Odkrivanje znanj iz podatkov

(zapiski predavateljev, samo za interno uporabo)

Blaž Zupan

13. maj 2018

Kazalo

Uvod

Predmet Odkrivanje znanj iz podatkov na magistrskem študiju je nadaljevanje predmeta Uvod iz odkrivanja znanj iz podatkov iz dodiplomskega študija. Pred 2017 se je uvodni predmet imenoval Poslovna inteligenca. Pričujoči zapiski so zato nadaljevanje zapiskov iz uvodnega predmeta in s tem predpostavijo, da se je bralec z osnovnimi koncepti in terminologijo srečal že prej. Kjer temu ni tako, priporočam pregled uvodnih zapiskov.

Poglavje 1

Večrazredna klasifikacija

Eden od ključnih klasifikacijskih algoritmov, ki smo jih spoznali pri uvodnih predavanjih (Poslovna inteligenca oziroma, po novem, Uvod v odkrivanje znanj iz podatkov) je bila logistična regresija. Ta na podlagi vrednosti atributov ocenjuje verjetnost ciljnega razreda. Na primer, kateri uporabniki telefonskih storitev bodo v naslednjem mesecu dni prekinili naročniško razmerje? Ali pa, kdo od zaposlenih bo v naslednjem pol leta dal odpoved? Kateri artikli iz prodajnega kataloga bodo uporabniku, ki je opisan z vektorjem značilk, všeč? Bo popoldanski avtobus številka 10 danes prišel pravočasno do končne postaje? Ciljni razred smo tu opisali z besedami (na primer odpoved razmerja, ali pa pravočasnost avtobusa), v strojnem učenju pa zanj uvedemo razredno (odvisno) spremenljivko y, ki ima za i-ti primer lahko vrednost $y^{(i)} \in \{0, 1\}$, kjer je tipično 1 ciljna vrednost razreda (dogodek se je zgodil, na primer, zaposleni je dal odpoved) in z 0 vrednst razreda, kjer se ciljni dogodek ni zgodil (na primer, zaposleni ni dal odpovedi). Pri logistični regresiji, tako kot pri vseh drugih klasifikacijskih algoritmih, nas seveda za dani primer zanima verjetnost ciljnega razreda, torej $p(y = 1|x)$.

Zgoraj smo našteli nekaj primerov, kjer je klasifikacija binarna. Torej tam, kjer imamo slučajno spremenljivko (razred), ki lahko zavzame dve vrednosti. A je za praktične primere ta omejitev precej omejujoča. Na primer, iz končnih ocen v srednji šoli bi radi napovedali, na katero fakulteto se bo dijakinja vpisala. Iz ročno zapisane pismenke oziroma njene digitalizirane slike bi radi razbrali, za kateri znak (črko) gre. Čivku bi želeli določiti primerni emotikon. Napovedali bi radi vreme (sončno, oblačno, deževno). V vseh naštetih primerih lahko razredna spremenljivka zavzame eno od možnih vrednostih. Njena zaloga vrednosti torej ni več binarna.

Naj bo število možnih vrednosti, ki jo razredna spremenljivka zavzame, enako K. Vrednost ciljnega razreda bomo tu zapisali torej kar z indeksom vrednosti, torej $y \in \{0, 1, \ldots, K\}$. Ta zapis uporabimo zaradi enostavnejšega enačb in formalnih modelov. Pri sporazumevanju z uporabniki oziroma pri delu z uporabniško prijaznim programom pa seveda pričakujemo, da bodo vrednosti razredne spremenljivke zavzele neko opisno vrednost, oziroma da bomo lahko

poročali o modelih in njihovih napovedih v obliki, ki je blizu uporabniku[1].

Tu se pojavi problem. Logistična regresija je tehnika, ki lahko obravnava samo binarne probleme, torej samo probleme, kjer je razred dvovrednosten. Večvrednostnih problemov, torej problemov, kjer število vrednosti razredov presega dva, logistična regresija sama po sebi ne zna obravnavati. Omejitev na binarno klasifikacijo ni lastna samo logistični regresiji: podobno je tudi z drugimi popularnimi tehnikami, kot sta na primer metodi podpornih vektorjev in diskriminantna analiza. Po drugi strani pa smo že spoznali metode, ki so že v osnovi primerne za večvrednostno klasifikacijo. Med temi so metoda klasifikacijskih dreves, k-najbližjih sosedov, in klasifikacijski gozdovi.

V tem poglavju se bomo pravzaprav osredotočili na logistično regresijo in razmišljali o tem, kako to metodo prilagoditi v namene večvrednostne klasifikacije. In sicer na dva načina. Pri prvem bomo metodo samo pustili pri miru in poskušali večvrednostno klasifikacijo prevesti v binarni problem. Tehnike, ki jih bomo uporabili, bodo primerne za uporabo tudi za ostale binarne klasifikatorje. Pri drugem načinu pa bomo dejansko posegli v ogrodje logistične regresije in spremenili model tako, da bo primeren za ocenjevanje verjetnosti vrednosti večvrednostnega razreda.

1.1 Ovojnica za binarne klasifikatorje

Pri tem pristopu vzamemo binarni klasifikator, torej, metodo, ki se zna naučiti na primerih, ki pripadajo enemu od dveh razredov. Problem prilagodimo tako, da pripravimo podatke za binarno učenje, se naučimo niza binarnih klasifikatorjev in potem te združeno uporabimo za večvrednostno klasifikacijo. Dva najbolj znana, pa tudi morda najbolj enostavna pristopa sta:

Eden-proti-vsem. Za vsak razred $k \in \{1, 2, \ldots, \mathrm{K}\}$ konstruiraj klasifikator h_k, tako da je $y_k = h_k(\mathbf{X}, \mathbf{y}_k)$, kjer je vrednost vektorja razrednih vrednosti y_k za i-ti primer enaka:

$$y_k^{(i)} = \begin{cases} 1, & \text{if } y^{(i)} \equiv k \\ 0, & \text{otherwise} \end{cases}$$

Ko dobimo primer, za katerega moramo napovedati vrednost razreda in ki ga opišemo z vektorjem atributov x, uporabimo vse razvite modele h_k in napovemo vrednost razreda k za model, ki napove največjo verjetnost razreda 1. Torej:

$$\hat{y} = \arg\max_{k=1\ldots K} h_k(x)$$

Eden proti drugemu. Naučimo se $k(k-1)/2$ klasifikatorjev $h_{a,b}$ za vsak par (a, b) razredov. Pri

[1] Ta želja je sicer trivialna, a je mnoga, tudi popularna okolja za strojno učenje, zaobidejo. Okolje scikit-learn, na primer, zna delati samo s številskimi vrednostmi in je implementacija poročanja s simbolnimi vrednostmi prepuščena uporabniku te knjižnice oziroma razvijalcu.

napovedi uporabimo vse klasifikatorje in izberemo razred, ki je bil največkrat napovedan oziroma je največkrat zmagal.

Zgoraj smo opisali napovedovanja razredne vrednost, ne pa tudi postopke ocenjevanja njihovih verjetnosti. Bolj za silo in ne preveč formalno poglobljeno se da zgornja postopka prirediti tudi tako, da napovedujeta verjetnosti. In sicer, za postopek eden-proti-vsem tako, da za razredne verjetnosti vzamemo napovedane verjetnosti $h_k(x)$ ter jih normaliziramo, pri postopku eden-proti-drugemu pa ocenimo verjetnosti tako, da so te proporcionalne številu napovedi, kjer je določen razred zmagal. Ali pa so morda proporcionalne povprečni verjetnosti, s katero je bil ocenjen razred pri klasifikatorju, ki je ta razred vseboval. Kot rečeno, so vsi ti prijemi sicer inženirski in precej *ad hoc* in bi nam koristilo kaj boljšega, z nekaj več formalne podlage. O tem v nadaljevanju.

1.2 Nekaj razmišljanj o logistični regresiji

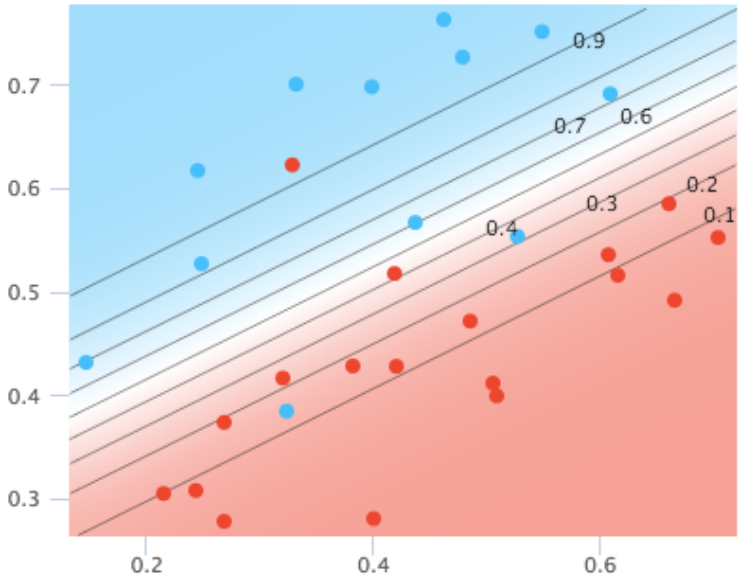
Še enkrat si poglejmo si logistično regresijo (za njeno izpeljavo glej zapiske pri predmetu Uvod v odkrivanje znanj iz podatkov), tokrat iz malce bolj inženirskega vidika. Logistična regresija v osnovi predpiše vsakemu primeru neko število iz intervala $(-\infty, +\infty)$. Vrednost atributov primera hranimo v vektorju x, za preslikavo iz atributnega prostora v prostor realnih števil pa bomo uporabili neko funkcijo, recimo $f(x)$. Število, ki nam ga vrne funkcija $f(x)$ logistična regresija potem pretvori v verjetnost ciljnega razreda, to je verjetno $p(y = 1|x)$. Verjetnosti so omejene na vrednosti med 0 in 1. Funkcija, ki je primerna za tako pretvorbo, je logistična:

$$g(z) = \frac{1}{1 + e^{-z}}$$

Logistični model oziroma verjetnost ciljnega razreda je torej $\hat{y} = g(f(x))$. Opazimo lahko, da je $g(-\infty) = 0$ in da je $g(+\infty) = 1$, ter da je $g(0) = 0.5$. Za primere v ciljnem razredu bi bilo torej dobro, da nam $f(x)$ vrača pozitivne, čim večje vrednosti, za primere v neciljnem razredu pa negativne, torej čim manjše vrednosti. Najbrž najbolj enostavna funkcija, ki izračuna $f(x)$ (spomnimo: x je vektor atributnih vrednosti) je linearna: $f(x) = \sum \theta_i x_i$. In prav to, linearno funkcijo, uporablja logistična regresija. Če bi bil vektor parametrov modela Θ enotski vektor, bi $f(x)$ vračala razdaljo primera do premice, ki jo določajo parametri Θ. Ker pa ne zahtevamo enotske dolžine tega vektorja na zahtevamo, lahko rečemo le, da je $f(x)$ premosorazmerna razdalji do premice. Kar je popolnoma v redu, saj enot, v katerih so zapisani atributi ne poznamo, in nam dejanska razdalja v izbranih enotah ne pove ničesar.

Oglejmo si to na primeru. Slika 1.1 prikazuje primere v dvo-atributnem prostoru in ločnico ($f(x)$) med njimi, ki smo jo dobili z logistično regresijo. Bolj, kot so točke oddaljene od ločnice, večja je verjetnost, da pripadajo enemu ali drugemu razredu. Idealno bi bilo, da je linearna ločnica taka, da loči vse točke enega razreda od točk drugega razreda. V našem primeru

taka ločnica ne obstaja, zato logistična regresija nekatere primere iz učne množice razvrsti napačno.



Slika 1.1: Podatki v prostoru dveh atributov in ločitvena meja med razredi, ki jo uporablja model logistične regresije.

1.3 Večrazredna logistična regresija

Postopek, ki ga bomo tu opisali, se imenuje tudi regresija softmax (angl. *softmax regression*)[2]. Nazivi so sicer rahlo zavajajoči, saj gre za klasifikacijsko in ne regresijsko metodo, a po drugi strani čisto informativni, saj v jedru uporabljajo linearno kombinacijo atributov. Za logistično regresijo smo zgoraj omenili, da pretvori atributni vektor x v neko število $f(x)$, tega pa potem v verjetnost ciljnega razreda. Pri večrazredni, tudi imenovani multinomski klasifikaciji, bi lahko izhajali iz predpostavke, da smo za nek atributni vektor za vsakega od razredov izračunali neko število. Recimo, napovedujemo vreme: jasno, oblačno ali deževno. Predstavljajmo si, da imamo nek predpis, ki nam iz dnevne karte Slovenije (vektor atributov) izračuna vektor števil za naš tri razrede. Naj bo ta vektor $z = [2.5, 3.7, 1.9]^T$. Naj nas tu zaenkrat ne bega, kako smo do tega vektorja prišli, a vsekakor opazimo, da to niso verjetnosti. Verjetnosti pa so ravno tiste, ki bi jih od našega napovednega modela želeli. Potrebujemo predpis, funkcijo, ki naš vektor pretvori v verjetnosti, torej predpis, ki stori nekaj z vektorjem z tako, da iz njena izračuna vektor istih dimenzij, kjer so vrednosti verjetnosti razredov, ki se seveda seštejejo v 1. Tak predpis je funkcija softmax:

[2]Pri izpeljavi regresije softmax smo se zgledovali po spletni strani `http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/`.

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Softmax nam torej naš vektor $z = [2.5, 3.7, 1.9]^T$ pretvori v vektor $\sigma(z) = [0.21, 0.68, 0.11]$. Funkcija sama poskrbi, da je vektor enotni, saj elemente vektorja (e^{z_j}) v funkcijskem predpisu deli z njihovo vsoto, torej jih normira.

V redu. A kako dobimo elemente vektorja z? Tako kot pri logistični regresiji, naj bo tudi ta izračunan z linearno kombinacijo atributnih vrednosti. Za j-ti razred in vektor atributnih vrednosti x zapišemo torej:

$$z_j(x) = \theta_{j,0} + \theta_{j,1}x_1 + \ldots + +\theta_{j,n}x_n$$

Z n smo tu označili število atributov. Za bolj kompakten zapis se dogovorimo, kot smo to storili pri logistični regresiji, da uvedemo še $x_0 = 1$. Vektorski zapis zgornje linearne kombinacije je:

$$z_j(x) = \Theta_j^T x$$

Naš mutlinomiski logistični model bo torej za primer opisan z vektorjem atributnih vrednosti x vračal vektor verjetnosti, kjer bo posamezen element ustrezal verjetnosti posameznemu razredu:

$$h_\Theta(x) = \begin{bmatrix} p(y=1|x;\Theta_1) \\ p(y=2|x;\Theta_1) \\ \vdots \\ p(y=k|x;\Theta_k) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\Theta_j^T x}} \begin{bmatrix} e^{\Theta_1^T x} \\ e^{\Theta_2^T x} \\ \vdots \\ e^{\Theta_k^T x} \end{bmatrix}$$

Za razred j in primer x je torej verjetnost razreda:

$$p(y=j|x;\Theta_j) = \frac{e^{\Theta_j^T x}}{\sum_{k=1}^{K} e^{\Theta_k^T x}}$$

Pozor: Θ_1, Θ_2 do Θ_k so vektorji in parametri modela, posamezen vektor parametrov za posamezen razred. Da bo vse skupaj bolj kompaktno, te parametre lahko združimo v matriko Θ:

$$\Theta = \begin{bmatrix} \text{——} \ \Theta_1^T \ \text{——} \\ \text{——} \ \Theta_2^T \ \text{——} \\ \vdots \\ \text{——} \ \Theta_k^T \ \text{——} \end{bmatrix}$$

Zgoraj smo določili strukturo modela. Model sam pa bo odvisen od vrednosti parametrov

modela, torej vrednosti elementov matrike Θ. Te vrednosti moramo določiti tako, da se bo model čim bolje prilegal učnim podatkom, torej podatkom zbranih v matriki atributov X in vektorju razredov y. Prileganje moramo opisati kvantitativno, torej določiti cenovno funkcijo $J(\Theta)$. Do te pridemo preko verjetja. Torej, kakšna je verjetnost, da bomo z našim modelom, ki ga določajo parametri Θ, napovedali pravo vrednost razredov:

$$\begin{aligned} L(\Theta) &= p(y|X;\Theta) \\ &= \prod_{i}^{m} p(y^{(i)}|x^{(i)};\Theta) \end{aligned} \tag{1.1}$$

Verjetje želimo maksimizirati, torej izbrati take parametre Θ, kjer je verjetje največje. Isto enačbo za verjetje smo že videli (poglej poglavje o logistični regresiji). Tudi takrat smo potožili, da je s produkti delati težko in da so vsote bolje. Nič se ne bo spremenilo, če namesto verjetja uporabimo logaritem verjetja:

$$\begin{aligned} l(\Theta) &= \log L(\Theta) \\ &= \sum_{i}^{m} p(y^{(i)}|x^{(i)};\Theta) \end{aligned} \tag{1.2}$$

Knjižnice z optimizacijskimi funkcijami tipično iščejo minimume. Naša cenovna funkcija bo zato:

$$\begin{aligned} J(\Theta) &= -l(\Theta) \\ &= -\log \sum_{i}^{m} p(y^{(i)}|x^{(i)};\Theta) \\ &= \sum_{i=1}^{m} \log \frac{\exp(\theta_{y^i}^T x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})} \\ &= \sum_{i=1}^{m} \sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})} \end{aligned} \tag{1.3}$$

Zgornja notacija z $\theta_{y^i}^T$ je malce čudna, a pomeni, da izberemo tisti θ vektor, ki ustreza razredu y^i. To je z vsoto in indikatorsko funkcijo morda še bolj čudno zapisano v naslednji vrstici. Indikatorska funkcija $1\left\{y^{(i)} = k\right\}$ vrne vrednost 1, če je izraz v oklepajo resničen, sicer pa vrednost 0. A je od vseh zapisov prav ta v zadnji vrstici najbolj uporaben. Verjetnosti v imenovalcu moramo tako ali tako izračunati. Shranimo jih v matriki. Potem iz matrike za števec izberemo samo tiste vrednosti, ki dejansko ustrezajo razredu i-tega primera.

Kriterijsko funkcijo imamo. Za izračun parametrov modela bomo uporabili gradientni sestop, oziroma katerokoli (boljšo) optimizacijsko metodo, ki pa potrebuje izračun gradienta. Ta je za k-ti vektor parametrov, torej za parametre, ki ustrezajo k-temu razredu, enak:

$$\nabla_{\Theta_k} J(\Theta) = -\sum_{i=1}^{m} \left[x^{(i)} \left(1\{y^{(i)} = k\} - p(y^{(i)} = k | x^{(i)}; \Theta) \right) \right] \tag{1.4}$$

Poiskati moramo vrednosti parametrov, ki zgornji izraz minimizirajo. Izraz v notranjem oklepaju gre za primere razreda k proti vrednosti 0, če je le ocenjena verjetnost za ta razred visoka (1 minus nekaj, kar gre proti 1, gre skupaj proti 0). Za primere, ki niso v razredu k, pa pričakujemo, da je verjetnost tega razreda čim manjša.

Od tu dalje je enostavno. Lahko uberemo metodo gradientnega sestopa in neko začetno matriko Θ (morda tako, kjer so vse vrednosti enake 0) osvežujemo:

$$\Theta_k \leftarrow \Theta_k - \alpha \nabla_{\Theta_k} J(\Theta)$$

ali pa (bolje) uporabilo postopek kot je L-BFGS iz za to dostopne knjižnice.

1.4 Predoločenost

Model, ki smo ga uvedli zgoraj, je predoločen. Ima preveč parametrov. Rešitev za Θ je poljubno mnogo. Zakaj? Vzemimo, da vse vektorjem Θ_k odštejemo nek poljuben vektor Ψ:

$$\begin{aligned} p(y = k | x; \theta) &= \frac{\exp((\theta_k - \psi)^T x)}{\sum_{j=1}^{K} \exp((\theta_j - \psi)^T x)} \\ &= \frac{\exp(\theta_k^T x) \exp(-\psi^T x)}{\sum_{j=1}^{K} \exp(\theta_j^T x) \exp(-\psi^T x)} \\ &= \frac{\exp(\theta_k^T x)}{\sum_{j=1}^{K} \exp(\theta_j^T x)}. \end{aligned} \tag{1.5}$$

Hm. Torej odštevanje vektorja Ψ ne vpliva prav nič in se iz vseh enačb pokrajša. Torej, lahko vstavimo, da je $\Psi = \Theta_1$ in je potem $\Theta_1 - \Psi = 0$. Torej, prva vrstica v matriki Θ ima vse vrednosti nastavljene na nič. In je pri optimizaciji sploh ne rabimo računati. Računamo potem vse ostale vrednosti. Namesto začetnih $k \times (n + 1)$ parametrov imamo tako sistem z $(k - 1) \times (n + 1)$ parametrov. In eno samo rešitev.

1.5 Večrazredna in dvorazredna logistična regresija

Premislimo, kaj je z večrazredno logistično regresijo v posebnem primeru, ko je $K = 2$. Naš vektor verjetnosti razredov bo:

$$h_\Theta(x) = \frac{1}{\exp(\Theta_1^T x) + \exp(\Theta_2^T x)} \begin{bmatrix} \exp(\Theta_1^T x) \\ \exp(\Theta_2^T x) \end{bmatrix} \tag{1.6}$$

Ta sistem, kot vemo iz prejšnjega razdelka, je predoločen. Izberemo vektor $\Psi = \Theta_1$ in ga odštejemo od Θ_1 in Θ_2:

$$\begin{aligned} h(x) &= \frac{1}{\exp((\theta_1 - \theta_2)^T x^{(i)}) + \exp(\vec{0}^T x)} \left[\exp((\theta_1 - \theta_2)^T x) \exp(\vec{0}^T x) \right] \\ &= \begin{bmatrix} \frac{1}{1+\exp((\theta_1-\theta_2)^T x)} \\ \frac{\exp((\theta_1-\theta_2)^T x)}{1+\exp((\theta_1-\theta_2)^T x)} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{1+\exp((\theta_1-\theta_2)^T x)} \\ 1 - \frac{1}{1+\exp((\theta_1-\theta_2)^T x)} \end{bmatrix} \end{aligned} \tag{1.7}$$

V zgornji enačbi nadomestimo $\theta_1 - \theta_2$ z θ in dobimo identični model kot pri logistični regresiji. Regresija softmax pri $K = 2$ je identična logistični regresiji.

1.6 Regularizacija

Tako, kot se logistična regresija lahko preveč prilagodi učnim podatkom, to velja tudi za mutlinomsko regresijo. Preveliko prilaganje pomeni visoke vrednosti v matriki Θ (to vemo že iz linearne regresije, pa tudi iz logistične; oboje iz lekcije o polinomski razširitvi učnih primerov). Zdravilo te anomalije je regularizacija. Cenovna funkcija z regularizacijo je:

$$\nabla_{\Theta_k} J(\Theta) = -\sum_{i=1}^{m} \left[x^{(i)} \left(1\{y^{(i)} = k\} - p(y^{(i)} = k | x^{(i)}; \theta) \right) \right] + \frac{\lambda}{2} \sum_{k=1}^{K} \sum_{j=1}^{n} \Theta_{kj}^2 \tag{1.8}$$

Gradient se nam skladno z zgornjo spremembo poveča za $\lambda\Theta_k$.

Poglavje 2

Ocenjevanje in izbor atributov

Ocenjevanje in izbor atributov sta osnovni opravili vsakega podatkovnega rudarja. Namreč, odkrivanje znanj iz podatkov temelji na tem, da ugotovimo, kateri atributi pomembni in prevladujoče vplivajo na obnašanje modela, kakršenkoli naj slednji bo. V tem poglavju se bomo sicer s primerom omejili na klasifikacijske probleme, a je koncepte, ki jih bomo spoznali, prav enostavno razširiti na kakršenkoli drugo nalogo.

Ocenjevanje informativnosti atributov je predvsem pomembno takrat, ko želimo podatke in iz teh porajajoče se modele razumeti. Na primer, želimo vedeti, kateri od atributov, s katerim opišemo naročnika mobilnih storitev, je tisti, ki je najbolj povezan z prekinitvijo naročniškega razmerja. Ali pa, v medicini, kateri od simptomov je najbolj povezan z diagnozo, ali pa prognozo. Morda celo, katere so tiste ključne besede v čivkih, ki so povezane s tečajem določene valute.

Z ocenjevanjem informativnosti atributov je povezan tudi njihov izbor. V splošnem bi iz začetne množice atributov, s katerimi smo popisali primere, radi izbrali množico atributov, ki je po nekem kriteriju najboljša. Recimo tako, na podlage katere nam izbrana metoda strojnega učenja zgradi model z visoko napovedno točnostjo. Na podlagi stopnje povezanosti oziroma vključenosti ocenjevanja v izgradnjo modelov tudi ločimo tri različne načine izbora atributov:

1. **Filter**. Atributom pripišemo stopnjo informativnosti, ki jo izračunamo neposredno iz podatkov. Pri tem lahko upoštevamo samo vrednosti atributa in razreda (univariatni pristop) ali pa kontekst, torej tudi vrednosti ostalih atributov (multivariatni pristop). Predvsem univariatne metode so lahko izjemno časovno učinkovite, a je tu težava pri določanju praga, koliko atributov izbrati. Kratkovidnost univariatnih metod rešujemo z multivariatnimi pristopi (npr. metoda Relief), ki pa so časovno potratnejše. Tudi pri njih ostaja problem, koliko atributov izbrati.

2. **Ovojnica**. Atribute izberemo tako, da optimiziramo napovedno točnost modelov. Tipično zberemo neko začetno množico atributov, preverimo napovedno točnost modela, ki ga zgradimo na tej množici, in tej množici dodajamo ali odvzemamo atribute s ciljem iz-

boljšanja točnosti. Metode te vrste so tipično časovno potratne.

3. **Metoda učenja.** Veliko metod strojnega učenja z uteževanjem ali kar neposredno z uporabo samo nekaterih atributov na ta način, torej znotraj metode, vrši izbor značilk. Primer takih tehnik so regresijska in klasifikacijsko drevesa, naključni gozdovi ter linearna in logistična regresija. Pri slednjih je še posebej zanimiva kombinacija z regularizacijo L1.

Obstaja vrsta odličnih pregledov različnih vrst in tipov zgoraj omenjenih metod. Cilj tega poglavja je samo osnovni pregled, morda še ožje, relacija med izborom značilk in nevarnostjo, da se z njo preveč prilagodimo podatkom. A pojdimo po vrsti in pričnimo s primerom.

2.1 Primer podatkov

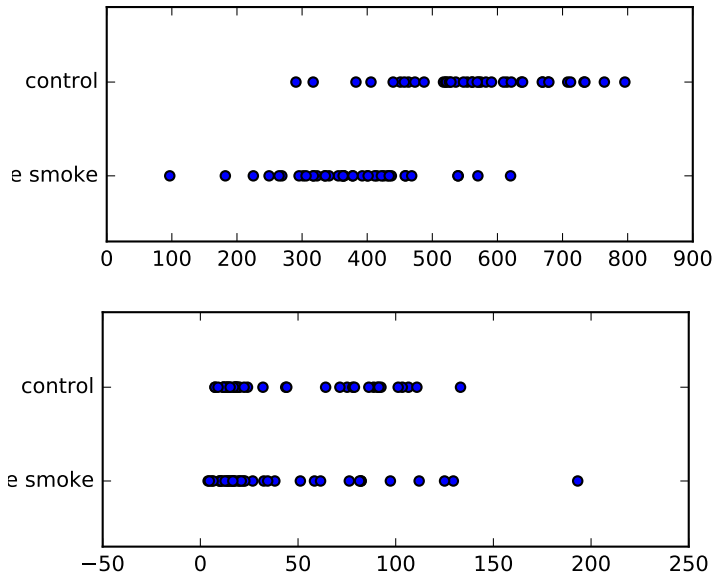
Za primer si poglejmo podatke, iz katerih lahko morda ugotovimo, ali ima kajenje sploh kakšne posledice. Recimo, na [limfocite](https://sl.wikipedia.org/wiki/Limfocit), to je na vrsto belih krvnih telesc. Še najbolj sistematično lahko posledice kajenja opazujemo tako, da pogledamo, kaj se dogaja v celicah limfocitov. Biologi danes dogajanja v celicah opazujejo tudi preko izražanja vseh genov v celici (človekov genom jih ima okoli 20.000). To je, z meritvami, koliko so geni aktivni pri proizvajanju proteinov. Še bolj direktno bi stanje v celici lahko ocenili s koncentracijo vseh proteinov, a je ta tehnologija dražja in manj dostopna. Tu nas je morda malce zaneslo pri domeni vsled piščeve navdušenosti nad biotehnolgoijo, ampak na koncu je enostavno: imamo primere (79 žensk, med njimi 40 kadilk in 39 nekadilk) in meritve izražanj 14.093 genov v njihovih limfocitih (atributi). Podatki so prosto dostopni v zbirki GEO Data Sets [1]. Naš cilj je ugotoviti, koliko atributov nam pove kaj o razredu. Oziroma, po biološko, koliko je takih genov, katerih izražanje se spremeni pri kadilkah.

2.2 Ocenjevanje informativnosti

Pod informativnost tu razumemo povezanost med vrednostjo atributa in razredom. Z drugimi besedami, v kakšni meri lahko iz vrednosti atributa napovemo razred. Kot kaže slika 2.1, so lahko atributi bolj ali manj povezani z razredom.

Pri zveznih atributih in diskretnem razredu za informativen atribut pričakujemo, da bo ta zavzel podobne vrednosti pri primerih istega razreda, ki bodo drugačne od vrednosti atributov za primere ostalih razredov. Za binarno klasifikacijo nam to razliko dobro kvantificira Studentovo t-statistika za ocenjevanje razlik med dvema skupinama meritev X_0 in X_1, kjer so v matriki X_0 zapisane atributne vrednosti primerov enega razreda ($y = 0$) in v matriki X_1 atributne vrednosti primerov drugega razreda ($y = 1$). Želimo torej, da bi bila povprečna vrednost med tema skupinama čim bolj drugačna, torej absolutna razlika med njima čim večja,

[1] https://www.ncbi.nlm.nih.gov/sites/GDSbrowser?acc=GDS3713



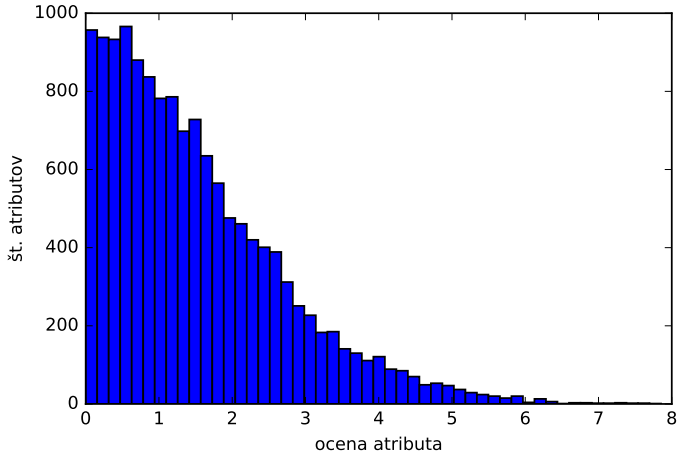
Slika 2.1: Porazdelitev vrednosti informativnega (zgoraj) in neinformativnega (spodaj) atributa pri primerih v prvem (“control”) in drugem razredu (“cigarette smoke”).

ter da bi razsipanje vrednosti znotraj skupin bilo čim manjše. Pri slednjem bi bilo dobro upoštevati tudi število primerov v posamezni skupine. Vse skupaj strnemo v enačbo za že omenjeno t-statistiko:

$$t = \frac{\overline{X_0} - \overline{X_1}}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}}$$

kjer sta s_0 in s_1 standardna odklona vzorcev X_0 in X_1. Zanima nas povprečna vrednost t-statistike, torej $|t|$. Za naš primer kadilk in nekadilk je porazdelitev vrednosti te statistke, torej vrednosti ocen informativnosti atributov takšna, kot jo prikazuje slika 2.2.

Tu je na mestu opozorilo. Zgoraj smo uvedli eno od metod za univariatno ocenjevanje informativnosti atributov. Ta je priročna samo v primeru, ko je atribut zvezen, razred pa binaren. V primeru večvrednostnega razreda bi bilo potrebno uporabiti statistiko ANOVA. Če bi bili atributi diskretni, bi lahko uporabili informacijski prispevek, ali pa χ^2. Univariantnih za različne kombinacije tipov atributov in razredov je veliko in bi bilo naštevanje na tem mestu neprimerno enciklopedično. Raje nadaljujmo z vprašanjem, kaj nam sploh ocene atributov povedo oziroma kateri atributi so potem sploh povezani z razredom. Ali bolje, kje, na sliki 2.2 je meja, od katere naprej bi lahko trdili, da je atribut informativen.



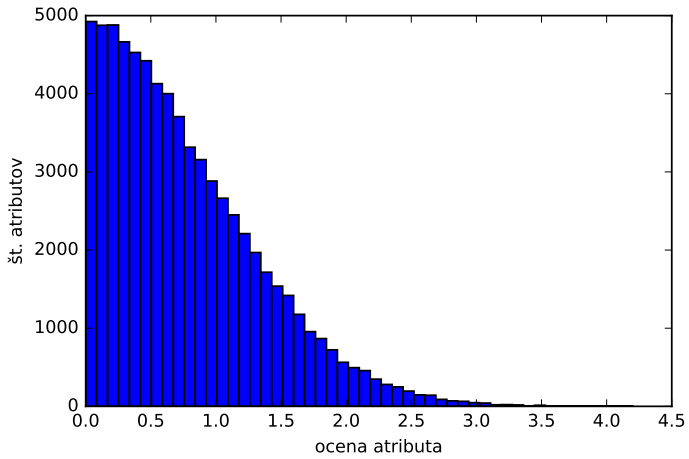
Slika 2.2: Porazdelitev ocen informativnosti atributov.

2.3 Permutacijski test

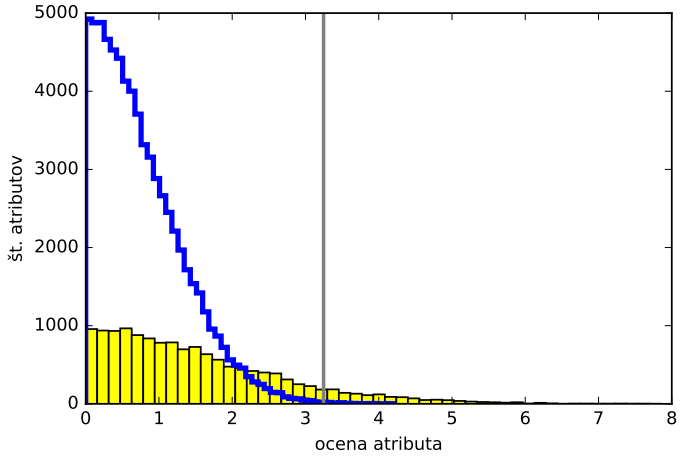
Naši podatki vsebujejo veliko število atributov in relativno majhno število primerov. Prav lahko bi se zgodilo, da je kakšen od teh atributov informativen popolnoma naključno, torej, da so meritve bile slučajno take, da so ravno dovolj ločile izbrani dve skupini oziroma razreda. Kakšna bi bila potem porazdelitev atributnih ocen, torej porazdelitev, če bi bile meritve naključne meritvah? Ali pa, če bi laborant pomešal oznake vzorcev in bi bili razredi primerov napačni. Do naključnih meritev lahko pridemo tako, da vrednosti za dani atribut premešamo, še lažje pa to storimo za vse atribute naenkrat tako, da premešamo vrednosti razredne spremenljivke. Po tej permutaciji ocenimo informativnost atributov. Da vse skupaj ne bo odvisno od ene same permutacije razredov, lahko postopek nekajkrat ponovimo. Za naše podatke in oceno s t-statistiko dobimo porazdelitev ocen, kot nam jo prikazuje graf na sliki 2.3.

Porazdelitev ocen informativnosti atributov, ki jih izračunamo nad ponaključenimi podatki, je dosti ožja od te pridobljene iz originalnih, "nepokvarjenih" podatkov. Slika 2.4 nam prikazuje obe distribuciji hkrati. Na njej smo z navpično nakazali na mejo, od katere dalje je na permutiranih podatkih zelo majhno število ocen oziroma je delež takih meritev samo $p = 0.001$. Za vse atribute, ki so bili ocenjeni desno od te meje, lahko trdimo, da je vrednost njihove ocene taka, da bi jo težko dobili na naključnih podatkih. Verjetnost, da bi se to lahko zgodilo, je enaka vrednosti p, na podlagi katere smo mejo določili. Če bi radi to verjetnost še zmanjšali, ustrezno zmanjšamo vrednost p. Koliko je sploh takih atributov? Za našo izbrano vrednost meje oziroma napake p je takih 1327 atributov, se pravi, nekako desetina vseh genov, ki so bili vključeni v meritve. Pri veliko strožji meji, na primer $p = 0.0001$, se število takih atributov zmanjša na 734.

V podatkih je torej samo približno desetina atributov, za katere kaže, da so povezani z razredom. To je pravzaprav za naš problem zelo veliko. Spomnimo se, preučujemo učinke



Slika 2.3: Porazdelitev ocen informativnosti atributov, ki smo jih dobili na "pokvarjenih" podatkih, kjer smo premešali vrednosti razredne spremenljivke oziroma premetali kolono z razredom.



Slika 2.4: Primerjave porazdelitve ocen informativnosti atributov na ponaključenih podatkih (rumena črta) in pravih podatkih (histogram v modrem). Siva navpičnica nakazuje mejo, od katere desno je samo zelo majhen delež ocen ($p = 0.001$), ki so bile pridobljene na naključnih podatkih.

kajenja in kot vse kaže obstaja kar ena desetina vseh opazovanih genov, katerih izražanje v limfocitih se zaradi kajenja spremeni.

Seveda se tu lahko vprašamo, kakšna meja je smiselna, oziroma kakšna je smiselna verjetnost napake, s katero trdimo, da je atribut informativen, čeprav bi lahko njegovo kvantitativno vrednost informativnost dobili tudi na naključnih podatkih. Odgovora za to ni. Vrednost p je pač nek parameter, katerega intuitivno interpretacijo smo podali zgoraj, nekega pravila zanj pa ni. Meja, ki smo jo postavili zgoraj, torej $p = 0.001$, je tipično dovolj ostra, da na realnih primerih z uporabo nje dobimo z razredom dobro povezane atribute.

2.4 Izbor atributov

Zgoraj smo ravno pridelali metodo za ocenjevanje in izbor značilk po principu filtriranja. Ocenjevanje nam pomaga pri razpoznavi tistih značilk, ki so za naš problem pomembne. Če bi bili biologi, bi nas seznam genov, ki so najbolj različno izraženi pri dveh skupinah (kadilke, nekadilke) močno zanimali. A nismo, tako da bomo tu razlago izbranih atributov zaenkrat preskočili (se boste pa s tem pozabavili pri domači nalogi na problemu, kjer bo informativnost značilk moč enostavno interpretirati). Tu razmislimo o drugačni uporabi filtriranja značilk. In sicer, v povezavi z napovednimi modeli. Ideja je, da lahko dober izbor značilk pomaga h gradnji bolj točnih modelov, ali pa vsaj pomaga k poenostavitvi problema tako, da so lahko ti odvisni od manj parametrov in jih je zato hitreje naučiti in se morda lahko manj prilegajo k učnim podatkom.

Za poskus, ali zgornje drži, naredimo nekaj poskusov. Pri prvem izberimo manjši nabor najbolj informativnih značilk in z prečnim preverjanjem ocenimo kvaliteto napovednih modelov na tako pridobljenih podatkih. Rezultate kaže tabela 2.1.

atributov	log. reg.	k-NN
14093	0.924	0.721
1000	0.936	0.886
100	0.924	0.873
10	0.784	0.886

Tabela 2.1: Klasifikacijska točnost logistične regresije in metode najbližjih sosedov ocenjena s prečnim preverjanjem pri podatkih, kjer smo pred prečnim preverjanjem izbrali najbolj informativne atribute.

Zanimivo. Dobre točnosti lahko dosežemo že na podatkih, ki vsebujejo veliko (veliko!) manjše število atributov. Točnost metode najboljših sosedov se z izborom celo poveča.

Pa je naš pristop res pravi? Naredimo (en malce odštekan) eksperiment. Uničimo podatke, izberimo nekaj najboljših značilk in poženimo prečno preverjanje. Podatke bomo “uničili”

tako, kot smo to naredili pri permutacijskem testu: premešali bomo vrednosti razredne spremenljivke.

atributov	log. reg.	k-NN
14093	0.468	0.443
10	0.835	0.797

Tabela 2.2: Klasifikacijska točnost logistične regresije in metode najbližjih sosedov ocenjena s prečnim preverjanjem na "uničenih" podatkih, kjer smo premešali vrednosti v koloni razredne spremenljivke. V drugi vrstici tabele smo pred prečnim preverjanjem izbrali deset najbolj informativnih atributov.

Rezultati (tabela 2.2) na tako uničenih podatkih so kar se da presenetljivi. Naša taktika izbora se namreč obnese celo na čisto zanič podatkih. Pa je to res tisto, kar smo želeli? Seveda ne. Na naključnih podatkih bi morala točnost napovedi biti slaba. Tako pa je na izboru deset najbolj informativnih atributov skoraj odlična. Le zakaj? Kaj smo storili z izborom destih najboljših atributov na naključnih podatkih? Čemu pravzaprav služi prečno preverjanje po opravljenem izboru?

Prečno preverjanje nam tu pokaže le, da smo med številnimi atributi izbrali tiste, ki so, čeprav po naključju, dobro povezani z razredom. Seveda med razredom obstaja, saj smo ravno take atribute izbrali. In pri tem upoštevali vrednost razreda. V naslednjem koraku pa smo se delali, torej pri prečnem preverjanju, da poznamo samo del primerov (učna množica), drugega dela pa ne (testna množica). A ta del smo že videli. Kje? Pri predhodnem izboru atributov, seveda.

Taktika predhodnega izbora atributov, ki ji sledi prečno preverjanje, je goljufija. Predprocesiranje podatkov je namreč korak, ki je del učenja, in ga moramo torej poganjati znotraj prečnega preverjanja. Torej, prečno moramo preveriti celoten postopek, tako ocenjevanje atributov, izbor le teh, in potem gradnjo modelov. Vse to lahko torej počnemo le na učni množici, testiramo pa na popolnoma ločeni množici, ki je pri ocenjevanju in izboru atributov nismo upoštevali. Pri takem, pravilno implementiranem izboru atributov v kombinaciji s prečnim preverjanjem dobimo klasifikacijske točnost, ki so slabe oziroma približno take, kot če bi napovedovali večinski razred učnih podatkov.

2.5 Multivariatne ocenjevanje značilk

Za konec in za razmislek si ustvarimo en na videz enostaven, a v resnici težek nabor podatkov. Težek zato, ker na njem pade velika večin učnih algoritmov. In sicer za matriko atributov X naključno generirajmo vrednosti med 0 in 1, razred, ki ga pripišemo primerom, pa naj bo binaren in enak $x_1 > 0.5 \wedge x_2 > 0.5$. Za začetek na ta način ustvarimo podatke z 300 primeri in

10 atributi ter na teh podatkih s prečnim preverjanjem opazujmo točnost logistične regresije in naključnega gozda.

log. reg.	gozd
0.537	0.843

Tabela 2.3: Točnost logistične regresije in naključnega gozda na umetno ustvarjenem naboru primerov z dvema povezanima atributoma.

Logistična regresija zgreši popolnoma (tabela 2.4). Njena točnost je podobna modelu, ki vedno napoveduje večinski razred učne množice. Se pa na tem primeru presenetljivo dobro obnašajo naključni gozdovi. Ti delujejo po principu "slepa kura zrno najde", saj v korenskem vozlišču pri vsakokratni gradnji le po naključju zberejo ali prvi ali drugi atribut. Pri desetih atributih je verjetnost za tak izbor enaka 20%. Ko, oziroma če je v zgornjem vozlišču izbran eden od teh dveh atributov, bo drevo "pravo", saj bo v naslednjem vozlišču izbran komplementarni atribut, ker ravno ta nosi največ informacije in ga lahko univariatne mere ocenjevanje atributov, ki jih sicer uporabljajo klasifikacijska drevesa, prepoznajo pri pogoju, da smo že odkrili prvi atribut (kar smo, tega smo dali v koren). Model bom v tem primeru popoln. Varianta je tudi, da do takega izbora pride v nekorenskih vozliščih, kar bi vodilo do manj popolnih modelov. Ni pa izbor "pravega" atributa tu načrten oziroma je popolnoma naključen, saj temelji na tem, da bo metoda izmed vseh enako pomembnih atributov skladno z univariatno oceno naključno izbrala enega od dveh atributov, ki nosita informacijo o razredu.

Naključni gozd lahko uporabimo za ocenjevanje atributov. Načinov za to je več. Izumitelj Leo Brieman je na primer predlagal, da v te namene zgradimo gozd na bootsrap vzorcu podatkov, ter ga vrednotimo na preostalih podatkih, torej teh, ki niso zajeti v vzorcu (t.im. vzorec *out-of-bag*). Za izdelani model napovedno vrednost atributo ocenimo tako, da v out-of-bag vrednosti tega atributa premešamo (tako, kot smo to počeli pri permutacijskem testu). Atributi, pri katerih bo na ta način točnost na testni množici padla, so bolj pomembni. Drug, hitrejši način pa je, da pregledamo drevesa v gozdovih in preštejemo, kolikokrat je posamezni atribut v drevesu uporaben in kje, pri čemer damo višjo težo uporabi bližje korenu drevesa. Ta način ocenjevanja atributov je odvisen samo od modela in ne potrebuje ločene testne množice.

Zanima nas, kako dobre so take ocene. Prvi in drugi atribut, iz katerih smo izračunali vrednost razreda, bi morala biti najbolje ocenjena in biti na prvem mestu. Ocene lahko primerjamo z metodo Relief, katere avtorji izpopolnjene različice so iz FRI: Kononenko in Robnik Šikonja! Ideja ocenjevanja s to oceno je naslednja. Vzemi nek naključen referenči primer in bližnji primer z istim (nearHit) ter bližnji primer z različnim razredom (nearMis). Pri primeru nearHit za informativen atribut pričakujemo, da se vrednost atributa ne bo spremenila prav dosti, pri nearMiss pa, da je sprememba atributa precejšnja. Iz te intuicije lahko izpišemo

enačbo za oceno atributov, oceno samo pa izračunamo na podlagi večkratnega vzorčenja referenčnih primerov.

V tabeli 2.4 so rezultati testiranja ocenjevanj atributov z metodo Relief in naključnim gozdom. Merili smo, kakšen je delež dejansko neinformativnih atributov, ki so bili slabše ocenjeni od znanih dveh informativnih atributov. Meritve smo ponovili petdesetkrat in izračunali povprečje ocenjevanj.

atributov	Relief	gozd
10	1.00	1.00
20	1.00	0.97
50	1.00	0.82
100	0.98	0.91
500	0.82	0.69
1000	0.72	0.03
5000	0.58	0.50
10000	0.74	0.69

Tabela 2.4: Zmožnost prepoznavanja dveh informativnih atributov v interakciji v množici neinformativnih atributov z metodo Relief in naključnim gozdom na podatkih z danim številom vseh atributov (prva kolona). Ocena pove, kakšen je delež neinformativnih atributov, ki je bil slabše ocenjen od dveh znanih informativnih atributov.

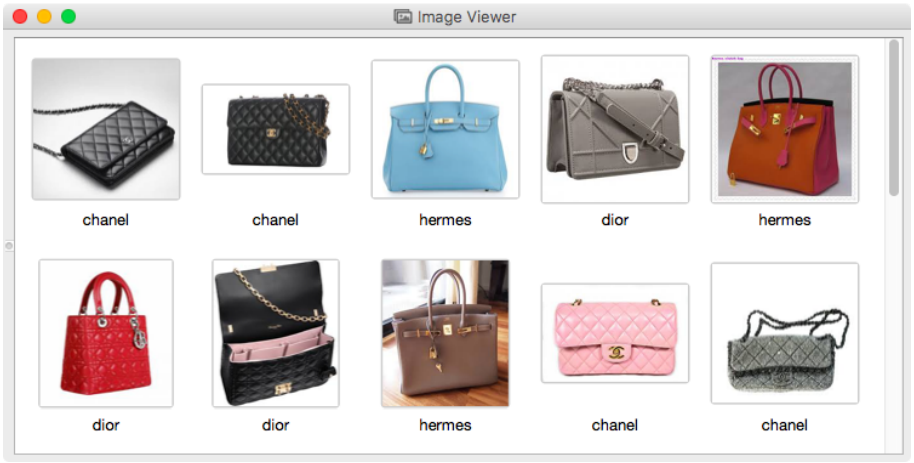
Relief očitno bolje prepoznava atribute, ki so v interakciji od naključnih gozdov. Vrednosti sicer nihajo (morali bi izvesti več kot petdeset ponovitev), a je očitno, da metodi delata odlično do nekako 100 atributov, potem pa obema naraščajoče število atributov predstavlja problem. Problem odkrivanja atributov v interakcijah je težek in je zanj znano, da hitre rešitve ne obstajajo. Še več, število potrebnih primerov za odkrivanje interakcij je eksponentno odvisno od števila atributov. Če smo bili pri našem primeru podatkov z 300-timi primeri radodarni za manjšo množico atributov, ta učna množica ni zadoščala odkrivanju interakcij, ko je bilo atributov več.

Poglavje 3

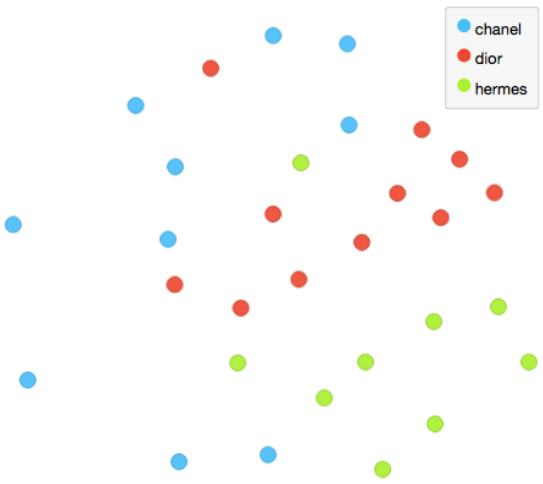
Projekcije in zmanjšanje dimenzionalnosti podatkov

Modeli, ki jih gradimo v strojnem učenju, povzemajo podatke tako, da v nekem formalnem zapisu predstavijo glavne vzorce, ki so te podatke oblikovali. V primeru večdimenzionalnih podatkov nas tako na primer zanima, ali bi podatke lahko popisali z manjšim številom spremenljivk. Z izborom značilk smo se ukvarjali že v prejšnjem poglavju, a nas bo tu zanimalo, če lahko podatke predstavimo z novimi značilkami, ki bi podatke lahko predstavili bolj kompaktno, pri tem odkrili kakšno zanimivo preslikavo starih v nove značilke, ter morda celo dosegli, da je novih značilk izjemno malo. Recimo, dve, tako da lahko vse primere izrišemo v razsevnem diagramu in tam s pomočjo vizualizacije razmišljamo o njihovih podobnosti in strukturi prostora primerov. Ljudje smo vizualna bitja in nam izris kart primerov lahko intuitivno pove več kot pa recimo matematične enačbe, ki morda te povezujejo.

Začnimo s primerom. Na sliki 3.1 so ženske torbice različnih proizvajalcev. Z uporabo že zgrajenih globokih mrež za razvrščanje slik lahko slike torbic pretvorimo v vektorje. Mreža Inception v3 nam na svojem predzadnjem nivoju na primer za vsako sliko vrne vektor dolžine 2048, oziroma sliko popiše z 2048 atributi. S postopki, ki jih bomo opisali v tem poglavju, lahko take opise zmanjšamo in predstavimo torbice v nizko dimenzionalnem prostoru. Slika 3.2 tako prikazuje predstavitev primerov v dvodimenzionalnem prostoru, kjer je razvidno, da ena Hermesova torbica bolj podobna torbicam ostalih proizvajalcev, in da se je Dior pri eni od torbic zgledoval po torbicah Chanela. Morda. Vsekakor bi ta opažanja morali preveriti, a je zanimivo, kako hitro nas vizualizacije navdahnejo z idejami. Prav zato so vizualizacije v odkrivanju znanj iz podatkov tako pomembne in zato so pomembne tudi tehnike zmanjšanja dimenzionalnosti in odkrivanja nizko dimenzionalnih projekcij podatkov.



Slika 3.1: Ženske torbice različnih proizvajalcev.



Slika 3.2: Predstavitev ženskih torbic s slike 3.1 v dvodimenzionalnem prostoru.

3.1 Metoda glavnih komponent

Prva, morda tudi najbolj znana metoda za zmanjšanje dimenzij in odkrivanje zanimivih projekcij podatkov je metoda glavnih komponent. Predpostavimo, da imamo dano matriko učnih primerov $X \in \mathbb{R}^{m \times n}$, kjer je m število primerov in n število atributov. Primer $x^{(i)} \in X$, torej i-ti primer v učni množici primerov, bo tako opisan z n atributi $x \in \mathbb{R}^n$, za katere bomo tu privzeli, da so njihove vrednosti realna števila. V splošnem iščemo projekcije podatkov tako, da atributni zapis primerov z n atributi nadomestimo z atributnim zapisom z k novimi atributi tako, da je $k \ll n$ in da nam novi atributi povedo čim več o primerih iz učne množice.

Pričnimo s $k = 1$, torej s projekcijo v eno samo dimenzijo. Dogovorimo se, da bo naša projekcija linearna in da bomo vrednost novega atributa tvorili kot linearno kombinacijo originalnih atributov. Smer projekcijske ravnine oziroma smer premice, na katero bomo projicirali podatke označimo z enotskim vektorjem u_1, za katerega torej velja $u_1^\intercal u_1 = 1$.

Naj bo $x^{(i)}$ primer iz učne množice. Njegova projekcija na projekcijsko ravnino je skalar, $u_1 x^{(i)} \in \mathbb{R}$. Naj bo $\bar{x}$ središčna točka podatkov:

$$\bar{x} = \frac{1}{m} \sum_i x^{(i)}$$

Projekcija središčne točke na projekcijsko ravnino je $u_1 \bar{x}$. Primeri v učni množici odstopajo od središčne točke, eni bolj, drugi manj. Primere bi želeli projicirati na ravnino tako, da ta odstopanja čim bolj zajamemo, torej, da so tudi v projicirani ravnini čim večja. Povprečno kvadratno odstopanje v projekciji od projekcije središčne točke imenujemo tudi varianca točk v projekciji:

$$\mathrm{Var}(u_1^\intercal X^\intercal) = \frac{1}{m} \sum_{i=1}^{m} \left(u_1^\intercal x^{(i)} - u_1^\intercal \bar{x} \right)^2$$

Z drugimi besedami, želimo poiskati tak projekcijski vektor u_1, ki maksimizira varianco projekcije $\mathrm{Var}(u_1^\intercal X^\intercal)$. Poigrajmo se malce z enačbo za varianco projekcije:

$$\begin{aligned} \mathrm{Var}(u_1^\intercal X^\intercal) &= \frac{1}{m} \sum_{i=1}^{m} \left(u_1^\intercal x^{(i)} - u_1^\intercal \bar{x} \right)^2 \\ &= \frac{1}{m} \sum_{i=1}^{m} \left(u_1^\intercal x^{(i)} u_1^\intercal x^{(i)} - 2 u_1^\intercal x^{(i)} u_1^\intercal \bar{x} + u_1^\intercal \bar{x} u_1^\intercal \bar{x} \right) \end{aligned} \tag{3.1}$$

Tu upoštevamo, da je $u_1^\intercal x$ skalarni produkt dveh vektorjev, in da je transponirana vrednost realnega števila enaka temu številu. Torej je na primer $u_1^\intercal x = (u_1^\intercal x)^\intercal = x^\intercal u_1$. Z upoštevanjem

tega se nam zgornji izraz primerno poenostavi:

$$\begin{aligned}\mathrm{Var}(u_1^\intercal X^\intercal) &= \frac{1}{m}\sum_{i=1}^{m} u_1^\intercal\Big(x^{(i)}x^{(i)\intercal} - 2x^{(i)}\overline{x}^\intercal + \overline{x}\,\overline{x}^\intercal\Big)u_1 \\ &= u_1^\intercal\Big(\frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \overline{x})(x^{(i)} - \overline{x})^\intercal\Big)u_1\end{aligned} \tag{3.2}$$

V oklepaju je kovariančna matrika! Prejšnji stavek smo končali s klicajem zato, ker je kovarianca znan in mnogokrat uporabljan koncept v statistiki in nam pove, kako sta dve naključni spremenljivki povezani. Označimo kovariančno matriko s S:

$$\mathrm{S} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \overline{x})(x^{(i)} - \overline{x})^\intercal \tag{3.3}$$

V našem primeru imamo n atributov, kar pomeni, da je kovariančna matrika dimenzije $\mathrm{S} = \mathbb{R}^{n\times n}$. Zapišimo še enkrat dobljeno enačbo za varianco:

$$\mathrm{Var}(u_1^\intercal X^\intercal) = u_1^\intercal \mathrm{S} u_1 \tag{3.4}$$

Spomnimo, da je naš namen poiskati projekcijski vektor u_1 pri katerem je zgornja varianca maksimalna. Za vektor u_1 zahtevamo, da je to enotni vektor (sicer maksimizacija zgornje enačbe ne bi imela smisla, saj bi to dosegli z neskončno dolgim vektorjem u_1). Maksimizacijo variance, kjer iščemo ustrezen vektor u_1 z omejitvijo $u_1^\intercal u_1 = 1$ rešimo z uporabimo Lagrangeovih multiplikatorjev. Maksimiziramo torej izraz:

$$f(u_1) = u_1^\intercal \mathrm{S} u_1 + \lambda_1(1 - u_1^\intercal u_1) \tag{3.5}$$

V maksimumu bo odvod zgornje enačbe enak nič:

$$\frac{\partial f(u_1)}{\partial u_1} = \mathrm{S}u_1 - \lambda_1 u_1 = 0 \tag{3.6}$$

kar pomeni,

$$\mathrm{S}u_1 = \lambda_1 u_1 \tag{3.7}$$

Tu je S matrika katere vrednosti poznamo, u_1 je vektor, ki ga iščemo, λ_1 pa skalar. Poznano? Seveda! Zgornjemu pogoju ustreza le tak u_1, ki je lastni vektor matrike S. Ampak korelacijska matrika ima več lastnih vektorjev. Kateri med njimi je pravi? Množimo zgornjo enačbo z $u_1^\intercal$:

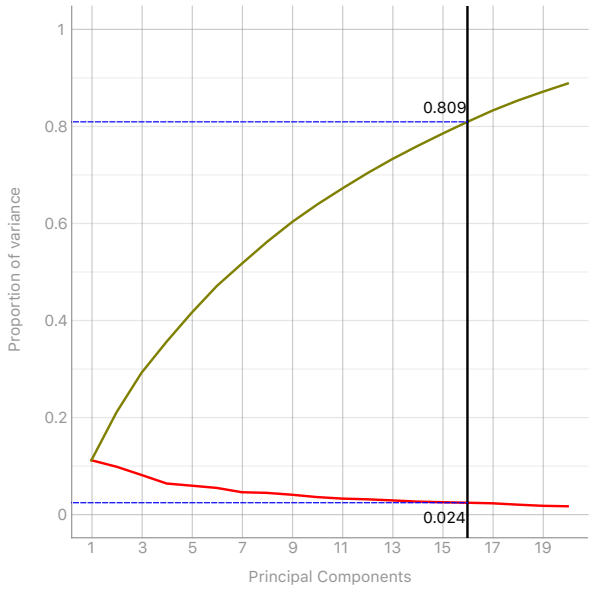
$$u_1^\intercal \mathrm{S} u_1 = u_1^\intercal \lambda_1 u_1 = \lambda_1 u_1^\intercal u_1 = \lambda_1 \tag{3.8}$$

Na levi je izraz za našo varianco $\mathrm{Var}(u_1^\intercal X^\intercal)$, ki jo skušamo maksimizirati. Vemo, da mora biti

u_1 lastni vektor kovariančne matrike. Skladno z zgornjo enačbo pa sedaj tudi vemo, da je to lastni vektor z najvišjo pripadajočo lastno vrednostjo.

V smeri vektorja u_1 se lega naših primerov X v n-dimenzionalnem prostoru torej najbolj spreminja, njihova varianca je v tej smeri največja. Z $u_1^\intercal X^\intercal$ dobimo pravokotno projekcijo vsakega od primerov na to os. S to projekcijo, oziroma legi v smeri u_1, smo pojasnili največjo varianco primerov. Kaj še ostane? Vse, kar je pravokotno na smer u_1. Največjo varianco na pravokotni hiperravnini pa je ravno v smeri drugega lastnega vektorja, saj je ta pravokoten na u_1, to je vektorja, ki ima drugo največjo lastno vrednost. Pojasnjen delež te variance je λ_2, skupaj pa s pravokotno projekcijo na prvi in drugi lastni vektor pojasnimo $\lambda_1 + \lambda_2$ variance množice primerov.

Lastni vektorji kovariančne matrike nam določajo nov koordinatni sistem, kjer so koordinate, po vrsti, tiste, po katerih se lege točk v večdimenzionalnem prostoru najbolj spreminjajo, oziroma je njihova varianca največja. Nove koordinate, urejene skladno z lastnimi vrednostmi vektorjev, imenujemo komponente primerov v novem, transformiranem sistemu. Ker nas bodo zanimale samo te z visokimi deleži razložene variance jih bomo imenovali *glavne komponente*.



Slika 3.3: Graf odvisnosti razložene variance od števila glavnih komponent (zgornja črta) za podatke o torbicah iz slike 3.1.

V splošnem nas zanima, koliko najvišje rangiranih dimenzij v novem koordinatnem sistemu zares potrebujemo za opis podatkov. Pri tem moramo seveda določiti, kakšen je naš ciljni delež pojasnjene variance. Tipično smo zadovoljni, če izbrano število komponent pojasni vsaj 80% skupne variance. Ker skupno varianco poznamo (enaka je vsoti kvadratov razdalj do centra), je potrebno torej le določiti, koliko začetnih urejenih lastnih vrednosti moramo

sešteti, da njihova vsota predstavlja želeni delež razložene variance. Primer grafa, ki kaže odvisno deleža razložene variance glede na število glavnih komponent kaže slika 3.3.

Pristop glavnih komponent se mnogokrat uporablja tudi samo za namene vizualizacije, kjer primere, opisane z mnogimi atributi, želimo predstaviti v dvodimenzionalni ravnini. Pri tem se moramo seveda zavedati, da prvi dve komponenti razložijo morda le majhen del celotne variance.

Za predstavitev podatkov z glavnimi komponentami je potrebno podatke najprej osrediščiti in nato poiskati kovariančno matriko. Lastni vektorji kovariančne matrike so bazni vektorji novega, transformiranega sistema, njihove lastne vrednosti pa povedo, kakšen delež variance nam razloži posamezna komponenta.

Še praktičen razmislek: v primerih velikega števila atributov bo računanje vseh lastnih vektorjev in vrednosti zamudno. Še posebej, če nas zanima samo projekcija podatkov v dvodimenzionalni prostor. Prvi lastni vektor kovariančne matrike M lahko rajši (hitreje) določimo numerično, s potenčno metodo. Vzamemo nek naključni vektor (npr. x), ga transformiramo z matriko M (torej, izračunamo $x \leftarrow Mx$), in to ponavljamo do konvergence. Ob vsakem koraku tako dobljeni x normaliziramo, torej, $x \leftarrow \frac{Mx}{\|Mx\|}$. Na ta način dobimo lastni vektor. Kako izračunamo pripadajočo lastno vrednost?

$$Mu = \lambda u \tag{3.9}$$

$$u^\intercal Mu = u^\intercal \lambda u = \lambda u^\intercal u = \lambda \tag{3.10}$$

Kako določimo naslednjo komponento? Ena od možnosti je, da uporabimo ravnokar izračunani lastni vektor, nanj projiciramo podatke, te vrednosti odštejemo od podatkov (torej dobimo projekcijo na lastnemu vektorju pravokotno hiperravnino) in za te podatke ponovno izračunamo kovariančno matriko ter s potenčno metodo njej lastni vektor z največjo lastno vrednostjo. V primeru, da potrebujemo več komponent, postopek ustrezno ponovimo.

Za primer, ko nas zanima samo projekcija podatkov v ravnino lahko potenčno metodo namesto na vektorju izvedemo na sistemu dveh vektorjev U. Vektorja matrike U morata biti pravokotna. To dosežemo tako, da po vsakem koraku potenčne metode uporabimo Gram-Schmidtovo ortogonalizacijo.

3.2 Večrazredno lestvičenje

Pri zmanjšanju dimenzij nam je lahko cilj, da ohranimo razdalje med posameznimi primeri. Tu predpostavimo, da za vsak par primerov znamo med njimi oceniti razdaljo. Se pravi, če imamo primere predstavljene v atributnem zapisu, potem uporabimo recimo Evklidsko razdaljo ali pa kosinusno podobnost. Lepota tehnike, ki jo bomo predstavili v tem razdelku je, da nas ne zanima, kako je bila podobnost oziroma razdalja med primeri ocenjena. Lahko

smo na vhodu na primer imeli tekstovne dokumente, za katere razdaljo recimo izračunamo s kakšno kompresijsko tehniko. Ali pa filme, za katere razdaljo izračunamo z ozirom na njihovo gledanost v spletni sposojevalnici. Ali pa čivke, ki jih primerjamo med sabo glede na to, kdo jih je všečkal.

Naj bo razdalja med primeroma enaka δ_{ij}. Sedaj te primeri vložimo, recimo, v dvodimenzionalno projekcijo tako da vsakemu primeru i pripišemo koordinati $\boldsymbol{\theta}^{(i)} = (\theta_1^{(i)}, \theta_2^{(i)})$. Želeli bi, da se v njej razdalje ohranjajo, to je, da je projekcijska razdalja, ki naj bo d_{ij}, čimbolj podobna originalni razdalji med primeroma. Razdalja primerov v projekcijskem prostoru je:

$$\delta_{ij} = \left\| \boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}^{(j)} \right\|$$

Računalniška izvedba algoritma, ki poišče tako vložitev, imenujemo večrazsežnostno lestvičenje (*multidimensional scaling*, MDS). Formalno ciljno funkcijo problema zastavimo tako, da določimo energijo sistema ali napetost (*stress*) kot, recimo,

$$J(\mathbf{X}) = \sum_{i \neq j} (d_{ij} - \delta_{ij})^2,$$

kjer je $\mathbf{X}$ trenutni razpored točk, d_{ij} in δ_{ij} pa trenutna projekcijska in želena razdalja med primeroma i in j. Večrazsežnostno lestvičenje poišče razpored $\mathbf{X}$ z najmanjšo napetostjo $J(\mathbf{X})$ oziroma najmanjšo vrednostjo kriterijske funkcije.

Kako minimizirati takšno funkcijo? Analitično ne bo šlo. Za to bi bilo potrebno odvesti $\sigma(\mathbf{X})$ po vseh koordinatah $\theta_1^{(i)}$ in $\theta_2^{(i)}$ in poiskati ničle:

$$\begin{aligned}
\frac{\partial \sigma(\mathbf{X})}{\partial \boldsymbol{\theta_k}} &= \sum_{j \neq i} \frac{\partial \sigma(\mathbf{X})}{\partial d_{ij}} \frac{d_{ij}}{\partial \boldsymbol{\theta_k}} \\
&= -\sum_{j \neq i} 2 \left(\sqrt{(\theta_1^{(i)} - \theta_1^{(j)})^2 + (\theta_2^{(i)} - \theta_2^{(j)})^2} - \delta_{ij} \right) \frac{(\theta_k^{(i)} - \theta_k^{(j)})}{\sqrt{(\theta_1^{(i)} - \theta_1^{(j)})^2 + (\theta 2^{(i)} - \theta_2^{(j)})^2}} \\
&= -\sum_{j \neq i} 2 \left(d_{ij} - \delta_{ij} \right) \frac{(\theta_k^{(i)} - \theta_k^{(j)})}{d_{ij}}
\end{aligned} \tag{3.11}$$

Pri stotih točkah bi dobili sistem z dvesto takšnimi enačbami; ideji o direktnem napadu se torej raje odpovemo.

Drug, izvedljiv postopek je gradientna metoda: kot pri dosedanjih pristopih z gradientnim spustom (recimo, linearna regresija) izračunamo gradient funkcije kriterijske funkcije $J(\mathbf{X})$. To smo pravzaprav storili že zgoraj. Izberemo si poljuben začetni razpored točk, vstavimo koordinate v odvode, dobimo gradient (vektor odvodov) in se pomaknemo v smeri nasprotni gradientu, se pravi, vse točke pomaknemo v smer, kamor jih potiskajo odvodi. Gradientna metoda bi delovala, vendar ne prav dobro. Bila bi počasna in velikokrat bi nas (lahko) pripeljala

le v lokalni minimum.

V svetu optimizacijskih problemov je gradientna metoda le metoda grobe sile, ki jo bomo uporabili, če se ne moremo spomniti ničesar boljšega. Eden od boljših postopkov je "majorizacija". Označimo funkcijo, katere minimum iščemo, z $f(x)$. Recimo, da si znamo izmisliti, neko drugo funkcijo $g(x, y)$, ki je stalno večja ali enaka $f(x)$ (torej, za vsak y velja $g(x, y) \geq f(x)$ pri vsakem x), v točki $g(x, x)$ pa velja $g(x, x) = f(x)$. Funkcija $g(x, y)$ naj bo takšna, da znamo dobro poiskati vrednost x, pri katere doseže minimum. Zdaj lahko počnemo tole: izmislimo si začetni x_0. Pri njem, vemo, velja $g(x_0, x_0) = f(x_0)$. Minimiziramo funkcijo g in dobimo nov, boljši x - označimo ga z x_1. Vrednost $g(x_1, x_0)$ je manjša ali enaka vrednosti $g(x_0, x_0)$, saj smo minimizirali g po prvem argumentu. Po drugi strani, spet po definiciji funkcije g, velja $f(x_1) \leq g(x_1, x_0)$. Imamo torej

$$f(x_1) \leq g(x_1, x_0) \leq g(x_0, x_0) = f(x_0)$$

Postopek ponovimo z x_1, da pridelamo naslednji, še boljši x_2.

Takšna optimizacija je, tako kot gradientna metoda še vedno iterativna, vendar hitrejša. Koliko hitrejša, je odvisno od tega, kako dobro $g(x, y)$ smo poiskali; $g(x, y)$ se mora čim tesneje prilegati $f(x)$, poleg tega pa jo moramo biti zmožni čim boljše optimizirati.

Algoritem večdimenzionalnega lestvičenja, ki deluje na ta način, se imenuje SMACOF (angl. *scaling by majorizing a complicated function*). SMACOF je hitrejši od gradientne metode, dela večje korake in se manjkrat zatakne v lokalnih ekstremih, zato je uporabnejši za (realistične) primere, v katerih je točk veliko.

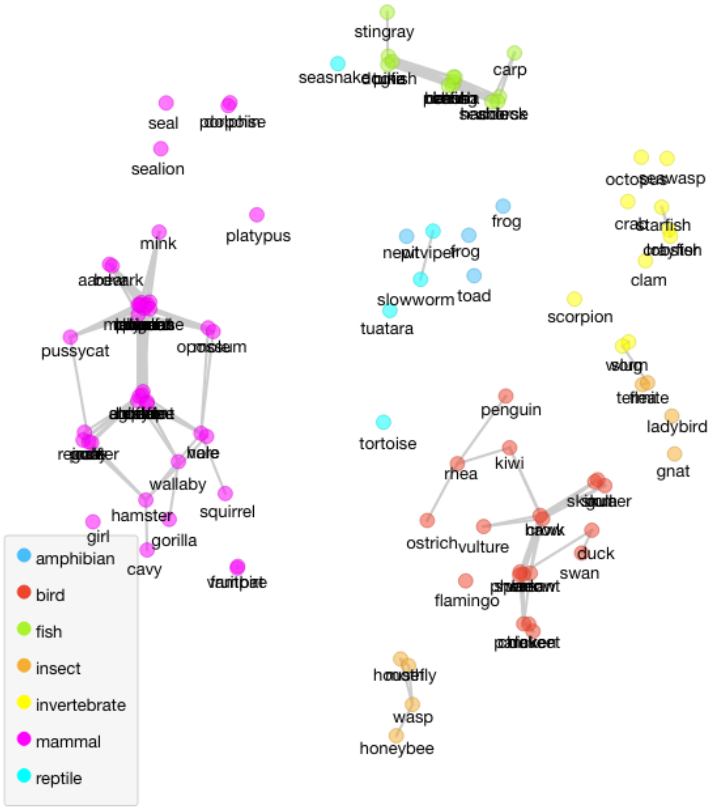
Za izračun napetosti lahko namesto gornje, naivne formule uporabljamo tudi takšne, ki uteži prispevke parov (predpišemo lahko, za katere pare si še posebej želimo, da bi razdalja ustrezala pravi), poleg tega pa lahko napetost posameznega para normiramo tako, da jo delimo z želeno ali s trenutno razdaljo.

Slika 3.4 kaže zemljevid živali. Razdalje med pari živali so izračunane kot evklidske razdalje med atributi, ki jih opisujejo. Točke, ki predstavljajo živali, smo obarvali glede na vrsto – vijolični so sesalci, rdeče ptice, oranžno žuželke. Povezani so pari živali, ki so si med seboj najbolj podobne.

Čeprav MDS ni uporabljal podatkov o vrstah živali, so različne vrste na zemljevidu dobro ločene. Vidimo tudi nekaj podobnosti med živalmi: morske sesalce je MDS postavil v bližino rib; dvoživke so blizu rib in nevretenčarjev, predvsem morskih; žuželke so pomešane s ptiči.

Vidimo tudi težavo: žuželke so razdeljene v dve skupini. MDS se je očitno znašel v lokalnem ekstremu, saj bi morale ose in čebele k ostalim žuželkam, vendar bi jih moral optimizacijski postopek peljati okrog ptičev. To se zgodi kar pogosto in pomagamo si tako, da podatke nekoliko potresemo, naključno premaknemo točke, ali pa začnemo celotno optimizacijo znova.

Večrazežnostno lestvičenje je torej postopek, ki vizualizira podatke v obliki zemljevida.



Slika 3.4: Zemljevid živali (podatkovna zbirka Zoo). Vrsta živali je označena z barvo. Nekatere oznake so zaradi prekrivanj neberljive in bi razvojnike programa potrebno opozoriti, da v vizualizacijo vključijo optimizacijo postavitve oznak.

Podatki so opisani s seznamom primerov in razdalj med njimi; lastnosti primerov niso potrebne in jih ne znamo upoštevati (razen, če iz le-teh računamo razdalje, kot smo storili pri živalih). Zemljevid je nezanesljiv v tem smislu, da bodo različne naključne začetne postavitve vodile v različne končne slike, a vsaka od njih nam lahko nudi drugačen pogled na podatke. Prav tako je nezanesljiv položaj posameznih točk; dogaja se, da je točka obtičala v lokalnem minimumu, čeprav v resnici ne sodi na to mesto. Takšne točke lahko prepoznamo, če zemljevidu dodamo povezave med najbolj podobnimi pari. Obenem nam takšne povezave pomagajo brati sliko. MDS ne sestavlja gruč, pač pa lahko gruče razberemo sami.

3.3 Stohastična vložitev sosedov

Je lahko še kaj boljšega za vložitev primerov v dvorazsežni prostor kot večrazredno lestvičenje? Oziroma, kaj bi bilo lahko sploh narobe s tehniko MDS? Problem MDS-a je, da se ta osredotoča tako na primere, ki so si v originalnem prostoru blizu, kot tudi na primere, ki so si daleč. MDS skuša ohraniti razdaljo med primeri ne glede na to, kako velika je. Velikokrat pa nas v vizualizacijah zanima samo to, da so primeri, ki so se med seboj podobni, skupaj tudi v projekcijskem prostoru. Torej, ohranjanje razdalje med primere, ki so si med seboj drugačni, nas ne zanima, ali pa nas, recimo, zanima veliko manj kot ohranjanje bližine med seboj podobnih primerov.

Metoda SNE (angl. *stochastic neighbor embedding*), ki smo jo tu poslovenili v stohastično vložitev sosedov, temelji na verjetnostni oceni podobnosti dveh primerov. Za primera i in j oziroma njuni vektorski predstavitvi $x^{(i)}$ in $x^{(j)}$ določimo verjetnost $p_{j|i}$, da bi za primer i izbrali soseda j, če bi sosede izbirali skladno z Gaussovo verjetnostno porazdelitvijo, ki je centrirana v $x^{(i)}$:

$$p_{j|i} = \frac{\exp(-\left\|x^{(i)} - x^{(j)}\right\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\left\|x^{(i)} - x^{(k)}\right\|^2 / 2\sigma_i^2)} \tag{3.12}$$

kjer je σ_i varianca Gaussove porazdelitve za točko i. Ker nas zanimajo samo modeliranje podobnosti med različnimi točkami, je $p_{i|i}$ enaka nič. Zanima nas vložitev teh primerov v projekcijski prostor, kjer bo lega primerov kot pri MDS določena z vektorji $\boldsymbol{\theta}^{(i)}$ in bomo podobno kot zgoraj, le tokrat za projekcijski prostor, sosednost predstavili z verjetnostjo q_{j_i}.

Označimo verjetnostne porazdelitve preko vseh sosedov z za primer i in za originalni prostor z P_i in za projekcijski prostor z Q_i. Naša želja je, da bi bili ti porazdelitvi med sabo čim bolj podobni. Ker ti dve verjetnostni porazdelitve favorizirajo bližnje primere oziroma so za te verjetnosti visoke, podobnost porazdelitev P_i in Q_i pomeni, da je projekcija ohranila bližnje primere za primer i. Mera, ki pove, kako zvesto $q_{j|i}$ sledi $p_{j|i}$ je Kullback-Leibler-jeva divergenca, katere vsoto po primerih želimo minimizirati. Kriterijska funkcija je zato:

$$J(\boldsymbol{\Theta}) = \sum_i KL(P_i \parallel Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \tag{3.13}$$

Zaradi nesimetričnosti Kullback-Leibler-jeva divergence bodo napake obravnavane nesimetrično: cena za v projekciji oddaljene točke, ki bi sicer morale biti blizu (majhen q in velik p) bo večja kot za točke, ki so bližje v projekciji, a bi sicer morale biti daleč (velik q in majhen p). Metoda SNE primarno ohranja sosednost.

Med parametri kriterijske funkcije nam je ostal nedoločen še σ_i, parameter verjetnostne porazdelitve za primer i. Metoda SNE ga določi skladno z lokalno gostoto primerov tako, da je za gostejše okolice σ_i primerno manjši.

Da bi določili vložitev moramo poiskati lege primerov $\boldsymbol{\theta}^{(i)}$ v projekcijskem prostoru. To lahko vnovič storimo z gradientnim sestopom:

$$\frac{\partial \boldsymbol{\Theta}}{\partial \boldsymbol{\theta}^{(i)}} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}^{(j)})$$

Gradient (popravek) bo večji pri večji razliki med p in q (prvi člen v produktu) med primeroma in bo potekal v smeri vektorja med primeri (drugi člen produkta).

Danes je predvsem znana "popravljena" verzija metode SNE, ki jo imenujemo t-SNE (angl. *t-distributed stochastic neighbor embedding*). V njej Gaussovo porazdelitev za projekcijski prostor zamenjajo s Studentovo t-porazdelitvijo, ki ima daljši rep. S to zamenjavo t-SNE dovoli, da so primeri, ki so v srednji okolici v originalnem prostoru lahko daleč stran v projekcijski ravnini. Tehnika namesto pogojnih verjetnosti uporablja združene verjetnosti, in simetrično oceno sosednosti za primere v originalnem prostoru,

$$p_{ij} = \frac{\exp(-\left\|x^{(i)} - x^{(j)}\right\|^2 / 2\sigma^2)}{\sum_{k \neq l} \exp(-\left\|x^{(k)} - x^{(l)}\right\|^2 / 2\sigma^2)}$$

ter za projekcijski prostor

$$q_{ij} = \frac{(1 + \left\|x^{(i)} - x^{(j)}\right\|^2)^{-1}}{\sum_{k \neq l} (1 + \left\|x^{(k)} - x^{(l)}\right\|^2)^{-1}}$$

Tudi tu porazdelitvi primerjamo s Kullback-Leiblerjevo divergenco,

$$J(\boldsymbol{\Theta}) = \sum KL(P \parallel Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ji}}$$

za katero poiščemo gradient:

$$\frac{\partial \boldsymbol{\Theta}}{\partial \boldsymbol{\theta}^{(i)}} = 4 \sum_j (p_{ij} - q_{ij}) \frac{(\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}^{(j)})}{(1 + \left\|\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}^{(j)}\right\|^2)}.$$

Zgornja enačba je po strukturi presenetljivo podobna enačbi 3.11 za gradient pri večrazrednem lestvičenju, kar je glede na drugačen način izpeljave nekoliko presenetljivo, glede na intui-

tivno razumevanje cilja obeh projekcij pa morda sploh ne. Glavna razlika obeh metod je seveda ocenjevanje razdalj oziroma pojmovanje, oziroma bolje uteževanje sosednosti. Če pri večrazrednem lestvičenju razdalja ni utežena, je pri metodi SNE in izvedenkah stopnja sosednosti eksponencialno padajoča funkcija.

Metoda t-SNE je v zadnjih letih postala izjemno popularna. Njen problematičen del je predvsem v parametrizaciji (iskanje prave vrednosti za parametre Gaussove distribucije) in v relativni počasnosti metode oziroma pripadajočega gradientnega pristopa.

3.4 FreeViz

Denimo, da imamo primere, ki so opisani z atributi, za vsak primer pa je podan tudi razred. Zemljevid takšnih podatkov bi sicer lahko narisali z večrazežnostnim lestvičenjem, kot smo to storili pri živalih, vendar bi tako zanemarili podatke o pripadnosti razredom. Poleg tega si pri takšnih podatkih navadno želimo imeti model, s katerim bomo lahko uvrščali nove podatke; MDS nam tega ne omogoča.

Želimo si torej metodo, ki vsaki točki priredi njeno mesto na podlagi vrednosti njenih atributov, pri čemer naj bo mesto določeno tako, da bodo primeri iz istega razreda blizu eden drugemu.

Primer takšne metode je FreeViz. FreeViz je linearna projekcija. Če imajo primeri n atributov, bomo rekli, da se nahajajo v n-dimenzionalnem prostoru; vsak primer je točka in njene koordinate so pač vrednosti atributov. Primere, kot smo to počeli do sedaj, opišemo z vektorjem-kolono $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^\intercal$. Vsakemu atributu ustreza ena dimenzija, njej pa en bazni vektor v originalnem prostoru primerov. Linearne projekcije so določene s slikami baznih vektorjev. Naj bodo torej $\boldsymbol{a_1}, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n$ slike baznih vektorjev, $\boldsymbol{a}_i = [a_{i,x}, a_{i,y}]^\intercal$. Zložimo jih v matriko, $\boldsymbol{A} = [\boldsymbol{a}_0|\boldsymbol{a}_1|\ldots|\boldsymbol{a}_k]$. Projekcija primera $\boldsymbol{x}$ je tako enaka $\boldsymbol{\theta} = \boldsymbol{A}\boldsymbol{x}$. Naloga FreeViz-a je poiskati dobro projekcijsko matriko $\boldsymbol{A}$.

Kaj pomeni dobro matriko? Določiti moramo funkcijo, ki jo želimo optimizirati. Za razliko od MDSa tokrat ne bomo definirali energije, temveč silo (torej v resnici ne definiramo funkcije, ki jo želimo optimizirati, temveč njen gradient). Točki, ki ustrezata primeroma iz istega razreda, se bosta privlačili, točke, ki ustrezajo primerom iz različnih razredov, se bodo odbijale. Za razliko od tega, česar smo navajeni iz fizike, bodo privlačne sile naraščale z razdaljo. To je smiselno zato, ker je točke, ki so izgubljene nekje daleč od svojih, težko privleči mimo vseh točk, ki so jim na poti, zato jih bomo vlekli z veliko silo. Po drugi strani pa dveh točk iz istega razreda, ki so že prišle blizu ena drugi, nima smisla tlačiti še bližje. Z odbojnimi silami je ravno nasprotno: točke različnih razredov, ki so si blizu skupaj želimo na vsak način spraviti narazen. Točk različnih razredov, ki so že daleč narazen, pa nima smisla še bolj odbijati. Zato bodo odbojne sile z razdaljo padale.

Naj bo torej $\boldsymbol{F}_{ij}$ sila, s katero projiciran primer j deluje na točko i; definiramo jo lahko recimo, tako, da za točke istega razreda narašča in za točke različnih razredov pada pre-

mosorazmerno z razdaljo (po želji pa lahko izberemo tudi kvadrat ali kako drugo primerno funkcijo razdalje). Naj bo $\boldsymbol{F}_i = \sum_{j \neq i} \boldsymbol{F}_{ij}$ vsota vseh sil na točko i. Sprememba potencialna energija projicirane točke je določena s silo in spremembo lege točke:

$$dE_i = -\boldsymbol{F}_{\boldsymbol{i}}^{\top} \boldsymbol{\theta}^{(\boldsymbol{i})},$$

sprememba energije celotnega sistema pa

$$dE = -\sum_i \boldsymbol{F}_{\boldsymbol{i}}^{\top} \boldsymbol{\theta}^{(\boldsymbol{i})},$$

Naše točke, primeri, so fiksni, spreminja pa se projekcijska matrika $\boldsymbol{A}$. Ker je projekcija primerov določena z $\boldsymbol{\theta}^{(i)} = \boldsymbol{A}\boldsymbol{x}^{(i)}$, so spremembe lege v projekcijski ravnini plod sprememb v projekcijski matriki:

$$\boldsymbol{d\theta}^{(i)} = \boldsymbol{dA}\boldsymbol{x}^{(i)}$$

Ko se spreminja projekcijska matrika, se spreminja potencialna energija sistema:

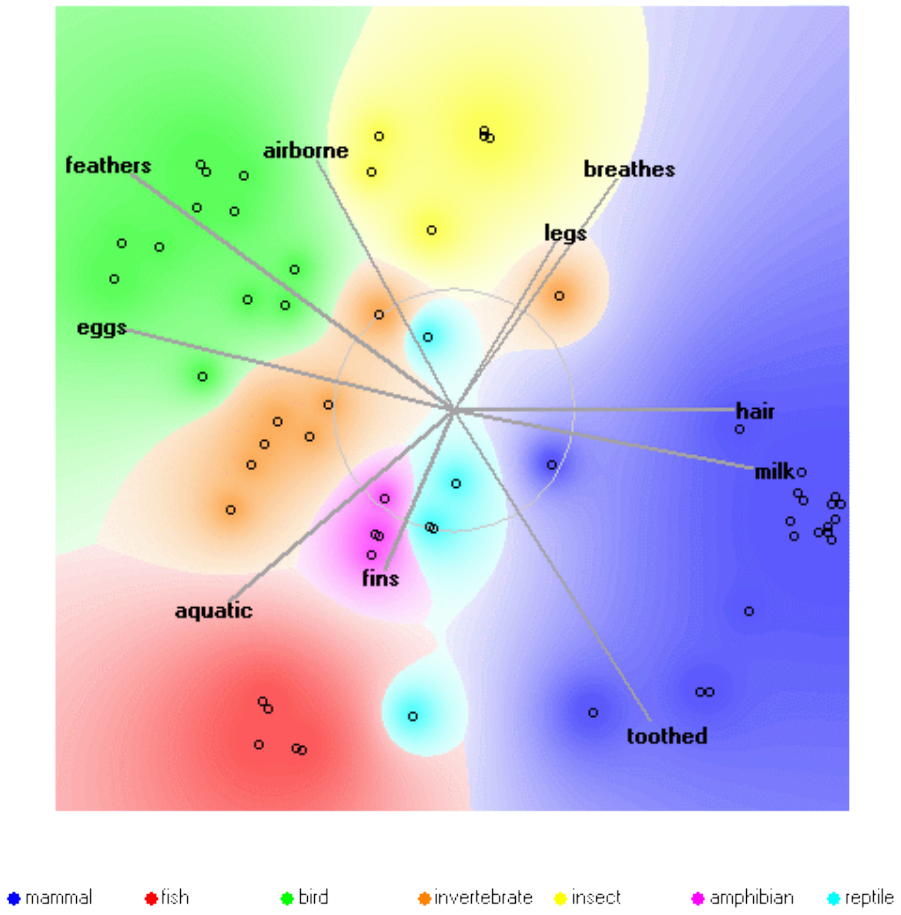
$$\begin{aligned} dE &= -\sum_i \boldsymbol{F}_i^{\top} \boldsymbol{d\theta}^{(i)} \\ &= -\sum_i \boldsymbol{F}_i^{\top} \boldsymbol{dA}\boldsymbol{x}^{(i)} \end{aligned}$$

Iščemo gradient, ki nam pove, kako sprememba projekcije i-tega primera vpliva na spremembe energije sistema. Ta znaša $\boldsymbol{F}_i x^{(i)\top}$, oziroma združeni gradient, ki pove, kako sprememba projekcije vseh primerov vpliva na skupno energijo sistema:

$$\begin{aligned} \frac{\partial E}{\partial \boldsymbol{A}} &= \sum_i \boldsymbol{F}_i x^{(i)\top} \\ &= \boldsymbol{F}\boldsymbol{X}, \end{aligned} \tag{3.14}$$

Algoritem FreeViz prične z neko naključno matriko $\boldsymbol{A}$ in potem v vsakem koraku izračuna sile na točke v projekcijski ravnini in gradiente ter projekcijo matriko ustrezno spremeni. Postopek ponavlja do konvergence.

Rezultat takšne projekcije na podatkih o živalih vidimo na sliki 3.5. Iz nje lahko, če znamo, razberemo veliko lastnosti podatkov. Na sliki so različna področja obarvana glede na prevladujočo živalsko vrsto - sesalci so modri, ribe rdeče in tako naprej. Osi ustrezajo atributom; smer osi nakazuje regijo, živalsko vrsto, ki jo sugerira visoka vrednost tega atributa. Vsi atributi, razen števila nog, so binarni, torej v tem primeru recimo dlake in zobje kažejo, da je žival verjetno sesalec. Iz osi torej razberemo povezavo med atributi in razredi, vidimo, katera živalska vrsta je značilna za posamezno lastnost, oziroma, katere lastnosti so značilne za posamezno živalsko vrsto.



Slika 3.5: Projekcija FreeViz iz podatkov o živalih

FreeViz lahko beremo – podobno kot MDS – kot zemljevid. Ptiči in žuželke so si med seboj podobne; oboji letijo, zato v njuno smer kaže atribut *airborne*. Prav tako so si podobne ribe in dvoživke.

Vidimo lahko tudi podobnosti med atributi: kosmate živali dajejo mleko; živali, ki letajo, imajo perje, živali, ki plavajo, plavuti in živali, ki dihajo, noge. Obenem razberemo tudi glavne osi, po katerih lahko ločimo živali: žival bodisi daje mleko, bodisi vali jajca (vodoravna os) in je vodna žival s plavutmi ali pa dihajoča žival z nogami.

Osi, ki predstavljajo atribute, so različno dolge; čim pomembnejši je atribut, tem daljša je pripadajoča os. Manj pomembni atributi, katerih osi bi bile krajše od kroga v sredini, zaradi preglednosti niso narisani.

Končno, FreeViz je mogoče uporabljati tudi za uvrščanje primerov. Nov primer, katerega vrednosti atributov poznamo, razreda pa ne, lahko projeciramo in preverimo, v katero regijo je padel. Če v modro, je najbrž sesalec, če v zeleno, ptič...

Ob uporabi FreeViza se moramo zavedati, da gre le za projekcijo v dvodimenzionalni prostor. Vse, kar vidimo, ni nujno resnično. Določeni atributi so morda tam, kjer so, samo zato, ker nekje pač morajo biti. Če uporabljamo FreeViz za iskanje vzorcev, moramo z njim dobljene vzorce preveriti, po možnosti na povsem novih podatkih ali pa jih primerjati z ekspertnim predznanjem o problemu.

FreeViza ne smemo uporabiti, kadar je število atributov primerljivo številu primerov ali celo večje. V tem primeru je mogoče kar analitično poiskati neskočno število rešitev, pri katerih so vse sile enake 0, razpored točk pa je v tem primeru nesmiseln. Slabost FreeViza je tudi, da v osnovi temelji na gradientni metodi in postane pri velikem številu točk počasen. Na lokalne ekstreme pa je, kakor je videti, dokaj neobčutljiv.

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 [**?**].

4.1 Bagging

Ansambelski pristop *bagging* predlaga Leo Breiman [**?**], 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 [**?**] 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 [**?**]. 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*) [**?**] 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 AdaBost

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 [**?**] 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

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

Poglavje 5

Mere za ocenjevanje uspešnosti klasifikatorjev

Naslov tega poglavja je presplošen: omejili se bomo namreč na klasifikacijo. Še več, celotno poglavje smo v ta del predavanj umestili, da predstavimo eno samo tehniko: površino pod krivuljo ROC. A da do nje pridemo in se poglobimo v to, kaj ta površina sploh pomeni, moramo uvesti nekaj drugih, pomembnih mer za ocenjevanje uspešnosti klasifikatorjev. V tem poglavju tudi ne bomo govorili o postopkih za ocenjevanje uspešnosti, saj te (prečno preverjanje, izloči enega, in testiranje po metodi stremena) že poznamo.

5.1 Tabela napak

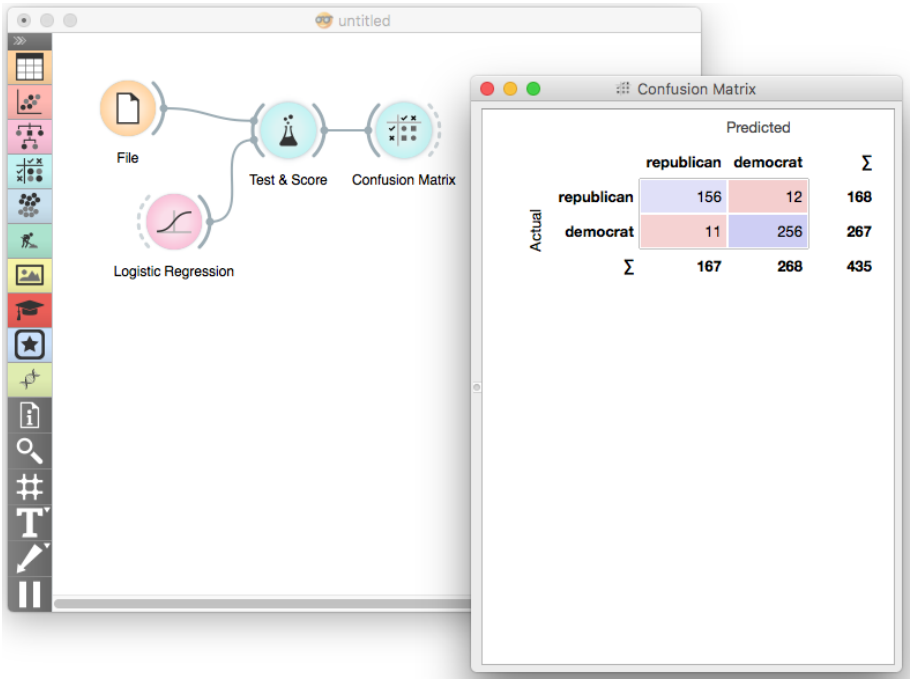
Predpostavimo, da je naš klasifikacijski problem binaren, torej da imamo dva razreda. Med njima izberemo pozitivni (ciljni) razred in negativni razred. Pozitivni razred je navadno razred, ki je povezan z študijem pojava, za katerega smo oblikovali učne podatke. Na primer, v medicini, kjer primerjamo zdrave osebke in osebke z določeno diagnozo bi bili slednji v ciljnem (pozitivnem) razredu. Ali pa pri industrijski aplikaciji, kjer želimo na podlagi meritev identificirati nezanesljive izdelke, so ti slednji pozitivni primeri oziroma pripadajo ciljnemu razredu.

Klasifikator, ki smo ga razvili na neki učni množici, bo na testni množici lahko pravilno razvrstil pozitiven primer (zadetek, angl. *true positive*), ali pa ga bo napačno razvrstil v negativni razred (pogrešek, angl. *false positive*). Za neciljni, negativni primer, bo lahko pravilno napovedal njegov razred (pravilna zavrnitev, angl. *true negative*) ali pa dal napačno napoved (napačno pozitiven, angl. *false positive*). Število takih pravilnih oziroma nepravilnih napovedi na testni množici lahko zberemo in prikažemo v tabeli napak, ki jo konceptualno kaže tabela 5.1, njen konkretni primer na podatkih o volitvah senatorjev (slavni podatkovni nabor `voting`) pa slika 5.1. Podatke za tabelo napak smo pri slednjem sestavili s pomočjo prečnega preverjanja, kjer smo tabelo napak najprej pridobili na vsaki od testnih množic posebej, na

koncu pa skupaj prikazali vsote posameznih celic za vse testne množice oziroma za celoten postopek prečnega preverjanja skupaj.

Tabela 5.1: Tabela napak (angl. *confusion matrix*).

		Dejanska vrednost		
		+	−	
Napovedana vrednost	+	True Positive (TP)	False Positive (FP)	P'
	−	False Negative (FN)	True Negative (TN)	N'
		P	N	



Slika 5.1: Shema za ocenjevanje in tabela napak v programu Orange.

Na osnovi tabele napak oziroma v njej predstavljenih števil pravilnih in napačnih napovedi lahko sestavimo nekaj mer, ki upoštevajo več celic v tabeli napak in s kombinacijo njih tvorijo smiselne ocene uspešnosti. Skupaj z samimi že opisanimi celicami tabele napak so te mere naslednje:

zadetek (TP), število primerov v ciljnem razredu, pri katerih je bil napovedan pravilen razred (angl. *true positive*),

pravilna zavrnitev (TN), število primerov negativnega razreda s točno napovedjo, (angl. *true negative*),

napačno pozitiven, lažni alarm (FP), število primerov v negativnem razredu, ki so bili napačno napovedani kot pozitivni, tudi napaka I. reda (angl. *false positive*),

pogrešek (FN), število primerov v negativnem razredu, ki so bili napačno prepoznani kot pozitivni primeri, tudi napaka II. reda (angl. *false negative*),

občutljivost, priklic (angl. *sensitivity, true positive rate, recall*), $TPR = TP/P = TP/(TP + FN)$, delež pozitivnih primerov, ki smo jih pravilno napovedali oziroma jih pravilno priklicali,

delež napačno pozitivnih, odpadek (FPR), (angl. *false positive rate, fall-out*) $FPR = FP/N = FP/(FP + TN)$, delež napačno klasificiranih negativnih primerov,

točnost (ACC), (angl. *accuracy*), $ACC = (TP + TN)/(P + N)$, delež primerov, za katere smo pravilno napovedali razred,

specifičnost , stopnja pravilne zavrnitve (SPC), (angl. *specificity, true negative rate*), $SPC = TN/N = TN/(FP + TN) = 1 - FPR$, dežel negativnih primerov, za katere smo pravilno napovedali razred,

natančnost , delež pravilnih pozitivnih (PPV), (angl. *positive predictive value, precision*) $PPV = TP/(TP + FP)$, delež pravilno napovedanih pozitivnih primerov med vsemi pozitivnimi napovedi,

delež pravilnih negativnih (NPV), (angl. *negative predictive value*), $NPV = TN/(TN + FN)$, delež pravilno napovedanih primerov med vsemi negativnimi primeri,

mera F1 (F1) (angl. *F1 score*), $F1 = 2 \times (\frac{1}{\text{recall}} + \frac{1}{\text{precision}})^{-1}$ = je harmonično povprečje priklica in natančnosti.

Nekatere med zgornjimi merami so komplementarne; z optimizacijo (višanjem) ene od mer bomo zmanjšali vrednost druge. Na primer, če želimo zvišati priklic, nam je najbolj enostavno spustiti kriterij za klasifikacijo v pozitivni razred oziroma čim več primerov iz testne množice razglasiti za pozitivne. A bomo na ta način prav gotovo zmanjšali delež pravilnih pozitivnih napovedi. Podobno je s specifičnostjo in občutljivostjo: tudi tu za povečanje občutljivosti lahko spustimo mejo, ko primer razglasimo za pozitivnega, a bomo na ta način zmanjšali delež pravilno napovedanih negativnih primerov.

Zgornji kompromisi oziroma optimizacije izbranih mer na račun drugih so seveda pri strojnem učenju povezani s tem, da v praksi ne napovedujem razredov marveč njihove verjetnosti. Uporabljamo torej verjetnostne napovedi. Pri teh se moramo odločiti za prag oziroma vrednost, nad katero primeru določimo, da pripada ciljnemu razredu.

Tabela 5.2: Rezultati meritev uspešnosti dveh radaristov, kjer so uspešnost pri prvem (leva tabela) merili pri petih različnih pogojih in pri drugem (tabela na desni) pri štirih različnih pogojih.

#	*FPR*	*TPR*
1	0.20	0.20
2	0.25	0.30
3	0.40	0.60
4	0.70	0.80
5	0.90	0.85

#	*FPR*	*TPR*
1	0.10	0.15
2	0.20	0.40
3	0.50	0.80
4	0.95	0.95

5.2 Površina pod krivuljo ROC

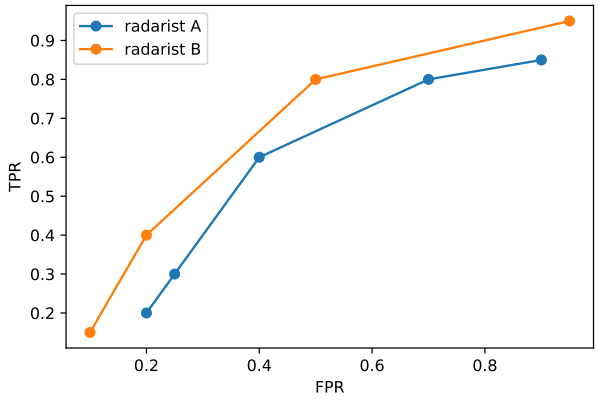
Prav posebna mera za ocenjevanje napovedne točnosti klasifikatorjev je površina pod krivuljo ROC (angl. *area under receiving operator characteristics*) in zato zasluži poseben razdelek. Mero so najprej uporabili tekom druge svetovne vojne in z njo skušali oceniti radariste ter njihovo točnost pri razlikovanju med zavezniškimi in agresorjevimi letali na radarskem zaslonu. Ocenjevali so jih v različnih pogojih (dopoldne, zvečer, po neprespani noči, ...) in ob različnih časih ter skušali, ne glede na njihovo stanje, radariste kar najbolj objektivno rangirati, od teh zelo uspešnih do drugih, ki so bolj često delali napake. Pri tem so opazili, da je uspešnost radaristov zelo odvisna od pogojev in psihofizičnega stanja, in da je posamezne meritve uspešnosti med seboj zelo težko primerjati.

Denimo, da so vsakič beležili nekaj, recimo okoli dvajset, poskusov prepoznave agresorjevih letal, in za dva radarista zabeležili priklic (*TPR*) in delež napačno pozitivnih napovedi (*FPR*) kot prikazuje tabela 5.2. Meritve lahko prikažemo v grafu (slika 5.2). Želimo, da bi *TPR* bil čim višji in *FPR* čim manjši. Meritve, ki segajo proti zgornjem levem kotu grafa so zato najbolj zaželene. Vidimo, da so rezultati radarista B boljši, saj njihova Paretto optimalna fronta prekrije Paretto optimalno fronto radarista B. Z drugimi besedami, konveksna ovojnica meritev za radarista B je levo in nad konveksno ovojnico radarista A.

Krivuljo ROC so ponovno odkrili v 1970-ih in jo najprej uporabljali v radiologiji [**?**], kasneje pa na celotnem področju medicine [**?**]. Da bi jo potem znova odkrili na prelomu stoletja in pričeli množično uporabljati na celotnem področju statistike in odkrivanja znanj iz podatkov.

Na enostavnem primeru si oglejmo si najprej konstrukcijo krivulje ROC, s katero opišemo rezultate testiranja izbranega postopka za strojno učenje. Denimo, da smo že zgradili model uvrščanja in bi radi ocenili njegovo točnost na testnih primerih. Naš klasifikacijski problem je dvovrednostni, napovedni model pa zna napovedati verjetnosti ciljnega razreda $p(y = 1|\boldsymbol{x})$, kjer je $\boldsymbol{x}$ atributno opisan primer.

Točnost uvrščanja primerov s tabele 5.3 je odvisna od praga verjetnosti T, nad katerimi bomo razglasili, da primeri pripadajo pozitivnemu razredu. Pri tem bomo opazovali *TPR*,



Slika 5.2: Krivulja ROC za rezultate meritev radaristov A in B iz tabele 5.2.

Tabela 5.3: Testni primeri z dejanskim razredom in verjetnostjo za razred “1”, kot jo je predlagal napovedni model, katerega točnost ocenjujemo. Primeri so urejeni skladno z napovedano verjetnostjo.

y	$p(y = 1\|X)$
1	0.89
1	0.80
1	0.80
0	0.80
1	0.63
0	0.33
1	0.33
0	0.10
0	0.10
0	0.10

to je delež pravilno napovedanih pozitivnih primerov med vsemi dejansko pozitivnimi (angl. *true positive rate*), in FPR, delež negativnih primerov, za katere je bila napoved napačna (angl. *false positive rate*). Izrazimo ti dve meri še z notacijo iz prejšnjega razdelka:

$$TPR = \frac{TP}{P}$$

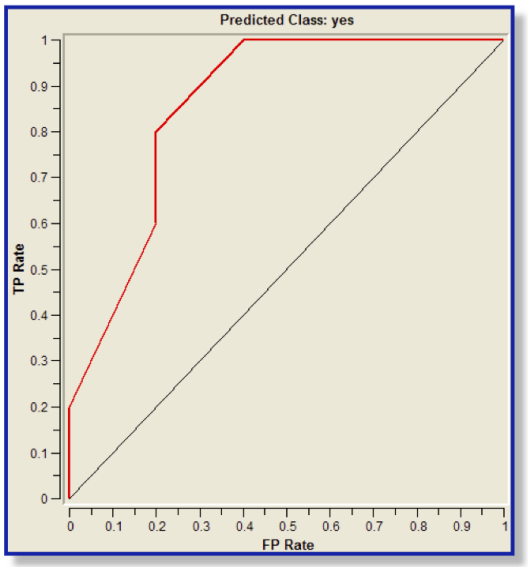
$$FPR = \frac{FP}{N}$$

Vzemimo, da vse primere razglasimo kot pozitivne ($T < 0.10$). Na ta način smo pravilno "ulovili" vse dejansko pozitivne primere in s tem dosegli najvišji delež pravilno napovedanih pozitivnih primerov med vsemi dejansko pozitivni primeri ($TP = P$, $TPR = 1.0$). Tudi FPR je s tem maksimalni, 1.0, saj smo vse negativne primere napačno uvrstili med pozitivne.

Drugače pa je, če vse primere iz testne razglasimo kot negativne ($T > 0.89$). Primerov, ki bi jih razglasili za pozitivne, ni, zato $TPR = 0$ in $FPR = 0$.

Vse ostale vrednosti TPR in FPR bodo, za prag, ki ga izberemo med zgornjima dvema skrajnima mejama, nekje med 0 in 1. Najbolj zaželen prag bi bil tam, kjer bi vse razvrstitve v pozitivni razred bile pravilne in nobena ne bi bila nepravilna. V tej točki bi bil $TPR = 1$ in $FPR = 0$. Ali za naš primer taka vrednost praga obstaja?

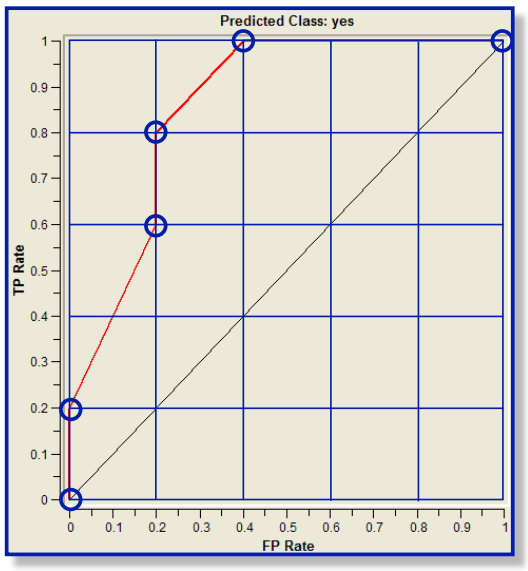
Preskusimo sedaj vse možne mejne vrednosti T, kjer bi se nam vrednosti TPR in FPR lahko spremenile. Za naše podatke imamo štiri intervale, kjer lahko poiščemo take meje. Med 0.89 in 0.80, med 0.80 in 0.63, med 0.63 in 0.33, in med 0.33 in 0.10. Kje v teh intervalih bo dejansko naša mejna vrednost ni pomembno. Za te štiri meje izračunajmo sedaj vrednosti TPR in FPR in te vnesimo v graf (slika 5.3).



Slika 5.3: Krivulja ROC za podatke s tabele 5.3.

Obstaja tudi hitrejši način izrisa krivulja *ROC*, ki zahteva ureditev primerov po napovedani verjetnosti ciljnega razreda (kot smo to storili v tabeli 5.3). Postopek je naslednji:

1. Izrišemo graf z mrežo $1/N$ horizontalno in $1/P$ vertikalno.
2. Uredimo primere v seznam L po padajoči verjetnosti napovedi v ciljni razred.
3. Pričnimo v točki (0,0).
4. Izberimo in iz seznama L izključimo primere z najvišjo napovedano verjetnostjo. Naj ti primeri vključujejo n_+ primerov iz pozitivnega razreda, in n_- primerov iz negativnega razreda. V mreži grafa se premaknimo n_+ razdelkov navzgor in n_- razdelkov desno.
5. Če L ni prazen, skoči na korak 4, sicer končaj.



Slika 5.4: Mreža za izris krivulje ROC in sprehod po njenih točkah skladno s primeri s tabele 5.3.

Opazimo, da zgoraj opisan algoritem pravilno razpozna obe meri *TPR* in *FPR* za vse možne vrednosti praga verjetnosti (slika 5.4).

Površino pod krivuljo ROC označimo z *AUC*. Pri naključnih napovedih pričakujemo, da bodo v obhodu zgornjega algoritma šli približno po diagonali grafi in bo $AUC = 0.5$. To je tudi spodnja meja za to mero in je karakteristična za neuspešen napovedni model. Dejansko je meja za uspešne modele tipično pri $AUC = 0.75$, modeli, ki imajo visoko napovedno točnost, pa imajo *AUC* nekje nad 0.9 [**?**].

Površina pod ROC krivuljo pa ima še eno zanimivo lastnost. Če opazujemo sprehod po mreži za izris ROC krivulje, opazimo, da površina pod vsakem segmentom ravno ustreza

številu napačno razvrščenih primerov, katerih verjetnost pozitivnega razreda je nad določeno mejo. Iz tega tudi sledi, da lahko *AUC* izračunamo z enkratnim sprehodom skozi seznam primerov, ki je urejen glede na napovedano verjetnost ciljnega razreda. Mera *AUC* ustreza verjetnosti, da za naključno izbrani primer $\boldsymbol{x}^+$ iz pozitivnega razreda in naključno izbrani primer $\boldsymbol{x}^-$ iz negativnega razreda velja $p(y = 1|\boldsymbol{x}^+) > p(y = 1|\boldsymbol{x}^-)$.

ROC krivuljo lahko rišemo in površino pod njo računamo za binarne klasifikatorje. Ko je razredov več, je potrebno večrazredni problem pretvoriti v dvorazrednega. Mero *AUC* lahko potem računamo tako, da vsak element vektorja napovedanih verjetnosti razredov smatramo za svojo napoved, kjer je pozitivni razred samo tisti z ustrezno vrednostjo razreda in so vse ostale verjetnosti povezane z negativnim razredom. Ta pristop imenujemo mikro povprečenje (angl. *micro-averaging*). Pri drugačnem pristopu pa obravnavamo večrazredno klasifikacijo kot sprego binarnih klasifikatorjev tip eden-proti-ostalim, ter za vsakega posebej izračunamo *AUC* oziroma krivulje ROC ter te povprečimo (t.im. makro povprečenje, angl. *macro-averagining*).

Poglavje 6

Nevronske mreže

V poglavju pregledamo arhitekturo in izpeljavo gradientov kriterijske funkcije za uteži najpreprostejšega model umetne nevronske mreže, tako imenovane usmerjene (angl. *feed forward*) nevronske mreže s polnimi povezavami med enotami sosednjih nivojev. Poleg strukture nevronske mreže uteži popolnoma določajo model nevronske mreže, izračun njihovih gradientov pa pri podanih učnih podatkih omogoča iskanje vrednosti uteži z gradientnim sestopom. Kot pri drugih modelih, ki smo jih pridobili na ta način, je tudi tu namen zgraditi mrežo, ki se čim bolj prilega učnim podatkom.

6.1 Arhitektura in notacija

Nevronska mreža je urejena mreža nevronov oziroma enot. Povezave med enotami označujejo funkcijske odvisnosti. Primer arhitekture mreže kaže slika 6.1. Vhod mreže sprejme atributno opisan primer oziroma vektor njegovih vrednosti atributov $\boldsymbol{x}$. Na izhodu mreža odda vrednosti izhodnih spremenljivk oziroma razredov. V strojnem učenju pravimo, da mreža za vhodni vektor $\boldsymbol{x}$ odda vektor napovedi $\boldsymbol{y}$. Vektor $\boldsymbol{y}$ je lahko eno ali več-dimenzionalen. Za regresijski model bo zadostovala ena enota, za klasifikacijo pa tipično izberemo toliko izhodnih enot, kolikor imamo razredov. Pri klasifikaciji enote poročajo o verjetnosti, da vhodnih primer $\boldsymbol{x}$ pripada določenemu razredu.

Vrednostim nevronov pravimo aktivacije. Aktivacije vhodnega nivoja mreže (L_1) nastavimo atributnim vrednostim primera, za katerega želimo izračunati vrednost razreda. Mreža z izračunom aktivacije enot njenem zadnjem nivoju (L_K) izračuna oceno vrednosti razrednih spremenljivk pripadajočega primera iz vhoda mreže. V preprosti nevronski mreži, ki jo obravnava to poglavje, so vse enote predhodnega nivoja povezane z vsemi enotami naslednjega nivoja.

Aktivacijo enote na nivoju L_i izračunamo kot uteženo vsoto vhodnih aktivacij nivoja L_{i-1}, ki jo transformiramo z neko nelinearno aktivacijsko funkcijo. Da bi linearne kombinacije vhodov lahko vključevale še konstantni člen, vsem nivojem nevronske mreže z izjemo zadnjega

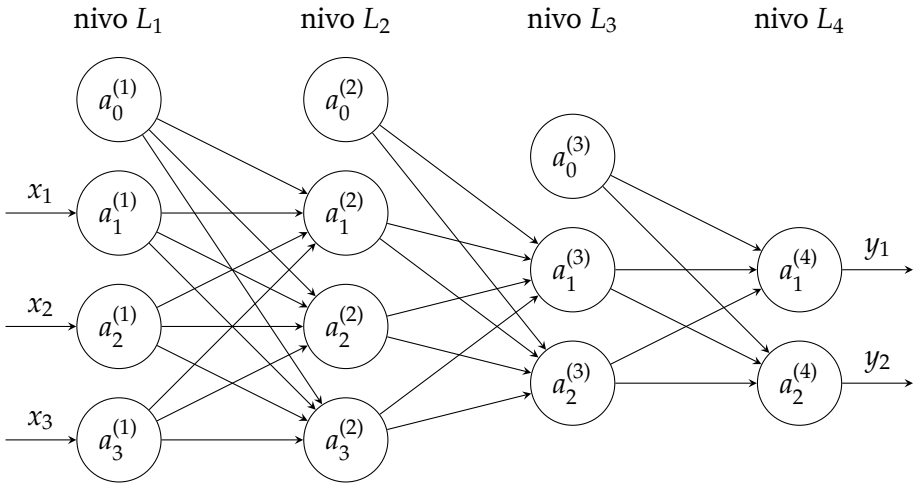
nivoja dodamo konstanten nevron $a_0^{(i)}$ ($i = \{1, 2, \ldots, K\}$), katerega aktivacija je enaka 1.

Aktivacijo j-tega nevrona na l-tem nivoju nevronske mreže izračunamo kot:

$$z_j^{(l)} = \sum_{k=0}^{n_{l-1}} w_{kj}^{(l)} \, a_k^{(l-1)} \tag{6.1}$$

$$a_j^{(l)} = \sigma(z_j^{(l)}), \tag{6.2}$$

kjer je n_k število enot nivoja L_k in σ aktivacijska funkcija. Dve tipični in pogosto uporabljani aktivacijski funkciji sta logistična funkcija, kjer je $\sigma(z) = 1/(1+\exp(-z))$ in popravljena linearna funkcija ali ReLU (angl. *rectified linear unit*), kjer je $\sigma(z) = \max(0, z)$. Aktivacijske funkcije morajo biti zvezne in odsekoma odvedljive.

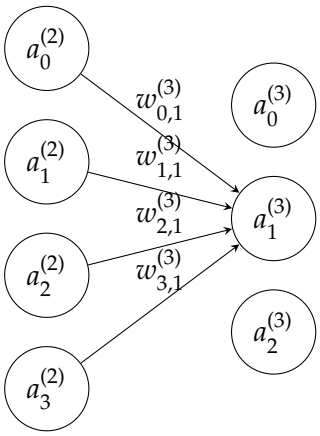


Slika 6.1: Primer štirinivojske nevronske mreže. Prvi nivo je nivo vhodnih podatkov, kjer aktivacijo nevronov nastavimo na vrednosti atributov ($a_i^{(1)} = x_i$). Sledita dva skrita nivoja, L_2 in L_3 in končni nivo, kjer so aktivacije nevronov vrednosti izračunanih ocen za razredne spremenljivke. V splošnem privzamemo, da imajo aktivacije $a_0^{(*)}$ na poljubnem nivoju konstantno vrednost 1.

V implementacijah nevronske mreže uteži na posameznem nivoju predstavimo z matrikami in aktivacije na posameznem nivoju z vektorji. Aktivacije $\boldsymbol{a}^{(l)}$ nivoja l tako izračunamo kot funkcijo skalarnega produkta uteži in aktivacij nivoja $l-1$:

$$\boldsymbol{a}^{(l)} = \sigma(\boldsymbol{a}^{(l-1)\mathsf{T}} \cdot W^{(l)}) \tag{6.3}$$

Primer: aktivacije drugega nivoja dela nevronske mreže s slike 6.2 naj zavzamejo vrednost



Slika 6.2: Primer sosednjih nivojev v nevronski mreži z oznakami uteži. Oznaka nivoja uteži ustreza oznaki nivoja, katerega aktivacijo računamo. Podobno je z vrstnim redom indeksov; prvi indeks je indeks nevrona, katerega vrednost je uporabljena pri računanju aktivacije, drugi pa indeks nevrona, za katerega aktivacijo računamo, drugi indeks pa indeks nevrona. Uteži, predstavljene na sliki, so del matrike uteži $W^{(3)}$ in predstavljajo njeno prvo vrstico.

$\boldsymbol{a}^{(2)} = (1\ 0\ 1\ 2)^\intercal$. Uteži tretjega nivoja predstavimo z matriko $W^{(l)}$, katere primer je:

$$W^{(l)} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

Utežena vsota aktivacij, ki jo potrebujemo za izračun aktivacij tretjega nivoja nevronske mreže je tako:

$$\begin{pmatrix} 1 & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 \end{pmatrix}$$

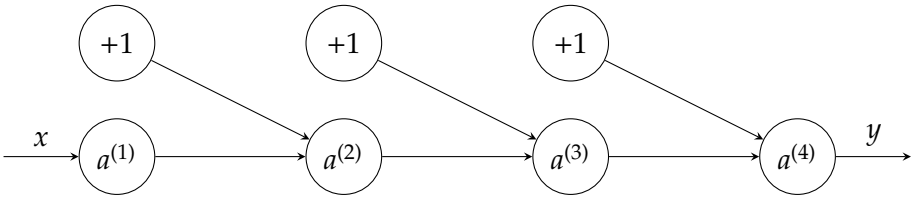
6.2 Učenje

Dan je nabor atributno opisanih primerov $\mathbf{X} \in \mathbb{R}^{m,n}$ s pripadajočo vrednostjo razreda. Nevronske mreže lahko obravnavajo primere z več razrednimi spremenljivkami, katerih vrednosti napovejo kot vrednost aktivacije enot v zadnjem nivoju nevronske mreže. Pri učenju želimo poiskati vrednosti uteži tako, da se napovedi mreže čimbolj ujemajo z dejanskimi vrednostmi razredov na učni množici primerov. Za učenje moramo tako določiti kriterijsko funkcijo, katere vrednost optimiziramo, in izračunati gradient kriterijske funkcije za vsako utež v mreži. Za optimizacijo vrednosti uteži uporabimo gradientni sestop. Slednjega že dobro poznamo, zato se bomo v tem poglavju ukvarjali samo z izračunom gradienta.

6.2.1 Enodimenzionalna nevronska mreža

Privzemimo, da imamo učno množico primerov, ki so opisani z enim samim atributom in kjer imamo en sam, zvezni razred. Najbolj enostavna nevronska mreža za tak primer vsebuje poleg enote s konstantno aktivacijo ((angl. *bias*)) eno samo aktivno aktivacijsko enoto na vsakem od nivojev. Primer take mreže je podan na sliki 6.2.1). Parametri te mreže so uteži $w_0^{(2)}$, $w_1^{(2)}$, $w_0^{(3)}$, $w_1^{(3)}$, $w_0^{(4)}$, $w_1^{(4)}$, kjer so z $w_0^{(*)}$ označene uteži, ki vodijo iz enot s konstantno aktivacijo in z $w_1^{(*)}$ označene uteži, ki vodijo iz enos s spremenljivo vrednostjo aktivacije. Za začetek si zamislimo, da imamo samo en učni primer (x, y). Kriterijska funkcija, ki jo optimiziramo oziroma za katero iščemo primerne parametre naj bo enaka kvadratu napake,

$$J\left(w_0^{(2)}, w_1^{(2)}, w_0^{(3)}, w_1^{(3)}, w_0^{(4)}, w_1^{(4)}\right) = \left(a^{(4)} - y\right)^2. \tag{6.4}$$



Slika 6.3: Nevronska mreža z enim samim aktivnim nevronom na vsakem od nivoju. Taka, poenostavljena, nam služi za pomoč pri izpeljavi gradientov uteži.

Iščemo gradiente kriterijske funkcije oziroma parcialne odvode kriterijske funkcije po vseh šestih utežeh. Najbolj enostavno nam je začeti čisto zadaj, pri zadnjem nivoju, ki ga označimo imenujmo nivo L. V našem primeru je $L = 4$. Odvisnost kriterijske funkcije od spremenljivk zadnjega nivoja so prikazane na sliki 6.4, izrazimo pa jih kot:

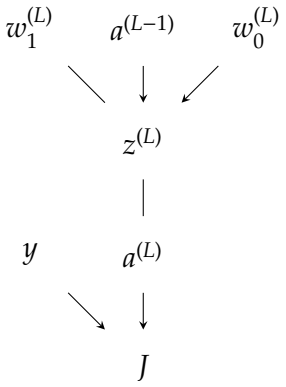
$$J = (a^{(L)} - y)^2, \tag{6.5}$$

$$a^{(L)} = \sigma(z^{(L)}), \tag{6.6}$$

$$z^{(L)} = w_1^{(L)} a^{(L-1)} + w_0^{(L)}. \tag{6.7}$$

Na tem, zadnjem nivoju, imamo dve uteži, $w_0^{(L)}$ in $w_1^{(L)}$. Izračunajmo gradient kriterijske funkcije po teh dveh utežeh. Najprej za utež $w_1^{(L)}$. Spomnimo so, da je kriterijska funkcija odvisna od aktivacije na nivoju L, ta od utežene vsote, in slednja od aktivacije nivoja $L-1$. Majhna sprememba uteži $w_1^{(L)}$ povzroči spremembo $z^{(L)}$, ta spremembo $a^{(L)}$, in slednja spremembo kriterijske funkcije J. Parcialni odvod kriterijske funkcije po uteži $w_1^{(L)}$ zato zračunamo z verižnim pravilom:

$$\frac{\partial J}{\partial w_1^{(L)}} = \frac{\partial z^{(L)}}{\partial w_1^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial J}{\partial a^{(L)}}, \tag{6.8}$$



Slika 6.4: Shematski prikaz odvisnosti kriterijske funkcije za zadnji nivo nevronske mreže.

kjer lahko naštete parcialne odvode dobimo iz zgoraj zapisanih funkcijskih odvisnosti:

$$\frac{\partial z^{(L)}}{\partial w_1^{(L)}} = a^{(L-1)}, \tag{6.9}$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)}), \tag{6.10}$$

$$\frac{\partial J}{\partial a^{(L)}} = 2(a^{(L)} - y). \tag{6.11}$$

Tu je σ' odvod aktivacijske funkcije po njeni edini spremenljivki. V primeru logistične aktivacijske funkcije ta odvod že poznamo:

$$\sigma'(z^{(L)}) = \sigma(z^{(L)})(1 - \sigma(z^{(L)})) = a^{(L)}(1 - a^{(L)}) \tag{6.12}$$

Izračun parcialnega odvoda kriterijske funkcije po $w_1^{(L)}$ je bil enostaven. Kaj pa parcialni odvod po $w_0^{(L)}$? Dobimo ga enako kot zgoraj, drugačen je le odvod utežene vsote aktivacij na vhodu enote, ki je sedaj:

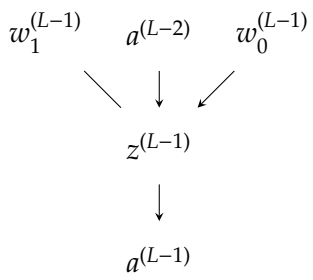
$$\frac{\partial z^{(L)}}{\partial w_0^{(L)}} = 1. \tag{6.13}$$

Do tu je šlo vse enostavno. Kaj pa predzadnji nivo. Tega (glej sliko 6.5) s kriterijsko funkcijo povezuje aktivacija enote $a^{(L-1)}$. Kako njena majhna sprememba učinkuje na spremembo kriterijske funkcije?

$$\frac{\partial J}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial J}{\partial a^{(L)}} \tag{6.14}$$

Zadnja dva parcialna odvoda smo že izračunali, manjka nam samo še prvi:

$$\frac{\partial z^{(L)}}{\partial a^{(L-1)}} = w_1^{(L)} \tag{6.15}$$



Slika 6.5: Shematski prikaz odvisnosti aktivacije zadnjega nivoja od elementov predzadnjega nivoja nevronske mreže.

Vse imamo pripravljeno za izračun gradientov za uteži na predzadnjem nivoju. Odvisnosti teh od aktivacije $a^{(L-1)}$ prikazuje slika 6.5, za to aktivacijo pa parcialni odvod že poznamo. Tudi tu parcialni odvod kriterijske funkcije po uteži $w_1^{(L-1)}$ dobimo z verižnim pravilom:

$$\frac{\partial J}{\partial w_1^{(L-1)}} = \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial J}{\partial a^{(L-1)}} \tag{6.16}$$

Zadnji parcialni odvod že poznamo, izračunati moramo le še prva dva:

$$\frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} = a^{(L-2)} \tag{6.17}$$

$$\frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} = \sigma'(z^{(L-1)}) = a^{(L-1)}(1 - a^{(L-1)}) \tag{6.18}$$

Podobno, z uporabo verižnega pravila, izpeljemo še gradiente za uteži preostalih nivojev ter pri tem uporabimo gradiente aktivacij, ki smo jih že izračunali. Izračun gradientov kriterijske funkcije po utežeh v nevronski mreži torej poteka od konca proti začetku. Postopku verižnega pravila v jeziku nevronskih mrež pravimo vzvratno razširanje napake oziroma (angl. *back propagation*).

Izpišimo torej splošne enačbe za izračun gradientov po tem postopku, in pri tem nek z l

označimo poljudni nivo nevronske mreže:

$$\frac{\partial J}{\partial w_1^{(l)}} = \frac{\partial z^{(l)}}{\partial w_1^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial J}{\partial a^{(l)}} \tag{6.19}$$

$$\frac{\partial J}{\partial a^{(l)}} = \begin{cases} 2(a^{(L)} - y), & l = L \\ \frac{\partial z^{(l+1)}}{\partial a^{(l)}} \frac{\partial a^{(l+1)}}{\partial z^{(l+1)}} \frac{\partial J}{\partial a^{(l+1)}}, & l = 1, 2, \ldots, L-1 \end{cases} \tag{6.20}$$

$$\frac{\partial z^{(l)}}{\partial w_1^{(l)}} = a^{(l-1)} \tag{6.21}$$

$$\frac{\partial a^{(l)}}{\partial z^{(l)}} = \sigma'(z^{(l)}) \tag{6.22}$$

Podobno kot zgoraj lahko zapišemo tudi parcialne odvode za $w_0^{(l)}$, le da je tu parcialni odvod $\frac{\partial z^{(l)}}{\partial w_0^{(l)}}$ enak 1.

Zgoraj smo privzeli, da imamo samo en učni primer. Za množico primerov je kriterijska funkcija lahko:

$$J = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2 \tag{6.23}$$

kjer smo tokrat z $\hat{y}^{(i)}$ zapisali ocenjeno vrednost razredne spremenljivke za i-ti primer $x^{(i)}$ in z $y^{(i)}$ njeno pravo vrednost. Skladno s to kriterijsko funkcijo se pri izračunu gradientov spremeni le $\frac{\partial J}{\partial a^{(L)}}$, ki je sedaj enak povprečni vrednosti napake:

$$\frac{\partial J}{\partial a^{(L)}} = \frac{1}{m} \sum_{i=1}^{m} \left(a^{(L)}(x^{(i)}) - y^{(i)}\right), \tag{6.24}$$

kjer smo z $a^{(L)}(x^{(i)})$ označili vrednost aktivacije enote na zadnjem nivoju nevronske mreže, ki jo ta zavzame pri vhodnem primeru $x^{(i)}$.

Vsota kvadratov napake je samo ena od možnih kriterijskih funkcij. Pravzaprav lahko uporabimo katerokoli kriterijsko funkcijo, ki pa mora biti odvedljiva po aktivaciji zadnjega nevrona.

6.2.2 Splošna polno-povezana nevronska mreža

Zgornji primer "enodimenzionalne" nevronske mreže razširimo na primer, kjer imamo na vsakem nivoju lahko več aktivnih enot. Utežena vsota aktivacij na vhodu enote j nivoja l bo enaka:

$$z_j^{(l+1)} = \sum_{i=0}^{n_l} w_{ij} a_i^{(l)} \tag{6.25}$$

To pa je tudi edina sprememba, ki jo moramo upoštevati pri izračunu novih parcialnih odvodov. Odvod kriterijske funkcije po dani uteži ostane pravzaprav enak kot prej (z izjemo nekaj dodanih indeksov):

$$\frac{\partial J}{\partial w_{kj}^{(l)}} = \frac{\partial z_j^{(l)}}{\partial w_{kj}^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial J}{\partial a_j^{(l)}}, \tag{6.26}$$

spremeni se le odvod kriterijske funkcije po aktivaciji a_k^l, saj ta sedaj vpliva na kriterijsko funkcijo po različnih poteh, ki vodijo preko enot nivoja $l+1$. Tudi tu uporabimo verižno pravilo, a upoštevamo vse poti vplivov aktivacije a_k^l:

$$\frac{\partial J}{\partial a_k^{(l)}} = \sum_{j=0}^{n_l} \frac{\partial z_j^{(l+1)}}{\partial a_k^{(l)}} \frac{\partial a_j^{(l+1)}}{\partial z_j^{(l+1)}} \frac{\partial J}{\partial a_j^{(l+1)}} \tag{6.27}$$

Odvode izračunamo enako enostavno kot v "enodimenzionalni" mreži, le da imamo tu opravka z vsoto verig odvodov.

6.2.3 Vse skupaj, matrično in vektorsko

Zgornje izpeljave so zaradi preprostosti funkcij, ki gradijo umetne nevronske mreže, precej enostavne. A se hitro zapleteš z indeksi. Veliko preprosteje je vse te enačbe zapisati v vektorsko-matrični obliki. Pričnimo z učno množico $\mathbf{X} \in \mathbb{R}^{m,n}$ in za vse te naenkrat izračunajmo aktivacije enot nevronske mreže. Podatkom $\mathbf{X}$bomo dodali kolono enic, tako spremenjene podatke $\mathbf{X}'$ pa vložili v enote prvega nivoja nevronske mreže:

$$\underset{m\times n_1}{\mathbf{A}^{(1)}} = \underset{m\times(n+1)}{\mathbf{X}'} \tag{6.28}$$

Od tu dalje aktivacije izračunamo po naslednji enačbi:

$$\underset{m\times n_{l+1}}{\mathbf{Z}^{(l+1)}} = \underset{m\times n_l}{\mathbf{A}^{(l)}} \times \underset{n_l\times n_{l+1}}{\mathbf{W}^{(l+1)}}, \tag{6.29}$$

$$\underset{m\times n_{l+1}}{\mathbf{A}^{(l+1)}} = \sigma\Big(\underset{m\times n_{l+1}}{\mathbf{Z}^{(l+1)}}\Big) \tag{6.30}$$

kjer skalarno aktivacijsko funkcijo σ apliciramo na vsakega od elementov matrike v argumentu posebej.

S to, matrično, notacijo zapišimo tudi izračun gradientov. Pričnemo z gradientno matriko $\mathbf{D}^{(L)}$ za uteži zadnjega nivoja:

$$\underset{m\times n_L}{\mathbf{d}^{(L)}} = \Big(\underset{m\times n_L}{\mathbf{A}^{(L)}} - \underset{m\times n_L}{\mathbf{Y}}\Big) \circ \underset{m\times n_L}{\mathbf{A}^{(L)}} \circ \Big(1 - \underset{m\times n_L}{\mathbf{A}^{(L)}}\Big) \tag{6.31}$$

$$\underset{n_{L-1}\times n_L}{\mathbf{D}^{(L)}} = \frac{1}{m} \Big(\underset{n_{L-1}\times m}{\mathbf{A}^{(L-1)}}\Big)^{\mathsf{T}} \times \underset{m\times n_L}{\mathbf{d}^{(L)}} \tag{6.32}$$

kjer je $\circ$ oznaka za Hadamardov produkt, oziroma produkt po elementih dveh matrik. Nadaljujemo s izračunom gradientnih matrik $\mathbf{D}^{(L)}$ za uteži preostalih nivojev:

$$\underset{m\times n_l}{\mathbf{d}^{(l)}} = \left(\underset{m\times n_{l+1}}{\mathbf{d}^{(l+1)}} \times (\underset{n_{l+1}\times n_L}{\mathbf{W}^{(l+1)}})^\intercal\right) \circ \underset{m\times n_L}{\mathbf{A}^{(l)}} \circ \left(1 - \underset{m\times n_L}{\mathbf{A}^{(l)}}\right) \tag{6.33}$$

$$\underset{n_{l-1}\times n_l}{\mathbf{D}^{(l)}} = \frac{1}{m} \left(\underset{n_{l-1}\times m}{\mathbf{A}^{(l-1)}}\right)^\intercal \times \underset{m\times n_l}{\mathbf{d}^{(l)}} \tag{6.34}$$

V implementaciji nastavimo matrike uteži na neko manjšo, neničelno vrednost (na primer med 0 in 0.1) in potem z gradientnim sestopom izračunamo vrednost uteži, ki za dani nabor učnih podatkov minimizira kriterijsko funkcijo. Prileganju učnim podatkom se izogibamo z regularizacijo. Regularizacija L2 tudi tu prišteje kriterijski funkciji vsoto kvadratov uteži v nevronski mreži, s tem da se uteži na povezavah iz konstantnih enot mreže ne regularizira.