TÌM HIỂU VÀ XÂY DỰNG MÔ HÌNH LOGISTIC REGRESSION

Nguyen Tien Manh

Tháng 1 năm 2026

Tóm tắt nội dung

Tóm tắt: Logistic Regression là kỹ thuật nền tảng trong bài toán phân loại nhị phân. Báo cáo này trình bày cơ sở lý thuyết từ trực giác xác suất đến việc xây dựng hàm mất mát Cross-Entropy.

Mục lục

1 Giới thiệu

Trong thực tiễn có rất nhiều vấn đề cần được phân loại, dự đoán như phân loại ảnh chó/mèo, dự đoán một người có mắc bệnh hay không.... Trong báo cáo này, chúng tôi tập trung triển khai thuật toán Logistic Regression từ đầu (from scratch) để hiểu rõ cơ chế hoạt động nội tại của mô hình trước khi áp dụng các thư viện sẵn có để áp dụng vào bài toán dự đoán bệnh tim mạch.

2 Cơ sở lý thuyết

2.1 Đặt vấn đề và trực giác

Trong bài toán phân loại nhị phân (0 hoặc 1), mô hình hồi quy tuyến tính (Linear Regression) $y = wx + b$ bộc lộ nhiều hạn chế:

1. Giá trị dự đoán có thể nằm ngoài khoảng $[0, 1]$, không mang ý nghĩa xác suất.
2. Nhạy cảm với các điểm dữ liệu nhiễu (outliers).

Do đó, ý tưởng cốt lõi là cần một "hàm nén"(squashing function) để ánh xạ đầu ra tuyến tính vào khoảng $(0, 1)$, biểu thị xác suất $P(y = 1|x)$.

2.2 Mô hình Logistic Regression

Xuất phát từ trực giác trên, chúng tôi sử dụng hàm kích hoạt **Sigmoid**:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

Trong đó $z = w^T x + b$. Khi đó, $\hat{y}$ được hiểu là xác suất mẫu dữ liệu thuộc lớp dương tính.

2.3 Xây dựng hàm chi phí

2.3.1 Tại sao không dùng Mean Squared Error (MSE)?

Trong Linear Regression, ta thường dùng MSE làm hàm mất mát. Tuy nhiên, nếu áp dụng MSE vào Logistic Regression (với hàm kích hoạt phi tuyến Sigmoid), hàm mất mát thu được sẽ trở thành hàm **không lồi** (non-convex). Điều này dẫn đến việc hàm số có nhiều cực trị địa phương (local minima), khiến thuật toán tối ưu không thể tìm được nghiệm tốt nhất toàn cục.

2.3.2 Tiếp cận theo ước lượng hợp lý cực đại

Để khắc phục, hàm mục tiêu của Logistic Regression được xây dựng dựa trên nguyên lý thống kê: **Ước lượng hợp lý cực đại** (Maximum Likelihood Estimation - MLE).

Giả sử biến mục tiêu y tuân theo phân phối Bernoulli. Xác suất để một mẫu dữ liệu x có nhãn y được viết gọn là:

$$P(y|x) = \hat{y}^y (1 - \hat{y})^{1-y} \tag{2}$$

(Nghĩa là: Nếu $y = 1$, xác suất là $\hat{y}$; nếu $y = 0$, xác suất là $1 - \hat{y}$).

Giả định các mẫu dữ liệu là độc lập (i.i.d), hàm hợp lý (Likelihood) cho toàn bộ tập dữ liệu m mẫu là tích xác suất của từng mẫu:

$$L(w,b) = \prod_{i=1}^{m} P(y^{(i)}|x^{(i)}) = \prod_{i=1}^{m} [\hat{y}^{(i)}]^{y^{(i)}} [1-\hat{y}^{(i)}]^{1-y^{(i)}} \tag{3}$$

2.3.3 Hàm mất mát

Mục tiêu là tìm w, b để cực đại hóa khả năng mô hình khớp với dữ liệu thực tế ($L(w,b) \to \max$). Để đơn giản hóa tính toán (chuyển tích thành tổng) và tránh lỗi tràn số, ta lấy logarit tự nhiên (Log-Likelihood):

$$\log L(w,b) = \sum_{i=1}^{m} [y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)})] \tag{4}$$

Trong bài toán tối ưu, ta thường thực hiện *cực tiểu hóa* thay vì cực đại hóa. Do đó, ta nhân biểu thức trên với $-\frac{1}{m}$ để thu được hàm mất mát **Binary Cross-Entropy**:

$$J(w,b) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)})] \tag{5}$$

Hàm này đã được chứng minh là hàm lồi (convex), đảm bảo Gradient Descent sẽ luôn tìm được cực trị toàn cục.

2.4 Thuật toán tối ưu

Sau khi thiết lập hàm mục tiêu, ta sử dụng Gradient Descent để cập nhật tham số theo hướng ngược chiều đạo hàm (hướng dốc xuống nhanh nhất) nhằm cực tiểu hóa sai số.

Quy tắc cập nhật tại mỗi bước lặp t:

$$w^{(t+1)} := w^{(t)} - \alpha \frac{\partial J}{\partial w} \tag{6}$$

$$b^{(t+1)} := b^{(t)} - \alpha \frac{\partial J}{\partial b} \tag{7}$$

Trong đó α là tốc độ học (learning rate). Chi tiết các bước biến đổi toán học để tính đạo hàm (Gradient) được trình bày tại **Phụ lục A**.

3 Thực nghiệm

3.1 Thiết kế thí nghiệm

- **Dữ liệu:** Tập dữ liệu `SAHeart.csv` bao gồm các đặc trưng: *sbp, tobacco, ldl, adiposity, famhist, typea, obesity, alcohol, age.*
- **Tiền xử lý:**
  - Mã hóa biến định danh: `famhist` (Present $\to$ 1, Absent $\to$ 0).
  - Chuẩn hóa (Normalization): Sử dụng Z-score scaling để đưa các đặc trưng về cùng phân phối chuẩn $\mathcal{N}(0,1)$, giúp Gradient Descent hội tụ nhanh hơn.
- **Cấu hình:** Learning rate $\alpha = 0.01$, số vòng lặp 1500. Tỷ lệ train/test là 80/20.

3.2 Kết quả thực nghiệm

Quá trình huấn luyện cho thấy hàm mất mát giảm đều đặn từ 0.693 xuống 0.518, chứng tỏ thuật toán hoạt động đúng lý thuyết.

Kết quả đánh giá trên tập kiểm thử (Test set):

Độ đo (Metric)	**Kết quả**
Accuracy (Độ chính xác)	0.742
Precision	0.571
Recall	0.444
F1 Score	0.500
ROC AUC	0.654

Bảng 1: Hiệu năng mô hình trên tập Test

3.3 Phân tích và đánh giá

Mô hình đạt độ chính xác tổng thể khá tốt ($\approx 74\%$). Tuy nhiên, chỉ số **Recall thấp (0.44)** cho thấy một hạn chế lớn của phương pháp hiện tại: mô hình bỏ sót hơn một nửa số ca bệnh thực tế. Nguyên nhân có thể do:

- Dữ liệu mất cân bằng giữa hai lớp bệnh và không bệnh.
- Đường phân chia tuyến tính (linear decision boundary) của Logistic Regression chưa đủ phức tạp để tách biệt dữ liệu trong không gian nhiều chiều.

4 Thảo luận và kết luận

4.1 Tổng kết

Bài báo cáo đã trình bày quy trình xây dựng hệ thống Logistic Regression hoàn chỉnh: từ cơ sở toán học, cài đặt thuật toán, đến thực nghiệm. Kết quả cho thấy mô hình tuyến tính đơn giản có thể hoạt động hiệu quả cho bài toán cơ bản nhưng còn hạn chế với các trường hợp phức tạp.

4.2 Hạn chế nghiên cứu và hướng phát triển

Trong khuôn khổ nghiên cứu này, chúng tôi chưa thực hiện:

- Tối ưu hóa siêu tham số (Hyperparameter Tuning) cho learning rate hay số vòng lặp.
- Xử lý vấn đề mất cân bằng dữ liệu (ví dụ: kỹ thuật SMOTE hay thay đổi ngưỡng threshold).

Hướng phát triển tiếp theo sẽ là áp dụng các kỹ thuật Regularization (L1/L2) để tránh Overfitting và thử nghiệm các mô hình phi tuyến (như Neural Networks) để cải thiện chỉ số Recall.

A Phụ lục: Đạo hàm Gradient

Để áp dụng Gradient Descent, ta cần tính đạo hàm riêng của hàm mất mát J theo trọng số w. Áp dụng quy tắc chuỗi (Chain rule):

$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_j} \tag{8}$$

Sau các bước biến đổi đại số (được lược bỏ để ngắn gọn), ta thu được công thức đóng gọn gàng:

$$\frac{\partial J}{\partial w} = \frac{1}{m} X^T (\hat{y} - y) \tag{9}$$

Đây chính là công thức vector hóa được sử dụng trong phương thức `fit()` của mã nguồn.