Poglavje 4

Ansambli

Več ljudi več ve. Oziroma, najbrž obstaja dober razlog, zakaj je pri televizijskih kvizih "Lepo je biti milijonar"[1] eden od zasilnih izhodov omogočal igralcu, da za odgovor na vprašanje povpraša gledalce. Idejo so v zadnjem desetletju prejšnjega stoletja skušali dodobra izpiliti tudi na področju strojnega učenja. Zakaj bi na podatkih gradili en sam napovedni model, ko pa je mogoče graditi različne modela, ki bi na koncu za napoved lahko glasovali? Obstaja kar nekaj različic metod z ansambli modelov, a je morda glavna razlika med njimi ta, da modelom (metodam, s katerimi gradimo modele) zaupamo enako ali pa jih želimo prej preizkusiti na delu podatkov in potem pri glasovanju utežimo glede na njihovo uspešnost. Po tretji različici gradimo modele v seriji tako, da se novi modeli osredotočijo na težke primere, torej na primere, ki so jih prejšnji modeli iz serije napačno napovedali.

Ansambli so uporabni tako za regresijske kot klasifikacijske modele, izvedenke za en ali drug problem pa so praktično identične. Če pri klasifikaciji z več modeli lahko slednji glasujejo, bomo na primer pri regresiji končno napoved dobili z povprečjem napovedi posameznih modelov.

Spodaj si oglejmo nekaj tipičnih predstavnikov ansamblov. Nekatere, na primer *bagging*, tu omenjamo morda bolj zaradi zgodovinskih razlogov, saj se v praksi skoraj več ne uporabljajo. Po drugi strani pa so v istem času kot *bagging* nastajali tudi naključni gozdovi, tehnika, ki ji na manjših podatkih danes skoraj ni para [3].

4.1 Bagging

Ansambelski pristop *bagging* predlaga Leo Breiman [1], sicer eden od izumiteljev metode klasifikacijskih in regresijskih dreves. Opazil je, da so drevesa precej nestabilna metoda in da je njihov vsakokratni model precej odvisen od nabora primerov. Ideja bagginga (angl. *bootstrap aggregating*) je vzorčenje po metodi stremena (angl. *bagging*), gradnja dreves iz

[1] Kviz je v Sloveniji do leta 2005 vodil legendarni Jonas, sicer pa je bil posnet po licenci franšize *Who Wants to Be a Millionare*.

vzorca podatkov, in potem, pri napovedovanju, glasovanje tako nastalih modelov. Vzorčenje po metodi stremena je naključno vzorčenje s ponavljanjem. Recimo da imamo 100 primerov. Vzorčimo tako, da vsakič zberemo naključni primer in da množice, iz katere vzorčimo, ne spreminjamo. V vzorcu bodo nekateri primeri nastopali večkrat.

Algoritem 1 Učenje z pristopom bagging

```
Require: data, learner, k
Ensure: a list of models
  models ← []
  for i = 1 to k do
    sample ← bootstrap sample of data
    model ← learner(sample)
    models.append(model)
  end for
```

Postopek gradnje modelov s tehniko bagging je predstavljen kot algoritem 1, algoritem 2 pa kaže, kako s skupino tako zgrajenih modelov napovedujemo. Pri regresijskih modelih napovemo srednjo vrednost napovedi posameznih modelov, pri klasifikaciji pa ali modeli glasujejo ali pa, v primeru verjetnostnih napovedi, napovemo povprečno napovedano verjetnost za vsakega od razredov.

Algoritem 2 Napovedovanje s pristopom bagging

```
Require: models, data-instance
Ensure: prediction
  p ← 0
  for m in models do
    p ← p+ model(data-instance)
  end for
  p ← p/|models|
```

Bagging deluje samo v primeru manj stabilnih modelov, torej takih, kjer se modeli pri različnih vzorcih podatkov lahko med sabo precej razlikujejo. Primer takih modelov so klasifikacijska in regresijska drevesa. Tu je izbor, kateri atribut bo v posameznem vozlišču, pri velikem številu atributov lahko zelo odvisen od podatkov, še posebej, če je v podatkih večje število podobno informativnih atributov. Zato lahko tudi manjše spremembe v podatkih vodijo k popolnoma različnim drevesom. Po drugi strani pa so druge metode precej bolj stabilne in manj odvisne od vzorca podatkov. Take so na primer linearna in logistična regresija in pa metoda najbližjih sosedov. Bagging na takih metodah ne bo deloval dobro, saj bodo s to tehniko pridelani modeli med sabo precej podobni. Eden od načinov, kako lahko pridobimo med sabo različne modele tudi takrat, kadar uporabljamo stabilnejše metode učenja je vzorčenje atributov. Torej, namesto vzorčenja podatkov učimo modele na vseh podatkih, od katerih pa izberemo le del atributov. Pri podatkih z mnogo atributi tipično izberemo le manjši del, na primer 20% ali 30% atributov.

4.2 Naključni gozdovi

V želji, da bi pridobil čimbolj različne modele iz osnovne množice podatkov, je Brieman [2] drevesa dodatno ponaključil. Na vsakem nivoju drevesa algoritem gradnje izbere najbolj informativen atribut. Namesto tega je Brieman predlagal, da vozliščni atribut naključno izberemo med nekaj, recimo μ najbolje ocenjenih atributov, kjer je privzeta nastavitev za $\mu = \sqrt{m}$, oziroma koren števila primerov v učni množici. Ta trik, skupaj z vzorčenjem po metodi stremena vodi k različnim drevesom, ki jih zložimo v tako imenovani naključni gozd. Drevesa v gozdu pri napovedovanju enakovredno glasujejo (klasifikacija) oziroma njihove napovedi povprečimo (regresija). Tipično v gozdove vključimo nekaj 100 dreves. Skladno z Breimenovim priporočilom drevesa gradimo do konca in jih ne obrezujemo.

Nekoliko je presenetljivo, da je enostavna metoda, opisana zgoraj, ena od danes ključnih metod za gradnjo modelov z visokimi napovednimi točnosti. Oziroma drugače povedano: napovedne točnosti naključnih gozdov z ostalimi metodami težko prekosimo [3]. Lepota te metode je tudi v njeni neparametričnosti: edini parameter, ki ga pravzaprav nastavljamo, je število dreves v gozdu, pa še tu so privzeti parametri (npr. $k = 500$) tipično čisto zadostni.

4.3 AdaBoost

Prilagodljiv boosting (angl. *adaptive boosting*) [4] je nekoliko drugačen kot zgoraj opisani ansambli. Osnovna ideja pri tem pristopu je, da se modeli motijo na delu atributnega prostora, torej, na primerih, ki so težki za razvrščanje. Učenje je zato potrebno usmeriti k modeliranju teh primerov in je zato smiselno, da imajo ti primeri pri učenju večjo težo. Modele zato pri AdaBoost-u gradimo v serijo. Pričnemo z gradnjo modela, pogledamo, kje se ta moti, napačno razvrščenim primerom povečamo težo, se na tako uteženih primerih naučimo novega modela, spet pogledamo, kje se ta moti, ustrezno spremenimo uteži primerov in postopek ponavljamo (algoritem 3).

AdaBoost napoveduje s uteženo povprečno napovedjo. Imamo torek k modelov, za utež modela pa vzamemo $-\log \frac{e_k}{1-e_k}$.

AdaBoost je primeren tako za klasifikacijske kot regresijske probleme. Pravzaprav je bil to eden od prvih ansambelskih postopkov in je bil predlagan pred naključnimi gozdovi. V devetdesetih letih je njegova objava povzročila pravo revolucijo, strojno učenje pa se je takrat od optimizacije posameznih metod usmerilo v raziskavo ansamblov. Danes je AdaBoost historično pomembna metoda, ki pa je v praksi praktično ne uporabljamo več. Prehitele so je druge tehnike, kot so naključni gozdovi in, morda ter če že, metoda gradientnega boostinga.

Algoritem 3 Učenje z pristopom bagging

Require: data, learner, k
Ensure: models, errors $\boldsymbol{e}$
  models← []
  $w_d \rightarrow 1/m$ for $d \in$data
  for $i = 1$ **to** k **do**
    model ← learner(data, $\boldsymbol{w}$)
    models.append(model)
    $e_k \leftarrow$ error of model on weighted data set
    if $e == 0$ or $e \geq 0.5$ **then**
      break
    end if
    for each d **in** data **do**
      $w_d \leftarrow w_d \times \frac{e_k}{1-e_k}$
    end for
    normalize weights $\boldsymbol{w}$
  end for

4.4 Zlaganje

Zlaganje (angl. *stacking*) je še ena tehnika kombiniranja različnih modelov. Pri njej je, drugače kot pri zgornjih, izjemno pomembno, da so modeli zelo različni, še najbolje pridobljeni z različnimi tehnikami. Predpostavimo, da imamo na voljo k učnih algoritmov in da je naš problem regresijski. Na učnih podatkih bomo izvedli prečno preverjanje, ter si za vse primere v testni množici zapomnili napovedi vsakega od modelov. Ker je pri prečnem preverjanju vsak primer enkrat v testni množici, smo na ta način ob vsakem primeru pridelali vektor k napovedi. Sedaj uporabimo te, napovedne vektorje, in s čim bolj enostavno metodo (npr. z linearno regresijo) iz njih zgradimo model s, ki napoveduje razred. Na celotni učni množici zgradimo še model z vsakim od učnih algoritmov, da dobimo množico modelov F_i, $i = 1 \ldots k$. Pri napovedovanju primera x uporabimimo zgrajene modele F_i in napovedi združimo z krovnim modelom S. Napoved za primer x je torej $\hat{y} = S(F_1(x), F_2(x), \ldots, F_k(x))$.

Zlaganje da tipično boljše napovedi kot bi bile napovedi posameznih modelov, oziroma se kot metoda obnaša bolje kot posamezne učne metode. Trik je v krovnem modelu S, ki se nauči primerne kombinacije osnovnih modelov. Če je kakšen od osnovnih modelov na učnih podatkih slab, bo njegova utež majhna in njegovih napovedi krovni model ne bo upošteval.

V praksi lahko slabši modeli, oziroma slave tehnike napovedovanja zlaganju škodujejo. Zlaganje nam bo dalo tipično dobre rezultate, če bodo modeli primerljivi po napovedni točnosti in čimbolj raznovrstni.

4.5 Gradientni boosting

Predpostavimo, da smo na podatkih zgradili model F_Θ. S Θ označimo parametre naučenega modela, model pa bomo od tu dalje poenostavljeno označevali kar z F. Na primeru x nam ta model vrne napoved $\hat{y} = F(x)$, ki je približek prave vrednosti y. Pri napovedi bomo naredili napako. Predstavljamo si, da obstaja funkcija $h(x)$, s katero našo napoved popravimo tako, da v splošnem velja:

$$F(x) + h(x) = y$$

oziroma za vse primere iz učne množice velja:

$$\begin{aligned} F(x^{(1)}) + h(x^{(1)} &= y^{(1)} \\ F(x^{(2)}) + h(x^{(2)} &= y^{(2)} \\ &\dots \\ F(x^{(m)}) + h(x^{(m)} &= y^{(m)} \end{aligned} \tag{4.1}$$

V vektorski obliki lahko tako funkcijo zapišemo kot

$$h(\boldsymbol{X}) = \boldsymbol{y} - F(\boldsymbol{X})$$

Desno stran zgornje enačbe imenuje tudi vektor residualov, oziroma napak napovedi modela F. Funkcije h seveda ne znamo zgraditi, lahko pa jo pridobimo iz podatkov kot model, ki povezuje atributni zapis primerov z napakami modela F, torej iz tabele, katere vrstice so:

$$\begin{aligned} x^{(1)},\ & y^{(1)} - F(x^{(1)}) \\ x^{(2)},\ & y^{(2)} - F(x^{(2)}) \\ & \dots \\ x^{(m)},\ & y^{(m)} - F(x^{(m)}) \end{aligned}$$

Vloga funkcije $h(\boldsymbol{X})$ je torej ublažitev napak začnega modela $F(\boldsymbol{X})$. S tem popravkom je naša napoved torej $\hat{y} = F(\boldsymbol{X}) + h(\boldsymbol{X})$. Spomnimo se: najprej smo se na podatkih naučili modela $F(\boldsymbol{X})$, nato pa za napovedovanje napake, ki jih je model F naredil pri napovedovanju razreda učne množice, zgradili še model $h(\boldsymbol{X})$.

Ampak, kaj če tudi model $F(\boldsymbol{X}) + h(\boldsymbol{X})$ napoveduje z napakami? V realnem svetu tudi ta kombinacija prav gotovo ne bo imela idealnih napovedi. Postopek ponovimo! Tudi za model $F(\boldsymbol{X}) + h(\boldsymbol{X})$ izračunamo residualne vrednosti na učni množici, ter te modeliramo. Za poenostavitev notacije pričnimo spuščati oznako h in jo nadomestimo z modelom prvega reda. Torej, naš začetni model označimo z $F_0(x)$, model zgrajen na njegovih rezidualih pa z $F_1(x)$. Spomnimo, prej smo $F_1(x)$ označili z $h(x)$. Sedaj lahko tudi na rezidualih modela

$F_0(x) + F_1(x)$ zgradimo nov model iz primerov:

$$\begin{aligned} &x^{(1)},\ y^{(1)} - F_0(x^{(1)}) - F_1(x^{(1)}) \\ &x^{(2)},\ y^{(2)} - F_0(x^{(2)}) - F_1(x^{(2)}) \\ &\dots \\ &x^{(3)},\ y^{(3)} - F_0(x^{(3)}) - F_1(x^{(3)}) \end{aligned}$$

Postopek lahko ponavljamo še naprej, in se pri nivoju i učimo iz atributnega opisa primerov $\boldsymbol{X}$, rezidualov $\boldsymbol{r}_{i-1}$ nivoja $i-1$ in kompenzacijskega modela F_{i-1}:

$$\boldsymbol{X},\ \boldsymbol{r}_{i-1} - F_{i-1}(\boldsymbol{X}) \rightarrow F_i$$

Končna napoved bo vsota napovedi zgrajenih modelov:

$$\hat{\boldsymbol{y}} = \sum F_i(\boldsymbol{X})$$

V teoriji lahko zgoraj opisani postopek izboljša napovedi kakršnegakoli regresijskega modela. V praksi pa se uporablja predvsem v namene izboljšanja napovedovanja z regresijskimi drevesi. Tipično se napovedi regresijskih dreves z zgornjo tehniko močno izboljšajo. Parameter metode je globina gnezdenja funkcij oziroma število modelov, ki jih v postopku zgradimo.

Razmislimo, kam pravzaprav vodi zgoraj opisani postopek. Privzamemo, da želimo z našim aproksimacijskim modelom minimizirati kvadrat napake. Za primer i zapišimo:

$$L(y^{(i)}, F(x^{(i)})) = \frac{\left(y^{(i)} - F(x^{(i)}\right)^2}{2}$$

in ta zapis uporabimo v kriterijski funkciji,

$$J = \sum_{i=1}^{m} L(y^{(i)}, F(x^{(i)}))$$

Potem je odvod zgornje kriterijske funkcije v smeri vrednosti modela za primer i:

$$\begin{aligned} \frac{\partial J}{\partial F(x^{(i)})} &= \frac{\partial \sum_{i=1}^{m} L(y^{(i)}, F(x^{(i)}))}{\partial F(x^{(i)})} \\ &= \frac{\partial L(y^{(i)}, F(x^{(i)}))}{\partial F(x^{(i)})} \\ &= F(x^{(i)} - y^{(i)} \end{aligned}$$

Zadnje je ravno naša (negativna) napaka iz enačbe 4.1.

Postopek gradnje ansamblov z gradnjo modelov na napakah modelov je enakovreden gra-

dientnemu sestopu, kjer je kriterijska funkcija enaka vsoti kvadratov. Torej,

$$\begin{aligned} F(\boldsymbol{X} \leftarrow & F(\boldsymbol{X}) + h(\boldsymbol{X}) \\ \leftarrow & F(\boldsymbol{X}) + \boldsymbol{y} - F(\boldsymbol{X}) \\ \leftarrow & F(\boldsymbol{X}) - \frac{\partial J}{\partial F(\boldsymbol{X})} \end{aligned}$$

Zgornje primerjajmo s spodnjo enačbo, ki smo jo sicer uporabljali za gradientne sestope,

$$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \lambda \frac{\partial J(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}}$$

Z gradientnim boostingom [5] torej izboljšujemo model oziroma njegove parametre, a pri tem ne uporabljamo analitičnega zapisa odvodov po posameznih parametrih modela, čeprav je naš gradient temu enak. Za tehnike modeliranja kot so regresijska drevesa analitičnega gradienta sploh ne bi mogli zapisati, saj zapisa modela z njegovimi parametri ne poznamo. Gradientni boosting nam kljub temu omogoča optimizacijo takih modelov in s tem izboljšanje modela na učni množici.

Izboljšave in razširitve opisane tehnike gredo v smeri drugačnih in razširjenih zapisov kriterijske funkcije. Naštejmo nekaj različnih, začenši s kvadratom napake, ki ga že poznamo. Zaradi preglednosti smo spodaj opustili indeksiranje po primerih:

$$\begin{aligned} L &= \frac{(y-F)^2}{2} \\ L &= \frac{|y-F|}{1} \\ L &= \begin{cases} \frac{1}{2}(y=F)^2 & |y-F| \leq \delta \\ \delta|y-F| - \frac{\delta}{2} & |y-F| > \delta \end{cases} \end{aligned}$$

Prva podaja nam že znano kvadratno napako, ki pa je občutljiva na osamelce in te želi vključiti v model. Druga, morda v praksi boljša, je absolutna napaka. Za njo je gradient enak $\text{sign}(y^{(i)} - F(x^{(i)}))$. Tretja pa je Huberjeva napaka, kjer je gradient enak $y - F(x)$ pri prvem pogoju in $\delta(y^{(i)} - F(x^{(i)}))$ pri drugem. Konceptualno lahko te kriterijske funkcije primerjamo z različnimi tipi regularizacije, to je L1 in L2 regularizacijo in regularizacijo, ki kombinira tip L1 in L2.

Zgornji postopki so primerni za regresijske modele. Kako pa postopamo pri klasifikaciji? Podobno. Tu seveda uporabimo verjetnostne modele, katerih izhod je verjetnostna porazdelitev po razredih. Naj v izogib detaljem tu za konec omenimo le, da za kriterijsko funkcijo vzamemo Kullback-Leiblerjevo divergenco, njen gradient pa izračunamo kot

$$\frac{\partial L}{\partial F} = \sum_{i=1}^{m} \left[\ln \frac{F(x^{(i)})}{y^{(i)}} + 1 \right]$$

Literatura

[1] L. Breiman. Bagging Predictors. *Machine Learning*, 24(421):123–140, 1996.

[2] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[3] M. Fernández-Delgado, E. Carnadas, and S. Barro. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

[4] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[5] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29(5):1889–1232, 2001.