XGBoost

Jakub Tyrek

16 maja 2018

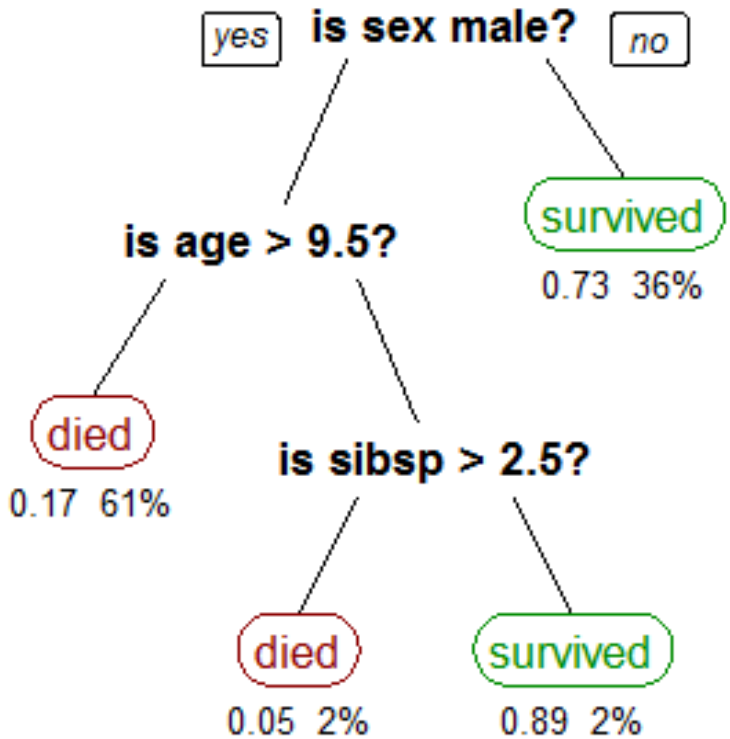
XGBoost - co to jest?

XGBoost - extreme gradient boosting - to bibiloteka (algorytm + implementacja) służąca do klasyfikacji / predykcji.

- boosting - metoda składania słabych klasyfikatorów w jeden silny.
- gradient boosting - metoda składania słabych klasyfikatorów w jeden silny z wykorzystaniem gradientu funckji kosztu.
- extreme - obliczeniowe sztuczki implementacyjne.

Drzewo decyzyjne

Słabym klasyfikatorem w XGBoost jest drzewo decyzyjne.



Boosting - ogólna idea

- Dana jest procedura tworzenia słabego klasyfikatora (f_k) (np. drzewo decyzyjne).
- Silny klasyfikator (ϕ_k) tworzony jest iteracyjnie.
- W pierwszej iteracji silny klasyfikator jest słabym klasyfikatorem.
- W kolejnych iteracjach silny klasyfikator jest poprawiany słabym klasyfikatorem $\phi_{k+1} = \phi_k + \alpha_k f_k$.
- Silny klasyfikator jest sumą słabych klasyfikatorów $\phi_K = \sum_{k=1}^{K} f_k$.

Sformułowanie zadania

Dany jest dataset $D = \{(x_i, y_i)\}$, $x_i \in \mathbb{R}^m$, $y_i \in \mathbb{R}$, $|D| = n$.
Zdefiniujmy $F = \{f(x) = w_{q(x)}\}$ - przestrzeń drzew decyzyjnych, gdzie T to liczba liści drzewa, $q : \mathbb{R}^m \to [T]$ funkcja przyporządkowująca przykładowi odpowiedni liść, a $w \in \mathbb{R}^T$ to wartość liścia. Predykcja $\hat{y}_i$ zadana jest wzorem $\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i)$, $f_k \in F$.
Prócz tego dana jest zregularyzowana funkcja kosztu $L(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$, gdzie l to różniczkowalna, wypukła funkcja kosztu, a $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_j w_j^2$ to czynnik regularyzujący.
Naszym zadaniem jest znalezienie ϕ, które minimalizuje L.

Oznaczenia

- $x_i \in \mathbb{R}^m$ - i - ty przykład w datasecie
- $y_i \in \mathbb{R}$ - wartość / klasa i - tego przykładu
- $D = \{(x_i, y_i)\}$ - dataset
- $n = |D|$ - liczba przykładów
- $\hat{y}_i$ - predykcja dla i - tego przykładu
- $q : \mathbb{R}^m \rightarrow [T]$ - funkcja wyboru liścia
- w_j - wartość w j - tym liściu
- F - przestrzeń słabych klasyfikatorów
- f_k - k - ty słaby klasyfikator
- ϕ - silny klasyfikator
- l - funkcja kosztu
- Ω - czynnik regularyzujący
- L - zregularyzowana funkcja kosztu

Gradient boosting

Załóżmy, że wykonano już $t-1$ iteracji. Niech $\hat{y}_i^{(t)} = \phi_t(x_i)$. Naszym zadaniem jest znalezienie f_t minimalizującego

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

Gradient boosting

Wprowadźmy $g_i = \frac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}}$, $h_i = \frac{\partial g_i}{\partial \hat{y}^{(t-1)}}$. Przybliżmy $L^{(t)}$ przez szereg Taylora

$$L^{(t)} \approx \sum_{i=1}^{n} \left(l(y_i, \hat{y}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right) + \Omega(f_t).$$

Po opuszczeniu tego, co nie zależy od f_t

$$\widetilde{L}^{(t)} = \sum_{i=1}^{n} \left(g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right) + \Omega(f_t).$$

Szukamy f_t, który minimalizuje $\widetilde{L}^{(t)}$.

Gradient boosting

Niech $I_j = \{i : q(x_i) = j\}$ będzie zbiorem indeksów tych przykładów, które wpadają w j - ty liść. Przepiszmy

$$\begin{aligned}\widetilde{L}^{(t)} &= \sum_{i=1}^{n} \left(g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right) + \gamma T + \frac{1}{2} \lambda \sum_{j=1} T w_j^2 \\ &= \sum_{j=1}^{T} \left((\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right) + \gamma T.\end{aligned}$$

Gradient boosting

Wybranie optymalnego f_t to wybranie odpowiednich splitów i w_j. $\widetilde{L}^{(t)}$ to funkcja kwadratowa od w_j. Stąd dla ustalonych splitów możemy wybrać optymalne wartości w_j^*

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

Po podstawieniu dostajemy

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

Gradient boosting

Wiemy już jak dobrać w_j. Teraz czas na splity. Struktura drzewa decyzyjnego powstaje iteracyjnie. Zaczynamy od pojedynczego liścia i możemy dokonywać splitów w sposób zachłanny. Niech $I = I_L \sqcup I_R$, I_L - indeksy, które trafiają po splicie do lewego liścia, I_R - analogicznie. Do oceny splitu wykorzystujemy poprzednie równianie. Spadek $\widetilde{L}^{(t)}$ po splicie to

$$L_{split} = \frac{1}{2}\Big(\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\Big) - \gamma.$$

Inne techniki regularyzacyjne

Prócz regularyzacji za pomocą wymienionych wcześniej, korzysta się jescze z dwóch technik.

- Shrinkage - skalowanie k - tego słabego klasyfikatora przez η^k, czyli $\phi = \sum_{k=1}^{K} \eta^k f_k$.
- Column subsampling - losujemy kolumny, które będą brane pod uwagę przy splitach.

Tworzenie splitów

Mamy trzy sposoby na budowanie drzewa:

- Splity dokładne
- Splity przybliżone
- Histogram

Splity dokładne

Powyższe pozwala nam sformułować dokładny sposób tworzenia drzewa.

- Dla każdej rozpatrywanej kolumny i dla każdej wartości w tej kolumnie liczymy wartość L_{split}.
- Dokonujemy podziału dla kolumny i wartości maksymalizującej L_{split}.
- Dokonujemy splitów dopóki nie osiągniemy maksymalnej głębokości.
- Dokonujemy pruningu splitów, dla których $L_{split} < 0$.
- Wartości NA kierowane są do tego liścia, który daje większy L_{split}.

Splity przybliżone

Poprzedni sposób działania wymagał sprawdzenia każdej wartości. Zamiast tego, można rozpatrywać tylko splity w wartościach pewnych percentyli. Ściślej niech $D_k = \{(x_{1k}, h_1), \ldots, (x_{nk}, h_n)\}$ będzie multi-zbiorem k - tej cechy. Definiujemy ważony kwantyl $r_k : \mathbb{R} \to [0, +\infty)$ wzorem

$$r_k(z) = \frac{1}{\sum_{(x,h)\in D_k} h} \sum_{(x,h)\in D_k, x<z} h.$$

Splity przybliżone

Dlaczego ważymy h? Inspiracja pochodzi z innej formy $\widetilde{L}^{(t)}$:

$$\widetilde{L}^{(t)} = \frac{1}{2}\sum_{i=1}^{n} h_i\left(f_t(x_i) - \frac{g_i}{h_i}\right)^2 + \Omega(f_t) + const.$$

Kandydatami na wartości splitu są te liczby $\{s_{k1}, \ldots, s_{kl}\}$, które spełniają

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon,\ s_{k1} = \min_i x_{ik},\ s_{kl} = \max_i x_{ik}.$$

Od parametru ϵ zależy liczba kandydatów na wartości. Ta metoda generowania splitów generuje nowych kandydatów w każdej iteracji. Splity wybieramy jak w poprzedniej metodzie.

Histogram

Trzecią metodą wybierania splitów jest histogram. Jest on bardzo podobny do splitów przybliżonych. Zamiast generowania kandydatów co iterację, każdy feature jest dyskretyzowany przed treningiem.

Parametry

- `eta` - η - parametr shrinkage.
- `gamma` - γ - parametr stojący przy liczbie liści w czynniku regularyzacyjnym.
- `max_depth` - maksymalna głębokość drzewa.
- `min_child_weight` - minimalna wartość sumy h_i w liściu potrzebna do przeprowadzenia splitu.
- `max_delta_step` - maksymalna wartość absolutna wartości w liściu.

Parametry

- `subsample` - jaka część przykładów jest wybierana do treningu.
- `colsample_bytree` - jaka część kolumn jest wybierana do treningu pojedynczego drzewa.
- `colsample_bylevel` - jaka część kolumn jest wybierana do pojedynczego splitu.
- `lambda` - λ - w czynniku regularyzacyjnym $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_j w_j^2 + \alpha \sum_j |w_j|$ przy normie L2.
- `alpha` - α - w czynniku lregularyzacyjnym przy normie L1.

Parametry

- `tree_method` - sposób tworzenia splitów
  - `'exact', 'gpu_exact'` - splity dokładne.
  - `'approx'` - splity przybliżone.
  - `'hist, 'gpu_hist'` - histogram.
  - `'auto'` - heurystyka wybierająca jedno z powyższych.
- `sketch_eps` - ϵ - używany tylko przy splitach przybliżonych
- `scale_pos_weight` - liczba przez jaką są mnożone wagi przykładów pozytywnych.
- `grow_policy` - używany w przypadku histogramu - w jakiej kolejności budować drzewo
  - `'depthwise'` - dziel od korzenia w dół
  - `'lossguide'` - dziel tam, gdzie największy L_{split}.
- `max_leaves` - maksymalna liczba liści.
- `max_bin` - maksymalna liczba binów w histogramie.