Gradient boosted trees

Lukáš Brchl

21. června 2018

1 Motivace

Gradient boosting je jedna z aktuálně hojně diskutovaných metod strojového učení, která patří do kategorie ensemble přístupů. V posledních letech získala na popularitě především díky knihovně zvané Extreme Gradient Boosting [1] (zkráceně XGBoost), kterou lze často nalézt na stupni vítězů v různých soutěžích zaměřených na strojové učení (např. Kaggle, KDDCup). Ensemble přístupy v dnešní době představují de facto standard pro jakékoliv vítězné řešení podobných soutěží a jejich značnou výhodou je, že při vynaložení minimálního úsilí na předzpracování dat dosahují nadprůměrných výsledků.

Vysoké popularitě též nasvědčuje fakt, že knihovna XGBoost vznikla v roce 2014 a již rok po jejím zveřejnění bylo 17 z 29 výherních řešení v soutěži Kaggle založeno právě na XGBoost. Navzdory všeobecné oblíbenosti gradient boosting metod, většina soutěžících často netuší jak fungují a využívají je pouze jako black-box k řešení libovolného problému, čímž snižují svou šanci na úspěch. Právě k osvětlení obecného fungování této metody a její implementace v knihovně XGBoost by se měla pokusit tato semestrální práce.

2 Základní princip

K tomu, aby bylo možné přejít k matematickým formulacím a pozdějším pokročilým trikům v XGBoost, je nejdříve nutno získat základní intuici a ujasnit si elementární pojmy, které se v tomto tématu vyskytují. Ty jsou stručně vysvětleny v této sekci.

2.1 Boosting

Prvním krokem k úspěchu je porozumění ensemblových metod založených na konceptu boosting. Cílem boostingu je vytvoření robustního modelu, který je složen z několika (klidně i tisíce) jednodušších klasifikátorů (taky nazývané weak learners). Jak boosting, tak i gradient boosting není z definice omezen na konkrétní druh klasifikátoru. Nejčastěji se však využívají rozhodovací stromy (konkrétněji klasifikační a regresní stromy CART [2]). Jednodušší klasifikátory si tak lze snadno představit jako například stromy hloubky jedna. Učení finálního modelu probíhá iterativně, přidáváním nových jednoduchých klasifikátorů, které jsou učeny na vhodně upravené distribuci původních dat. Přidávání modelů probíhá do doby, dokud není splněna zastavovací podmínka.

Nevýhodou tohoto postupu je, že výsledný model se již nedá snadno interpretovat, jako je tomu například u metody bagging v rozhodovacích lesích. Vzhledem k závislostem aktuálního modelu na předchozích jsou omezeny i způsoby paralelizace. Jedním ze zástupců klasického boostingu a předchůdce gradient boostingu je algoritmus AdaBoost [3].

2.2 Gradient boosting

V metodice gradient boostingu je výsledný model též budovaný iterativní formou, kdy v každé další iteraci usilujeme o vylepšení přesnosti předchozího modelu. Rozdílem je, že místo vážení jednotlivých přidávaných klasifikátoru jako je tomu u reprezentanta klasického boostingu AdaBoost [3], gradient boosting učí nový jednoduchý model na reziduích z předchozí iterace za účelem minimalizace ztrátové funkce (loss function). Tato formulace odpovídá známému algoritmu gradientního sestupu [4], odtud název gradient boosting.

Naivní formulace v matematické řeči
Cílem gradient boostingu je najít model F, pro jehož predikce $\hat{y} = F(x)$ platí, že součet ztrátové funkce $l(y_i, \hat{y}_i)$ přeš všechna trénovací data, je minimální. V každé iteraci $m, 0 \leq m \leq M$ algoritmu předpokládáme, že existuje nedokonalý model F_m a k jeho vylepšení se snažíme zkonstruovat model h_m takový, že nový model

$$F_{m+1}(x) = F_m(x) + h_m(x) \tag{1}$$

predikuje přesnější výsledky. K nalezení modelu h_m nám poslouží pozorování, že pro ideální volbu h_m platí

$$y = F_m(x) + h_m(x) \tag{2}$$

nebo také

$$h_m(x) = y - F_m(x). \tag{3}$$

$y - F_m(x)$ nazýváme rezidua, neboli výsledky se kterými si aktuální model F_m nedovedl vhodně poradit. Cílem modelu h_m je, se tyto nedostatky naučit kompenzovat a zlepšit tak přesnost v novém modelu F_{m+1}.

Jestliže nový model F_{m+1} stále není uspokojivý, můžeme opakovat operaci přidání nového modelu h_{m+1}. Tento proces opakujeme, dokud nedosáhneme požadované přesnosti nebo M iterací. Prvotní model ve většině implementací bývá dán jako $F_0(x) = \frac{\sum_{i=1}^{n} y_i}{n}$ nebo $F_0(x) = 0$. Výsledné predikce lze pak získat takto:

$$\hat{y} = F(x) = F_0(x) + \sum_{i=0}^{M} h_i(x). \tag{4}$$

Souvislost s gradientním sestupem
Pro jednoduchost nyní uvažujme kvadratickou ztrátovou funkci $l(y, F(x)) = \frac{1}{2}(y - F(x))^2$. Cílem je minimalizovat funkci $L = \sum_i l(y_i, F(x_i))$. Všimněme si, že $F(x_1), F(x_2), \ldots, F(x_n)$ jsou obyčejná čísla. Můžeme tedy $F(x_i)$ považovat za proměnou podle které funkci L zderivujeme a získáme

$$\frac{\partial L}{\partial F(x_i)} = \frac{\partial \sum_i l(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial l(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i. \tag{5}$$

Z toho lze vidět, že

$$y_i - F(x_i) = -\frac{\partial L}{\partial F(x_i)} \tag{6}$$

a to znamená, že rezidua v každé iteraci jsou rovny zápornému gradientu, který udává směr největšího poklesu funkce L. Po prostém dosazení do (1) dostáváme

$$F_{m+1}(x) = F_m(x) + y - F_m(x) = F_m(x) - 1 * \frac{\partial L}{\partial F_m(x)}. \tag{7}$$

Tím jsme se dostali k tomu, že model vlastně v každém kroku upravujeme pomocí algoritmu gradientního sestupu [4]. Výhoda této formulace je, že se jedná o zobecnění, která nám dovoluje využít libovolnou diferencovatelnou funkci l a zachovat vlastnosti původního algoritmu. Pro obecnou funkci l se už ale rezidua nemusí rovnat zápornému gradientu, proto v dalším textu už bude uvažován pouze koncept gradientu.

2.3 Gradient boosted trees

Gradient boosted trees (GBT) byly prvně popsány v článku [5] a mezi další používané názvy patří Gradient Boosting Machine (GBM) a Multiple Additive Regression Trees (MART). GBT lze již zařadit mezi popis konkrétní implementace, která je například zastoupena v balíčku scikit-learn. Dost často je pojem GBT komunitou bráno jako synonymum gradient boostingu, ale gradient boosting je obecnější pojem, který nespecifikuje druh weak learnerů.

Obecný algoritmus
Předpokládejme trénovací data ve formě $\{(x_i, y_i)\}_{i=1}^n$, funkci $l(y, F(x))$ se spojitou první derivací a číslo M značící počet iterací. Inicializujeme první model jako

$$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} l(y_i, \gamma), \tag{8}$$

což je v případě námi oblíbené kvadratické ztrátové funkce klasický průměr. Následně v každé iteraci $m, 1 \leq m \leq M$ opakuj:

1. Výpočet parciálních derivací (stejně jako v případě (5))

$$r_{i,m} = -\frac{\partial l(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}, \quad \text{pro} \quad \forall i = 1, \ldots, n. \tag{9}$$

2. Naučení jednoduchého stromu $h_m(x)$ na trénovacích datech $\{(x_i, r_{i,m})\}_{i=1}^n$ (v případě kvadratické ztrátové funkce se jedná přímo o rezidua, tedy $r_{i,m} = y_i - F_m(x_i)$.

3. Výpočet multiplikátoru γ_m pomocí algoritmu line search, který určuje ideální velikost kroku ve směru záporného gradientu, neboli

$$\gamma_m = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} l(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)). \tag{10}$$

4. Vytvoření nového model, který snižuje chybu modelu z předchozí iterace jako

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x). \tag{11}$$

Výsledný model je pak dán $F(x) = F_0(x) + \sum_{i=1}^{M} \gamma_i h_i(x)$.

Kromě této základní varianty existuje ještě několik vylepšení, které využívají vlastnosti rozhodovacích stromů a vylepšují robustnost budovaných modelů. Jedno z vylepšení je známé jako TreeBoost a jeho popis je k nalezení v originálním článku [5].

Regularizace
Stejně jako v ostatních modelech, požadujeme, aby se model při trénování příliš nepřizpůsoboval trénovacím datům (přeučení). V GBT je možné využít různých způsobů regularizace, ale jednou z nejdůležitějších je smršťování přidávaných modelů (shrinkage). Tento postup odpovídá drobné úpravě v rovnici (11) přidáním násobku $\alpha, 0 < \alpha \leq 1$ známe jako koeficient učení (learning rate). Model se tedy buduje takto:

$$F_m(x) = F_{m-1}(x) + \alpha\gamma_m h_m(x). \tag{12}$$

Další způsob regularizace spočívá například v tom, že v každé iteraci se nový model $h_m(x)$ neučí na všech datech, ale pouze na určité podmnožině. Tento způsob je známy jako stochastic gradient boosting. Poslední stojí za zmínku penalizace složitosti jednotlivých stromů, která bude následně k vidění v XGBoost.

3 XGBoost

XGBoost je open-source knihovna implementující algoritmus gradient boostingu na stromových strukturách. Je multiplatformní a poskytuje programová rozhraní pro C++, Javu, Python a R. Mezi její hlavní výhody patří, že s pomocí knihoven Hadoop a Spark lze využít na distribuovaných systémech.

Slovo extreme v názvu XGBoost ve skutečnosti nesouvisí s žádnou razantní modifikací originálního algoritmu, ale odkazuje na technický cíl maximálního zefektivnění všech nutných výpočtů. Skutečně, pokud bychom totiž detailněji studovali originální článek XGBoost [1], lze pozorovat, že se autoři opravdu snažili nalézt všechna možná slabá místa, která pomocí různých neočividných triků optimalizují.

V původním článku se občas vyskytuje na první pohled neintuitivní značení. Aby byla tato práce snadno pochopitelná i pro čtenáře originálního článku, je původní značení následováno. V nejasných případech je však vždy uveden vysvětlující text.

Základní terminologie
XGBoost rozšiřuje původní koncept gradient boostingu přidáním regularizace rovnou v minimalizované funkci. Tato funkce je v terminologii XGBoost označena jako *Obj* (objective function, cílová funkce). Předpokládejme model ve tvaru

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad x_i \in R^d, f_k \in F, \quad d, i, K \in N \tag{13}$$

kde $\hat{y}_i$ jsou jednotlivé predikce modelu, číslo K je počet stromů, $f_k(x)$ je regresní rozhodovací strom a F je množina všech možných regresních stromů. Pak lze cílovou funkci *Obj* vyjádřit takto:

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k), \tag{14}$$

kde $l(y, \hat{y})$ je libovolná ztrátová funkce, která musí mít spojité druhé derivace a Ω je regularizace měřící složitost jednotlivých stromů.

K tomu, aby bylo možné snáze vypočíst složitost stromu a následně provést některé další úpravy cílové funkce, je nutné znát reprezentaci jednotlivých stromů f_k. Mějme strom s T očíslovanými listy, kde každému listu přísluší jeho váha $w_i, 1 < i \leq T$, pak lze vyjádřit tímto způsobem:

$$f_k(x) = w_{q(x)} \quad w \in R^T, q : R^d \to \{1, 2, \ldots, T\}. \tag{15}$$

Tento vztah není na první pohled intuitivní, ale jeho smyslem je, že pro vstupní data vrátí funkce $f_k(x)$ váhu listu, do kterých data přísluší. Funkce $q(x)$ vrací pořadové číslo listu, ke kterému data náleží a definuje tak vlastně strukturu stromu. w_i jsou číselné hodnoty vah listů.

Nyní již máme způsob, jak můžeme vyjádřit funkci složitosti modelu $\Omega(f_k)$. Tato složitost může mít více podob, ale opět existuje jeden výhodný tvar, který nám později dovolí dělat zjednodušující úpravy cílové funkce. Funkce $\Omega(f_k)$ obsahuje uživatelsky definované hyper-parametry γ a λ a její vhodný tvar je například

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2. \tag{16}$$

Po dosazení do cílové funkce tak celkem získáváme

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \left(\gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \right), \tag{17}$$

kterou se snažíme minimalizovat. V tuhle chvíli to zatím ještě nevypadá příliš vábně a řešitelně, ale v další části se výraz pomocí několika úprav značně zjednoduší.

Zjednodušení tvaru cílové funkce
K tomu abychom snadněji viděli, co přesně se bude náš model učit, je vhodné v cílové funkci provést několik zjednodušujících úprav. Ve výsledku pak bude v každé iteraci snazší vyčíslit hodnotu cílové funkce, což v implementaci nabízí značné zrychlení.

Trénování modelu probíhá ve stejném duchu jako klasický gradient boosting. V každé iteraci t pracujeme s predikcemi $y^{(t-1)}$ z předchozí iterace a snažíme se najít strom $f_t(x)$, který minimalizuje výraz

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t), \quad \hat{y_0} = 0. \tag{18}$$

To obecně není triviální, neboť $f_t(x)$ jsou stromy a navíc l může být libovolná (za podmínky spojité druhé derivace). Namísto toho můžeme funkci l aproximovat pomocí Taylorova polynomu 2. stupně (vlastně parabolou) a hledat minimum zjednodušené aproximované funkce. Pro vetší přehlednost si nyní definujme g_i a h_i takto:

$$g_i = \frac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}} \quad \text{a} \quad h_i = \frac{\partial^2 l(y_i, \hat{y}^{(t-1)})}{\partial (\hat{y}^{(t-1)})^2} \tag{19}$$

Funkce Obj po aproximaci l Taylorovým polynomem 2. stupně vypadá nyní takto:

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \tag{20}$$

Provádíme minimalizaci vůči $f_t(x)$, takže výraz můžeme zjednodušit vynecháním konstantního členu nezávisejícím na $f_t(x)$, navíc ještě dosadíme regularizaci z rovnice (16)

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2. \tag{21}$$

Definujme nyní množinu I_j všech indexů i trénovacích dat které náleží listu j jako

$$I_j = \left\{ i \mid q(x_i) = j \right\} \quad 1 \leq j \leq T. \tag{22}$$

Například pro x_1, x_3, x_4 patřící do prvního listu a pro x_2, x_5 do druhého, budou množiny I_1 a I_2 vypadat takto:

$$I_1 = \{1, 3, 4\} \quad \text{a} \quad I_2 = \{2, 5\}. \tag{23}$$

Díky tomu můžeme ve vztahu (21) vhodně přeskládat sumy a získáme tak

$$Obj^{(t)} \simeq \sum_{j=1}^{T} \left[\left(\sum\nolimits_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum\nolimits_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T, \tag{24}$$

což je T nezávislých kvadratických funkcí (sčítáme po listech). Tuto úpravu můžeme udělat, neboť data patřící do stejných listů mají vždy stejné predikce w_j. Pouze si připomeneme, že T je počet listů stromu, w_j jsou váhy listů, I_j množina indexů trénovacích dat listu j. Pro větší přehlednost ještě provedeme poslední substituci $G_j = \sum_{i \in I_j} g_i$ a $H_j = \sum_{i \in I_j} h_i$

$$Obj^{(t)} \simeq \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T. \tag{25}$$

Nyní již můžeme snadno nalézt optimální váhy jednotlivých listů jako:

$$\begin{aligned} \frac{\partial (G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2)}{\partial w_j} &= 0 \\ G_j + (H_j + \lambda) w_j &= 0 \\ w_j &= -\frac{G_j}{H_j + \lambda}. \end{aligned} \tag{26}$$

A po dosazení optimálních vah w_j do (25) dostáváme:

$$Obj^{(t)} \simeq -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T \tag{27}$$

což už je finální tvar vyjadřující minimální hodnotu cílové funkce v iteraci t, pro předem danou strukturu stromu.

Nyní stručné shrnutí toho, co jsme výše uvedenými úpravami získali. Pro strukturu stromu definovanou pomocí $q(x)$ jsme nyní schopni nalézt optimální volbu vah listů, tak aby cílová funkce byla minimální. Navíc známe její optimální hodnotu včetně zahrnuté regularizace.

Hledání optimální struktury stromu

V tuhle chvíli už jen zůstává otázkou, jak ideální strukturu stromu budovat. Ideálně bychom chtěli projít všechny možné struktury a vybrat tu nejlepší. Jenže všech možných struktur je nekonečně mnoho a není tedy možné projít všechna různá dělení. Námi známý způsob budování běžných stromů je hladová heuristika, využívající metod jako je například informační zisk nebo Gini index.

Zde si můžeme všimnout, že hodnota $\frac{G_j^2}{H_j + \lambda}$ nám vlastně říká, jak je daná struktura stromu dobrá. Při budování struktury stromu se tedy v každém listu můžeme dívat na to, zda pro nás dělení

na další uzly znamená přidanou hodnotu, respektive zda sníží hodnotu *Obj* funkce. V terminologii XGBoost lze toto kritérium nalézt jako

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L^2+G_R^2)^2}{H_L+H_R\lambda}\right] - \gamma, \tag{28}$$

kde $\frac{G_L^2}{H_L+\lambda}$ je skóre levého potomka, $\frac{G_R^2}{H_R+\lambda}$ je skóre pravého potomka, $\frac{(G_L^2+G_R^2)^2}{H_L+H_R\lambda}$ je skóre v případě když v uzlu neprovedeme dělení, γ je penalizace za případné přidání dalšího listu. Z rovnice pro *Gain* je též vidět, že výraz může být díky regularizaci γ záporný. Což znamená, že tento způsob dělení nesnižuje hodnotu funkce *Obj*.

Stejně jako v běžných stromech, zde tedy budování struktury probíhá hladovým způsobem. V každém listu hledáme optimální dělení jako:

1. Pro každý příznak, proběhne seřazení instancí dle hodnot příznaku (např. vzestupně dle věku).
2. Průchodem hodnot je nalezena nejlepší dělící hodnota pro konkrétní příznak.
3. Výběr nejlepšího řešení napříč všemi příznaky pro rozdělení aktuálního listu.

Toto dělení provádíme dokud splňujeme podmínku pro maximální hloubku stromu. Po dosažení maximální hloubky pak ještě proběhne prořezaní uzlů se zápornou hodnotou *Gain* (prunning).

3.1 Výhody XGBoost proti GBT

Obě metody bezpochyby následují klíčové koncepty gradient boostingu, tak jak byly původně popsány v originálním článku [5]. Implementačně se však velmi liší a jedním nejdůležitějších příspěvků XGBoostu je jeho škálovatelnost. Na běžném počítači je ve všech možných scénářích několikanásobně rychlejší. V případě paralelního distribuovaného systému je pak možné pracovat s mnohonásobně většími instancemi problémů (v řádech milionů záznamů). Toto zrychlení je dosaženo zejména pomocí struktur pro řídké uložení dat, out-of-core způsoby využívající vhodnou strukturu cache pamětí a paralelizaci při vytváření jednotlivých stromů (pro zajímavost: paralelizace vytváření rozhodovacího stromů je hezky ilustrována v [6]).

Mimo jiné je možné nalézt i některé algoritmické odlišnosti, například XGBoost v každém kroku při hledání minima funkce používá k její lokální aproximaci druhou derivaci. Navíc je v knihovně možné vybírat z několika předem definovaných ztrátových funkcí nebo dokonce definovat svou vlastní (podmínka je spojitá druhá derivace). Velkou výhodou XGBoost je regularizace obsažena přímo v minimalizované cílové funkci. Za zmínku stojí i pokročilejší metody regularizace jako je DART [7] (dropout regularizace pro stromy). Poslední z cenných výhod jsou zabudované rutiny pro práci s chybějícími daty.

3.2 Úskalí

XGBoost je stále bezesporu jedním z nejlepších a nejpoužívanějších ensemble modelů, který lze snadno využít prakticky pro jakýkoliv klasifikační a regresní problém. Je ale vhodné, znát dopředu jeho slabá místa.

Jednou z jeho hlavních nevýhod je, že nativně nepodporuje kategorická data. Tyto data je tedy nutno nejdříve převést pomocí nějakého předzpracování na čísla. Nejjednodušší způsob je aplikovat kódování 1 z N, ale existují samozřejmě i pokročilejší techniky. Problém nastává v případě, kdy jednotlivé atributy obsahují velké množství unikátních hodnot. V případě atributu s 1000 kategorickými hodnotami je pro jeho zakódování nutno vytvořit 1000 příznaků. Tím se znatelně zpomalí proces učení, zvýší se paměťová náročnost a následně zhorší jakákoliv další manipulace s daty. Nehledě na to, že kategorických atributů s unikátními volbami může být mnoho a tím může vzniknout klidně i statisíce nových příznaků. Za zmínku stojí, že novější knihovny, které se snaží XGBoostu konkurovat mají tento nedostatek už nativně vyřešen. Příkladem těchto knihoven je LightGBM [8] a CatBoost [9].

Dále je nutno brát v potaz:

- Úlohy, které nejsou vhodné pro rozhodovací stromy ani XGBoost nezachrání.

- K dosažení optimálních výsledků je potřeba ladit velké množství hyper-parametrů, což je velmi časově náročné.
- V porovnání s novějšími metodami LightGBM [8] a CatBoost [9] je učící proces v některých případech i několikanásobně pomalejší.
- Za XGBoost nestojí žádná firma, která by se aktivně podílela na vývoji.

Reference

[1] Chen, T.; Guestrin, C.: XGBoost: A Scalable Tree Boosting System. *CoRR*, ročník abs/1603.02754, 2016, 1603.02754. Dostupné z: http://arxiv.org/abs/1603.02754

[2] Breiman, L.: *Classification and regression trees*. Routledge, 1984.

[3] Freund, Y.; Schapire, R. E.: A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, ročník 55, č. 1, 1997: s. 119–139.

[4] Wikipedia contributors: Gradient descent — Wikipedia, The Free Encyclopedia. 2018, [Online; accessed 10-June-2018]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Gradient_descent&oldid=845070621

[5] Friedman, J. H.: Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 2001: s. 1189–1232.

[6] Fang, Z.: Parallel Gradient Boosting Decision Trees. [online], [cit. 2018-06-09]. Dostupné z: http://zhanpengfang.github.io/418home.html

[7] Rashmi, K. V.; Gilad-Bachrach, R.: DART: Dropouts meet Multiple Additive Regression Trees. *CoRR*, ročník abs/1505.01866, 2015, 1505.01866. Dostupné z: http://arxiv.org/abs/1505.01866

[8] Ke, G.; Meng, Q.; Finley, T.; aj.: LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems 30*, editace I. Guyon; U. V. Luxburg; S. Bengio; H. Wallach; R. Fergus; S. Vishwanathan; R. Garnett, Curran Associates, Inc., 2017, s. 3146–3154. Dostupné z: http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf

[9] Dorogush, A. V.; Gulin, A.; Gusev, G.; aj.: Fighting biases with dynamic boosting. *CoRR*, ročník abs/1706.09516, 2017, 1706.09516. Dostupné z: http://arxiv.org/abs/1706.09516