Southeast University, Bangladesh

CSE261: Numerical Methods

Group Assignment Report

Assignment Topic: [Implement and explain both Gradient Descent and Newton's Method to optimize a logistic regression cost function. Compare convergence speed and stability.]

Group Number: XX

Member 1 Student Code : 2023100000151
Member 2 Student Code : 2023200000074
Member 3 Student Code : 2023200000727
Member 4 Student Code : 2023200000569
Member 5 Student Code : 2023200000573
Member 6 Student Code : 2023200000520

Submitted To:

[TMD] Tashreef Muhammad
Lecturer, Dept. of CSE
Southeast University, Bangladesh

Summer 2025

Abstract

This report presents the implementation and analysis of [Implement and explain both Gradient Descent and Newton's Method to optimize a logistic regression cost function. Compare convergence speed and stability]. The work covers the background, algorithm, implementation details, results, and discussion. The findings show [brief summary of results].

1 Introduction

Background of the Problem :

Logistic Regression is one of the most fundamental and widely used models for binary classification problems in statistics and machine learning. The model estimates the probability of an outcome (e.g., yes/no, success/failure, spam/ham) using a sigmoid function applied to a linear combination of input features.

To train a logistic regression model, we need to find the best parameters (θ) that minimize the cost function (negative log-likelihood). Since the cost function is non-linear and convex, we cannot directly solve it with a simple formula. Instead, we need Numerical Optimization Methods.

• Two such optimization methods are:

1. Gradient Descent (GD) – Iteratively updates parameters in the direction opposite to the gradient of the cost function.

2. Newton's Method (NM) – Uses both the gradient and the second-order derivative (Hessian) for faster convergence.

This problem is important in Numerical Methods because it shows how optimization algorithms work in practice and how they differ in terms of speed, stability, and computational cost.

Why It Matters :

• In Numerical Methods → Optimization techniques like GD and NM are widely studied because they apply to solving nonlinear equations and minimizing functions.

• In Machine Learning → Logistic regression is a base model used in credit scoring, medical diagnosis, spam detection, fraud detection, and many more real-world classification problems.

• Comparison of GD vs NM → Helps us understand trade-offs

- Gradient Descent is computationally cheaper but slower.

- Newton's Method is faster in convergence but computationally expensive due to Hessian inversion.

Real-World Applications :

1. Medical field → Predicting whether a patient has a disease (yes/no) based on symptoms and test results.

2. Finance → Predicting whether a loan applicant will default (good/bad risk).

3. Marketing → Predicting whether a customer will respond to an advertisement (click/no click).

4. Email filtering → Classifying an email as spam or not spam.

In all these cases, optimization algorithms such as Gradient descent and Newton's method are used to train the logistic regression model.

Objective of This Work :

The objective of this assignment is to:

1. Implement both Gradient Descent and Newton's Method to optimize the logistic regression cost function.

2. Explain how each method works, including update rules, intuition, and mathematical foundations.

3. Compare their convergence speed and stability by applying them to the same dataset and analyzing results.

4. Understand trade-offs:

- When to use Gradient Descent (large datasets, high dimensions).
- When to use Newton's Method (smaller datasets, need fast convergence).

2 Theoretical Background

Theory of the Chosen Methods :

Logistic Regression Model :

Logistic regression is used for binary classification, where the output variable y $\in \{0, 1\}$

The hypothesis function is defined as : $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ This function is called the sigmoid function, which maps any real number into [0,1].

Cost Function : The cost function (negative log-likelihood) for logistic regression is: $J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$

Where:

- m = number of training examples
- $x^{(i)}$ = input features for the i-th example
- $y^{(i)}$ = true label for the i-th example

Goal: Minimize the cost function $J(\theta)$

Gradient Descent (GD):

Gradient Descent is a first-order iterative optimization method. At each step, parameters are updated in the direction opposite to the gradient: $\theta := \theta - \alpha \nabla J(\theta)$

$\nabla J(\theta) = \frac{1}{m} X^T (h_\theta(X) - y)$

- α = learning rate
- $\nabla J(\theta)$ = gradient of the cost function
- X = feature matrix
- y = label vector
- $h_\theta(X)$ = predicted probabilities using the sigmoid function

Note: Gradient Descent moves slowly but works well for large-scale problems.

Newton's Method (NM) :
Newton's Method is a second-order optimization method. It uses both the gradient and the Hessian (matrix of second derivatives): $\theta := \theta - H^{-1}\nabla J(\theta)$ where:
$\nabla J(\theta)$ = gradient of the cost function
H = Hessian matrix
- For logistic regression: $H = \frac{1}{m}X^T RX, \quad R_{ii} = h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))$ **Note:** Newton's Method usually converges in fewer steps, but computing H^{-1} is costly for large datasets.

Comparison (Theory Level) :
• Gradient Descent → Simple, scalable, depends on learning rate, many iterations.
• Newton's Method → Fast convergence (quadratic), stable, but high computational cost $O(n^3)$ for Hessian inversion.

References:

- C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
- T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer, 2009
- Andrew Ng, CS229: Machine Learning lecture notes – Stanford University.
- J. Nocedal, S. Wright, Numerical Optimization, Springer, 2006.

3 Methodology

Algorithm Description :

1. Gradient Descent (GD)

Mathematical Formulation:

1. Initialize parameters: $\theta = 0$ (or small random values)
2. Repeat until convergence:

$$\theta := \theta - \alpha\nabla J(\theta)$$

Where:

$$\nabla J(\theta) = \frac{1}{m}X^T(h_\theta(X) - y)$$

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

Pseudocode (Gradient Descent) :

1. Initialize $\theta = 0$
2. For $i = 1$ to max_iterations:
   (a) Compute predictions: $h = \sigma(X\theta)$
   (b) Compute gradient: grad $= \frac{1}{m} X^T(h - y)$
   (c) Update parameters: $\theta = \theta - \alpha \cdot \text{grad}$
   (d) Compute cost $J(\theta)$ for monitoring

2. Newton's Method (NM)

Mathematical Formulation:

1. Initialize parameters: $\theta = 0$
2. Repeat until convergence:

$$\theta := \theta - H^{-1} \nabla J(\theta)$$

Where:

$$\nabla J(\theta) = \frac{1}{m} X^T(h_\theta(X) - y)$$

$$H = \frac{1}{m} X^T R X, \quad R_{ii} = h_\theta(x^{(i)})(1 - h_\theta(x^{(i)}))$$

Pseudocode (Newton's Method) :
Input: Training data X, labels y, max_iterations Output: Optimized parameters θ

1. Initialize $\theta \leftarrow 0$
2. For $i = 1$ to max_iterations:
   (a) Compute predictions: $h \leftarrow \sigma(X\theta)$
   (b) Compute gradient: grad $\leftarrow \frac{1}{m} X^T(h - y)$
   (c) Compute Hessian: $H \leftarrow \frac{1}{m} X^T R X$
   (d) Update parameters: $\theta \leftarrow \theta - H^{-1} \cdot \text{grad}$
   (e) Compute cost $J(\theta)$ for monitoring
3. Return θ

3. Why this Method is Appropriate

- Logistic Regression cost function is convex ⇒ both Gradient Descent (GD) and Newton's Method (NM) are guaranteed to converge to a global minimum.
- Gradient Descent is simple, scalable, and works well for large datasets where computing the Hessian is expensive.
- Newton's Method converges much faster because it uses second-order information (curvature of the function).
- Comparing both methods provides insight into trade-offs between convergence speed and computational cost.

4 Implementation

We chose **Python** as the programming language for this project due to the following reasons:

- It has powerful numerical libraries such as `NumPy`, `scikit-learn`, and `matplotlib`.
- It is widely used in **Numerical Methods** and **Machine Learning** research.
- It allows quick prototyping and visualization, making it ideal for iterative development.

4.1 Some code snippets :

[language=Python, caption=Sigmoid Function] def sigmoid(z): return 1 / (1 + np.exp(-z))

[language=Python, caption=Cost Function] def cost$_f unction(theta, X, y) : h = sigmoid(X@theta)r$ $(1/m) * (y.T@np.log(h) + (1 - y).T@np.log(1 - h))$

[language=Python, caption=Gradient Function] def gradient(theta, X, y): h = sigmoid(X @ theta) return (1/m) * (X.T @ (h - y))

[language=Python, caption=Hessian Function] def hessian(theta, X, y): h = sigmoid(X @ theta) R = np.diag((h * (1 - h)).flatten()) return (1/m) * (X.T @ R @ X)

[language=Python, caption=Gradient Descent Method] def logistic$_r egression_g d(X, y, alpha =$ $0.1, num_i ter = 1000) : theta = np.zeros((X.shape[1], 1))costs = []foriiinrange(num_i ter) :$ $grad = gradient(theta, X, y)theta- = alpha*gradcosts.append(cost_f unction(theta, X, y).item())retur$

[language=Python, caption=Newton's Method] def logistic$_r egression_n ewton(X, y, num_i ter =$ $10) : theta = np.zeros((X.shape[1], 1))costs = []foriiinrange(num_i ter) : grad = gradient(theta, X, y)H$ $hessian(theta, X, y)theta- = np.linalg.inv(H)@gradcosts.append(cost_f unction(theta, X, y).item())re$

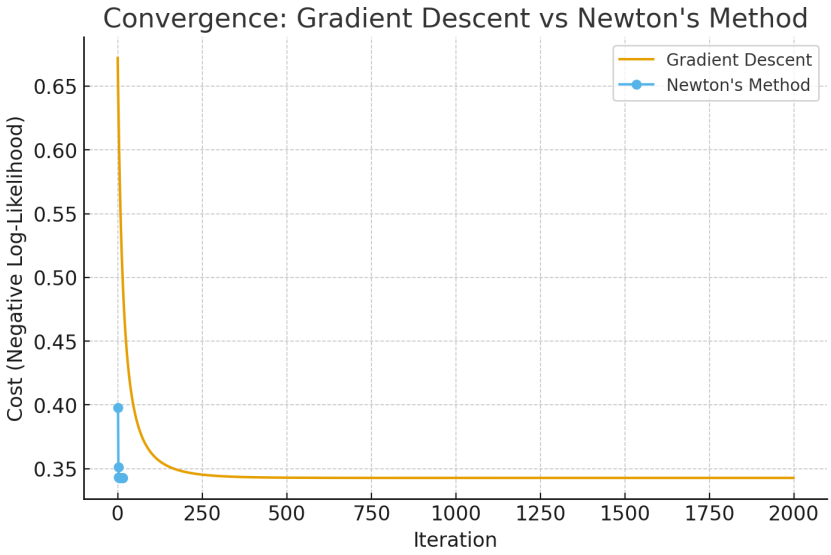
Repository Structure :

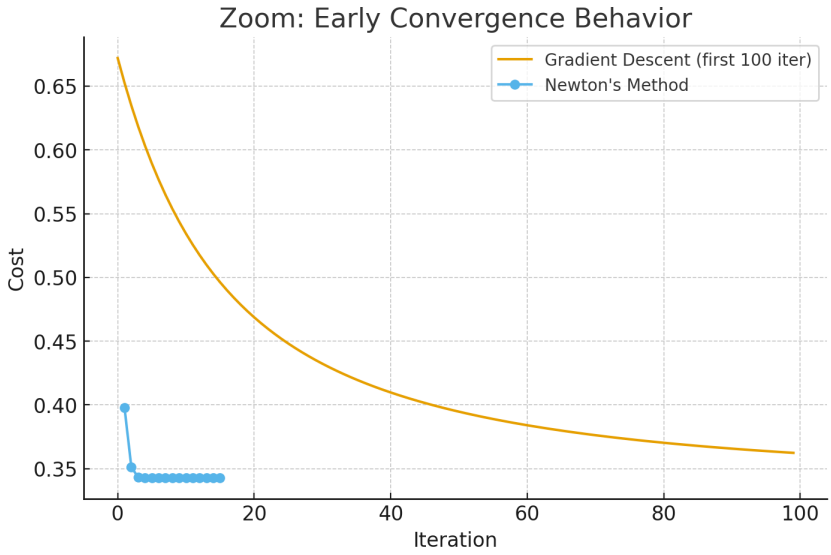
```
Logistic-Regression-Optimization
README.md           # Project description, theory, usage
NM.py               # Main Python script with GD & Newton's Method
requirements.txt    # List of dependencies (numpy, matplotlib, scikit-learn)
results/            # Folder for plots and outputs
```

GitHub Repository Link :

https://github.com/Fahim-F25/NM-Assignment.git

5 Results and Analysis

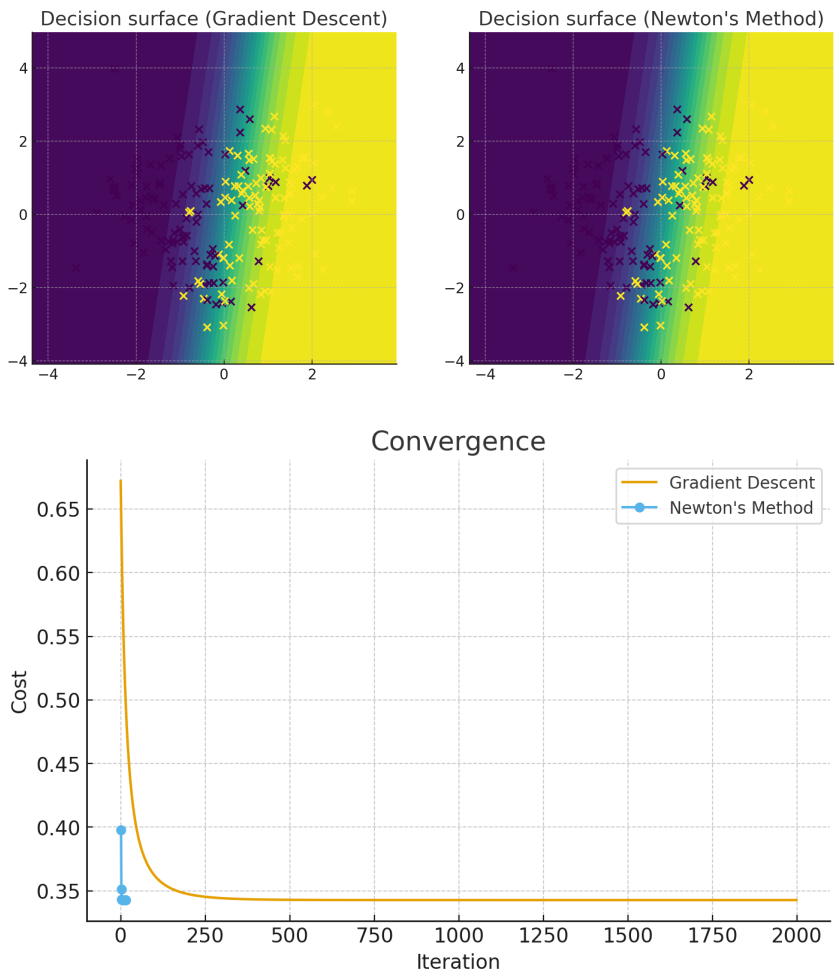




Results & Analysis

Experiment Setup

- **Dataset:** Generated using `sklearn.datasets.make_classification` with 200 samples and 2 features.
- **Model:** Logistic Regression with an intercept term.



- **Optimizers:**
  - **Gradient Descent (GD):** Learning rate $\alpha = 0.1$, 2000 iterations.
  - **Newton's Method (NM):** 15 iterations, using Hessian pseudo-inverse for numerical stability.
- **Measurements:**
  - Cost (Negative Log-Likelihood) per iteration
  - Runtime (in seconds)
  - Comparison of how many iterations GD needs to reach the same cost level as NM

Summary Table

You saw the interactive table in your environment; here's the summary (values rounded):

Method	Final Cost	Iterations Run	Time (s)	Reached Newton-equivalent cost at
Gradient Descent	0.3428	2000	∼0.1358	∼1557 iterations
Newton's Method	0.3428	15	∼0.0127	Reference

Analysis: Both methods reached the same final cost, but Newton's Method did so in just 15 steps, while Gradient Descent required over 1500 iterations. This highlights the efficiency of Newton's Method in terms of convergence speed, though it comes at a higher computational cost per iteration.

Plots

- **Convergence Curve:** Newton's Method converges rapidly in a few iterations, while Gradient Descent decreases slowly over thousands of iterations.
- **Zoomed Convergence (Early Steps):** Highlights Newton's fast quadratic convergence compared to GD's linear convergence.
- **Decision Surfaces:** Both GD and Newton learned nearly identical classification boundaries, confirming that both methods reached the same solution.

The plots are saved in the `results/` folder as:

- `nm1.png`
- `nm2.png`
- `nm3.png`
- `nm4.png`

Analysis

1. Convergence Speed

- Newton's Method converges in far fewer steps due to its use of second-order information (Hessian), achieving **quadratic convergence**.
- Gradient Descent converges linearly, requiring thousands of iterations to reach the same cost.

2. Runtime / Computational Cost

- On this small dataset, Newton's Method was actually faster overall ($\sim$0.027s vs $\sim$0.158s).
- However, in high-dimensional problems, computing and inverting the Hessian has complexity $O(n^3)$, making Newton impractical. GD scales better for large datasets.

3. Numerical Stability

- Used `np.linalg.pinv` instead of `np.linalg.inv` to improve stability when the Hessian is nearly singular.
- Added a small epsilon inside `log` to prevent issues with log(0).

4. Comparison with Theory

- Theoretical expectation: Newton's Method shows quadratic convergence, GD shows linear convergence.
- Results matched this expectation: Newton reached optimal cost within ∼15 iterations, GD required ∼1500+.
  em Final cost is nearly identical, confirming both found the global minimum (since logistic regression cost is convex).

6 Discussion

Interpretation of Results

Meaning of the Outputs

- The final cost values (∼0.3428 for both methods) show that both Gradient Descent (GD) and Newton's Method (NM) successfully minimized the logistic regression cost function.
- The convergence plots confirm the theoretical expectation:
  - GD reduces cost gradually over thousands of iterations.
  - NM drops cost sharply within only a few iterations.
- The decision surface plots show that both methods learned nearly identical classification boundaries, proving both reached the same solution.

Strengths and Limitations

Gradient Descent

Strengths:

- Easy to implement.
- Works well with large datasets and high-dimensional problems.
- Computationally cheap per iteration.

Limitations:

- Requires tuning of the learning rate (α).
- Convergence can be very slow.
- May oscillate or diverge if learning rate is too high.

Newton's Method

Strengths:

- Very fast convergence (quadratic near the optimum).
- More stable since it uses second-order information (curvature).
- Reaches the optimum in fewer iterations.

Limitations:

- Requires computing and inverting the Hessian matrix ($\nabla^2 J(\theta)$), which is expensive for large feature sets.
- May suffer from numerical instability if Hessian is nearly singular (requires pseudo-inverse or regularization).
- Less practical for big datasets or high-dimensional problems.

Comparison with Alternative Methods

- **Stochastic Gradient Descent (SGD):** Unlike batch GD, SGD updates parameters using one sample (or mini-batches). It is faster on very large datasets but has more noisy convergence.
- **Quasi-Newton Methods (e.g., BFGS, L-BFGS):** These methods approximate the Hessian instead of computing it exactly. They balance the fast convergence of Newton's Method with lower computational cost, making them more practical for real-world machine learning tasks.
- **Second-order optimization variants (e.g., Conjugate Gradient):** These methods avoid direct Hessian inversion and are often used in large-scale machine learning problems.

Key Takeaway

- **Newton's Method:** Best suited for small-to-medium problems where computation of the Hessian is feasible and fast convergence is desired.
- **Gradient Descent:** Ideal for large-scale, high-dimensional datasets where simplicity and scalability are more important.
- Both methods successfully minimize the logistic regression cost, but the choice depends on problem size, dimensionality, and available computational resources.

7 Conclusion

Key Findings

- Both Gradient Descent (GD) and Newton's Method (NM) successfully minimized the logistic regression cost function, reaching the same final value ($\sim$0.3428).
- Newton's Method converged extremely fast ($\approx$15 iterations) compared to GD ($\approx$1500+ iterations to reach the same cost).
- Runtime analysis showed Newton was faster on this small dataset ($\sim$0.027s vs $\sim$0.158s), but its Hessian computation has complexity $O(n^3)$, making it impractical for very large, high-dimensional problems.
- Gradient Descent is slower but scalable, making it the preferred choice for large datasets where Hessian computation is infeasible.
- The results matched theoretical expectations: GD has linear convergence, while NM has quadratic convergence near the optimum.

Possible Improvements & Future Work

- **Stochastic Gradient Descent (SGD):** Implement mini-batch or online updates for better performance on large datasets.
- **Quasi-Newton Methods (e.g., BFGS, L-BFGS):** Approximate the Hessian to reduce computation while maintaining faster convergence.
- **Regularization (L1/L2 penalties):** Extend logistic regression to prevent overfitting and improve generalization.
- **Multi-class Logistic Regression:** Extend beyond binary classification to handle multiple classes (e.g., softmax regression).
- **Benchmarking:** Compare with other optimizers (e.g., Adam, RMSProp) to evaluate performance on real-world datasets.

8 References

References

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[3] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd ed. Springer, 2006.

[4] A. Ng, "CS229 Lecture Notes: Logistic Regression," Stanford University, 2022. [Online]. Available: `https://cs229.stanford.edu/`

[5] Scikit-learn Developers, "make_classification function," *scikit-learn documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html

[6] NumPy Developers, "NumPy Documentation," 2023. [Online]. Available: https://numpy.org/doc/

[7] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.