

Co bude v dnešní přednášce

- Ensemble metody – základní myšlenka
- Bagging – náhodné lesy
- Boosting – AdaBoost

35 Ensemble metody obecně

Ensemble metody: základní myšlenka

- Základní myšlenka spočívá v tom, že namísto jednoho modelu (např. rozhodovacího stromu) použijeme **více modelů** a jejich predikce nějakým způsobem zkombinujeme do finálního rozhodnutí.
- My si ukážeme dva nejobvyklejší způsoby: *bagging* (bootstrap aggregating) a *boosting*.
- Každý z těchto dvou přístupů si ilustrujeme na nejznámějších reprezentantech, v obou případech spočívajících ve skládání rozhodovacích stromů, které už známe.
- Tito dva reprezentanti jsou: *náhodný les* (angl. *Random Forest*) pro bagging a *AdaBoost* (Adaptive Boosting) pro boosting.

36 Bagging: náhodný les

Náhodný les pro klasifikaci: základní myšlenka

Pro jednoduchost předpokládejme, že máme binární klasifikační problém, tj. rozhodujeme jestli $Y = 0$ nebo $Y = 1$.

1. Ze vstupního trénovacího datasetu \mathcal{D} vytvoříme n datasetů $\mathcal{D}_1, \dots, \mathcal{D}_n$ obvykle stejně velkých jako \mathcal{D} (příp. mohou být i větší či menší) pomocí metody **bootstrap**, neboli pomocí výběru *s opakováním*.
2. Na každém datasetu \mathcal{D}_i naučíme rozhodovací strom tak, jak jsme si ukázali v přednášce o rozhodovacích stromech.
Typicky pro trénování tohoto stromu použijeme pouze omezenou náhodně vzatou podmnožinu příznaků. Strom může, ale nemusí být málo hluboký, klidně hloubky dva nebo tři (někdy se používá i hloubka jedna).
Tyto stromy označíme T_1, \dots, T_n a budeme je nazývat *podmodely* (angl. base learners).
3. Při predikci daný datový bod \mathbf{x} proženeme všemi stromy T_1, \dots, T_n a od každého z nich si uložíme predikci $\hat{Y}_1, \dots, \hat{Y}_n$.
4. Všechny tyto stromy T_1, \dots, T_n tvoří **náhodný les** a jeho finální rozhodnutí o hodnotě Y je dané většinovým rozhodnutím stromů, je-li např. v množině $\{\hat{Y}_1, \dots, \hat{Y}_n\}$ více jedniček než nul, je predikce náhodného lesa $\hat{Y} = 1$.

Bootstrap

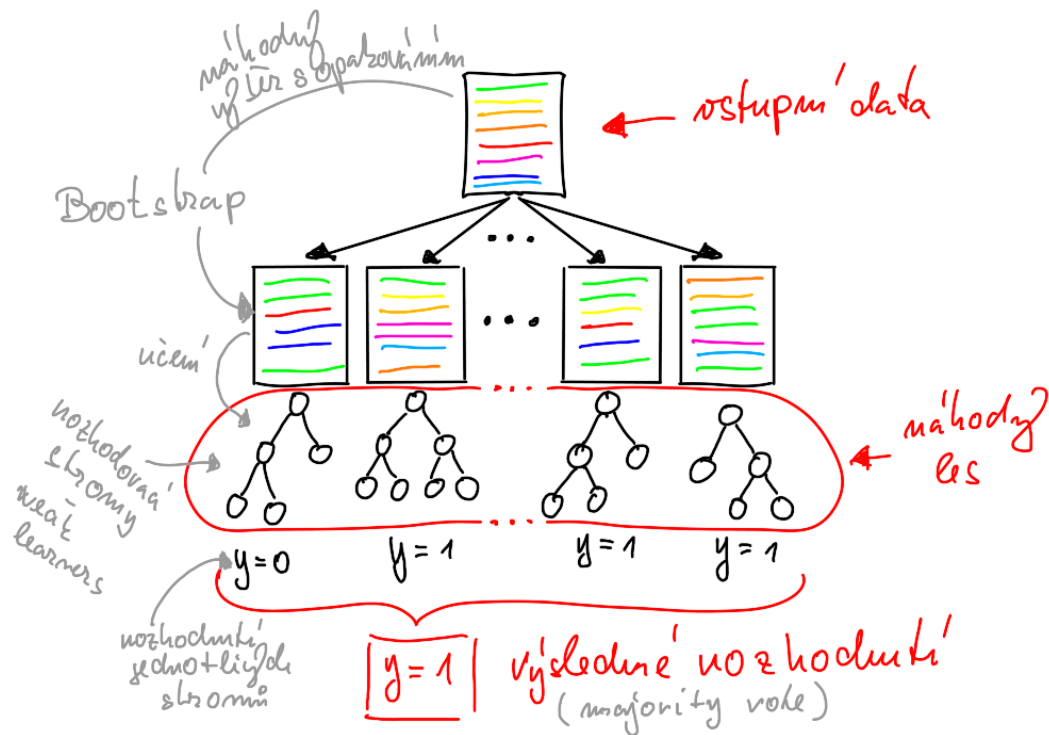
Ukážeme si, jak funguje *bootstrap* na jednoduchém příkladu a našem rýmičkovém datasetu:

id	rýmička	pohlaví	> 39°C	vstal(a)?
1	ano	muž	ne	ne
2	ne	žena	ano	ano
3	ne	muž	ne	ano
4	ano	žena	ano	ne

Chceme-li vytvořit „bootstrapem“ dataset velikosti pět, pětkrát si **náhodně vybereme řádek s tabulky** s tím, že se řádky v našem výběru **mohou opakovat**. Vybereme-li např. řádky s id 1,4,3,3,1, dostaneme dataset

id	rýmička	pohlaví	> 39°C	vstal(a)?
1	ano	muž	ne	ne
4	ano	žena	ano	ne
3	ne	muž	ne	ano
3	ne	muž	ne	ano
1	ano	muž	ne	ne

Náhodný les pro klasifikaci: základní myšlenka na obrázku



Predikce náhodného lesa

- V případě regresního problému (spojité Y) se postupuje opět analogicky jako u regresního rozhodovacího stromu: predikce náhodného lesa se bere jako průměr z predikcí jednotlivých stromů lesa:

$$\hat{Y} = \frac{1}{n} \sum_{i=1}^n \hat{Y}_i.$$

- Implementace `RandomForestClassifier` ve `scikit-learn` pracuje na místo predikcí tříd s predikcemi pravděpodobností tříd od jednotlivých podmodelů (relativní četnosti tříd v listech).
- Tj. např. u binární klasifikace, pokud označíme jako $\hat{p}_i = \hat{P}(Y = 1 | T_i, \mathbf{X} = \mathbf{x})$ predikci pravděpodobnosti příslušnosti bodu \mathbf{x} ke třídě 1 podmodelem T_i , pak finální predikovaná pravděpodobnost třídy 1 pro náhodný strom je určena průměrem:

$$\hat{p} = \frac{1}{n} \sum_{i=1}^n \hat{p}_i.$$

- Finální predikce vysvětlované proměnné Y je pak určena obvyklým porovnáním této pravděpodobnosti s číslem $1/2$ jako

$$\hat{Y} = \begin{cases} 1 & \text{pro } \hat{p} > 0.5, \\ 0 & \text{jinak.} \end{cases}$$

Základní hyperparametry

Základními hyperparametry metod `RandomForestClassifier` a `RandomForestRegressor` v `sklearn` jsou:

- `n_estimators`: určuje počet stromů v náhodném lese,

- `max_depth`: určuje maximální hloubku stromů v lese (typicky spíše nižší hodnota),
- `max_features`: počet (náhodně vybraných) příznaků, ze kterých si hladový algoritmus vybírá ten, podle kterého bude v aktuálním kroku data „větvit“. Defaultně se volí hodnota \sqrt{p} , tj. odmocnina počtu příznaků,
- je možné nastavovat i další obvyklé hyperparametry rozhodovacích stromů jakožto podmodelů.

Poznámky

- U baggingu, který skládá rozhodnutí z více podmodelů, je vhodné, aby jednotlivé podmodely nebyly stejné, ale naopak co **nejpestřejší**.
- Toho se jak docílí nějakým druhem **randomizace**; v případě náhodných lesů je tento náhodný prvek daný bootstrap metodou pro generování jednotlivých trénovacích datasetů a také hodnotou parametru `max_features`.
- Jak víme, **rozhodovací stromy** jsou velice citlivé na změny v trénovacích datech (**mají velký rozptyl**), a tak často stačí odlišnost datasetů daná bootstrapem, abychom získávali velice odlišné rozhodovací stromy.
- Tím, že **při baggingu** počítáme průměry predikcí těchto rozdílných podmodelů snažíme se **redukovat rozptyl** (a povětšinou se to úspěšně daří).
- Přestože mohou být jednotlivé stromy samy o sobě slabé modely (podmodelům v ensemble metodách se proto často říká *weak learners*), jejich kolektivní rozhodování dává až překvapivě dobré výsledky.
- Náhodné lesy jsou na rozdíl od rozhodovacích stromů velice robustní a poměrně odolné vůči přeučení. Bohužel ale částečně ztrácejí jejich jednoduchost a snadnou interpretovatelnost.

37 Boosting: AdaBoost

Rozhodovací stromy a `sample_weight` (1/2)

- Abychom mohli vysvětlit, jak funguje boosting a konkrétně algoritmus AdaBoost, potřebujeme pochopit použití vážených dat u rozhodovacích stromů.
- V řeči `sklearn` se jedná o použití hyperparametru `sample_weight` v metodě `.fit(X,y,sample_weight)` tříd `DecisionTreeClassifier` a `DecisionTreeRegressor`.
- Pokud máme v trénovací množině N datových bodů (řádků v tabulce, poloangl. „samplů“), je proměnná `sample_weight` pole w_1, \dots, w_N nezáporných vah jednotlivých bodů.
- Tyto váhy určují, jak moc je který bod „důležitý“. Defaultní hodnotou je $w_i = 1$ pro $i = 1, \dots, N$. V některých případech bývají znormované na jedničku, ale nemusí to platit.

Rozhodovací stromy a `sample_weight` (2/2)

- Při učení stromu se váhy projeví v kroku, ve kterém se počítá informační zisk

$$H(\mathcal{D}) - t_L H(\mathcal{D}_L) - t_R H(\mathcal{D}_R)$$

kde $t_L = \frac{\#\mathcal{D}_L}{\#\mathcal{D}}$ a $t_R = \frac{\#\mathcal{D}_R}{\#\mathcal{D}}$.

- Konkrétně se váženým způsobem napočítají jak entropie $H(\mathcal{D})$, $H(\mathcal{D}_L)$ a $H(\mathcal{D}_R)$, tak hodnoty t_L a t_R .
Např. pro výpočet entropie $H(\mathcal{D}_L)$ se v případě binární klasifikace použije odhad pravděpodobnosti třídy 1 určený součtem vah bodů ve třídě 1, které spadají do \mathcal{D}_L , poděleným sumou vah všech bodů v \mathcal{D}_L .
Dále např. hodnota t_L je rovna součtu vah bodů, které spadají do \mathcal{D}_L , poděleným sumou vah všech bodů z \mathcal{D} .
- Jsou-li tedy v \mathcal{D}_L body z řádků s indexy 3, 10, 15, 25 a odpovídajícími třídami 0, 1, 1, 1 a v \mathcal{D} body s indexy 3, 10, 15, 17, 21, 25, dostáváme:

$$p_1 = \frac{w_{10} + w_{15} + w_{25}}{w_3 + w_{10} + w_{15} + w_{25}} \quad \text{spolu s} \quad H(\mathcal{D}_L) = -p_1 \log p_1 - (1 - p_1) \log(1 - p_1)$$

a

$$t_L = \frac{w_3 + w_{10} + w_{15} + w_{25}}{w_3 + w_{10} + w_{15} + w_{17} + w_{21} + w_{25}}$$

Výsledkem použití vah je to, že strom se učí tak, aby správně predikoval zejména datové body s vyšší vahou. (Toto si rozmyslete!)

AdaBoost (Adaptive Boosting): základní myšlenka

- Stejně jako při baggingu **konstruujeme více podmodelů** (opět uvažujeme rozhodovací stromy, i když AdaBoost může používat i jiné typy modelů) a finální rozhodnutí je (váženou) kompozicí rozhodnutí jednotlivých modelů.
- Na rozdíl od baggingu ale při *boostingu* **nejsou tyto modely nezávislé**, ale jsou seřazené a každý další je ovlivněn těmi předchozími.
- Tento vliv je při AdaBoostu realizován pomocí vah datových bodů: Při konstrukci n -tého stromu je **zvýšena váha těm bodům, které předchozí $(n - 1)$ -tý strom klasifikoval špatně**.
- Takovýmito změnami vah je zajištěno, že se další model soustředí více na ty datové body, se kterými si předchozí modely neporadily.
- Přibližně takto funguje boosting obecně, my si dále ukážeme konkrétní implementaci této myšlenky známou jako algoritmus *AdaBoost* [Freund, Schapire (1997)].

AdaBoost (Adaptive Boosting): popis algoritmu (1/3)

- Na začátku máme dataset \mathcal{D} s N datovými body.
- Pro jednoduchost opět uvažujeme binární klasifikaci. Existují ale i modifikace algoritmu pro více než dvě třídy a i pro regresi.
- Počet zkonstruovaných stromů je zadán uživatelem v hyperparametru `n_estimators`.

AdaBoost:

1. Nastavme váhy rovnoměrně, tedy $w_i = \frac{1}{N}$ a položme $m = 1$.
2. Pokud $m \leq \text{n_estimators}$, naučme strom $T^{(m)}$ na datech \mathcal{D} s váhami w_i .
3. Do proměnné $e^{(m)}$ uložíme součet vah těch bodů z \mathcal{D} , které jsou špatně klasifikované stromem $T^{(m)}$.
4. Pokud je $e^{(m)} = 0$ skončíme, všechna data jsou klasifikována správně.

AdaBoost (Adaptive Boosting): popis algoritmu (2/3)

5. Položíme

$$\alpha^{(m)} = \text{learning_rate} \cdot \log \frac{1 - e^{(m)}}{e^{(m)}},$$

kde `learning_rate` je hyperparametr zadaný uživatelem; používá se ke zpomalení trénování a k zabránění přeučení (je-li menší než jedna).

6. Pro stromem $T^{(m)}$ špatně klasifikované body nastavme nové váhy¹

$$w_i \leftarrow w_i \exp(\alpha^{(m)}).$$

7. Znормalizujeme váhy tak, aby jejich součet byl jedna.
8. Zvětšíme m o jedna a vraťme se do bodu 2.

Výsledkem algoritmu je tedy až `n_estimators` rozhodovacích stromů $T^{(1)}, T^{(2)}, \dots$

¹Výraz $\exp(x)$ značí staré známé e^x . Často se to v literatuře zapisuje takto, tak si zvykejte ;).

AdaBoost (Adaptive Boosting): popis algoritmu (3/3)

Abychom určili rozhodnutí tohoto modelu pro nějaký datový bod \mathbf{x} , postupujeme následovně:

1. Každému stromu $T^{(m)}$ přiřadíme váhu danou číslem $\alpha^{(m)}$ z kroku 5.
2. Sečti váhy $\alpha^{(m)}$ všech stromů, které pro \mathbf{x} predikují $Y = 1$ a to samé udělej pro stromy predikující $Y = 0$.
3. Rozhodni se pro tu z možností, pro kterou je součet vah vyšší.

Poznámky

- Verze algoritmu pro „více než binární“ klasifikaci se nazývá *AdaBoost-SAMME* [Zhu, Rosset, Zou, Hastie (2006)]. Tato verze je implementována v `sklearn`.
- Varianta pro regresi se zase nazývá *AdaBoost.R2* [Drucker (1997)].
- AdaBoost nemusí nutně používat rozhodovací stromy; je možné použít jakýkoli model, který umí pracovat s parametrem `sample_weight`.
- V `sklearn` implementaci jsou rozhodovací stromy (hloubky 3) výchozí volba (parametr `base_estimator`).
- Parametr `learning_rate` je obvykle zaváděn u většiny ensemble metod spadajících do kategorie Boostingu.
- Jedná se o tzv. **regularizaci**: čím je `learning_rate` nižší, tím je model odolnější vůči přeučení. Nevýhodou je, že je pak obvykle nutné zvýšit počet stromů (parametr `n_estimators`).
- Při boostingu dochází k postupnému korigování chyb předchozích modelů, což vede na **redukcí vychýlení** (bias). Podmodely se typicky konstruují jako slabé modely (weak learners) - tj. např. jako poměrně mělké stromy.

ChangeLog

Verze	Datum	Autor	Log
-------	-------	-------	-----