Enhanced Logistic Regression on the Titanic Dataset

Soorya s

October 3, 2025

1 Introduction

This report presents a logistic regression model applied to the Titanic dataset. The model includes:

- L_2 regularization to control overfitting,
- momentum gradient descent to smooth convergence,
- learning rate decay for stability,
- grid search for hyperparameter tuning.

We preprocess the data, derive the mathematics, implement in Python, and present the results.

2 Mathematical Formulation

Given $X \in \mathbb{R}^{m\times n}$ (feature matrix), $\vec{w} \in \mathbb{R}^n$ (weights), and $b \in \mathbb{R}$ (bias):

$$\vec{z} = X\vec{w} + b\mathbf{1}, \qquad \hat{\vec{p}} = \sigma(\vec{z}) = \frac{1}{1+e^{-\vec{z}}}.$$

The loss function is binary cross-entropy with L_2 penalty:

$$L(\vec{w}, b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\ln \hat{p}^{(i)} + (1-y^{(i)})\ln(1-\hat{p}^{(i)})\right] + \frac{\lambda}{2m}\|\vec{w}\|^2.$$

Gradients are:

$$\nabla_{\vec{w}}L = \frac{1}{m}X^\top(\hat{\vec{p}} - \vec{y}) + \frac{\lambda}{m}\vec{w}, \qquad \frac{\partial L}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(\hat{p}^{(i)} - y^{(i)}).$$

Momentum Update

Momentum accumulates gradients:

$$\vec{v}_t = \beta\vec{v}_{t-1} + (1-\beta)\nabla_{\vec{w}}L, \quad \vec{w} \leftarrow \vec{w} - \alpha_t\vec{v}_t, \quad b \leftarrow b - \alpha_t v_{b,t}.$$

Learning Rate Decay

We use reciprocal decay:

$$\alpha_t = \frac{\alpha_0}{1 + kt}.$$

3 Preprocessing

We use z-score normalization:

$$x_j'^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}, \qquad \sigma_j = 0 \;\Rightarrow\; \sigma_j = 1.$$

The dataset is split:

$$\text{Train: } 80\%, \quad \text{Validation: } 10\%, \quad \text{Test: } 10\%.$$

4 Implementation

Data Loading and Preprocessing

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Load Titanic dataset
df = pd.read_csv("train.csv")
df = df[["Survived", "Pclass", "Sex", "Age", "SibSp", "Parch", "Fare"]]

# Handle missing values
df["Age"].fillna(df["Age"].mean(), inplace=True)

# Encode categorical variable
df["Sex"] = (df["Sex"] == "male").astype(int)

# Build feature matrix and target
X_full = df[["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare"]].
    to_numpy(float)
y_full = df["Survived"].to_numpy(float)
m_full, n = X_full.shape

# Split 80/10/10
rng = np.random.default_rng(42)
perm = rng.permutation(m_full)
m_train = int(0.80 * m_full)
m_val = int(0.10 * m_full)
idx_train = perm[:m_train]
idx_val = perm[m_train:m_train+m_val]
idx_test = perm[m_train+m_val:]
X_train, y_train = X_full[idx_train], y_full[idx_train]
X_val, y_val = X_full[idx_val], y_full[idx_val]
X_test, y_test = X_full[idx_test], y_full[idx_test]

```

```python
# Normalize
mu = X_train.mean(axis=0)
sigma = X_train.std(axis=0, ddof=0)
sigma[sigma == 0] = 1.0
X_train = (X_train - mu) / sigma
X_val = (X_val - mu) / sigma
X_test = (X_test - mu) / sigma
```

Model and Loss

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

def f(X, w, b):
    return sigmoid(X @ w + b)

def loss(X, y, w, b, lam):
    m = len(y)
    p = f(X, w, b)
    eps = 1e-12
    p = np.clip(p, eps, 1 - eps)
    ce = -np.mean(y*np.log(p) + (1-y)*np.log(1-p))
    l2 = (lam / (2*m)) * np.sum(w*w)
    return ce + l2
```

Momentum Step and LR Decay

```python
def step_mom(X, y, w, b, v_w, v_b, lr, lam, beta=0.9):
    m = len(y)
    p = f(X, w, b)
    e = p - y
    g_w = (X.T @ e) / m + (lam/m) * w
    g_b = e.mean()

    v_w = beta * v_w + (1-beta) * g_w
    v_b = beta * v_b + (1-beta) * g_b

    w = w - lr * v_w
    b = b - lr * v_b
    return w, b, v_w, v_b

def lr_decay(alpha0, t, k=1e-3):
    return alpha0 / (1 + k*t)

def accuracy(y, p, thr=0.5):
    yhat = (p >= thr).astype(float)
    return (yhat == y).mean()
```

Grid Search

```python
lam_vals = [0.0, 0.005, 0.01, 0.05, 0.1]
```

```python
alpha0_vals = [0.05, 0.1, 0.2, 0.3]
beta_vals = [0.85, 0.9, 0.95]

best_val = 0
best = {}

for lam in lam_vals:
    for a0 in alpha0_vals:
        for beta in beta_vals:
            w = np.zeros(n); b = 0.0
            v_w = np.zeros(n); v_b = 0.0
            prev_L = None
            for epoch in range(1, 3001):
                lr = lr_decay(a0, epoch, k=1e-3)
                w, b, v_w, v_b = step_mom(X_train, y_train, w, b, v_w,
                    v_b, lr, lam, beta)
                L = loss(X_train, y_train, w, b, lam)
                if prev_L is not None and abs(prev_L - L) < 1e-7: break
                prev_L = L
            val_acc = accuracy(y_val, f(X_val, w, b))
            if val_acc > best_val:
                best_val = val_acc
                best = {"lam": lam, "a0": a0, "beta": beta}
```

Final Training and Evaluation

```python
lam, a0, beta = best["lam"], best["a0"], best["beta"]
w = np.zeros(n); b = 0.0
v_w = np.zeros(n); v_b = 0.0

for epoch in range(1, 5001):
    lr = lr_decay(a0, epoch, k=1e-3)
    w, b, v_w, v_b = step_mom(X_train, y_train, w, b, v_w, v_b, lr, lam
        , beta)
    if epoch % 500 == 0:
        print("Epoch", epoch, "Train Loss", loss(X_train,y_train,w,b,
            lam))

p_tr = f(X_train, w, b)
p_va = f(X_val, w, b)
p_te = f(X_test, w, b)

print("Train Acc:", accuracy(y_train, p_tr))
print("Val Acc:", accuracy(y_val, p_va))
print("Test Acc:", accuracy(y_test, p_te))
```

5 Results

Grid search gave best parameters: $\lambda = 0$, $\alpha_0 = 0.1$, $\beta = 0.85$. Final accuracies:

$$\text{Train: } \approx 82.9\%, \quad \text{Validation: } \approx 77.5\%, \quad \text{Test: } \approx 78.9\%.$$

6 Conclusion

The logistic regression model with momentum and learning rate decay converges smoothly and avoids oscillations. Hyperparameter tuning confirmed that regularization was not needed, since the dataset was large enough and momentum already stabilized training.