

Gradient Boosting: Regression, Classification

Regression 1.

Input: Data $\{(x_i, y_i)\}_{i=1}^n$, and a differentiable loss function $L(y_i, F(x))$

for regression: $L(y_i, F(x)) = \frac{1}{2}(y_i - \overset{\text{predicted}}{F(x)})^2$

Step 1. Initialize a model with a constant value $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^n L(y_i, \gamma)$

Step 2. For $m=1, \dots, M$:

(A) Compute $\underset{\text{pseudo residuals}}{r_{im}} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)}$ for $i=1, \dots, n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm} for $j=1, \dots, J_m$
we build a regression tree to predict the residuals instead of the weights.

(C) For $j=1, \dots, J_m$ compute $r_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, \underbrace{F_{m-1}(x_i)}_{\text{previous predict}} + \gamma)$

(D) Update $F_m(x) = F_{m-1}(x) + \underset{\text{learning rate } \in (0,1)}{\eta} \sum_{j=1}^{J_m} r_{jm} \mathbb{1}\{x \in R_{jm}\}$

Step 3. Output $F_M(x)$

Classification 2.

Input: Data $\{(x_i, y_i)\}_{i=1}^n$, differentiable loss function $L(y_i, F(x))$

$-\ell(\text{Data}) = -\sum_{i=1}^n y_i \log(p) + (1-y_i) \log(1-p)$
log-likelihood negative $L(y_i, F(x)) \uparrow$

Step 1. Initialize a model with a const. value: $\underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^n L(y_i, \gamma)$

Step 2. For $m=1, \dots, M$:

(A) Compute $r_{im} = \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)}$ for $i=1, \dots, n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm} , for $j=1, \dots, J_m$

(C) For $j=1, \dots, J_m$ compute $r_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$

Solving loss for an optimal γ is a mess. Thus, we can approximate with a Taylor expansion of L over γ .

$$L(y_1, F_{m-1}(x_1) + \gamma) \approx L(y_1, F_{m-1}(x_1)) + \frac{d}{dF} L(y_1, F_{m-1}(x_1)) \gamma + \frac{1}{2} \frac{d^2}{dF^2} L(y_1, F_{m-1}(x_1)) \gamma^2$$

Basically $\frac{dL(y_i, F_{m-1}(x_i) + \gamma)}{dF}$ \leftarrow 2nd-order Taylor polynomial \leftarrow grad of the old loss + current prediction

$\frac{d^2 L(y_i, F_{m-1}(x_i) + \gamma)}{dF^2} \leftarrow$ hessian of old loss + γ

Especially XGBoost does it like that

J_m - leaves

$$\frac{d}{d\gamma} \mathcal{L}(y_1, \hat{F}_{m-1}(x_1) + \gamma) \approx \frac{d}{d\hat{F}(1)} \mathcal{L}(y_1, \hat{F}_{m-1}(x_1)) + \frac{d^2 \mathcal{L}(y_1, \hat{F}_{m-1}(x_1))}{d\hat{F}(1)^2} \gamma$$

Set it to 0 \rightarrow solve for γ : $\gamma = - \frac{\frac{d}{d\hat{F}(1)} \mathcal{L}(y_1, \hat{F}_{m-1}(x_1))}{\frac{d^2}{d\hat{F}(1)^2} \mathcal{L}(y_1, \hat{F}_{m-1}(x_1))}$ Can be solved analytically

(D) Update $\hat{F}_m(x) = \hat{F}_{m-1}(x) + \gamma \sum_{j=1}^m \gamma_{jm} 1\{x \in R_{jm}\}$

Step 3. Output $\hat{F}_m(x)$

L_1, L_2 Regularization (Lasso, Ridge)

1. Ridge Regression

Ridge regression = penalty for least squares fit = $\lambda \cdot \text{slope}^2$ It adds some bias, but != smaller variance

Given $D = \{(x_i, y_i)\}_{i=1}^n$

$$\mathcal{L}(y, \hat{y}) = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \beta^2$$

Matrix notation: $RSS(\beta) = (\underbrace{y}_{\text{target}} - \underbrace{X\beta}_{\text{coefficients}})^T (\underbrace{y - X\beta}_{\text{target}}) + \lambda \beta^T \beta$ regularizer \Rightarrow will penalize large β 's.

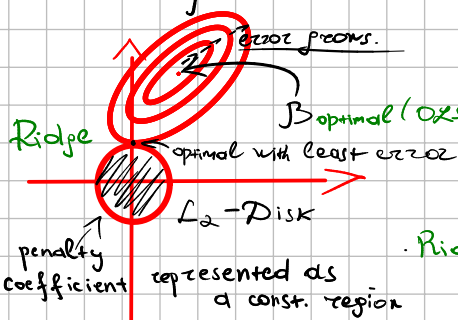
For an optimal solution: \min_{β}

Expanding loss $y^T y - y^T X \beta - X^T \beta^T y + X^T \beta^T X \beta + \lambda \beta^T \beta$

Now for min: $\frac{\partial RSS}{\partial \beta} = 0 \Rightarrow 0 - \underbrace{y^T X - X^T y}_{= -2 y^T X} + X^T X \cdot 2\beta + 2\lambda \beta = 0 \Rightarrow \underbrace{-y^T X + X^T X \hat{\beta}}_{+ \lambda \hat{\beta}} = 0$

$$X^T X \hat{\beta} + \lambda \hat{\beta} = y^T X$$

$$\hat{\beta} (X^T X + \lambda I) = y^T X \Rightarrow \hat{\beta}_{\text{ridge}} = (X^T X + \lambda I)^{-1} \cdot y^T X$$



Ridge is smooth \Rightarrow shrink all variables equally, none = 0. May cause overfitting (high variance) \Rightarrow If data is noisy it will chase every tiny bit of noise for min. error \Rightarrow high β .

Ridge - shrinks

Lasso - selects

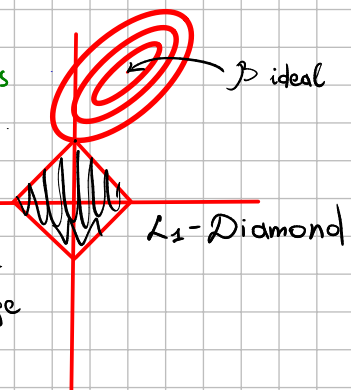
Actually, the ridge problem is $\hat{\beta}_{\text{ridge}} = \arg \min_{\beta} \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j)^2$

s.t. $\sum_{j=1}^p \beta_j^2 \leq t$, solving it by

1. Lagrangian

2. KKT \rightarrow 2. Objective

more strict than Ridge



2) Lasso Regression

Our penalty term: $\lambda \cdot |\text{slope}|$ L_1 -norm; λ - use Cross-Validation

Ridge and Lasso do the same thing, they make our model less sensitive.

Lasso is more strict, some features = 0

$RSS(\beta) = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda |\beta|$ We don't have a closed-form analytical solution like in Ridge

$|\beta|$ is not differentiable at 0. We want to minimize $L(y, \hat{y}) = \|X\beta - \hat{y}\|_1 = \frac{1}{n} \sum_{i=1}^n (y_i - \sum_{j=1}^p X_{ij}\beta_j)^2 + \lambda \|\beta\|_1$

Then, we do: 1. Compute the partial residual $r_i = y_i - \sum_{k \neq j} X_{ik}\beta_k$; 2. Compute the "strength" \rightarrow

\rightarrow of feature j $\beta_j = \sum_{i=1}^n X_{ij}(y_i - \hat{y}_{i, \text{not } j})$; 3. Apply soft thresholding: $\beta_j = S(\beta_j, \lambda) = \rightarrow \rightarrow$

$\rightarrow \begin{cases} (\beta_j - \lambda) / \sum X_{ij}^2, & \text{if } \beta_j > \lambda \\ (\beta_j + \lambda) / \sum X_{ij}^2, & \text{if } \beta_j < -\lambda \\ 0, & \text{if } |\beta_j| \leq \lambda \end{cases}$ We can transform the objective into a more convenient form:

$$\min_{\beta} L(y, \hat{y}) = \frac{1}{2n} \sum_{i=1}^n (y_i - \sum_{j=1}^p X_{ij}\beta_j)^2 + \lambda \|\beta\|_1$$

 \swarrow for simpler differentiation, possible to do cause min is the same β

for lasso: $\hat{\beta}_{\text{lasso}} = \arg\min_{\beta} \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p X_{ij}\beta_j)^2$
 s.t. $\sum_{j=1}^p |\beta_j| \leq t$

XGBoost: Regression, Classification

A - анимация, α_N - композиция на N -ой итерации $\nwarrow \alpha_{N-1}(x_i)$

На каждой итерации над бустинга вычисляется вектор ошибок S , - показывает как корректно \hat{x}_i анимация чтобы как можно сильнее уменьшить ошибку:

$$S = \left(-\frac{\partial L}{\partial z} \Big|_{z = \alpha_{N-1}(x_i)} \right)_{i=1}^L = -\nabla_z \sum_{i=1}^L L(y_i, z_i) \Big|_{z = \alpha_{N-1}(x_i)}$$

После этого новый базовый алгоритм $b_N(x)$ обучается путем минимизации MSE от вектора ошибок S :

$$b_N(x) = \arg\min_{b \in A} \sum_{i=1}^L (b(x_i) - S_i)^2$$

 \leftarrow мы используем MSE потому что прощаешь локальные оптимизации
 нет $\frac{1}{n}$ потому что не может min

вектор ошибок $\vec{S} = \nabla_z \alpha_{N-1}(x_i)$
 просто над. индекс обнуляют

Наша задача: $\sum_{i=1}^L L(y_i, \alpha_{N-1}(x_i) + b(x_i)) \rightarrow \min_b$

Разложим в ряд Тейлора: $\sum_{i=1}^L L(y_i, \alpha_{N-1}(x_i) + b(x_i)) \approx \sum_{i=1}^L (L(y_i, \alpha_{N-1}(x_i)) - S_i b(x_i) + \frac{1}{2} h_i b(x_i)^2)$

$$h_i = \frac{\partial^2 L(y_i, z)}{\partial z^2} \Big|_{z = \alpha_{N-1}(x_i)}$$

В контексте задачи оптимизации, 1-й член не важен \rightarrow убери

Задачу можно теперь представить так, $\sum_{i=1}^L (-S_i b(x_i) + \frac{1}{2} h_i b(x_i)^2) \rightarrow \min_b$

Он похож на MSE, $\sum_{i=1}^L (b(x_i) - S_i)^2 = \sum_{i=1}^L (b^2(x_i) - 2b(x_i)S_i + S_i^2) \Rightarrow \sum_{i=1}^L (b^2(x_i) - 2b(x_i)S_i)$

можно преобразовать еще раз, $= 2 \sum_{i=1}^L (-S_i b(x_i) + \frac{1}{2} b^2(x_i)) \rightarrow \min_b$ не зависит от b можно убрать

То есть, мы увидели что разложение в ряд Тейлора - хорошая аппроксимация. Здесь $h_i = 1$. Мы отбрасываем info о вторых производных, используем аппрокс. 2-го порядка, то есть считаем что функция имеет одинаковую кривизну по всем направлениям

Регуляризация

Будем работать с функцией $\sum_{i=1}^L (-S_i b(x_i) + \frac{1}{2} h_i b^2(x_i)) \rightarrow \min_b$

Сложность зависит от:

- 1: Число листьев $J \rightarrow$ больше параметров \rightarrow overfitting
- 2: Нормы coeffs в листьях $\|b\|_2^2 = \sum_{j=1}^J b_j^2$, чем сильнее $> 0 \Rightarrow$ больше веса у алгоритма

измеряет L после добавления нового $b(x_i)$ но! не штрафует

В классическом граф. бустинге мы пруним деревья (max-depth), но можно порогать более гибко.

Добавим регуляризаторы: $\sum_{i=1}^L (-S_i b(x_i) + \frac{1}{2} h_i b^2(x_i)) + \gamma J + \frac{1}{2} \sum_{j=1}^J b_j^2 \rightarrow \min_b$ \swarrow L_2 -pen, убеждает обученная

для каждого листа R_j и объектов в нем - одинаковое предсказание (вес) $f(x_i) \rightarrow b_j$

для мат. удобства

Мы можем упростить функционал и суммировать по всем листьям (J), а не по всей выборке i-тым объектам

Поэтому:
$$\sum_{j=1}^J \left\{ \left(-\sum_{i=1}^L s_i \right) \cdot b_j + \frac{1}{2} \left(\lambda + \sum_{i=1}^L h_i \right) b_j^2 + \gamma \right\} \rightarrow \min_b$$

Каждый терм можно минимизировать независимо по b_j . Везде этот функционал — парабола с $a > 0$ — ветви вверх

Значит оптимальный $b_j = -\frac{b}{2a} \Rightarrow b = -\sum_{i=1}^L s_i = -S_j$, $a = \frac{1}{2} \left(\lambda + \sum_{i=1}^L h_i \right) = \frac{1}{2} (\lambda + H_j) \Rightarrow$

$\Rightarrow b_j = \frac{S_j}{\lambda + H_j} = \frac{S_j}{\lambda + H_j}$ Наш функционал был: $\sum_{j=1}^J \left\{ -S_j b_j + \frac{1}{2} (\lambda + H_j) b_j^2 + \gamma \right\} \rightarrow \min_b$

Упростим подставляя оптимальный b_j : $\sum_{j=1}^J \left\{ -S_j \cdot \frac{S_j}{\lambda + H_j} + \frac{1}{2} (\lambda + H_j) \cdot \frac{S_j^2}{(\lambda + H_j)^2} + \gamma \right\} \rightarrow \min_b$

Получаем, $\sum_{j=1}^J \left\{ \frac{-S_j^2}{\lambda + H_j} + \frac{1}{2} \frac{S_j^2}{\lambda + H_j} + \gamma \right\} \Rightarrow \sum_{j=1}^J \left(\frac{-2S_j^2 + S_j^2}{2(\lambda + H_j)} + \gamma \right) = \left(\frac{1}{2} \sum_{j=1}^J \frac{-S_j^2}{H_j + \lambda} + \gamma J \right) = H(b)$

Обучение $b(x_i)$ — Decision Tree

$H(b) = \min_b \mathcal{L}(y_i, a_{n+1}(x_i) + b(x_i))$

С его помощью можно строить дерево $b(x_i)$

То есть можно сказать снижение функции потерь в узле R = выигрыш информации

Значит наша задача максимизировать Gain — прирост информации = Q

$Q = H(R) - H(R_L) - H(R_R) \rightarrow \max$ Выбираем такое разбиение чтобы прирост инфы (снижение потерь) $\rightarrow \max$

$H(R) = -\frac{1}{2} \left[\left(\sum_{(h_i, s_i) \in R} s_i \right)^2 / \left(\lambda + \sum_{(h_i, s_i) \in R} h_i \right) + \gamma \right]$

Особенности XGBoost: 1. Базовый алгоритм приближает направление, постигаемое с учетом вторых производных функции потерь; 2. Регуляризация функционала, L_2 для весов λ , и γ для кол-ва листьев J; 3. При построении дерева используется критерий информативности, зависит от оптимального вектора сдвига (антиградиента);

Стекинг.

Постройка ансамбля \rightarrow Бэггинг \rightarrow Бустинг \rightarrow Стекинг!

Независимо обучим N базовых алгоритмов $b_1(x), \dots, b_N(x)$ на выборках X_1, \dots, X_N теперь хотим обучить мета-алгоритм $a(x)$ на прогнозах базовых алго.

$b_j^{-K}(x)$ — базовый алгоритм, обученный по всем блокам кроме K -го

Функционал будет таким: $\sum_{k=1}^K \sum_{(x_i, y_i) \in X_k} \mathcal{L}(y_i, a(b_1^{-K}(x_i), \dots, b_N^{-K}(x_i))) \rightarrow \min_a$

В таком случае мы избегаем переобучения, мы тестируем ошибку на объекте x_i который не входит в K -ый блок, то есть b_j^{-K} — этот объект при обучении не видели

Категориальные и Текстовые признаки!

Для ансамблей над деревьями — проблема! — TF-IDF + Binary Encoding = Огромная размерность

Решение: (стемки, где $a(x)$ обучается град. функцией, а каждый кат./текст. признак = одно число базовым алгоритмом) Например: для кат. признаков = count — b_j , или линейные модели.