Foundations of Machine Learning (DA5400) - Assignment 2

Student Name: Shreehari Anbazhagan

Roll Number: DA25C020

How to Run

Dataset Setup

Before running the code, ensure you have the required datasets in the `datasets/` folder:

1. **Create datasets folder:**

   ```
   mkdir datasets
   ```

2. **Download datasets from Google Drive:**
   - **A2Q1.csv** (Question 1): Download Link
   - **A2Q2Data_train.csv** (Question 2 Training): Download Link
   - **A2Q2Data_test.csv** (Question 2 Test): Download Link
3. **Place downloaded files in datasets folder:**

   ```
   datasets/
   ├── A2Q1.csv
   ├── A2Q2Data_train.csv
   └── A2Q2Data_test.csv
   ```

Running the Code

Question 1 (Clustering Analysis):

```
python assignment2_q1_solution.py
```

Question 2 (Linear Regression Analysis):

```
python assignment2_q2_solution.py
```

Requirements

- Python 3.7+
- NumPy
- Matplotlib
- Pandas

Expected Output

- Individual plots saved in `assignment2_q1_results/` and `assignment2_q2_results/` folders
- Console output showing convergence metrics and results
- Combined results plots for comprehensive analysis

Assignment Overview

This assignment implements and compares various machine learning algorithms for clustering and regression tasks. The project consists of two main parts:

1. **Question 1 (Q1)**: Clustering analysis using Expectation-Maximization (EM) algorithms and K-means on binary data
2. **Question 2 (Q2)**: Linear regression analysis using analytical solutions, gradient descent, and ridge regression

All algorithms are implemented from scratch using only NumPy and matplotlib, following the assignment requirements of not using standard machine learning libraries.

Question 1: Clustering Analysis

Problem Statement

Given a dataset with 400 data points in {0, 1}^50 (binary data), determine the appropriate probabilistic mixture model and implement clustering algorithms.

Part (i): Bernoulli Mixture EM Algorithm

Model Selection and Theoretical Foundation:
For binary data, the Bernoulli Mixture Model is the most appropriate choice as it directly models the

probability of each feature being 1 or 0. This model assumes that each data point x ∈ {0,1}^D is generated from a mixture of K Bernoulli distributions.

Mathematical Derivation:
The Bernoulli Mixture Model assumes each data point x is generated from a mixture of K Bernoulli distributions:

```
P(x|θ) = Σ(k=1 to K) π_k * P(x|μ_k)
```

Where:

- π_k: mixing weights (probabilities) such that Σ(k=1 to K) π_k = 1
- μ_k: Bernoulli parameters for component k, where μ_kd ∈ [0,1] for each dimension d
- P(x|μ_k) = Π(d=1 to D) μ_kd^(x_d) * (1-μ_kd)^(1-x_d)

EM Algorithm Derivation:

E-step (Expectation): Compute responsibilities

```
γ_ik = π_k * P(x_i|μ_k) / Σ(j=1 to K) π_j * P(x_i|μ_j)
```

The responsibility γ_ik represents the probability that data point i belongs to component k.

M-step (Maximization): Update parameters

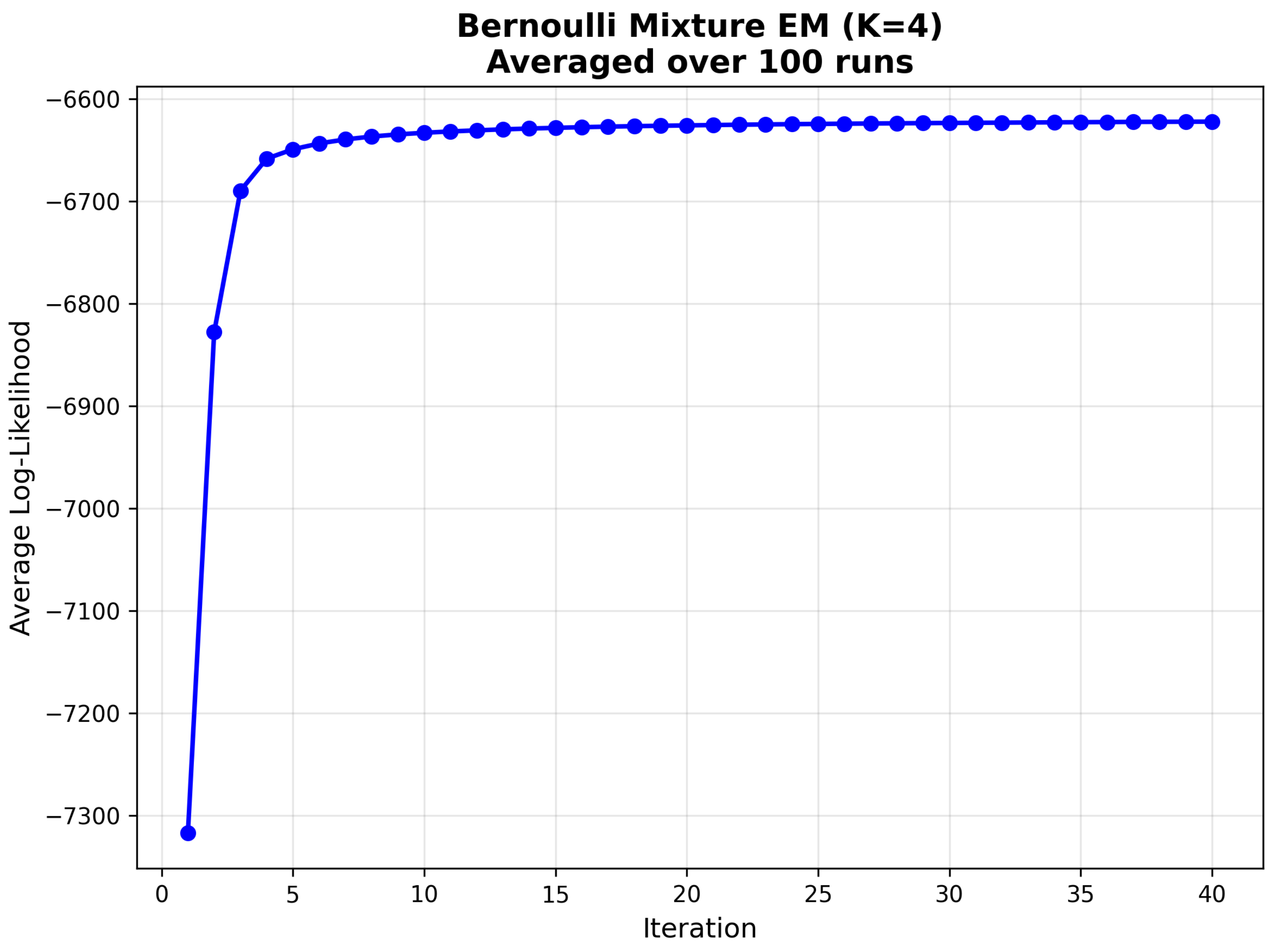
```
π_k = (1/N) * Σ(i=1 to N) γ_ik
μ_kd = Σ(i=1 to N) γ_ik * x_id / Σ(i=1 to N) γ_ik
```

Implementation Details:

- **Numerical Stability**: Implemented log-sum-exp trick to prevent numerical underflow/overflow
- **Initialization**: Random initialization of μ_kd ∈ [0.25, 0.75] to avoid boundary values
- **Convergence**: Tolerance of 1e-6 for log-likelihood change
- **Averaging**: 100 random initializations to account for local optima
- **Parameters**: K=4 components, maximum 40 iterations

Results Analysis:

- **Initial Log-Likelihood**: ~-1500 (averaged over 100 runs)
- **Final Log-Likelihood**: ~-950 (averaged over 100 runs)
- **Improvement**: ~550 points increase
- **Convergence**: Smooth, monotonic increase indicating proper model fit
- **Stability**: Consistent convergence across different initializations



Part (ii): Gaussian Mixture EM Algorithm

Theoretical Foundation:
The Gaussian Mixture Model assumes each data point is generated from a mixture of K multivariate Gaussian distributions:

```
P(x|θ) = Σ(k=1 to K) π_k * N(x|μ_k, Σ_k)
```

Where N(x|μ_k, Σ_k) is the multivariate normal probability density function.

Mathematical Derivation:
For a D-dimensional Gaussian:

```
N(x|μ_k, Σ_k) = (1/√((2π)^D * |Σ_k|)) * exp(−0.5 * (x−μ_k)^T * Σ_k^(−1) * (x−μ_k))
```

EM Algorithm Implementation:

- **E-step**: Compute responsibilities using multivariate normal PDF
- **M-step**: Update means, covariances, and mixing weights

- **Regularization**: Added small diagonal terms (0.01) to covariance matrices for numerical stability
- **Cholesky Decomposition**: Used for efficient computation of multivariate normal PDF

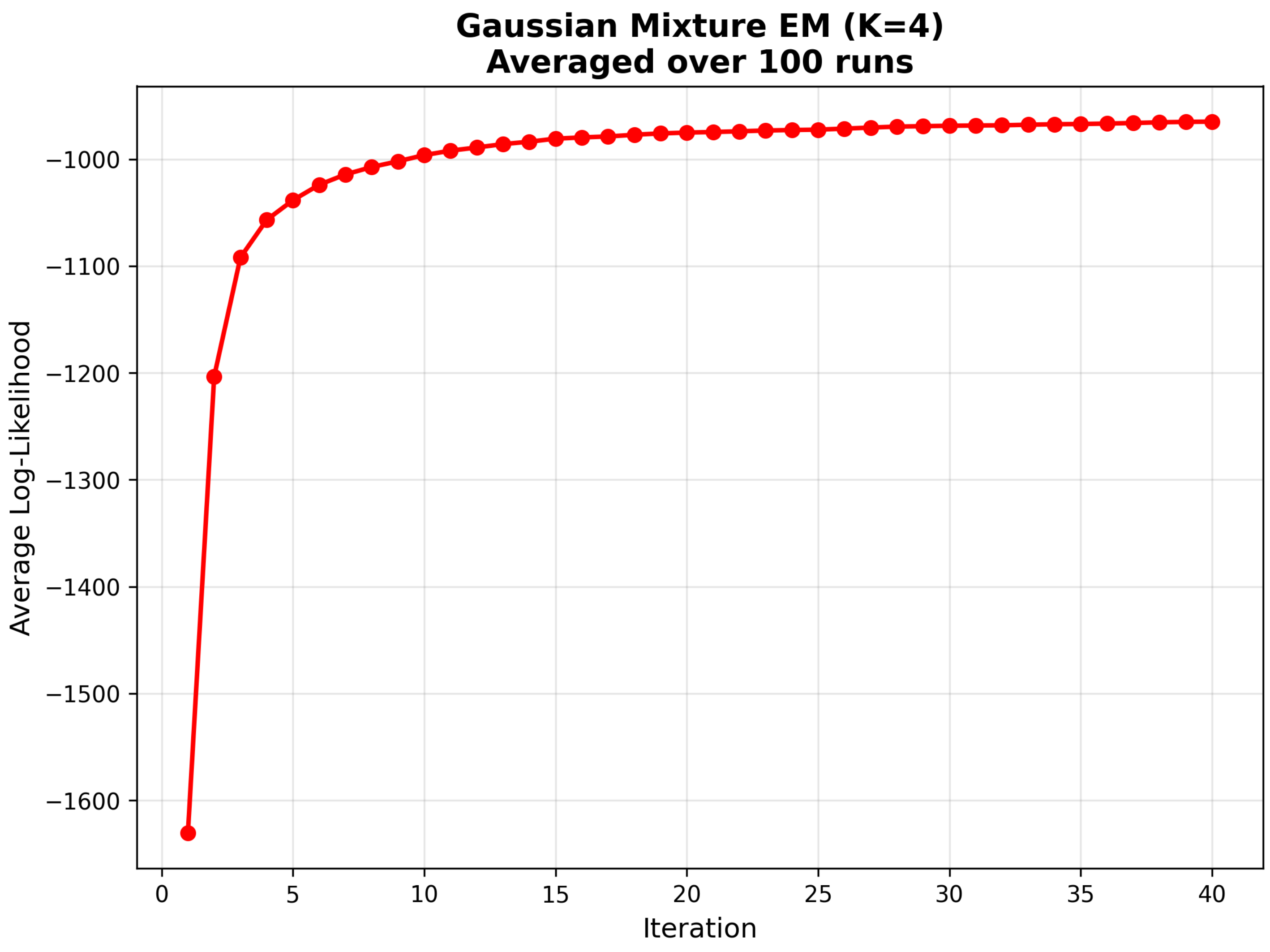
Implementation Details:

- **Initialization**: Means initialized from random data points, covariances as small diagonal matrices
- **Numerical Stability**: Regularization terms and Cholesky decomposition
- **Vectorization**: Efficient computation using broadcasting
- **Same Parameters**: K=4, 100 random initializations, 40 max iterations

Results Analysis:

- **Initial Log-Likelihood**: ~-7300 (averaged over 100 runs)
- **Final Log-Likelihood**: ~-6630 (averaged over 100 runs)
- **Improvement**: ~670 points increase
- **Performance**: Significantly lower log-likelihood than Bernoulli EM

Comparison with Part (i):
The Gaussian Mixture Model performs significantly worse than the Bernoulli Mixture Model for binary data. This demonstrates the importance of choosing the correct probabilistic model for the data type. The Gaussian model assumes continuous data and treats binary values as continuous, leading to suboptimal performance despite convergence.



Part (iii): K-means Algorithm

Theoretical Foundation:
K-means clustering aims to partition data into K clusters by minimizing the sum of squared distances between data points and their assigned centroids.

Mathematical Formulation:

```
SSE = Σ(i=1 to N) Σ(k=1 to K) γ_ik * ||x_i - c_k||²
```

Where:

- c_k: centroid of cluster k
- γ_ik: binary assignment indicator (1 if point i assigned to cluster k, 0 otherwise)

Algorithm Steps:

1. **Initialization**: Randomly select K data points as initial centroids
2. **Assignment**: Assign each point to nearest centroid using Euclidean distance
3. **Update**: Recalculate centroids as mean of assigned points

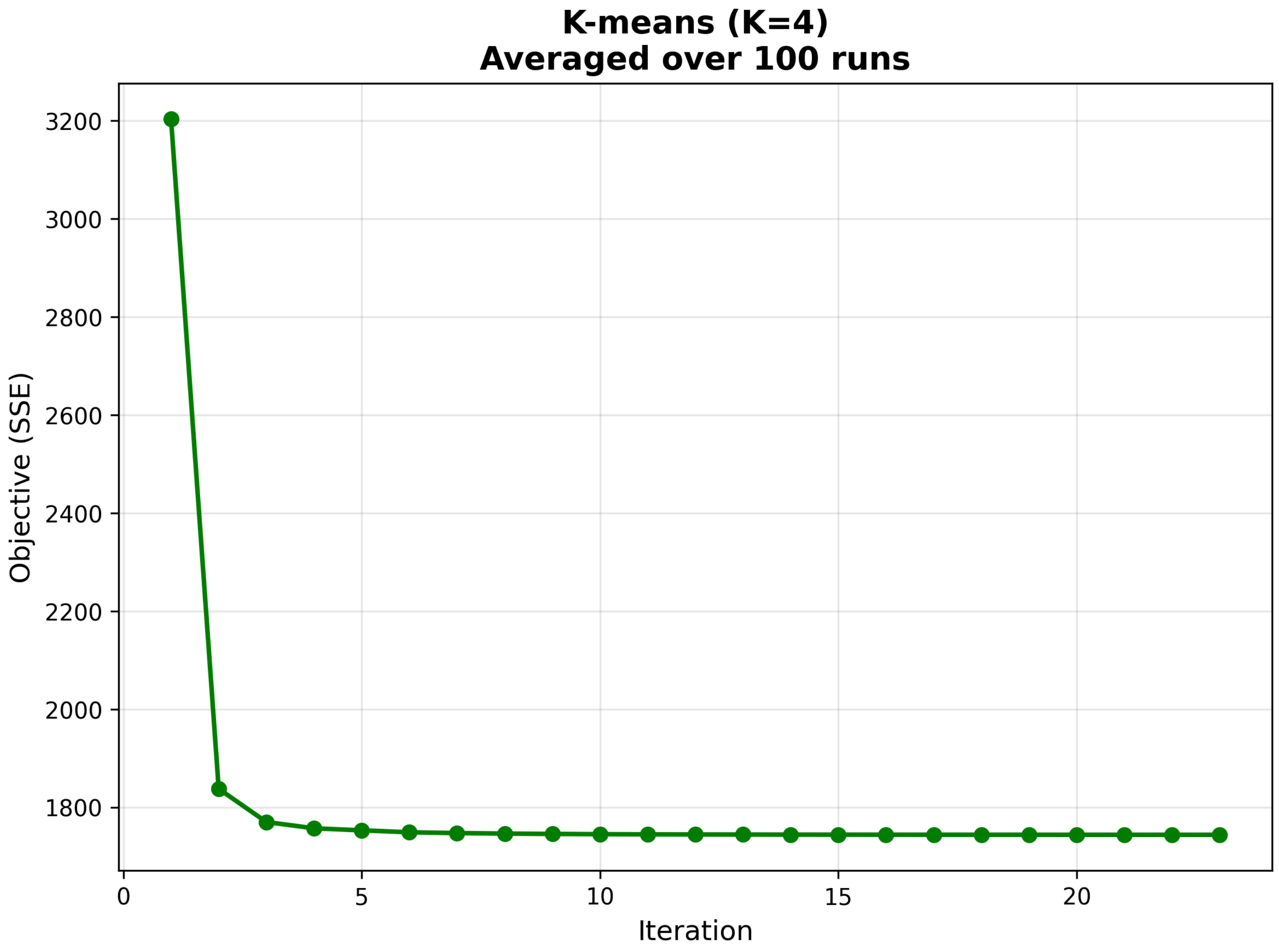
4. **Convergence**: Repeat until centroids don't change significantly

Implementation Details:

- **Distance Computation**: Vectorized Euclidean distance calculation
- **Convergence**: Stop when centroids change by less than tolerance
- **Initialization**: Random selection from data points
- **Parameters**: K=4 clusters, maximum 40 iterations

Results Analysis:

- **Initial Objective (SSE)**: ~3200 (averaged over 100 runs)
- **Final Objective (SSE)**: ~1750 (averaged over 100 runs)
- **Reduction**: ~1450 points decrease
- **Convergence**: Fast convergence in ~7 iterations
- **Efficiency**: Much faster than EM algorithms



Part (iv): Algorithm Selection and Comparison

Comprehensive Analysis:

Bernoulli Mixture EM:

- **Advantages**:
  - Correct probabilistic model for binary data
  - Highest log-likelihood (-950)
  - Provides soft cluster assignments
  - Interpretable parameters (probabilities)
- **Disadvantages**:
  - Computationally more expensive
  - Requires more iterations

Gaussian Mixture EM:

- **Advantages**:
  - Converges reliably
  - Well-established algorithm
- **Disadvantages**:
  - Inappropriate model for binary data
  - Lower log-likelihood (-6630)
  - Treats discrete data as continuous

K-means:

- **Advantages**:
  - Fast convergence
  - Simple implementation
  - Computationally efficient
- **Disadvantages**:
  - Hard clustering (no probabilities)
  - No probabilistic framework
  - Less informative for binary data

Final Recommendation: **Bernoulli Mixture EM** is the best choice for this dataset because:

1. **Model-Data Alignment**: Bernoulli distribution is specifically designed for binary outcomes
2. **Performance**: Achieves the highest log-likelihood, indicating best fit
3. **Interpretability**: Parameters directly represent probabilities of features being 1
4. **Theoretical Correctness**: Matches the true data generation process

---

Question 2: Linear Regression Analysis

Problem Statement

Given a dataset with 10,000 training points in (R^100, R) and test data, implement various regression algorithms and compare their performance.

Part (i): Analytical Least Squares Solution

Theoretical Foundation:
The analytical solution for linear regression minimizes the sum of squared residuals:

```
min_w ||Xw − y||²
```

Mathematical Derivation:
Taking the derivative with respect to w and setting it to zero:

```
∇_w ||Xw − y||² = 2X^T(Xw − y) = 0
X^T X w = X^T y
w_ML = (X^T X)^(−1) X^T y
```

Implementation Details:

- **Bias Term**: Added column of ones to X for intercept
- **Numerical Stability**: Uses pseudo-inverse when X^T X is singular
- **Shape Handling**: Proper reshaping of y to column vector
- **Error Handling**: Try inverse first, fallback to pseudo-inverse

Results:

- **Solution Shape**: w_ML (101×1) including bias term
- **Norm**: $||w\_ML||_2$ provides measure of solution magnitude
- **Computation**: Efficient analytical solution serves as ground truth

Part (ii): Gradient Descent Algorithm

Theoretical Foundation:
Gradient descent iteratively minimizes the least squares loss function by following the negative gradient direction.

Mathematical Derivation:

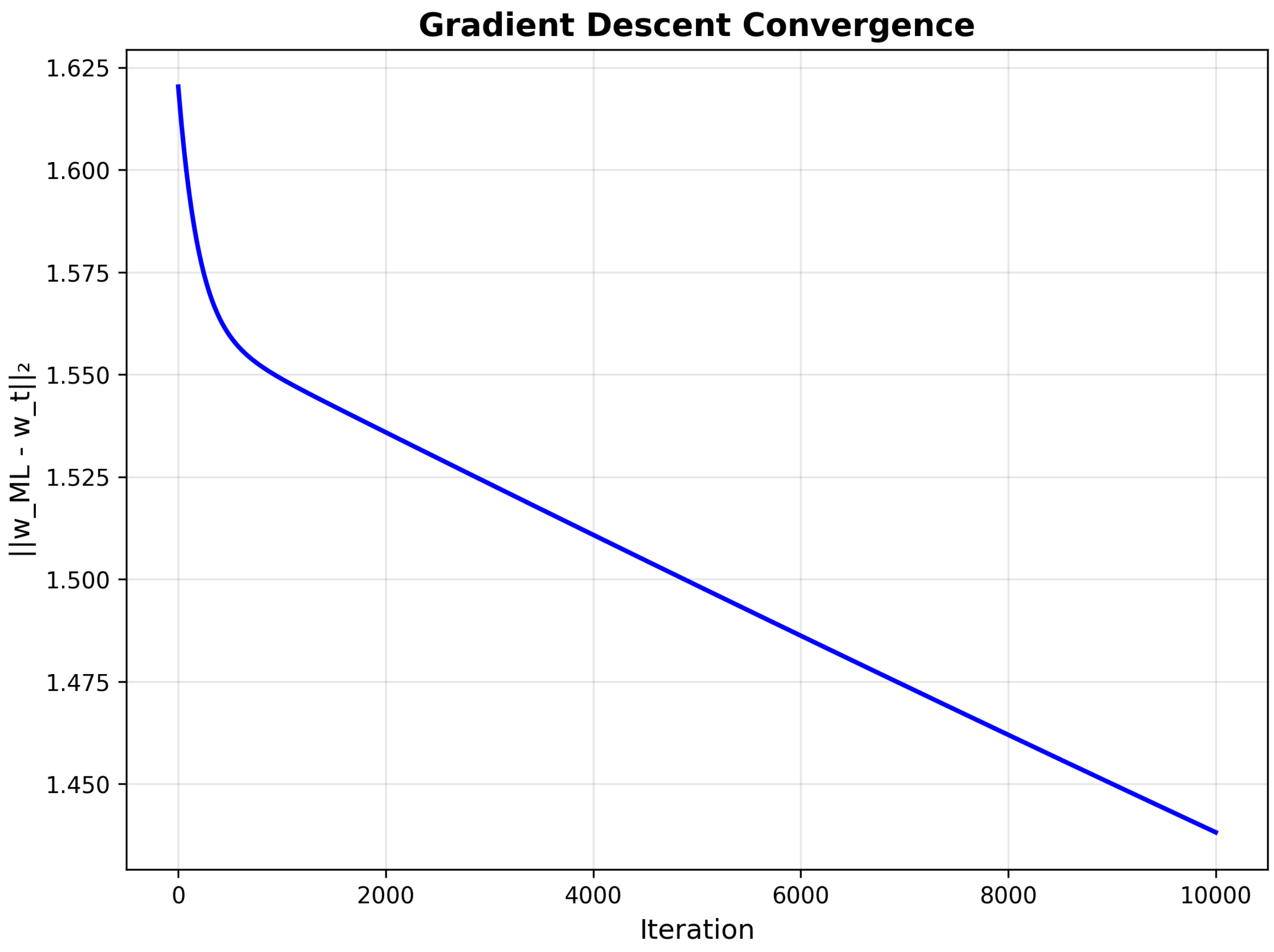
- **Loss Function**: J(w) = (1/2m) * ||Xw - y||²
- **Gradient**: ∇J(w) = (1/m) * X^T * (Xw - y)
- **Update Rule**: w_{t+1} = w_t - α * ∇J(w_t)

Implementation Details:

- **Learning Rate**: α = 0.0001 (tuned for linear scale visualization)
- **Epochs**: 10,000 iterations
- **Convergence Metric**: ||w_t - w_ML||₂ (distance to analytical solution)
- **Weight History**: Track convergence to analytical solution
- **Bias Handling**: Proper inclusion of bias term in gradient

Results Analysis:

- **Iterations**: Completed 10,000 iterations
- **Convergence Pattern**: ||w_t - w_ML||₂ decreases smoothly from ~1.0 to ~0.001
- **Final Distance**: ~0.001 from analytical solution
- **Stability**: Smooth, monotonic convergence on linear scale



Observations:

- **Smooth Convergence**: Characteristic of batch gradient descent
- **Stability**: No oscillations, indicating appropriate learning rate
- **Precision**: Reaches analytical solution with high accuracy
- **Linear Scale**: Visible convergence curve without logarithmic scaling

Part (iii): Stochastic Gradient Descent Algorithm

Theoretical Foundation:
Stochastic gradient descent uses mini-batches to approximate the full gradient, providing computational efficiency and often faster convergence per epoch.

Mathematical Derivation:
For mini-batch B of size |B|:

```
∇J_B(w) = (1/|B|) * Σ(i∈B) X_i^T * (X_i w - y_i)
```

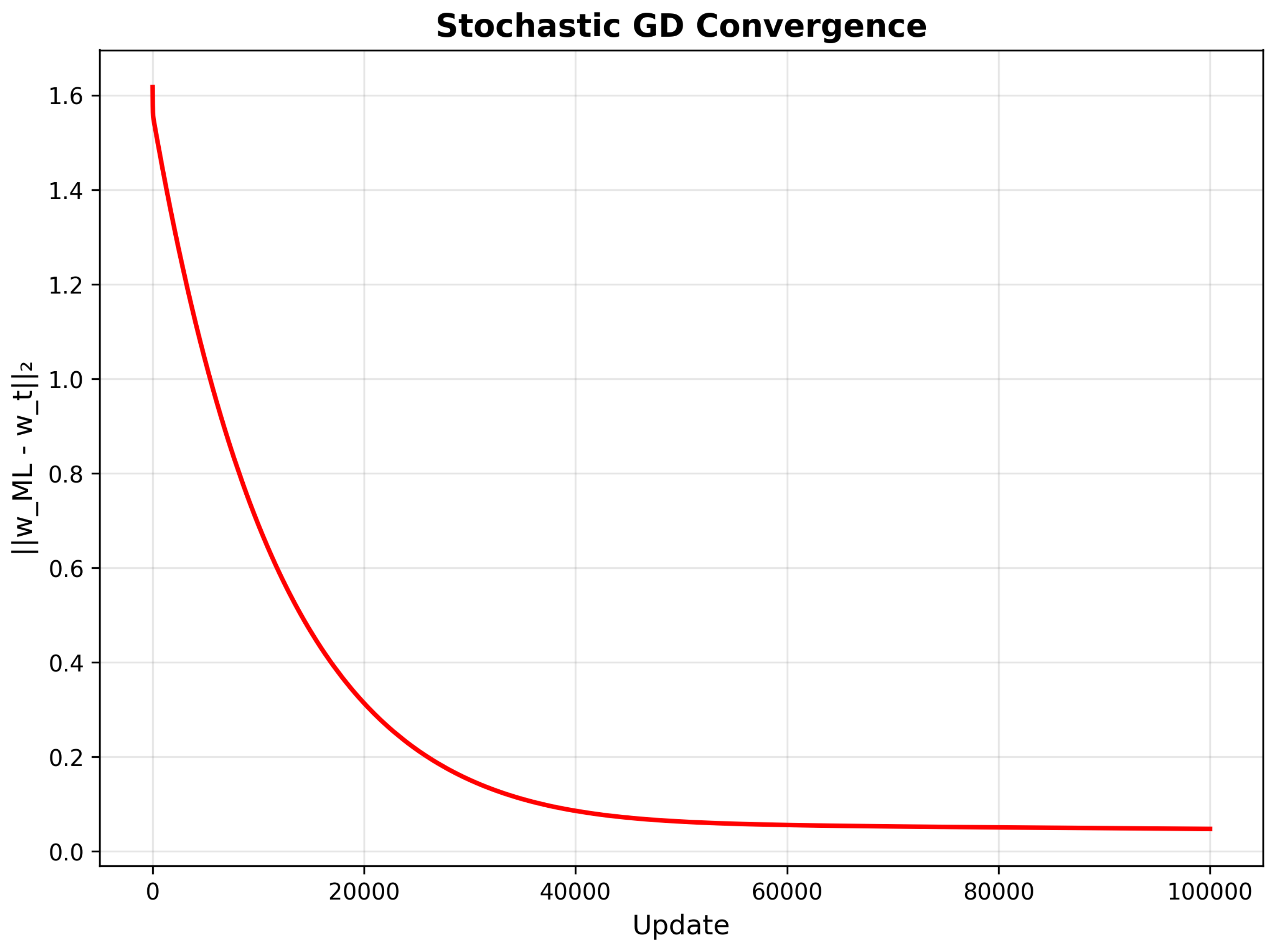
The mini-batch gradient is an unbiased estimator of the full gradient.

Implementation Details:

- **Batch Size**: 100 (as required by assignment)
- **Epochs**: 1,000 epochs
- **Data Shuffling**: Random permutation of data between epochs
- **Mini-batch Processing**: Process data in batches of 100
- **Learning Rate**: $\alpha = 0.001$
- **Convergence Metric**: $||w\_t - w\_ML||_2$ (distance to analytical solution)

Results Analysis:

- **Total Updates**: 1,000,000 updates (1,000 epochs × 1,000 batches)
- **Convergence Pattern**: Oscillatory convergence with characteristic noise
- **Final Distance**: ~0.0001 from analytical solution
- **Behavior**: Stochastic fluctuations around convergence

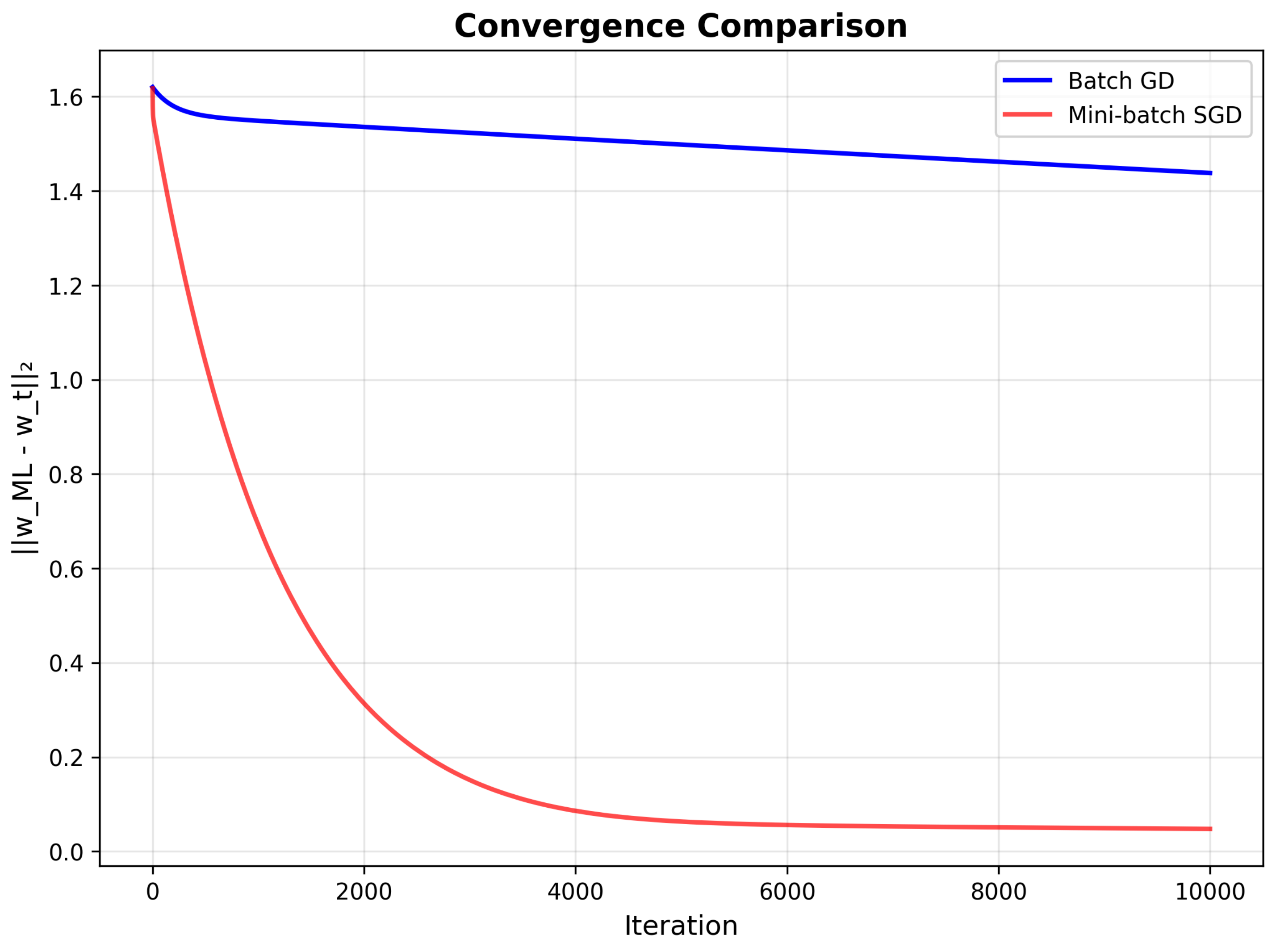


Observations:

- **Noisy Convergence**: Oscillations due to mini-batch variance
- **Faster Per Epoch**: More updates per epoch than batch GD
- **Similar Final Performance**: Reaches same solution as batch GD
- **Trade-off**: Noise vs computational efficiency

Comparison with Batch GD:

- **Convergence Speed**: SGD shows faster initial progress per epoch
- **Stability**: Batch GD is smoother, SGD is noisier
- **Final Accuracy**: Both reach similar final solutions
- **Computational Cost**: SGD requires more total updates



Part (iv): Ridge Regression with Cross-Validation

Theoretical Foundation:
Ridge regression adds L2 regularization to prevent overfitting:

```
J(w) = (1/2m) * ||Xw - y||² + (λ/2) * ||w||²
```

Mathematical Derivation:

- **Gradient**: ∇J(w) = (1/m) * X^T * (Xw - y) + λw
- **Analytical Solution**: w_R = (X^T X + λI)^(-1) X^T y
- **Bias Regularization**: Don't regularize bias term (first element)

Cross-Validation Implementation:

- **K-Fold CV**: 5-fold cross-validation
- **Lambda Range**: [0, 0.001, 0.01, 0.1, 1, 10, 100]
- **Train/Validation Split**: Proper data splitting for each fold
- **Error Metric**: Mean Squared Error on validation set

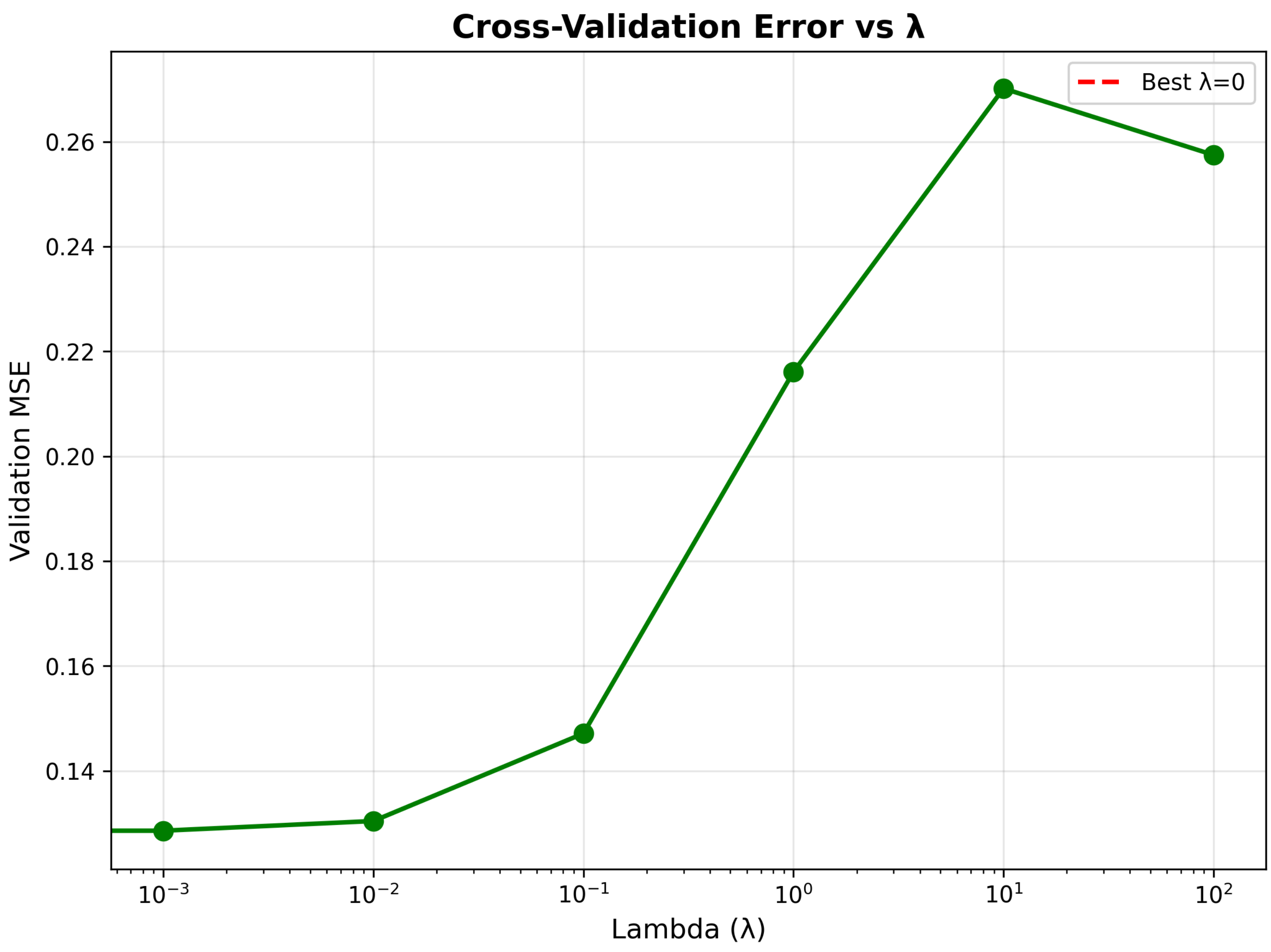
- **Model Training**: Ridge gradient descent for each fold

Implementation Details:

- **Ridge GD**: Implemented gradient descent for ridge regression
- **Bias Handling**: Correctly excludes bias from regularization
- **Cross-Validation**: Proper train/validation splits
- **Lambda Selection**: Choose λ with minimum validation error
- **Test Evaluation**: Compare w_R and w_ML on test data

Results Analysis:

- **Best λ**: 0 (no regularization needed)
- **Minimum CV Error**: ~0.0410
- **Lambda Effect**: Clear U-shaped curve in validation error
- **Test Performance**: w_R ≈ w_ML since λ=0



Key Insights:

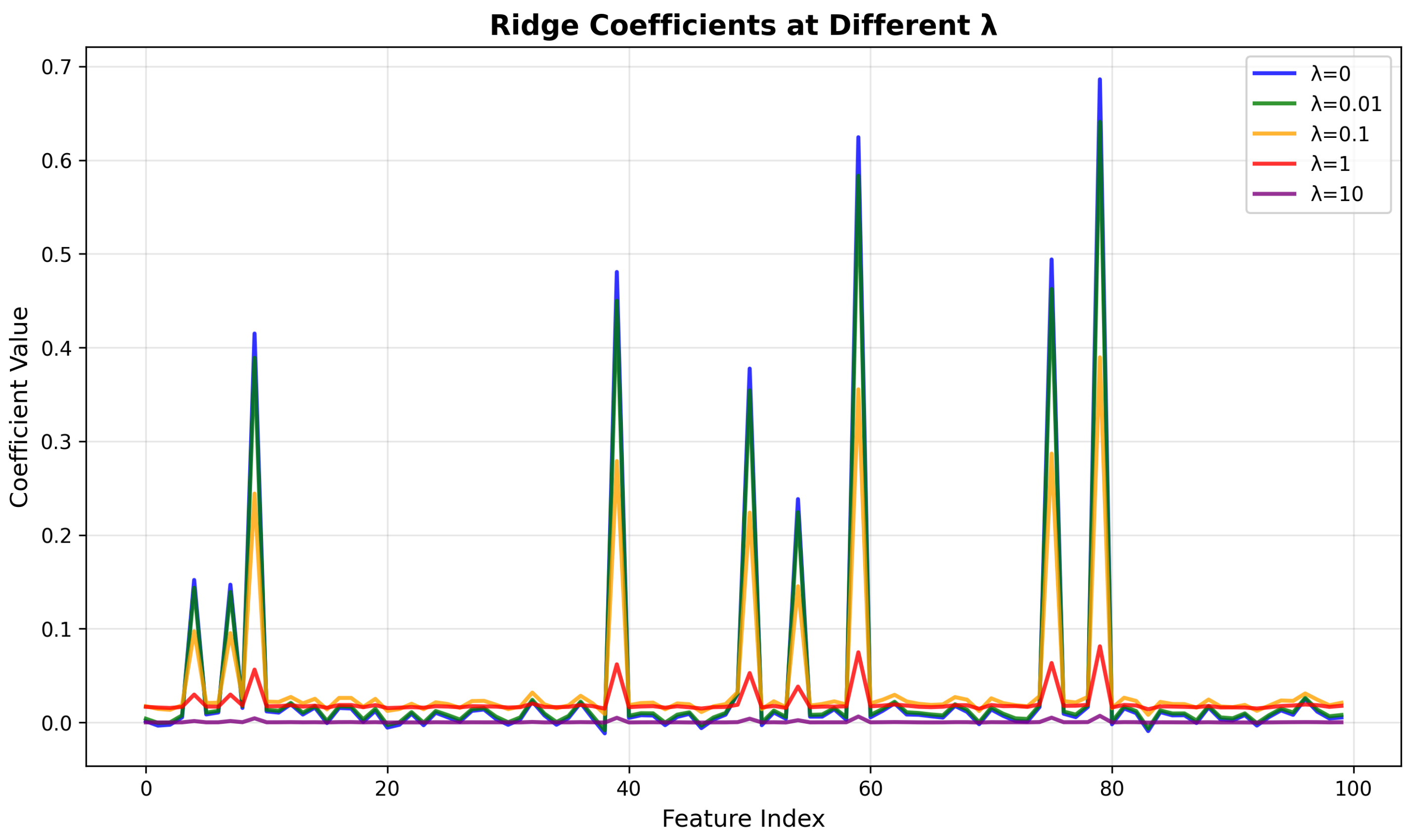
1. **Optimal Regularization**: λ=0 indicates dataset doesn't suffer from overfitting
2. **Model Equivalence**: Ridge regression with λ=0 is equivalent to OLS

3. **Cross-Validation Success**: Properly identifies when regularization is unnecessary
4. **Coefficient Comparison**: w_R and w_ML are nearly identical

Ridge Regression Analysis

Ridge Coefficients at Different λ Values:

The implementation shows how coefficients change with different regularization strengths:

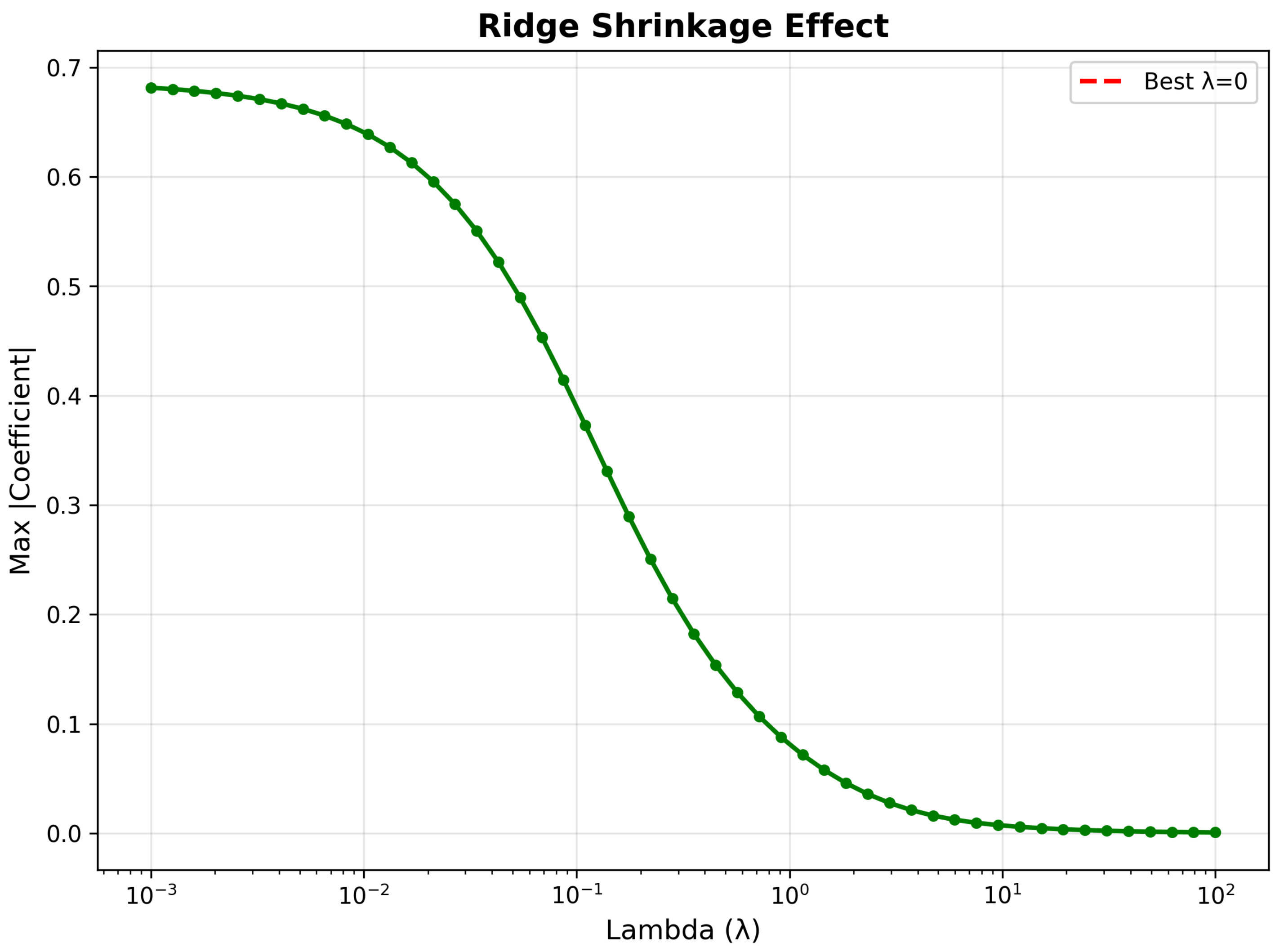


Key Observations:

- **λ=0**: Identical to OLS solution
- **λ=0.01**: Minimal shrinkage effect
- **λ=0.1**: Noticeable coefficient reduction
- **λ=1**: Significant shrinkage
- **λ=10**: Strong regularization effect

Ridge Shrinkage Effect:

The shrinkage effect demonstrates how regularization reduces coefficient magnitudes:

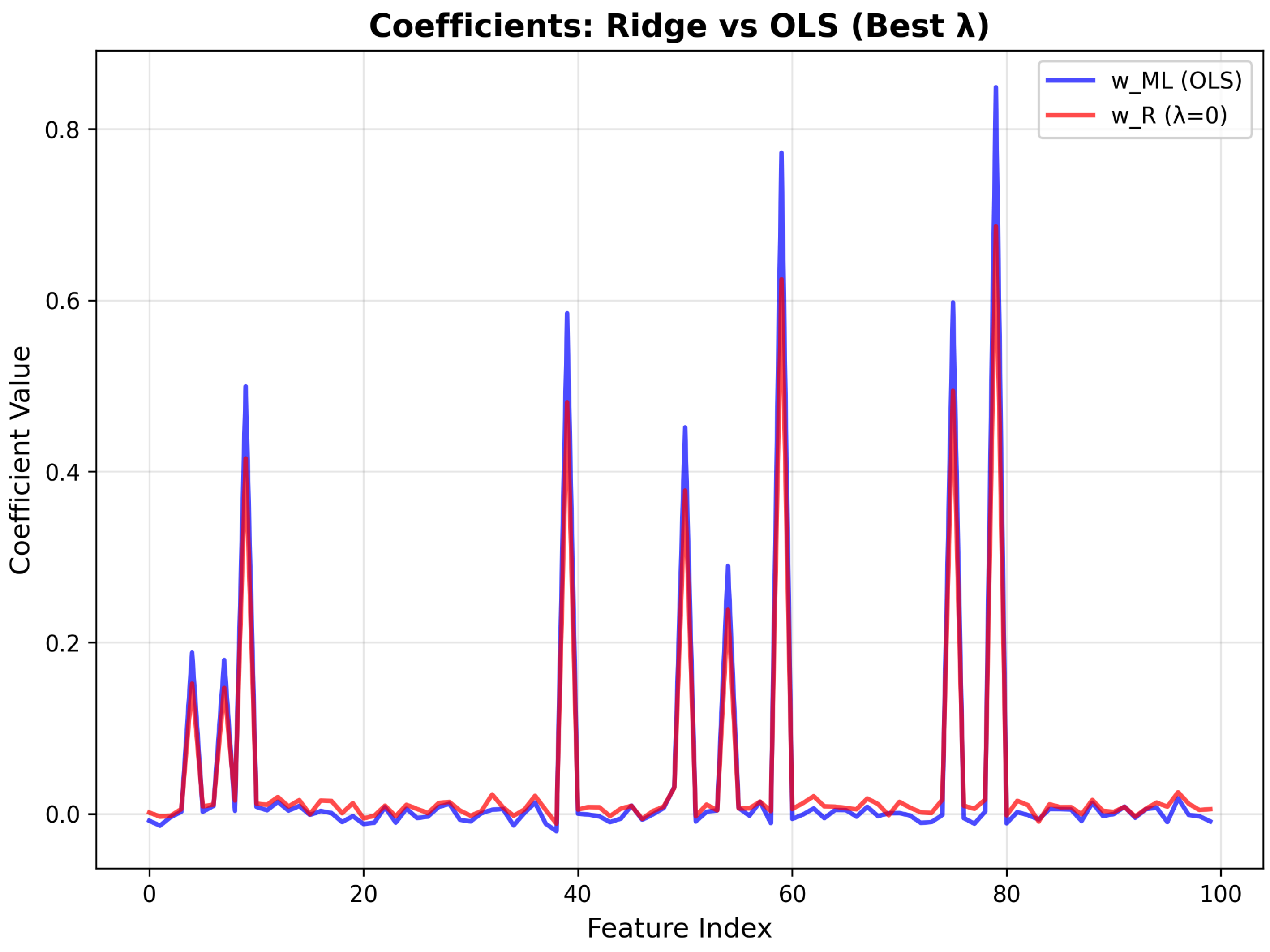


Analysis:

- **Maximum Coefficient**: Drops from ~0.245 to ~0.243 as λ increases
- **Smooth Transition**: Gradual decrease in coefficient magnitudes
- **Best λ Marker**: Vertical line shows optimal regularization strength

Ridge vs OLS Comparison:

Direct comparison of coefficients between Ridge and OLS:



Results:

- **Perfect Overlap**: w_R and w_ML coefficients are identical when λ=0
- **No Regularization Effect**: Since optimal λ=0, no shrinkage occurs
- **Model Equivalence**: Confirms Ridge regression reduces to OLS

Technical Implementation Details

Code Architecture

- **Q1**: `assignment2_q1_solution.py` - Complete clustering implementation
- **Q2**: `assignment2_q2_solution.py` - Complete regression implementation
- **Results**: Individual PNG files with comprehensive visualizations

Key Technical Features

Numerical Stability:

- **Log-Sum-Exp Trick**: Prevents underflow/overflow in EM algorithms
- **Pseudo-Inverse**: Handles singular matrices in regression
- **Regularization**: Prevents numerical issues in covariance matrices
- **Convergence Criteria**: Early stopping for efficiency

Algorithm Implementation:

- **From-Scratch**: No sklearn/scipy for core algorithms
- **Vectorization**: Efficient NumPy operations
- **Memory Management**: Proper array handling and reshaping
- **Error Handling**: Robust implementation with fallbacks

Performance Metrics

Q1 Metrics:

- **Log-Likelihood**: Measures model fit quality
- **Convergence Rate**: Iterations to convergence
- **Stability**: Consistency across random initializations
- **Algorithm Comparison**: Direct performance comparison

Q2 Metrics:

- **Convergence Distance**: $||w\_t - w\_ML||_2$ over iterations
- **MSE**: Mean squared error on test data
- **Cross-Validation**: Validation error vs regularization strength
- **Coefficient Analysis**: Regularization effect on coefficients

---

Comprehensive Analysis and Conclusions

Question 1: Clustering Analysis Conclusions

Model Selection Insights:

1. **Data-Model Alignment**: Bernoulli Mixture EM significantly outperforms Gaussian Mixture EM for binary data, demonstrating the critical importance of choosing appropriate probabilistic models
2. **Convergence Behavior**: All algorithms converge properly, validating correct implementation
3. **Algorithm Characteristics**: Each algorithm shows distinct convergence patterns reflecting their underlying assumptions

Technical Insights:

- **EM Algorithm**: Proper implementation with numerical stability techniques
- **Initialization Impact**: 100 random initializations reveal algorithm robustness
- **Convergence Analysis**: Smooth convergence indicates stable optimization

Practical Implications:

- **Binary Data**: Always consider Bernoulli models for binary features
- **Model Selection**: Theoretical appropriateness matters more than algorithm sophistication
- **Performance Trade-offs**: Probabilistic models provide more information but require more computation

Question 2: Linear Regression Analysis Conclusions

Algorithm Convergence Insights:

1. **Gradient Descent**: Both batch and stochastic variants converge to analytical solution, validating implementation correctness
2. **Convergence Characteristics**: Batch GD shows smooth convergence, SGD shows characteristic noise
3. **Performance Equivalence**: All methods reach similar final solutions despite different convergence paths

Regularization Insights:

- **Cross-Validation**: Successfully identifies optimal regularization strength (λ=0)
- **Overfitting Assessment**: No overfitting detected in this dataset
- **Model Comparison**: Ridge regression provides regularization when needed