AI VIET NAM – Course 2024

Insight into Logistic Regression

Hoàng-Nguyên Vũ

Ngày 21 tháng 3 năm 2024

1. **Tóm lược:**

   - **Hồi quy Logistic** là một mô hình thống kê được sử dụng để dự đoán xác suất của một sự kiện xảy ra. Nó phù hợp với các vấn đề phân loại, đặc biệt là các vấn đề phân loại nhị phân chỉ có hai kết quả có thể xảy ra. Ví dụ:
     - Dự đoán một bệnh nhân có bị ung thư hay không dựa trên các triệu chứng của họ.
     - Dự đoán một khách hàng có mua sản phẩm hay không dựa trên dữ liệu hành vi của họ.
   - **Cách thức hoạt động:** Hồi quy Logistic sử dụng một hàm logistic để mô tả mối quan hệ giữa các biến độc lập (biến dự đoán) và biến phụ thuộc (biến mục tiêu). Hàm logistic có dạng hình chữ S, với đầu vào là một giá trị thực và đầu ra là một xác suất nằm trong khoảng từ 0 đến 1.

     $$P(y = 1|x) = \frac{e^x}{1 + e^x} \Leftrightarrow \frac{1}{1 + e^{-x}} \tag{1}$$

     Trong đó:
     - P(y = 1 | x): Xác suất của biến phụ thuộc y bằng 1 cho một giá trị cụ thể của biến độc lập x.
     - x: Giá trị của biến độc lập.
     - e: Hằng số Euler (khoảng 2.71828).

2. **Vectorization :**

   - **Vectorization** là kỹ thuật sử dụng các phép toán vector để thực hiện các phép toán trên nhiều giá trị cùng lúc. Kỹ thuật này giúp tăng hiệu quả và tốc độ xử lý dữ liệu, đặc biệt là khi sử dụng với các mô hình học máy và các ứng dụng khoa học tính toán.
   - **Lợi ích của Vectorization:**
     - Tăng hiệu quả: Vectorization cho phép thực hiện các phép toán trên nhiều giá trị cùng lúc, giúp giảm thời gian xử lý dữ liệu.
     - Tăng tốc độ: Việc sử dụng các thư viện lập trình tối ưu hóa cho phép thực hiện vectorization một cách hiệu quả, giúp tăng tốc độ xử lý dữ liệu.
     - Dễ dàng sử dụng: Các thư viện lập trình cung cấp các hàm vectorization dễ dàng sử dụng, giúp đơn giản hóa việc viết mã.

Bảng 1: Bảng so sánh giữa cách cài đặt thông thường và sử dụng Vectorization

Tính năng	Vectorization	Không sử dụng Vectorization
Hiệu quả	Cao	Thấp
Tốc độ	Nhanh	Chậm
Dễ sử dụng	Dễ dàng (với thư viện)	Khó khăn
Độ phức tạp	Phức tạp (khó hiểu)	Dễ hiểu
Khả năng tương thích	Thấp (không phải thư viện nào cũng hỗ trợ)	Cao
Sử dụng bộ nhớ	Ít	Nhiều
Khả năng mở rộng	Dễ dàng mở rộng cho tập dữ liệu lớn	Khó khăn mở rộng cho tập dữ liệu lớn
Lượng mã	Ít	Nhiều

Ví dụ:

* **Sử dụng Vectorization:**

```
import numpy as np

X = np.array([[1, 2], [3, 4], ..., [100, 200]])
y = np.array([0, 1, ..., 1])

def logistic_regression(X, y):
  w = np.zeros((X.shape[1], 1))

  for i in range(len(X)):
    z = X[i] @ w
    p = sigmoid(z)
    loss = -y[i] * log(p) - (1 - y[i]) * log(1 - p)

    # Cập nhật tham số w

  return w

w = logistic_regression(X, y)
```

* **Không sử dụng Vectorization:**

```
import numpy as np

X = np.array([[1, 2], [3, 4], ..., [100, 200]])
y = np.array([0, 1, ..., 1])

def logistic_regression(X, y):
  w = np.zeros((X.shape[1], 1))

  for i in range(len(X)):
    z = 0
    for j in range(X.shape[1]):
      z += X[i][j] * w[j]
    p = sigmoid(z)
    loss = -y[i] * log(p) - (1 - y[i]) * log(1 - p)

    # Cập nhật tham số w

  return w

```

```
w = logistic_regression(X, y)
```

- **Kết luận:** Vectorization là kỹ thuật hữu ích giúp tăng hiệu quả và tốc độ xử lý dữ liệu trong Logistic Regression. Việc sử dụng vectorization giúp đơn giản hóa việc viết mã và tăng hiệu quả của mô hình.

3. **Ôn tập Toán cơ bản dùng trong Vectorization:**

- **Tích vô hướng hai ma trận:** Phép nhân tích vô hướng ma trận là phép toán kết hợp hai ma trận có cùng kích thước để tạo ra một số thực. Phép toán này được thực hiện bằng cách tính tổng tích các phần tử tương ứng của hai ma trận.
  Công thức: Cho hai ma trận A và B có cùng kích thước (m, n), phép nhân tích vô hướng được tính toán như sau:

$$A \cdot B = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij} B_{ij} \tag{2}$$

  Ví dụ: Giả sử ta có hai ma trận A và B như sau: A $= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ và B $= \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$. Phép nhân tích vô hướng của hai ma trận A và B là C $= \begin{bmatrix} (1*5+2*7) & (1*6+2*8) \\ (3*5+4*7) & (3*6+3*8) \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$
  Trong Python, chúng ta sẽ dùng như sau để tính tích vô hướng 2 ma trận:

```
import numpy as np

A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

dot_product = np.dot(A, B)

print(dot_product)
```

- **Ma trận chuyển vị:** là một ma trận được tạo ra bằng cách đổi vị trí các hàng và cột của một ma trận ban đầu. Giả sử có một ma trận A có kích thước (m, n). Ma trận chuyển vị của A được ký hiệu là A^T (hoặc A') và có kích thước (n, m). Ma trận A^T được tạo ra bằng cách hoán đổi vị trí của hàng i và cột i của ma trận A với mọi i từ 1 đến m.
  Ví dụ: Giả sử có một ma trận A như sau: $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ Ma trận chuyển vị của A được tính toán như sau: $A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$.
  Trong Python, chúng ta sẽ dùng như sau để chuyển vị ma trận:

```
import numpy as np

mat_A = np.array([[1, 2, 3], [4, 5, 6]])

mat_A_Transpose = mat_A.T

print(mat_A_Transpose)
```

4. **Ứng dụng Vectorization trong Logistic Regression:**

Vectorization (N-Features)

```
def predict(X, theta):
    return sigmoid_function( np.dot(X.T
    , theta) )

def loss_function(y_hat, y):
    return -y*np.log(y_hat) - (1 - y)*
    np.log(1 - y_hat)

def compute_gradient(X, y_hat, y):
    return X*(y_hat - y)

def update(theta, lr, gradient):
    return theta - lr*gradient
```

Cách thông thường (N-Features)

```
def predict(x1, x2, b, w1, w2):
    z = x1*w1 + x2*w2 + b
    y_hat = sigmoid_function(z)

    return y_hat

def compute_gradient(x1, x2, y_hat, y):
    db  = (y_hat - y)
    dw1 = x1*(y_hat - y)
    dw2 = x2*(y_hat - y)

    return (db, dw1, dw2)
```

Kết luận: Việc sử dụng vectorization giúp đơn giản hóa việc viết mã và tăng hiệu quả của mô hình

5. **Binary Cross-entropy và Mean Squared Error:**

   (a) **Binary Cross-entropy (BCE):** BCE là thước đo mức độ khác biệt giữa hai phân phối xác suất trong trường hợp phân loại nhị phân. Nó được sử dụng phổ biến trong học máy để đánh giá hiệu suất của các mô hình phân loại nhị phân. Công thức như sau:

$$\mathcal{L} = -\sum_i y_i \log(\hat{y}_i) - (1 - y_i) \log(1 - \hat{y}_i) \tag{3}$$

   Trong logistic regression, Loss function phổ biến được sử dụng là binary cross-entropy (hay còn gọi là log loss) vì nó phản ánh tốt hơn sự khác biệt giữa các xác suất dự đoán và nhãn thực tế.

   (b) **Mean Squared Error (MSE):** MSE là thước đo mức độ sai lệch giữa giá trị dự đoán và giá trị thực tế. Nó được sử dụng phổ biến trong học máy để đánh giá hiệu suất của các mô hình hồi quy. Công thức như sau:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4}$$

   Bên cạnh hàm binary cross-entropy, Mean Squared Error (MSE) có thể được sử dụng làm Loss-function nhưng sẽ không phù hợp trong logistic regression vì nó không đảm bảo Loss function là hàm phi tuyến và giới hạn các dự đoán trong khoảng [0, 1], dẫn đến các vấn đề trong tối ưu hóa và hiệu suất của mô hình.

Bảng 2: Bảng so sánh BCE và MSE

Khía cạnh	**BCE**	**MSE**
Loại bài toán	Phân loại nhị phân	Hồi quy
Mục đích	Đo lường mức độ khác biệt giữa hai phân phối	Đo lường mức độ sai lệch giữa giá trị dự đoán và giá trị thực tế
Công thức	Phức tạp hơn	Đơn giản hơn
Ý nghĩa	Phù hợp cho đánh giá hiệu suất mô hình phân loại	Phù hợp cho đánh giá hiệu suất mô hình hồi quy

(c) **Kết luận:** Trong Logistic Regression, lựa chọn phù hợp là BCE. Nó trực tiếp đo lường mức độ khác biệt giữa xác suất thực tế và xác suất dự đoán, đồng thời phạt nặng những sai lệch lớn. Mặc dù MSE có ưu điểm về đơn vị đo lường dễ diễn giải, nhưng nó lại không phù hợp với bản chất của bài toán phân loại nhị phân.

6. **Optimiztion:**

- **Optimization** là quá trình tìm kiếm các giá trị của các tham số trong mô hình logistic regression sao cho mô hình phù hợp nhất với dữ liệu. Phương pháp này hoạt động bằng cách lặp đi lặp lại điều chỉnh các giá trị tham số theo hướng dốc âm của hàm mất mát cho đến khi đạt được điểm tối thiểu cục bộ. Hàm mất mát thường được sử dụng là cross-entropy loss hoặc mean squared error.
- Các bước Optimzation trong Logistic Regression đối với 1+ Samples, Mini-batch và Batch như sau:

Require: Training data X (n × m matrix), labels y (n × 1 vector), learning rate η, maximum iterations T
Ensure: Weights vector θ (m × 1 vector)
Initialize $\theta = \mathbf{0}_m$, bias $b = 0$
for $t = 1$ T **do**
  for $i = 1$ n **do**
    $x_i \leftarrow$ i-th row of X
    $y_i \leftarrow y[i]$
    $z_i = \theta^T x_i + b$
    $\hat{y}_i = \sigma(z_i)$ (sigmoid function)
    $error_i = y_i - \hat{y}_i$
    for $j = 1$ m **do**
      $\theta_j \leftarrow \theta_j + \eta error_i x_{ij}$
    end for
    $b \leftarrow b + \eta error_i$
  end for
end for
Return θ

1: **function** SIGMOID(z)
2: Return $\frac{1}{1+e^{-z}}$
3: **end function**

- Hết -