Acad. Year

2020-21

Project No.

A123

Thuộc bài tập lớn Phát hiện gian lận tín dụng

XGBoost

- **Nguyễn Hải Long 18020032**
- **Nguyễn Việt Long 18020793**
- **Trịnh Tuấn Tú 18021352**

Đại học Công Nghệ - Đại học Quốc Gia Hà Nội

XGBoost

Nguyễn Hải Long 18020032

Nguyễn Việt Long 18020793

Trịnh Tuấn Tú 18021352

Đại học Công Nghệ & khoa Khoa học Máy Tính

Báo cáo kết quả nghiên cứu bài tập lớn môn Học Máy

Bài toán phát hiện gian lận tín dụng

Năm học 2020/21

Mục lục

1 Hàm mục tiêu cho Graident Tree Boosting

Tree Boosting là một thuật toán thuộc lớp Ensemble Tree. Ý tưởng cơ bản của lớp thuật toán này đó chính là thay vì đi tìm một mô hình phức tạp thì ta sử dụng nhiều mô hình đơn giản và kết hợp kết quả của chúng lại. Việc kết hợp này đã được các nhà nghiên cứu chứng minh là đem lại sự cải tiến không hề nhỏ về độ chính xác của mô hình. Thuật ngữ "Boosting" chỉ ra cách ta kết hợp kết quả các cây lại với nhau bằng một hàm cộng. Cụ thể của thuật toán sẽ được trình bày ở phần này.

1.1 Regularized hàm mục tiêu

Cho một tập dữ liệu có n mẫu, và mỗi mỗi có m đặc trưng $\mathcal{D} = \{(x_i, y_i)\}(|\mathcal{D}| = n, x_i \in \mathbb{R}^m), y_i \in \mathbb{R}$. Một hình Tree-Boosting là việc đưa x_i qua K cây quyết định rồi sau đó sử dụng một hàm cộng tất cả các kết quả đầu ra của các cây đó làm kết quả. Công thức thường hay được sử dụng được mô tả như công thức 1.

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}, \tag{1}$$

Trong đó:

$$\mathcal{F} = \{f(x) = w_{q(x)}\}(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T)$$

$\mathcal{F}$ là không gian các cây hồi quy (CART) có tác dụng mapping từ một vector đầu vào có m chiều cho ra kết quả là chỉ số của lá tương ứng với dữ liệu đó. T ở đây chính là số lá, mỗi lá có chứa một trọng số w_i liên quan tới kết quả phân lớp tại lá đó. Đối với mỗi điểm dữ liệu, ta cho dữ liệu đó qua các cây hồi quy trong tập cây đang có, và cho ra kết quả cuối cùng là tổng tất cả các trọng số tại các lá kết quả của mỗi cây. Để học được tập hàm $\mathcal{F}$ sử dụng trong mô hình (bao gồm cả cấu trúc cây và trọng số của lá) thì ta sử dụng hàm mục tiêu đã được regularized như công thức 2.

$$\mathcal{L}(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \tag{2}$$

Trong đó

$$\Omega(f_k) = \gamma T + \frac{1}{2}||w||^2$$

l ở đây chính là một hàm lồi hai biến, ước lượng độ khác nhau của kết quả dự đoán của mô hình $\hat{y}_i$ và nhãn thật của dữ liệu y_i. Còn lại, hàm Ω có tác dụng phạt độ phức tạp của cây. Việc sử dụng hàm phạt này sẽ giúp tránh được hiện tượng Overfitting của mô hình.

1.2 Tối ưu hàm Loss (Gradient Tree Boosting)

Hàm mục tiêu (loss) được chỉ ra ở công thức 2 nhận tham số đầu vào là một hàm khác (hàm Ω - hàm số biểu diễn cho 1 cây hồi quy) và không thể tối ưu như với các hàm ở không gian Euclid như bình thường. Nếu ta giả sử $\hat{y}_i^{(t)}$ là kết quả dự đoán của mẫu thứ i tại cây thứ t. Ta sẽ cần tối ưu hàm mất mát tại bước thứ t như công thức 3.

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{3}$$

Điều này có nghĩa là ta sẽ tối ưu theo chiến thuật tham lam, thêm f_t sao cho cải thiện được hàm mất mát tại bước đó càng nhiều càng tốt. Ta sử dụng ước lượng bậc 2 để có thể tối ưu được hàm mất mát tại công thức 3 nhanh hơn.

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} [l(y^i, \hat{y}^{y-1}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{4}$$

Trong đó:

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, y^{t-1})$$

và

$$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, y^{t-1})$$

Bỏ đi biến là hằng số, ta được:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{5}$$

Nếu như định nghĩa $I_j = i|q(x_i) = j$ là tập các mẫu thuộc lá j, ta có thể khai triển công thức 5 như sau:

$$\begin{aligned}\mathcal{L}^{(t)} &= \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}[w_j * \sum_{i\in I_j} g_i + \frac{1}{2}w_j^2(\sum_{i\in I_j} h_i + \lambda)] + \gamma T\end{aligned} \tag{6}$$

Với một cây cố định về cấu trúc, ta có thể tính ra được tham số w_i tối ưu dựa vào việc đạo hàm công thức 6, và cho ra kết quả như công thức 7.

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda} \tag{7}$$

Như vậy giá trị hàm mất mát tối ưu tại thời điểm t như trong công thức.

$$\hat{\mathcal{L}}^{(t)} = -\frac{1}{2}\sum_{j=1}^{t}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T \tag{8}$$

Công thức tối ưu của hàm mất mát được biểu diễn trên được sử dụng để đánh giá độ tốt của một cây phân lớp / hồi quy. Trên thực tế, trong khi xây dựng cây phân lớp/ hồi quy, ta khó có thể thử hết tất cả cấu trúc của cây để tìm ra cấu trúc nào là tốt nhất. Thay vào đó, ta sử dụng chiến thuật tham lam để xây dựng cây. Bắt đầu từ một nút lá, ta lần lượt thêm các nhánh cho cây. Độ tốt của việc phân nhánh sẽ được tính toán sử dụng công thức 8. Nếu giả sử I_L và I_R là tập các mẫu dữ liệu ở nhánh con trái và nhánh con phải, và $I = I_L \cup I_R$, độ giảm của hàm mất mát được tính bằng công thức 9.

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{9}$$

Công thức trên thường được sử dụng để đánh giá các phép chia dữ liệu ở một nút.

2 Thuật toán phân nhánh dữ liệu

Ở phần trước, ta đã thu được công thức ước lượng độ tốt của một phép chia dữ liệu tại một nút của cây. Tuy nhiên, bài toán tiếp theo cần giải quyết đó chính là chia dữ liệu như thế nào. Ở phần này, chúng tôi sẽ trình bày một số chiến lược chọn cách chia dữ liệu tại một nút của cây phân lớp/ hồi quy.

2.1 Thuật toán tham lam chính xác cơ bản (Basic Greedy Algorithm)

Thuật toán này đơn giản chỉ là thử từng trường hợp chia dữ liệu có thể có tại nút đó. Tuy nhiên, trước khi thử thì ta cần sắp xếp lại giá trị của đặc trưng sau đó thực thiện vòng lặp từ đầu đến cuối, giá trị hàm mất mát của con phải và con trái sẽ được tính bằng cách cộng tích luỹ liên tục để không mất thời gian tính lại. Cụ thể thuật toán trên sẽ được mô tả bằng đoạn giả code 1.

Algorithm 1 Thuật toán tham lam chính xác dùng để tìm phép chia dữ liệu tại mỗi nút

I: tập mẫu tại nút hiện tại
d: số chiều dữ liệu
procedure GREEDYSPLITFINDING(I, d)
    $gain \leftarrow 0$
    $G \leftarrow \sum_{i \in I} g_i, \quad H \leftarrow \sum_{i \in I} h_i$
    for $k = 1$ `to` m **do**
        $G_L \leftarrow 0$
        $H_L \leftarrow 0$
        for j `in` $sorted(I, byx_{jk})$ **do**
            $G_L \leftarrow G_L + g_j, \quad H_L \leftarrow H_L + h_j$
            $G_R \leftarrow G - G_L, \quad H_R \leftarrow H - H_L$
            $score \leftarrow \max\left(score, \frac{G_L^2}{H_L+\gamma} + \frac{G_R^2}{H_L+\gamma} - \frac{G^2}{H_L+\gamma}\right)$
    return `cách phân chia dữ liệu có score cao nhất`

2.2 Thuật toán ước lượng phân chia

Ở phần trước, thuật toán tham lam chính xác là một thuật toán rất mạnh vì nó kiểm tra hết tất cả các trường hợp phân chia dữ liệu có thể có và chọn ra cách có điểm cao nhất. Tuy nhiên, trong một số trường hợp dữ liệu quá lớn không thể đọc hết cùng một lúc thuật toán này sẽ không thể thực hiện được. Chính vì vậy, để phân chia dữ liệu một cách hiệu quả và hỗ trợ song song hoá trên nhiều bộ xử lý thì ta cần một thuật toán ước lượng phân chia.

Một cách tổng quan, thuật toán ước lượng phân chia sẽ ước lượng ra các điểm ứng viên, có thể dùng để phân chia dữ liệu dựa theo một tiêu chí đánh giá nào đó. Sau đó, ta chia tập dữ liệu thành các khối dựa vào những ứng viên đó, sử dụng các phép thống kê tổng hợp để chia dữ liệu.

Có hai tuỳ chọn trong thuật toán này. Tuỳ chọn thứ nhất đó chính là ta lựa chọn các ứng viên phân chia ngay từ bước đầu tiên trong quá trình xây dựng cây. Hai là với mỗi nút nhỏ ta lại thực hiện việc phân chia lại các ứng viên. Dễ dàng thây rằng, đối với tuỳ chọn thứ nhất, ta cần phải lựa chọn được càng nhiều ứng viên, chia dữ liệu ban đầu ra càng nhỏ càng tốt và ngược lại với tuỳ chọn thứ hai (chỉ cần chọn số ứng viên vừa đủ).

2.3 Weighted Quantile Sketch

Giả sử với đặc trưng k, ta định nghĩa tập sau $\mathcal{D}_k = \{(x_{1k}, h_1), (x_{2k}, h_2), ..., (x_{nk}, h_n)\}$. Ta định nghĩa hàm $r_k : R \to [0, +\infty]$ như sau:

$$r_k(z) = \frac{1}{\sum_{(x,h)\in\mathcal{D}_k} h} \sum_{(x,h)\in\mathcal{D}_k, x<z} \tag{10}$$

Hàm này có tác dụng biểu diễn phần trăm lượng mẫu có giá trị đặc trưng k bé hơn z. Mỗi mẫu dữ liệu ở đâu được gán cho một trọng số là h_i. Với một tham số chia là ε, ta cần tìm ta tập các ứng viên phân chia dữ liệu $s_{k1}, s_{k2}, ..., s_{kl}$ sao cho:

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \varepsilon, \;\; s_{k1} = \min_i x_{ik}, \;\; s_{kl} = \max_i x_{ik} \tag{11}$$

Dễ thấy, nếu theo cách phân chia này, dữ liệu có thể được chia thành $\frac{1}{e}$ phần. Tuy nhiên, tại sao lại chọn h_i làm trọng số cho mỗi mẫu dữ liệu? Câu trả lời nằm ở biểu thức 5 sau khi khai triển (thêm hằng số $(\frac{g_i}{h_i})^2$ để áp dụng hằng đẳng thức):

$$\hat{\mathcal{L}}^{(}t) = \sum_{i=1}^{n} \frac{1}{2} h_i \left(f_t(x_i) - \frac{g_i}{h_i} \right)^2 + \Omega(f_t) + constant \tag{12}$$

Công thức trên có thể coi như một hàm mất mát tổng bình phương nhỏ nhất với nhãn của mỗi dữ liệu là $\frac{g_i}{h_i}$ và mỗi dữ liệu được gán một trọng số là h_i tại hàm mất mát. Tuy nhiên, với một tập dữ liệu lớn, việc tìm các ứng viên để thoả mãn công thức 11 là một việc không hề đơn giản. Tuy nhiên, có một thuật toán được đề xuất bởi nhóm nghiên cứu của bài báo gốc có thể thực hiện việc này một cách hiệu quả. Thông tin chi tiết về thuật toán này được trình bày khá kĩ ở phần phụ lục của bài báo gốc.

2.4 Xử lý đặc trưng có giá trị thưa

Trên thực tế, có khá nhiều trường hợp giá trị của đặc trưng bị thưa vì nhiều lý do như: mất mát giá trị trong quá trình tiền xử lý, tần suất của giá trị 0 trong lúc thống kê, trích chọn đặc trưng sử dụng one-hot encoding. Để xử lý trường hợp này, mỗi khi dữ liệu bị mất (có giá trị Nan hoặc 0) sẽ được phân vào một nhánh mặc định nào đó. Nhánh mặc định này sẽ được học dựa trên dữ liệu. Thuật toán xử lý đối với giá trị bị thiếu sẽ được mô tả trong đoạn giả code 2. Sử cải tiến của thuật toán là chỉ phân nhánh những dữ liệu không bị thiếu và học ra được nhánh tối ưu nhất dành cho dữ liệu bị thiếu.

Algorithm 2 Thuật toán phân chia dữ liệu có xử lý trường hợp thiếu dữ liệu

I: tập mẫu tại nút hiện tại

$I_k = \{i \in I | x_{ik} \neq missing\ \}$

d: số chiều dữ liệu

procedure GREEDYSPLITFINDING(I, d)

    $gain \leftarrow 0$

    $G \leftarrow \sum_{i \in I} g_i, \quad H \leftarrow \sum_{i \in I} h_i$

    for $k = 1$ `to` m **do**

        $G_L \leftarrow 0$

        $H_L \leftarrow 0$

        for j `in` $sorted(I_k, ascentorderbyx_{jk})$ **do**

            $G_L \leftarrow G_L + g_j, \quad H_L \leftarrow H_L + h_j$

            $G_R \leftarrow G - G_L, \quad H_R \leftarrow H - H_L$

            $score \leftarrow \max\left(score, \frac{G_L^2}{H_L+\gamma} + \frac{G_R^2}{H_L+\gamma} - \frac{G^2}{H_L+\gamma}\right)$

        $G_R \leftarrow 0$

        $H_R \leftarrow 0$

        for j `in` $sorted(I_k, descentorderbyx_{jk})$ **do**

            $G_R \leftarrow G_R + g_j, \quad H_R \leftarrow H_R + h_j$

            $G_L \leftarrow G - G_R, \quad H_L \leftarrow H - H_R$

            $score \leftarrow \max\left(score, \frac{G_L^2}{H_L+\gamma} + \frac{G_R^2}{H_L+\gamma} - \frac{G^2}{H_L+\gamma}\right)$

    return `cách phân chia dữ liệu và nhánh mising-value có max score`