Logistic Regression

1.1 What is Logistic Regression?

Logistic Regression là một quá trình mô hình hóa xác xuất của một kết quả riêng biệt vơới biến đầu vào. Nó được sử dụng để dự đoán một biến phụ thuộc có giá trị rời rạc (nhị phân, đa lớp,..) dựa trên một tập hợp các biến độc lập. Hồi quy logistic là một kỹ thuật phân tích dữ liệu sử dụng toán học để tìm ra mối quan hệ giữa hai yếu tố dữ liệu. Sau đó, kỹ thuật này sử dụng mối quan hệ đã tìm được để dự đoán giá trị của những yếu tố đó dựa trên yếu tố còn lại. Dự đoán thường cho ra một số kết quả hữu hạn, như có hoặc không.

Ví dụ: giả sử bạn muốn đoán xem khách truy cập trang web của bạn sẽ nhấp vào nút thanh toán trong giỏ hàng của họ hay không. Phân tích hồi quy logistic xem xét hành vi của khách truy cập trước đây, chẳng hạn như thời gian dành cho trang web và số lượng các mặt hàng trong giỏ hàng. Quá trình phân tích này xác định rằng, trước đây, nếu khách truy cập dành hơn năm phút trên trang web và thêm hơn ba mặt hàng vào giỏ hàng, họ sẽ nhấp vào nút thanh toán. Nhờ vào thông tin này, sau đó, hàm hồi quy logistic có thể dự đoán hành vi của một khách mới truy cập trang web.

1.2 Binary Logistic Classification

Hồi quy Binary Logistic là mô hình phổ biến trong nghiên cứu dùng để ước lượng xác suất một sự kiện sẽ xảy ra. Đặc trưng của hồi quy nhị phân là biến phụ thuộc chỉ có hai giá trị: 0 và 1.

1.3 Training a Binary Classifier

Các phương pháp Support Vector Machine đã đề cập (Hard Margin, Soft Margin, Kernel) đều được xây dựng để giải quyết bài toán Binary Classification, tức bài toán phân lớp với chỉ hai classes. Việc này cũng giống như Percetron Learning Algorithm hay Logistic Regression vậy. Các mô hình làm việc với bài toán có 2 classes còn được gọi là Binary classifiers.

Đối với bài toán phân loại nhị phân, chúng ta có thể sử dụng Logistic Regression để dự đoán xác suất của một mẫu thuộc về một lớp nào đó. Nếu xác suất lớn hơn 50%, mẫu đó sẽ được dự đoán là thuộc về lớp đó, ngược lại thì không.

Để huấn luyện một mô hình Logistic Regression, chúng ta cần tính toán độ lỗi của mô hình. Để tính toán độ lỗi, chúng ta cần một hàm mất mát. Hàm mất mát thường được sử dụng trong Logistic Regression là Cross-Entropy Loss.

Về phần phương pháp tối ưu, chúng ta có thể sử dụng Gradient Descent để tối ưu hàm mất mát.

Để đánh giá mô hình, chúng ta có thể sử dụng các độ đo như Accuracy, Precision, Recall, F1-score

1.3.1 Cross-Entropy Loss

Cross-Entropy Loss được sử dụng để đo lường độ lỗi của mô hình phân loại. Nó được tính bằng cách tính toán tổng các điểm dữ liệu trong tập huấn luyện. Để tính toán Cross-Entropy Loss, chúng ta cần biết xác suất dự đoán của mô hình cho mỗi điểm dữ liệu. Để tính toán xác suất dự đoán, chúng ta cần sử dụng hàm Sigmoid.

Hàm Sigmoid được sử dụng để chuyển đổi giá trị đầu ra của mô hình thành một giá trị xác suất nằm trong khoảng từ 0 đến 1. Hàm Sigmoid được định nghĩa như sau:

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

Công thức Cross-Entropy Loss:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log\left(\phi\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(z^{(i)}\right)\right) \right]$$

Đạo hàm của Cross-Entropy Loss:

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(\phi \left(z^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

Trong đó:

$\phi \left(z^{(i)} \right)$ là xác suất dự đoán của mô hình cho điểm dữ liệu thứ i.
$y^{(i)}$ là nhãn của điểm dữ liệu thứ i.
$x_j^{(i)}$ là giá trị của thuộc tính thứ j của điểm dữ liệu thứ i.
θ_j là tham số thứ j của mô hình.

1.3.2 Gradient Descent

Gradient Descent (GD) là thuật toán tìm tối ưu chung cho các hàm số. Ý tưởng chung của GD là điều chỉnh các tham số để lặp đi lặp lại thông qua mỗi dữ liệu huấn luyện để giảm thiểu hàm chi phí.

Gradient Descent là một thuật toán tối ưu lặp (iterative optimization algorithm) được sử dụng trong các bài toán Machine Learning và Deep Learning (thường là các bài toán tối ưu lồi — Convex Optimization) với mục tiêu là tìm một tập các biến nội tại (internal parameters) cho việc tối ưu models. Trong đó:

Gradient: là tỷ lệ độ nghiêng của đường dốc (rate of inclination or declination of a slope). Về mặt toán học, Gradient của một hàm số là đạo hàm của hàm số đó tương ứng với mỗi biến của hàm. Đối với hàm số đơn biến, chúng ta sử dụng khái niệm Derivative thay cho Gradient.

Descent: là từ viết tắt của descending, nghĩa là giảm dần.

Gradient Descent có nhiều dạng khác nhau như Stochastic Gradient Descent (SGD), Mini-batch SDG. Nhưng về cơ bản thì đều được thực thi như sau:

Khởi tạo biến nội tại.

Đánh giá model dựa vào biến nội tại và hàm mất mát (Loss function).

Cập nhật các biến nội tại theo hướng tối ưu hàm mất mát (finding optimal points).

Lặp lại bước 2, 3 cho tới khi thỏa điều kiện dừng.

Công thức cập nhật cho GD có thể được viết là:

$$\theta_j := \theta_j - \eta \frac{\partial J}{\partial \theta_j}$$

Trong đó:

- η là learning rate.
- $\frac{\partial J}{\partial \theta_j}$ là đạo hàm của hàm mất mát theo tham số thứ j.

a) Stochastic Gradient Descent

Thay vì sử dụng toàn bộ tập huấn luyện thì Stochastics Gradient Descent (SGD) sẽ lấy ngẫu nhiên 1 phần tử ở tập huấn luyện và thực hiện tính lại vector độ dốc dựa chỉ dựa trên 1 điểm dữ liệu, sau đó lặp đi lặp lại đến khi kết thúc. Và việc tính toán dựa trên 1 điểm dữ liệu sẽ khiến thuật toán chạy nhanh hơn bởi có rất ít dữ liệu cần xử lý ở mỗi vòng lặp. Và điều này cũng giúp mô hình có thể được huấn luyện với những dữ liệu lớn vì mỗi vòng lặp chỉ cần đưa 1 điểm dữ liệu vào trong bộ nhớ.

Mặt khác do tính chất ngẫu nhiên của dữ liệu đưa vào nên trong quá trình huấn luyện, thay vì hàm chi phí giảm từ từ giống Batch GD thì hàm chi phí của SGD sẽ lúc tăng lúc giảm nhưng sẽ giảm dần theo khoảng thời gian. Dần dần nghiệm của bài toán nó sẽ tiệm cận rất gần với cực tiểu nhưng khi đã đạt được cực tiểu thì giá trị hàm chi phí sẽ liên tục thay đổi mà không giữ ổn định. Khi gặp điều kiện dừng ta sẽ được bộ tham số cuối cùng đủ tốt, nhưng chưa thật sự tối ưu.

Khi hàm chi phí liên tục thay đổi có thể giúp thuật toán nhảy ra khỏi cực tiểu địa phương. Vì vậy SGD có cơ hội để tìm được cực trị toàn cục hơn là Batch Gradient Descent. Vì vậy lựa chọn ngẫu nhiên dữ liệu sẽ giúp thoát khỏi nghiệm tối ưu cục bộ nhưng điều đó nghĩa là thuật toán cũng không bao giờ có nghiệm cực tiểu

$$\theta_j := \theta_j - \eta \left(\phi \left(z^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

Trong đó:

- x_i là điểm dữ liệu thứ i.
- θ là tham số của mô hình.
- η là learning rate.
- $\phi \left(z^{(i)} \right)$ là xác suất dự đoán của mô hình cho điểm dữ liệu thứ i.

b) Batch Gradient Descent

Batch Gradient Descent là phương pháp cập nhật tham số sau khi tính toán đạo hàm trên toàn bộ tập huấn luyện. Đây là phương pháp cập nhật tham số chậm nhất vì chúng ta phải tính toán đạo hàm trên toàn bộ tập huấn luyện trước khi cập nhật tham số.

Cách làm này có một vài hạn chế đối với cơ sở dữ liệu có vô cùng nhiều điểm.Việc phải tính toán lại đạo hàm với tất cả các điểm này sau mỗi vòng lặp trở nên cồng kềnh và không hiệu quả.

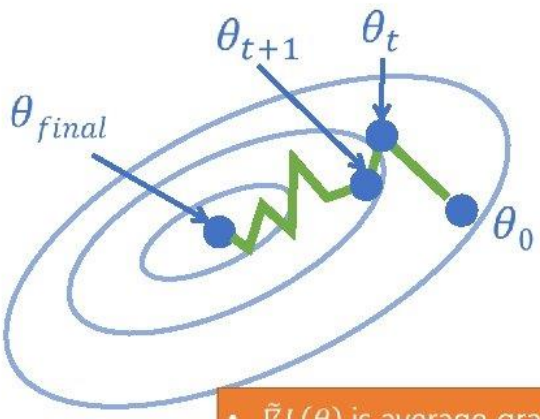
$$\theta_j := \theta_j - \eta \frac{1}{m} \sum_{i=1}^{m} \left(\phi \left(z^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

c) Mini-batch Gradient Descent

Thuật toán Gradient Descent cuối cùng mà chúng ta nghiên cứu đó là Mini-batch Gradient Descent. Nếu chúng ta hiểu về batch GD và SGD thì sẽ dễ dàng hiểu về Mini-batch GD: ở mỗi bước, thay vì tính toán vector độ dốc dựa trên toàn bộ tập huấn luyện (như thằng Batch GD) hoặc dựa trên 1 điểm dữ liệu (như thằng Stochastic GD) thì Mini-batch GD tính gradients dựa trên 1 tập nhỏ ngẫu nhiên các điểm dữ liệu được gọi là mini- batches. Ưu điểm chính của Mini-batch GD so với Stochastic GD đó là chúng ta có thể tận dụng tối đa được hiệu suất phần cứng khi thực hiện các phép toán trên ma trận, ví dụ như GPU (tận dụng khả năng tính toán song song của phần cứng)

Mini-batch GD sẽ tiến đến gần cực tiểu toàn cục hơn SGD nhưng sẽ khó để thoát khỏi cực tiểu địa phương hơn. Dù vậy SGD và Mini-batch GD sẽ tiến đến cực tiểu toàn cục nếu chúng ta áp dụng giảm dần learning rate hợp lý.

Minibatch stochastic gradient descent



- Initialize θ_0 randomly
- For t in 0,..., T_{maxiter}

$$\theta^{t+1} = \theta^t - \underbrace{\eta_t \cdot \tilde{\nabla}_B L(\theta)}_{\text{minibatch gradient}}$$

where minibatch B is chosen randomly

- $\tilde{\nabla} L(\theta)$ is average gradient over random subset of data of size B
- Per-iteration comp. cost = O(B)