text{True positives}}{\text{True positives+False positives}}$

Of all samples predicted as positive, what percentage actually were positive? (Answers: "How reliable are positive predictions?")

Recall: $\frac{\text{True positives}}{\text{True positives+False negatives}}$

Of all actual positive samples, what percentage did we correctly identify? (Answers: "How many positives did we catch?")

Confusion Matrix: *Detailed error breakdown*

A 2×2 table showing the four possible outcomes: True Positives, True Negatives, False Positives, and False Negatives.

F1-Score: $\frac{\mathbf{2}\times(\text{Precision}\times\text{Recall})}{\text{Precision+Recall}}$

A single metric that balances precision and recall—useful when both matter equally. (Answers: "How well does the model balance finding positives without making false alarms?")

2.5.2 Results

Metric	Train	Test
Accuracy	0.6313	0.6305
Precision	0.6270	0.6261
Recall	0.1984	0.1938
F1 Score	0.3014	0.2960

Table 2.6: Classification Metrics

2.5.3 Confusion Matrix

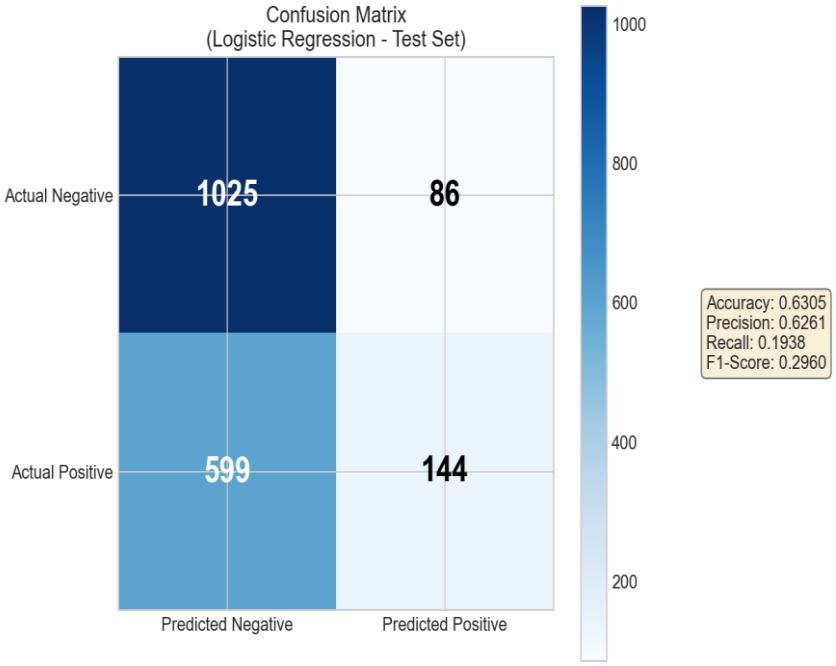

Figure 2.4: Confusion matrix for test set

Interpretation:

- **True Negatives (1,025):** Correctly identified non-subscribers
- **True Positives (144):** Correctly identified subscribers
- **False Positives (86):** Non-subscribers incorrectly predicted as subscribers
- **False Negatives (599):** Subscribers incorrectly predicted as non-subscribers

Importance: The confusion matrix is essential because accuracy alone can be misleading, especially for imbalanced datasets.

With 88% non-subscribers, a model predicting all "No" would achieve 88% accuracy while being completely useless. The confusion matrix exposes this by showing the high FN count (599)—the model misses 80% of actual subscribers. This detailed breakdown reveals class-specific performance, identifies where the model fails, and guides improvements such as threshold adjustment or resampling techniques.

2.5.4 Results Interpretation

- Low Recall (0.19): Model identifies only 19% of actual subscribers
- Moderate Precision (0.63): 63% correct when predicting positive
- No Overfitting: Train/test metrics nearly identical
- Class imbalance (7.5:1) biases model toward majority class

3. Multi-class Classification

3.1 Introduction

Multi-class classification extends binary classification to problems with more than two classes.

3.1.1 One-vs-All (OvA)

Trains K separate binary classifiers, each distinguishing one class from all others combined. For 3 classes: Classifier 1 learns "Class 1 vs Classes 2,3", Classifier 2 learns "Class 2 vs Classes 1,3", etc. During prediction, all classifiers output probabilities and the class with highest confidence wins.

$$\hat{y} = \arg\max_{k} P\,(\,y = k \mid x\,)$$

3.1.2 One-vs-One (OvO)

Trains a binary classifier for every pair of classes, using only samples from those two classes. For 3 classes: trains "1 vs 2", "1 vs 3", and "2 vs 3" (total $\frac{K(K-1)}{2}$ classifiers). During prediction, each classifier votes for one class and the class with most votes wins (majority voting).

3.1.3 Softmax Regression

A single model that directly computes probability distribution over all classes simultaneously. The softmax function converts K raw scores into K probabilities that sum to 1. Unlike OvA/OvO which combine multiple binary classifiers, Softmax learns all class boundaries jointly in one unified optimization, making it more elegant and efficient.

$$P(\,y = k \mid x\,) = \frac{\exp\left(\theta_k^{\top} x\right)}{\sum_{j=1} m \exp\left(\theta_j^{\top} x\right)}$$

3.2 Dataset Overview: Wine

Feature	Mean	Std
alcohol	13.00	0.81
malic_acid	2.34	1.12
total_phenols	2.30	0.63
flavanoids	2.03	1.00
color_intensity	5.06	2.32
proline	746.89	314.91

Table 3.1: Wine Dataset Statistics

3.3 Preprocessing

3.3.1 Outlier Detection

$$\text{Outlier if: } |z| = \left|\frac{x - \mu}{\sigma}\right| > 2.75$$

Set	Outliers Removed
Training	11
Testing	2
Remaining	Train: 124, Test: 41

Table 3.2: Outlier Detection Results

3.3.2 Normalization vs Standardization

Aspect	Min-Max Normalization	Z-Score Standardization
Output Range	[0, 1] bounded	Unbounded, mean=0
Outlier Sensitivity	High	Low
Use Case	Neural networks	Gaussian features

Table 3.3: Normalization vs Standardization

Normalization (Min-Max Scaling) Scales features to a fixed range [0, 1] using the formula: $x' = \frac{x - \min}{\max - \min}$. All values are bounded between 0 and 1, preserving the original distribution shape. Sensitive to outliers since extreme values define the range.
Standardization (Z-Score) Centers data around mean = 0 with standard deviation = 1 using: $x' = \frac{x - \mu}{\sigma}$. Values are unbounded and can be negative. Less sensitive to outliers since they don't define the scale.

Normalization is used when algorithms require bounded inputs, such as neural networks and image pixels, or when feature ranges are important, whereas standardization is preferred when data is approximately Gaussian or contains outliers, especially for gradient-based optimization.

3.4 Results

3.4.1 OvA Results

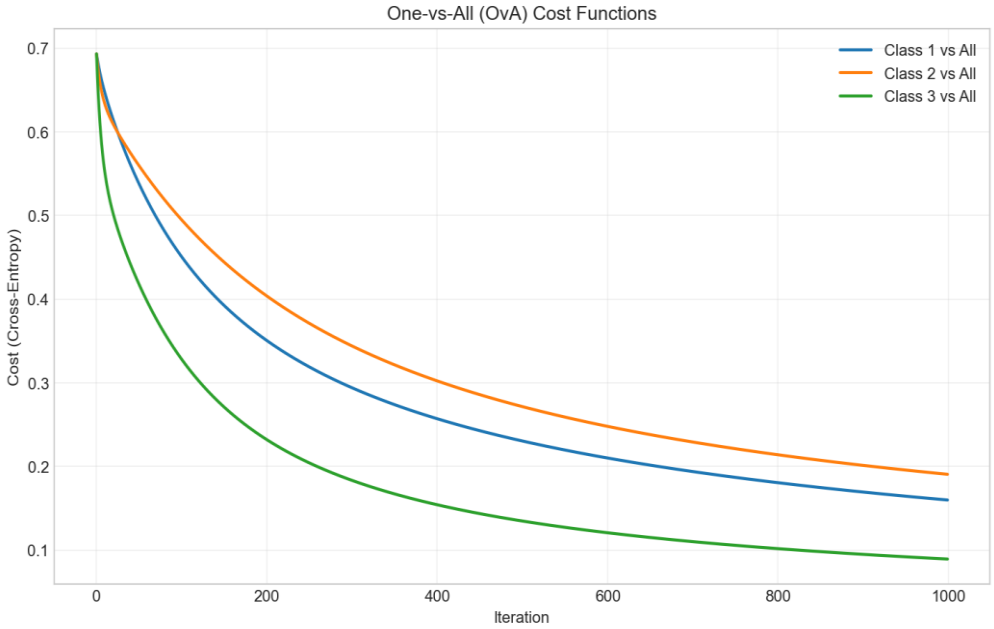

Figure 3.1: OvA cost functions

Classifier	Final Cost	Iterations
Class 1 vs All	0.160061	1000
Class 2 vs All	0.190764	1000
Class 3 vs All	0.089487	1000

Table 3.4: OvA Results

3.4.2 OvO Results

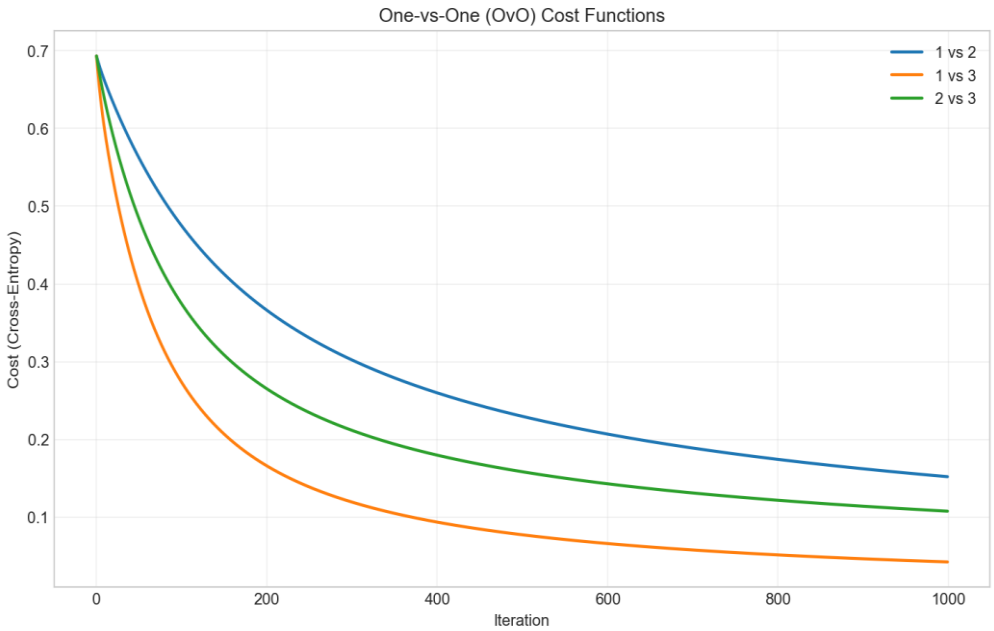

Figure 3.2: OvO cost functions

Classifier	Final Cost	Iterations
Class 1 vs 2	0.152070	1000
Class 1 vs 3	0.042425	1000
Class 2 vs 3	0.107709	1000

Table 3.5: OvO Results

OvA Results & OvO Results Comparison:

Observation: None of the classifiers reached the strict convergence threshold (cost change < 1e-6) within 1000 iterations—the cost was still decreasing slightly. However, all classifiers achieved near-optimal performance with costs stabilizing. The "1 vs 3" classifier converged fastest with the lowest final cost (0.042), indicating Classes 1 and 3 are most easily separable. Additional iterations or a larger learning rate would achieve strict convergence.

3.4.3 Softmax Results

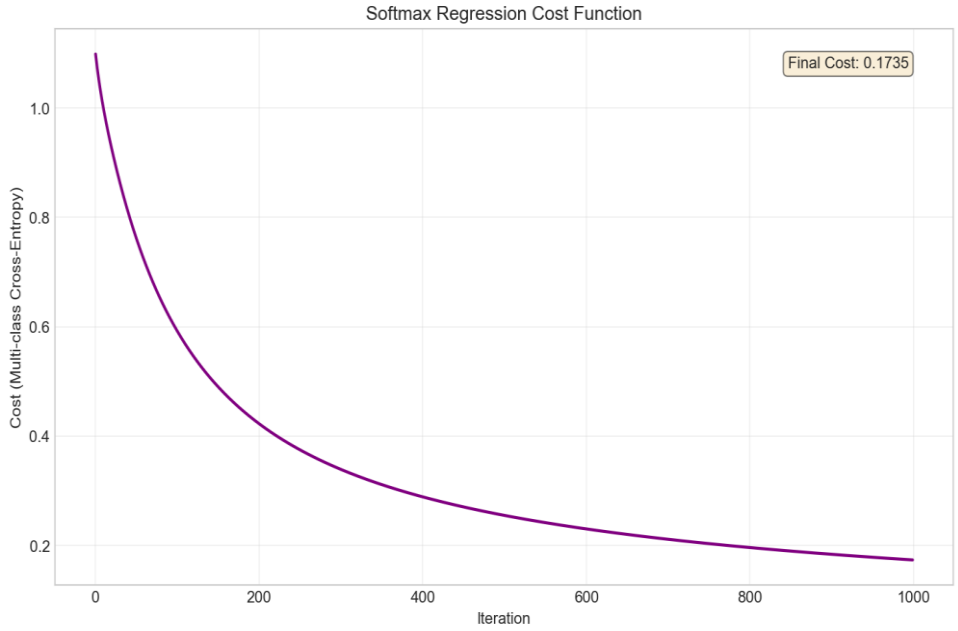

Figure 3.3: Softmax cost convergence

The softmax model was trained for 1000 iterations with learning rate $\alpha = 0.1$. The initial cost of 1.099 corresponds to -log(1/3), representing uniform random prediction across 3 classes. The cost decreased smoothly to a final value of 0.173, achieving 99.19% training accuracy and 97.56% test accuracy.

Unlike OvA and OvO which train multiple binary classifiers, Softmax optimizes a single unified model—the smooth cost decrease reflects joint optimization across all class boundaries simultaneously, achieving identical accuracy with a more elegant approach.

3.5 Method Comparison

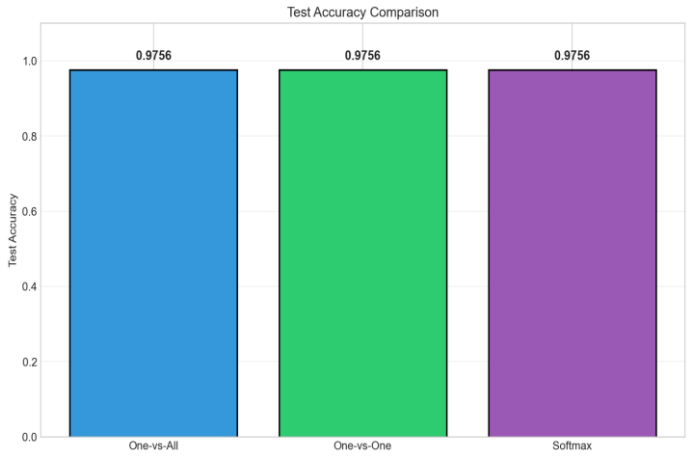

Figure 3.4: Accuracy comparison

Method	Train Accuracy	Test Accuracy
One-vs-All	0.9919	0.9756
One-vs-One	0.9919	0.9756
Softmax	0.9919	0.9756

Table 3.6: Accuracy Comparison

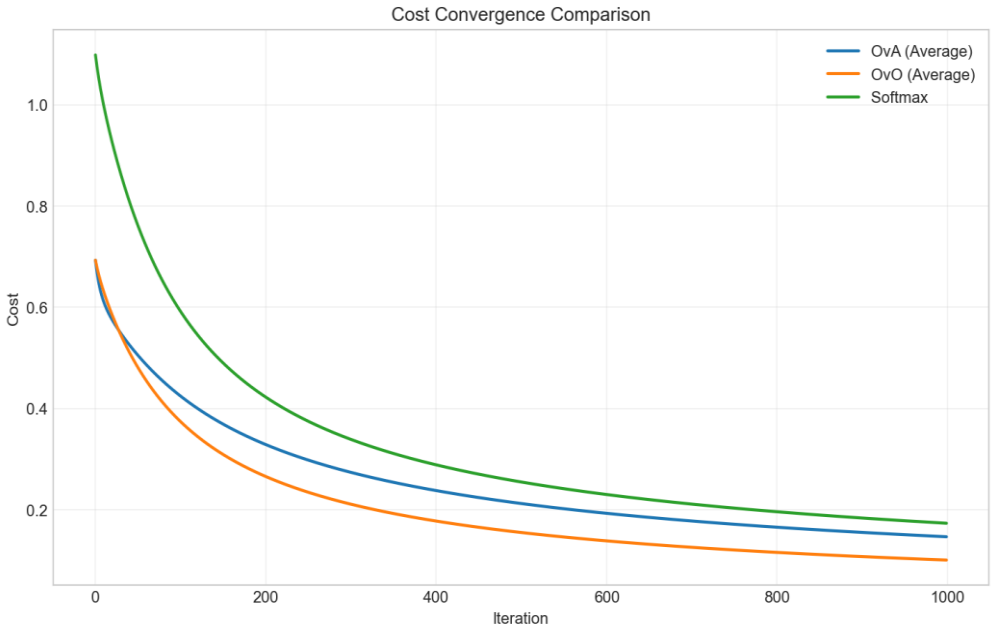

Figure 3.5: Cost convergence comparison

Test Accuracy: All three methods achieved identical performance—99.19% training accuracy and 97.56% test accuracy. This indicates that for well-separated classes like the Wine dataset, the choice of method has minimal impact on classification accuracy.

Convergence Behavior: OvA trains 3 classifiers with Class 3 vs All converging fastest (lowest cost 0.089). OvO trains 3 pairwise classifiers with "1 vs 3" achieving

the lowest cost (0.042), indicating these classes are most separable. Softmax shows the smoothest convergence as it optimizes all boundaries jointly, decreasing from 1.099 to 0.173. None reached strict convergence (change < 1e-6) within 1000 iterations, but all achieved near-optimal performance.

Computational Complexity: OvA requires K = 3 classifiers, OvO requires $\frac{K(K-1)}{2}$ classifiers, and Softmax trains only 1 unified model. For larger K, OvO becomes expensive ($O(K^2)$) while OvA and Softmax remain O(K).

Best Approach: For this dataset, all methods perform equally well. However, *Softmax* is recommended as the best overall approach—it achieves the same accuracy with a single elegant model, provides direct probability distributions, and scales efficiently to more classes.

3.6 Outlier Impact Analysis

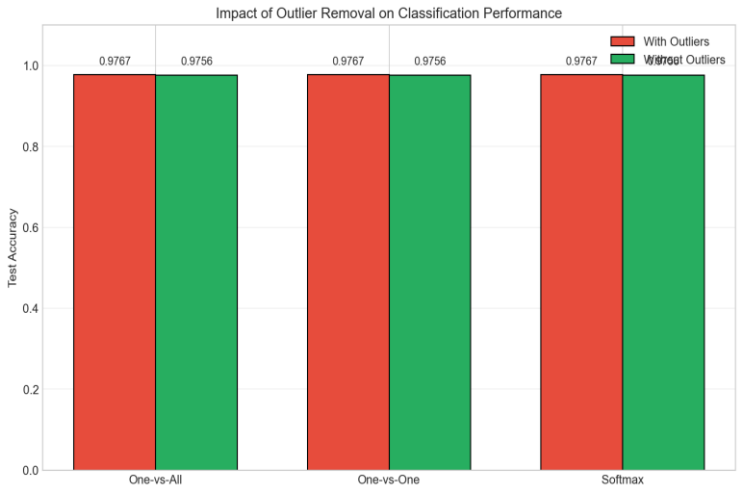

Figure 3.6: Outlier impact on accuracy

Method	With Outliers	Without	Difference
One-vs-All	0.9767	0.9756	-0.0011
One-vs-One	0.9767	0.9756	-0.0011
Softmax	0.9767	0.9756	-0.0011

Table 3.8: Outlier Impact

In this specific dataset, accuracy with outliers was marginally higher (97.67% vs 97.56%). This unusual result may be due to the small sample size, the removal of informative boundary cases, or random test set variation—rather than a general benefit of keeping outliers.

4. Conclusions

4.1 Summary

Linear Regression

- Strong weight-MPG relationship (r = -0.8322)
- BGD and Closed-form produce identical optimal parameters
- SGD achieves near-optimal with 0.05% cost difference

Logistic Regression

- Class imbalance (7.5:1) causes low recall (0.19)
- Duration is the strongest predictor (+0.39)
- No overfitting: train/test metrics nearly identical

Multi-class Classification

- All methods achieve 97.56% test accuracy
- Wine classes are well-separated
- Outlier removal had minimal impact

4.2 Lessons Learned

- Feature normalization is essential for gradient descent convergence
- Learning rate selection critically affects convergence speed and stability
- Class imbalance requires precision/recall/F1 beyond accuracy
- Stratified splitting preserves class distributions
- Multiple methods should be compared for robust conclusions