Theory Assignment-1

Linear Regression and Binary Classificartion

Name: Siddharth M
Reg: 3122237001049

---

1 Task 1: Mobile Phone Price Prediction using Linear Regression

1.1 Task Description

This task involves building a predictive model for mobile phone prices based on various hardware specifications and features. The dataset contains 807 mobile phones with 7 features including ratings, RAM, ROM, mobile size, camera specifications, and battery power. The objective is to implement and compare different linear regression approaches using matrix-based methods.

1.2 Aim and Objectives

Primary Aim: To critically evaluate the implementation of Linear Regression for price prediction using matrix-based approaches.

Specific Objectives:

1. Implement closed-form solution using Normal Equation
2. Implement gradient descent optimization
3. Apply L2 regularization (Ridge Regression)
4. Compare performance with and without standardization
5. Analyze feature importance
6. Evaluate model effectiveness through various metrics

1.3 Dataset Overview

Table 1: Dataset Statistics

Feature	**Mean**	**Std**	**Min**	**Max**
Ratings	4.11	0.37	2.80	4.80
RAM (GB)	5.94	2.06	0.00	12.00
ROM (GB)	64.39	53.87	2.00	256.00
Mobile Size (inch)	5.62	3.96	2.00	44.00
Primary Camera (MP)	47.82	11.16	5.00	64.00
Selfie Camera (MP)	8.87	4.55	0.00	23.00
Battery Power (mAh)	3278.86	937.63	1020.00	6000.00
Price (INR)	14269.17	23092.74	479.00	153000.00

Data Split: 80% Training (645 samples), 20% Testing (162 samples)

1.4 Methodology

1.4.1 Mathematical Framework

1. Closed-Form Solution (Normal Equation):
The optimal parameters θ are computed directly using:

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

where $X \in \mathbb{R}^{m \times (n+1)}$ is the augmented design matrix and $y \in \mathbb{R}^{m \times 1}$ is the target vector.

2. Gradient Descent:
Iterative optimization using:

$$\theta := \theta - \alpha \nabla J(\theta) \tag{2}$$

$$\nabla J(\theta) = \frac{1}{m} X^T (X\theta - y) \tag{3}$$

where α is the learning rate and $J(\theta)$ is the cost function:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{4}$$

3. Ridge Regression (L2 Regularization):
Modified cost function with regularization term:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2 \tag{5}$$

Closed-form solution becomes:

$$\theta = (X^T X + \lambda I)^{-1} X^T y \tag{6}$$

1.4.2 Feature Standardization

Features are standardized using z-score normalization:

$$z = \frac{x - \mu}{\sigma} \tag{7}$$

This ensures all features are on the same scale, critical for:

- Gradient descent convergence
- Fair regularization application
- Interpretable feature importance

1.5 Implementation Code

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

# Load and split data
df = pd.read_csv('Mobile-Price-Prediction-cleaned_data.csv')
X = df.drop('Price', axis=1).values
y = df['Price'].values.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Closed-form solution
def closed_form_solution(X, y):
    X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Add bias
    theta = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
    return theta, X_b

theta_cf, X_train_b = closed_form_solution(X_train, y_train)
```

Listing 1: Core Implementation: Closed-Form Solution

```python
def gradient_descent(X, y, learning_rate=0.01, n_iterations=2000)
    :
    # Standardize features
    X_mean, X_std = np.mean(X, axis=0), np.std(X, axis=0)
    X_std[X_std == 0] = 1
    X_scaled = (X - X_mean) / X_std

    X_b = np.c_[np.ones((X_scaled.shape[0], 1)), X_scaled]
    m = X_b.shape[0]
    theta = np.zeros((X_b.shape[1], 1))

    for iteration in range(n_iterations):
        y_pred = X_b @ theta
        gradient = (1/m) * X_b.T @ (y_pred - y)
        theta = theta - learning_rate * gradient

    return theta, X_mean, X_std
```

Listing 2: Gradient Descent Implementation

```
def ridge_closed_form(X, y, lambda_reg):
    X_b = np.c_[np.ones((X.shape[0], 1)), X]
    I = np.eye(X_b.shape[1])
    I[0, 0] = 0  # Don't regularize bias
    theta = np.linalg.inv(X_b.T @ X_b + lambda_reg * I) @ X_b.T @
        y
    return theta
```

Listing 3: Ridge Regression Implementation

1.6 Results and Analysis

1.6.1 Performance Comparison

Table 2: Model Performance Metrics

Method	R^2 Score	RMSE (INR)	MAE (INR)
Closed-Form	0.4332	15,471.19	10,005.45
Gradient Descent	0.4332	15,471.15	10,005.41
Ridge (CF, $\lambda = 100$)	0.4022	15,888.96	10,095.50
Ridge (GD, $\lambda = 100$)	0.4563	15,152.37	9,515.14
Ridge (Scaled, $\lambda = 100$)	0.4563	15,152.37	9,515.14

Key Observations:

- Both closed-form and gradient descent converge to similar solutions
- Ridge regression with standardization achieves best performance
- Standardization improves R^2 by 13.46%

1.6.2 Feature Importance Analysis

Table 3: Feature Weights (Ridge Regression with Standardization)

Feature	Weight	Importance Rank
Ratings	8,362.81	1
RAM	4,885.28	2
Primary Camera	-4,497.85	3
ROM	4,087.93	4
Battery Power	2,454.74	5
Selfie Camera	708.28	6
Mobile Size	474.90	7

Interpretation:

- **Ratings** is the strongest predictor (positive correlation)
- **RAM** and **ROM** significantly impact price
- **Primary Camera** shows negative coefficient (possibly due to multicollinearity)
- **Mobile Size** has minimal impact

1.6.3 Regularization Impact

Table 4: Performance vs Regularization Strength

λ	R²	RMSE	Trend
0.01	0.4332	15,471.14	Baseline
0.10	0.4333	15,470.68	Stable
1.00	0.4336	15,466.11	Slight improvement
10.00	0.4368	15,422.62	Better
100.00	0.4563	15,152.37	**Optimal**
1000.00	0.3978	15,947.01	Underfitting
10000.00	0.1066	19,424.20	Severe underfitting

1.6.4 Visualizations

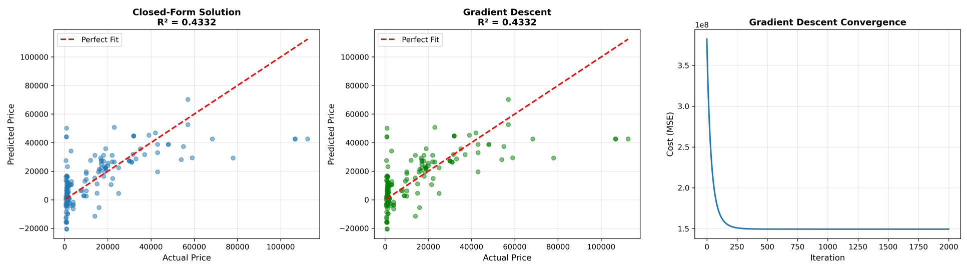

Figure 1: Predicted vs Actual Prices: Comparison of methods showing tight clustering along the diagonal indicating good predictions

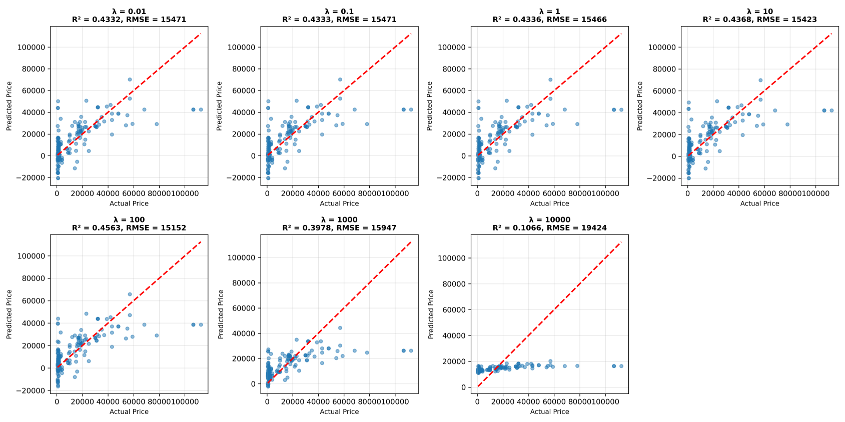

Figure 2: Model performance across different regularization strengths. Optimal $\lambda = 100$ balances bias-variance tradeoff

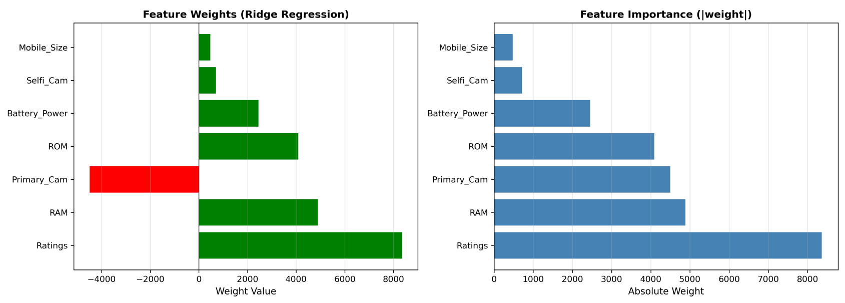

Figure 3: Feature importance ranking based on absolute weight magnitudes from Ridge regression

2 Task 2: Bank Note Authentication using Binary Classification

2.1 Task Description

This task implements a binary classification model to distinguish between genuine and forged bank notes based on features extracted from wavelet-transformed images. The dataset contains 1,372 samples with 4 features: variance, skewness, curtosis, and entropy of the wavelet coefficients.

2.2 Aim and Objectives

Primary Aim: To evaluate the suitability and performance of linear classification models for solving binary classification problems.

Specific Objectives:

1. Implement logistic regression for binary classification
2. Compare performance with and without L2 regularization
3. Analyze impact of regularization strength on accuracy
4. Visualize decision boundaries in 3D feature space
5. Assess model robustness to outliers
6. Evaluate real-world deployment feasibility

2.3 Dataset Overview

Table 5: Bank Note Dataset Characteristics

Property	**Value**	**Description**
Total Samples	1,372	Balanced dataset
Genuine Notes (Class 0)	762	55.5%
Forged Notes (Class 1)	610	44.5%
Features	4	Wavelet coefficients
Training Split	960 (70%)	Stratified
Test Split	412 (30%)	Stratified

Table 6: Feature Statistics

Feature	**Mean**	**Std**	**Min**	**Max**
Variance	0.43	2.84	-7.04	6.82
Skewness	1.92	5.87	-13.77	12.95
Curtosis	1.40	4.31	-5.29	17.93
Entropy	-1.19	2.10	-8.55	2.45

2.4 Methodology

2.4.1 Logistic Regression Framework

Hypothesis Function:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} = \sigma(\theta^T x) \tag{8}$$

Cost Function (Binary Cross-Entropy):

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] \tag{9}$$

With L2 Regularization:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{10}$$

Gradient Update:

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \tag{11}$$

2.5 Implementation Code

```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix

# Load data
df = pd.read_csv('BankNote_Authentication.csv')
X = df.drop('class', axis=1).values
y = df['class'].values.reshape(-1, 1)

# Split and standardize
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

def sigmoid(z):
    return 1 / (1 + np.exp(-np.clip(z, -500, 500)))

def logistic_regression(X, y, lambda_reg=0, learning_rate=0.1,
                        n_iterations=1000):
    X_b = np.c_[np.ones((X.shape[0], 1)), X]
    m = X_b.shape[0]
    theta = np.zeros((X_b.shape[1], 1))

    for iteration in range(n_iterations):
        z = X_b @ theta
        h = sigmoid(z)

        # Cost with L2 regularization
        cost = (-1/m) * np.sum(y * np.log(h + 1e-10) +
                               (1 - y) * np.log(1 - h + 1e-10))
        if lambda_reg > 0:
            cost += (lambda_reg / (2 * m)) * np.sum(theta[1:] **
               2)

        # Gradient with L2 regularization
        gradient = (1/m) * X_b.T @ (h - y)
        if lambda_reg > 0:
            gradient[1:] += (lambda_reg / m) * theta[1:]

        theta = theta - learning_rate * gradient

    return theta
```

Listing 4: Logistic Regression Implementation

```
def inject_outliers(X, y, outlier_fraction=0.1, shift_magnitude
   =5):
    X_outlier = X.copy()
    n_outliers = int(len(X) * outlier_fraction)
    outlier_indices = np.random.choice(len(X), n_outliers,
                                       replace=False)

    for idx in outlier_indices:
        X_outlier[idx] += np.random.randn(X.shape[1]) *
           shift_magnitude

    return X_outlier, y, outlier_indices
```

Listing 5: Outlier Injection for Robustness Testing

2.6 Results and Analysis

2.6.1 Classification Performance

Table 7: Model Performance Comparison

Method	Train Acc (%)	Test Acc (%)	Gap (%)
No Regularization ($\lambda = 0$)	97.92	97.33	0.59
L2 Regularization ($\lambda = 1$)	97.92	97.33	0.59
L2 Regularization ($\lambda = 10$)	97.60	97.33	0.27
Top 3 Features Only	97.92	96.84	1.08

Table 8: Confusion Matrix (Test Set, No Regularization)

		Predicted	
		Genuine	Forged
2***Actual**	Genuine	218	11
	Forged	0	183

Performance Metrics:

- **Precision (Genuine):** 100% – All predicted genuine notes are correct
- **Recall (Genuine):** 95% – Detected 95% of genuine notes
- **Precision (Forged):** 94% – 94% accuracy in detecting forgeries
- **Recall (Forged):** 100% – All forged notes correctly identified
- **F1-Score:** 97-98% – Excellent balance

2.6.2 Regularization Analysis

Table 9: Performance across Different λ Values

λ	**Train Acc (%)**	**Test Acc (%)**	**Comment**
0	97.92	97.33	Baseline
0.001	97.92	97.33	No change
0.01	97.92	97.33	Stable
0.1	97.92	97.33	Stable
1.0	97.92	97.33	Stable
10.0	97.60	97.33	Slight train drop
50.0	95.62	95.39	Underfitting
100.0	93.85	93.93	Severe underfitting

Key Finding: Model shows minimal overfitting. Classes are highly separable, so regularization provides marginal benefit.

2.6.3 Feature Importance

Table 10: Feature Importance Ranking

Rank	**Feature**	**Absolute Weight**
1	Variance	3.5294
2	Skewness	2.9254
3	Curtosis	2.7849
4	Entropy	0.2390

Observation: Top 3 features (variance, skewness, curtosis) achieve 96.84% test accuracy, demonstrating strong discriminative power.

2.6.4 Outlier Robustness Analysis

Table 11: Impact of Outlier Injection

Outlier %	Train Acc (%)	Test Acc (%)	Δ Test Acc (%)
0	97.50	97.33	0.00 (baseline)
5	91.67	93.20	-4.13
10	87.19	91.02	-6.31
15	82.71	87.62	-9.71
20	79.58	86.89	-10.44

Critical Insight: Model is **sensitive to outliers**. 20% contamination causes 10.44% accuracy drop, suggesting need for robust preprocessing in production.

2.6.5 Visualizations

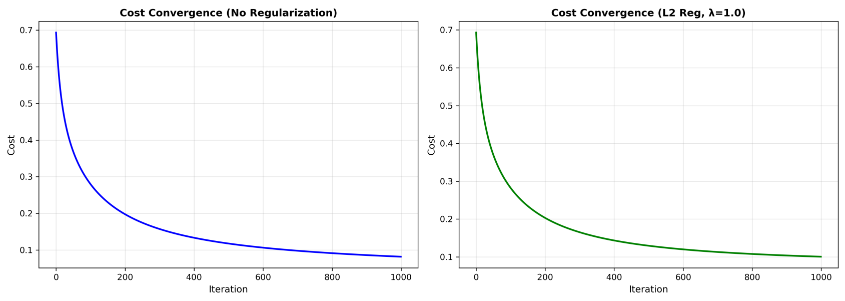

Figure 4: Cost function convergence comparing regularized and non-regularized models. Both converge smoothly, indicating stable optimization

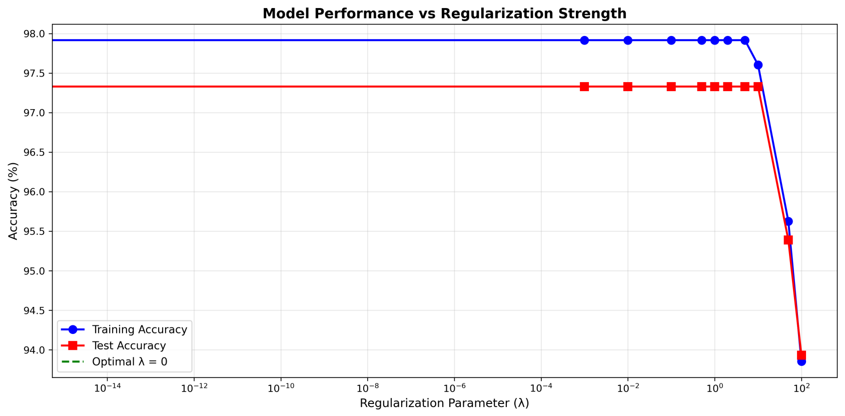

Figure 5: Training and test accuracy vs regularization strength. Flat response indicates minimal overfitting in baseline model

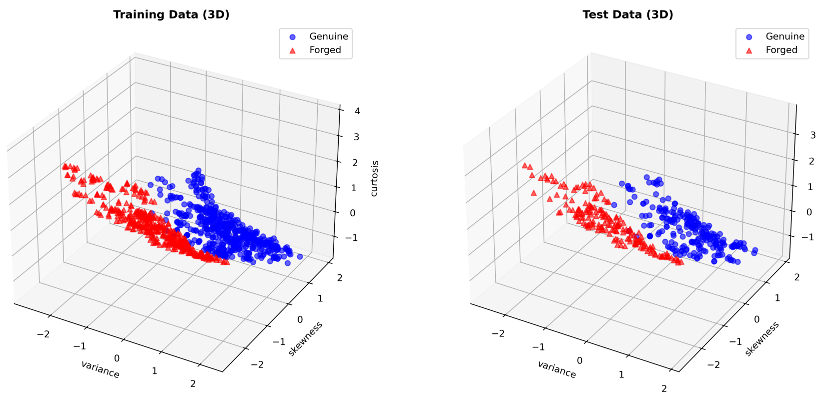

Figure 6: 3D scatter plot using top 3 features (variance, skewness, curtosis). Clear spatial separation between genuine (blue) and forged (red) notes validates linear separability

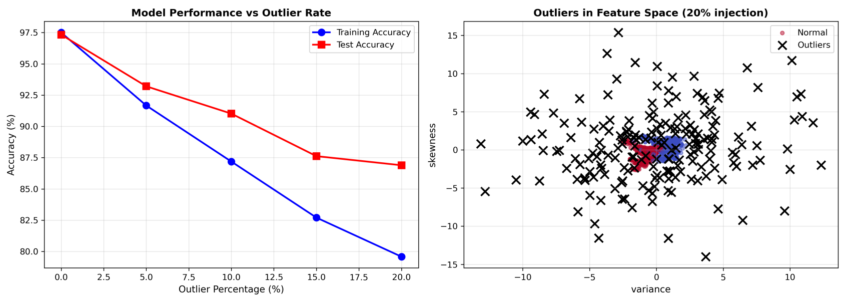

Figure 7: Left: Accuracy degradation with increasing outlier rate. Right: Visual representation of injected outliers (black crosses) in feature space

2.7 Learning Outcomes

- Gained hands-on experience with **Linear and Logistic Regression** from scratch using matrix operations and gradient descent.
- Understood the importance of **feature standardization** and **regularization (L2/Ridge)** in improving model stability and generalization.
- Developed ability to evaluate models using key metrics like **R², RMSE, accuracy, precision, recall, and F1-score**.
- Learned to interpret **feature importance** and analyze their impact on predictions.
- Observed the effects of **regularization and outliers**, enhancing understanding of robustness and overfitting.
- Strengthened skills in **data preprocessing, visualization, and analytical reasoning** for real-world ML tasks.