0 Ensemble metody

Ensemble metody spočívají v konstrukci velkého množství jednoduchých modelů, jejichž predikce se zkombinují do finálního rozhodnutí.

0.1 Bagging (bootstrap aggregating)

Jednou z metod konstrukce slabých podmodelů (weak learners), je trénování nad náhodným výběrem trénovací množiny. Tím se zajistí rozdílnost podmodelů, jejich pestrost a variabilita. Tato metoda se nazývá bootstrap.

0.2 Náhodné lesy

Velmi silným a mocným reprezentantem bagging metody je náhodný les, který agreguje stromy. Tyto stromy jsou zpravidla malé hloubky. Datasety vzniklé bootstrapem způsobí vyšší pestrost stromů, které jsou sami o sobě velmi citlivé a u kterých se malá změna dat často projevuje velmi rozdílnými výsledky. Stromům s rostoucí hloubkou klesá bias, ale roste variance. Kombinací stromů model v jistém smyslu rozptyl krotí. Díky této stabilitě funguje náhodný les v praxi velmi dobře.

0.2.1 Pestrost stromů

Dalším velmi důležitým krokem v konstrukci náhodného lesa je náhodný výběr příznaků při každém dělení listu během konstrukce jednotlivých stromů. Algoritmus náhodně volí nějakou podmnožinu příznaků (o velikosti např. $\sqrt{\cdot}$, log) mezi kterými hledá ten nelepší split. Silné prediktory budou stále hodně zastoupené, nicméně v případě, že se ve výběru nevyskytnou, mají šanci výslednou konstrukci ovlivnit i slabších prediktory (které se mohou nyní vyskytnout i v kořeni nebo jinde ve vyšších hladinách).

0.2.2 Predikce

Náhodný les kombinuje výsledky stromů v případě klasifikace majoritním hlasováním a v případě regrese průměrem.

0.2.3 Shrnutí

Rozhodovací stromy jsou díky kolektivnímu rozhodování velmi robustní, díky průměrování odolné vůči přeučení (s rostoucím počtem průměrovaných hodnot klesá rozptyl a roste spolehlivost). Naopak ztrácí interpretovatelnost a trénují se znatelně déle (konstrukce stromů lze však provádět paralelně).

U náhodného lesa ladíme jako hyperparametry četnost podmodelů, maximální

hloubku jednotlivých stromů, počet/poměr náhodně vybraných příznaků při větvení.

0.3 Boosting

0.3.1 Vážené hodnoty u rozhodovacího stromu

Princip vážených dat u rozhodovacího stromu je následovný: Každému prvku $\boldsymbol{x}_i$ v trénovací množině přiřadíme nezápornou váhu $w(\boldsymbol{x}_i)$ tak, že

$$\sum_{i=1}^{N} w(\boldsymbol{x}_i) = 1$$

Při učení se tak význam vah u výpočtu informačního zisku trochu mění.

$$IG(\mathcal{D}) = H(\mathcal{D}) - t_0 H(\mathcal{D}_0) - t_1 H(\mathcal{D}_1),$$

kde $\mathcal{D}_0, \mathcal{D}_1$ jsou příslušné podmnožiny $\mathcal{D}$, jsou nyní koeficienty definované jako

$$t_0 = \frac{\sum_{\boldsymbol{x}\in\mathcal{D}_0} w(\boldsymbol{x})}{\sum_{\boldsymbol{x}\in\mathcal{D}} w(\boldsymbol{x})} \quad \text{a} \quad t_1 = \frac{\sum_{\boldsymbol{x}\in\mathcal{D}_1} w(\boldsymbol{x})}{\sum_{\boldsymbol{x}\in\mathcal{D}} w(\boldsymbol{x})}.$$

Takhle naučený strom klade větší důraz na to, aby správně predikoval datové body s vyšší váhou.

0.3.2 AdaBoost

AdaBoost podobně jako bagging konstruuje velké množství stromů, jejichž výsledky pro finální predikci kombinuje. Na rozdíl od náhodného lesa však spolu vybudované stromy souvisí. Stromy se konstruují sekvenčně, přičemž každý strom v posloupnosti se soustředí na ty datové body, ve kterých předchozí strom chyboval (zvýší jim váhu).

Konstrukce stromů V algoritmu se vyskytuje hodnota λ (learning rate), která v případě, že je menší než jedna trénování zpomaluje a zabraňuje přeučení. Pro $\lambda \geq 1$ je konstrukce významných stromů rychlejší, ale zároveň volatilní (zběsile řeší "aktuální" problém, kterým rychleji vznikají špatně klasifikované hodnoty, na které se nesoustředil).

Algorithm AdaBoost

Require:

Trénovací data $\mathcal{D}$.

Learning rate $\lambda \geq 0$.

1: Přiřaď všem datovým bodům váhu $w_i = \frac{1}{N}$.
2: Polož $m = 1$ (indexuje stromy v posloupnosti).
3: **while** $m \leq$ `n_estimators` **do**
4: Natrénuj strom $T^{(m)}$ s váhami w_i.
5: Spočítej součet vah $e^{(m)}$ špatně klasifikovaných hodnot v $T^{(m)}$.
6: Pokud je $e^{(m)} = 0$, skonči.
7: Polož

$$\alpha^{(m)} = \lambda \log \frac{1 - e^{(m)}}{e^{(m)}}.$$

8: Špatně klasifikovaným prvkům v $T^{(m)}$ přiřaď nové váhy

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

9: Váhy znormalizuj, aby součet byl 1, a inkrementuj m.
10: **end while**
11: **return** $T^{(1)}, T^{(2)}, \ldots, T^{(m)}$

O regularizaci Regularizaci většinou doprovází nějaký tradeoff. V případě AdaBoostu se jedná o pomalejší trénování (je potřeba víc stromů), nebo v případě hřebenové regrese způsobuje vychýlení odhadu $\hat{\boldsymbol{w}}$.

Predikce Model každému stromu $T^{(m)}$ přiřadí váhu $\alpha^{(m)}$ a při klasifikaci spočte W_1 součet vah stromů, které predikují 1, a W_0 součet vah stromů, které predikují 0. Ve finále predikuje tu hodnotu, která má mezi všemi stromy vyšší celkovou váhu.

Existuje i verze AdaBoostu pro multiclass klasifikaci: AdaBoost-SAMME, a pro regresi: AdaBoost.R2. AdaBoost nemusí nutně používat stromy. Umí používat jakýkoliv model, který umí správně pracovat s váženými datovými body.