

2. Osnovni koncepti

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v2.2

1 Osnovni koncepti

Na uvodnom predavanju motivirali smo strojno učenje i ukratko predstavili osnovne pristupe: **nadzirano i nenadzirano učenje**. Danas, kao i u narednih deset tjedana, fokusirat ćemo se na nadzirano strojno učenje (engl. *supervised machine learning*). 1

Danas ćemo uvesti niz zanimljivih ideja i novih pojmova, i tako "postaviti scenu" za ono što slijedi idući tjedan. Ovo je predavanje malo apstraktno; taj problem pokušat ćemo ublažiti ubacivanjem ponešto primjera kako bi stvari postale plastičnije. Međutim, kako ćemo se na ove koncepte puno puta vraćati na svakom predavanju tijekom semestra, do kraja semestra dobit ćete vrlo dobar osjećaj za sve ove stvari o kojima ćemo danas pričati, pa ako danas ne sjedne sve na svoje mjesto, nemojte očajavati.

Glavni koncepti koje ćemo danas uvesti su **hipoteza, model, pogreška i funkcija gubitka, prenaučenost, unakrsna provjera i odabir modela**. Glavni glumac ovdje je **model**, koji će se stalno pojavljivati u svemu što ćemo raditi.

2 Primjena algoritma strojnog učenja

Za početak, razmotrimo širu sliku: koja ja tipična kuharica za primjenu strojnog učenja? Algoritme strojnog učenja tipično primjenjujemo kroz sljedećih osam koraka:

1. **Priprema podataka i preliminarna analiza** – kao i u statistici, prije nego što išta radimo, bitno je dobro se upoznati s podatcima i razumijeti s kakvim podatcima radimo;
2. **Ručno označavanje podataka za učenje i ispitivanje** – ovo nije uvijek potrebno, jer nekada su podaci već označeni, dok nekada radimo s algoritmima koji uopće ne trebaju označene podatke (nenadzirano strojno učenje);
3. **Ekstrakcija značajki** – na ulaz algoritma strojnog učenja dovode se podatci, bilo označeni (nadzirano strojno učenje) ili neoznačeni (nenadzirano strojno učenje), u obliku skupa primjera. Svaki je primjer opisan kao vektor značajki, odnosno ključnih karakteristika koje su indikativne za klasifikaciju/regresiju sličnih, budućih primjera. U ovom koraku potrebno je osmisliti na koji način prikazati primjer kao skup značajki te implementirati postupke ekstrakcije značajki. U većini slučajeva ovo je najkreativniji korak;
4. **Redukcija dimenzionalnosti** – ni ovo nije uvijek potrebno, nego samo ako imamo vrlo mnogo značajki i ako nam to iz nekog razloga predstavlja problem (npr., ako model radi loše jer imamo previše značajki a premalo primjera za učenje, ili ako zbog previše značajki gubimo mogućnost da shvatimo što se zapravo događa i da interpretiramo odluke modela); 2
5. **Odabir modela** (engl. *model selection*) – ovo je važan korak, koji početnici često ili propuste napraviti ili ga naprave pogrešno, pa ćemo ovome posvetiti mnogo paženje; 3

6. **Učenje modela** – ovdje algoritam odraduje svoj posao a mi trebamo samo pričekati da se učenje završi (to nekad traje sekundu-dvije, nekad više sati, a nekad i više tjedana, ovisno o algoritmu i podatcima);
7. **Vrednovanje modela** – kada je model naučen, želimo znati kako dobro radi, jer je model koji dobro radi naš glavni cilj u strojnog učenju. Postoji mnogo načina kako se model može vrednovati, od kojih su mnogi dobri a neki su pogrešni. Naš cilj će biti da vas naučimo kako ispravno vrednovati model;
8. **Dijagnostika i ispravljanje** (engl. *diagnostics and debugging*) – ako model ne radi (a uglavnom ne radi; iluzorno je očekivati da će stvar raditi isprve), treba znati kako debagirati model i natjerati da radi;
9. **Instalacija** (engl. *deployment*) – konačno, kada sve radi dobro, model se može ugraditi u produkciiju gdje će odradivati svoj posao.

Naš fokus na ovom predmetu bit će koraci 5–7. To su koraci koji su zapravo dosta isprepleteni i u praksi se više puta ponavljaju, sve dok ne izgradimo model s kojim smo zadovoljni, odnosno koji je dovoljno dobar za zadani zadatak.

Koracima 1–3 nećemo se baviti jer su ti koraci povezani s ekspertizom iz konkretnе domene (ako radite podatkovnu znanost, morat će se time baviti, ali i onda je dobro, a nekad i nužno, da tu imate pomoć domenskog stručnjaka). Korak 4 je vrlo zanimljiv i koristan, ali nam ne stane u ovaj predmet. Korak 9 može biti netrivijalan (npr. ako algoritam treba raditi s velikim količinama podataka, javljaju se raznorazni algoritamski izazovi), no taj korak pripada više programskom inženjerstvu, pa se ni njime nećemo baviti.

3 Primjeri, hipoteza, model

3.1 Prostor primjera

Kao i svaki algoritam, tako i algoritam strojnog učenja ima svoj ulaz i svoj izlaz. Na ulazu algoritma strojnog učenja – bilo da tek učimo model ili koristimo već naučeni model za predviđanje – nalazi se **primjer** (engl. *example, instance*). Primjer je jedna podatkovna točka (npr. jedna osoba, jedan putnik, jedna kuća, itd.), dakle jedan redak u našoj bazi podataka za koji želimo da model napravi neko predviđanje (klase, ako radimo klasifikaciju, ili brojčane vrijednosti, ako radimo regresiju).

Svaki ćemo primjer prikazati kao vektor značajki. **Značajka** (engl. *feature*) jest neka karakteristika primjera koja je bitna (ili mi mislimo da bi mogla biti bitna) za problem koji rješavamo, tj. to je neko svojstvo o kojem ovisi klasifikacija primjera (kod klasifikacije) ili njezina ciljna vrijednost (kod regresije). Npr., ako radimo klasifikaciju putnika Titanica (hoće li putnik preživjeti), onda bi značajke bile dob putnika, socio-ekonomski status, spol, itd. Značajke nekada (pogotovo u dubinskoj analizi podataka) nazivamo **atributi**. Dakle, svaki primjer bit će okarakteriziran s nizom značajki (atributa), i to ćemo formalizirati kao **vektor značajki** (engl. *feature vector*):

$$\mathbf{x} = (x_1, x_2, \dots, x_n)$$

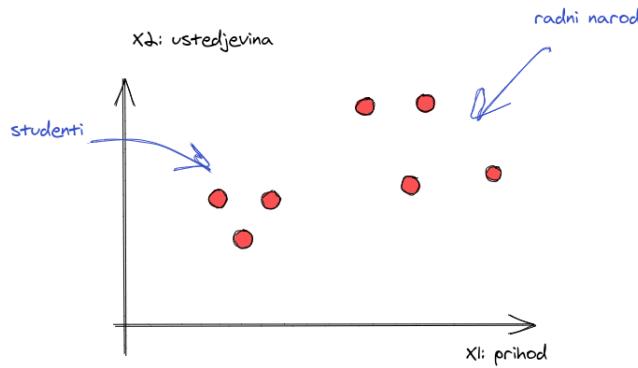
gdje su x_i pojedine značajke (atributi), a n je ukupan broj značajki (atributa). Svaki primjer će imati isti broj značajki. Premda to ne mora uvijek biti tako, mi ćemo si često pojednostaviti život i prepostaviti da su sve značajke numeričke, tj. $x \in \mathbb{R}$, pa onda $\mathbf{x} \in \mathbb{R}^n$. Vidjet ćemo kasnije da je ta prepostavka sasvim u redu, jer se numeričkim značajkama mogu kodirati i nenumeričke značajke.

Primjeri žive u **prostoru primjera** ili **ulaznom prostoru** (engl. *instance space, input space*). Taj prostor označavat ćemo s \mathcal{X} . Ako su značajke numeričke, tj. $x_i \in \mathbb{R}$, onda je $\mathcal{X} = \mathbb{R}^n$, tj. to je n -dimenzijski vektorski prostor. Njegova dimenzija je n ona odgovara broju značajki.

Dimenzionalnost n će u našim jednostavnim primjerima biti $n = 2$ ili $n = 3$ (samo zato jer nam je takav prostor primjera moguće vizualizirati), međutim u praksi ćemo (željeti) imati puno više značajki, 100 ili 1000, nekad više tisuća ili stotine tisuća, ovisno o problemu. Npr., ako radimo strojno učenje u bininformatici, računalnom vidu ili obradi prirodnoga jezika, lako ćemo imati na desetke tisuća značajki.

Dakle, ponovimo: svaki primjer je jedan vektor u ulaznom prostoru, $\mathbf{x} \in \mathcal{X}$, odnosno jedna točka u tom vektorskom prostoru.

► PRIMJER



Razmotrimo prostor primjera (ulazni prostor) za problem klasifikacije kreditno sposobnih klijenata banke. Banka treba odlučiti kome će dati kredit. Pojednostavimo problem i pretpostavimo da banke to rade na temelju samo dvije značajke: prihod i ušteđedvina. Onda je ulazni prostor dvodimenzijски ($x_1 \rightarrow$ prihod i $x_2 \rightarrow$ ušteđedvina), a svaki klijent banke je jedan dvodimenzijski vektor u tom prostoru. Studenti nažalost imaju niske prihode i malo ušteđedvine, dok oni koji uživaju plodove dužeg rada imaju naravno veće prihode i veće ušteđedvine, pa su bankama draži.

3.2 Označeni primjeri

Kod nadziranog učenja, svaki primjer ima svoju **oznaku** (engl. *label*). To će biti oznaka klase (kod klasifikacije) ili ciljna brojčana vrijednost (kod regresije). Oznaku primjera označit ćemo sa y , a skup svih mogućih oznaka u skupu podataka označit ćemo sa \mathcal{Y} .

Za klasifikaciju u K klasa imamo $\mathcal{Y} = \{0, \dots, K-1\}$. Kada imamo samo dvije klase, $K = 2$, govorimo o **binarnoj klasifikaciji**. Tipično, tada $\mathcal{Y} = \{0, 1\}$ ili $\mathcal{Y} = \{-1, +1\}$, već kako nam je (matematički) jednostavnije. Primjer za koji $y = 0$ ili ($y = -1$) nazivamo **negativan primjer**, dok primjer za koji $y = 1$ nazivamo **pozitivan primjer**. Za regresiju najopćenitiji (i tipičan) slučaj jest $\mathcal{Y} = \mathbb{R}$, tj. oznake su realni brojevi.

Jedan primjer nam naravno neće biti dovoljan za strojno učenje. Naprotiv, što više primjera, to bolje. Općenito, raspolažemo skupom primjera. Ukupan **broj primjera** označit ćemo sa N . Primijetite da koristimo veliko N za broj primjera, a malo n za broj značajki. Općenito, ta dva broja bit će različita. Tipično, iz razloga koji će postati jasni kasnije, voljeli bismo da primjera imamo više nego značajki, tj. $N > n$, a idealno da ih imamo mnogo više od značajki, tj. $N \gg n$.

Konačno, **skup označenih primjera** (engl. *labeled dataset*) označit ćemo sa \mathcal{D} . Formalno, \mathcal{D} je skup parova:

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N \subseteq \mathcal{X} \times \mathcal{Y}$$

U strojnem učenju često će nam biti lakše koristiti matrični zapis podataka. Tako skup označenih primjera \mathcal{D} možemo prikazati pomoću dvije komponente: matrica (neoznačenih)

primjera \mathbf{X} i vektor njihovih oznaka \mathbf{y} . Ta matrica i taj vektor izgledaju ovako:

$$\mathbf{X} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & & & \\ x_1^{(N)} & x_2^{(N)} & \cdots & x_n^{(N)} \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{pmatrix}$$

Matrica \mathcal{D} sastavljena je od matrice $\mathbf{X}_{N \times n}$ i vektora $\mathbf{y}_{N \times 1}$, tj. $\mathcal{D} = (\mathbf{X}, \mathbf{y})$. Retci matrice \mathbf{X} odgovaraju primjerima, a stupci odgovaraju značajkama. Dakle, svaki redak matrice \mathbf{X} je jedan vektor značajki odnosno primjer, $\mathbf{x}^{(i)}$. S druge strane, svaki stupac matrice \mathbf{X} je jedna značajka kroz sve primjere. Primijetite da indeks primjera pišemo u superskriptu (iznad simbola \mathbf{x}), a indeks značajke u supskriptu (ispod simbola \mathbf{x}). Npr., x_1^2 je prva značajka (od njih ukupno n) drugog primjera (od njih ukupno N). Analogno, komponente vektora \mathbf{y} odgovaraju oznaci za svaki pojedinačan primjer. Pretpostavka je, naravno, da su retci matrice \mathbf{X} i komponente vektora \mathbf{y} poravnate, tj. $y^{(1)}$ je oznaka za primjer $\mathbf{x}^{(1)}$, $y^{(2)}$ je oznaka za primjer $\mathbf{x}^{(2)}$, itd.

Inače, matrica \mathbf{X} naziva se **matrica dizajna** (engl. *design matrix*). Ovaj naziv pogotovo se često koristi u statistici.

3.3 Hipoteza

Svrha nadziranog strojnog učenja jest naučiti funkciju koja primjerima iz \mathcal{X} dodjeljuje oznake iz \mathcal{Y} . Drugim riječima, svrha je naučiti preslikavanje iz \mathcal{X} u \mathcal{Y} . Ta funkcija naziva se **hipoteza**. Označavat ćemo je sa h . Dakle h je:

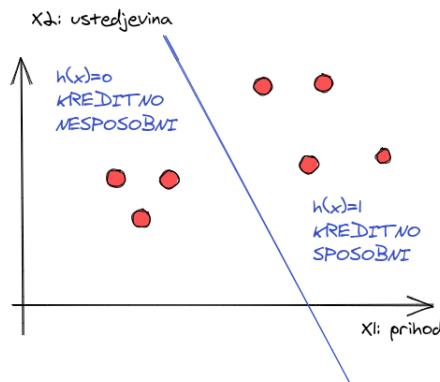
$$h : \mathcal{X} \rightarrow \mathcal{Y}$$

Ponovimo ovo: h je funkcija koja svakom primjeru (iz prostora primjera) dodjeljuje oznaku klase (kod klasifikacije) ili brojčanu vrijednost (kod regresije). Konkretno, za binarnu klasifikaciju hipoteza je:

$$h : \mathcal{X} \rightarrow \{0, 1\}$$

Ova funkcija zapravo dijeli prostor ulaznih primjera na dva poluprostora: jedan za koji je $h(\mathbf{x}) = 0$ i drugi za koji je $h(\mathbf{x}) = 1$. Pogledajmo primjer.

► PRIMJER



U dvodimenzionskom ulaznom prostoru hipotezu h mogli bismo definirati tako da odgovara pravcu u ravnini. Taj pravac razdjeljuje dvodimenzionalni prostor primjera na dva poluprostora. Poluprostor u kojem $h(\mathbf{x}) = 0$ odgovara kreditno nesposobnim klijentima banke (niski prihodi i niska uštedjedjina), a poluprostor u kojem $h(\mathbf{x}) = 1$ odgovara onim kreditno sposobnim (visoki prihodi i visoka uštedjedjina). Klasifikacija novih primjera (novih klijenata) bit će određena time s koje strane pravca će primjer sletjeti. Primijetite da ovako definirana hipoteza klasificira sve moguće primjere u $\mathcal{X} = \mathbb{R}^2$, a ne samo ovih sedam koje imamo ucrtane.

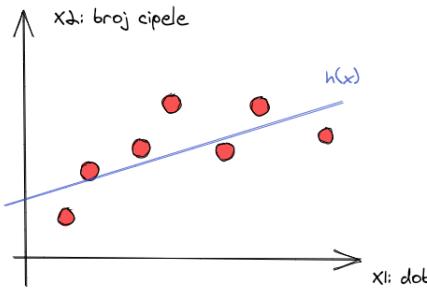
Općenito, mi nećemo unaprijed znati koja je točna funkcija h . Kada bismo to znali, ne bi nam trebalo strojno učenje. Međutim, znat ćemo otprilike – ili, ako ne znamo, nekako ćemo morati odlučiti – što su mogući kandidati za funkciju h . Možda se sjećate da smo u uvodu rekli da je zadatak algoritma strojnog učenja podesiti **parametre** modela, a da je model funkcija **definirana do na parametre**. Evo sad što to točno znači: mi ćemo imati funkciju h koja će biti definirana s nekim parametrima, koje ćemo označavati sa θ . To pišemo ovako:

$$h(\mathbf{x}; \boldsymbol{\theta})$$

(Koristimo ‘;’ kako bismo odvojili argument funkcije koji varira od argumenta funkcije koji je više-manje konstantan. Ova razlika je samo konvencija; formalno, naravno, oba su argumenta variabile dotične funkcije.)

Funkcija h parametrizirana je parametrima θ . Općenito, to neće biti samo jedan parametar, nego niz parametara, tako da zapravo govorimo o **vektor parametara $\boldsymbol{\theta}$** . U primjerima koji slijede vektor parametara sastojat će se od parametara koji realni brojevi. Međutim, općenito vektor parametara može sadržavati parametre različitog tipa (npr., matrice, vektore, skalare, i razne kombinacije tih tipova). Pogledajmo sada dva primjera.

► PRIMJER



Želimo predvidjeti broj (tj. veličinu) cipele s obzirom na dob osobe. Kakav je to problem: regresijski ili klasifikacijski? To je regresijski problem, jer želimo predvidjeti brojčanu vrijednost, a ne klasu. Naša hipoteza je funkcija $h : \mathcal{X} \rightarrow \mathbb{R}$, a ulazni prostor je jednodimenzionalni, $n = 1$, jer imamo samo jednu značajku (dob osobe). Hipotezu bismo mogli definirati ovako:

$$h(x; \theta_0, \theta_1) = \theta_1 x + \theta_0$$

Ovime smo zapravo definirali **pravac**. To je u redu, jer očekujemo (donekle) **linearnu ovisnost** između dobi i veličine cipela.

Funkcija h (a ovdje je to jednadžba pravca) ima dva parametra, θ_1 i θ_0 , odnosno naš vektor značajki je $\boldsymbol{\theta} = (\theta_0, \theta_1)$. Jedan parametar (θ_1) određuje nagib pravca, a drugi (θ_0) odsječak na osi y . Trebaju nam oba parametra (odsječak na osi y nam treba jer tek rođena osoba s praktički 0 godina nema broj cipele 0).

► PRIMJER

Ovo je bio primjer za regresiju. Pogledajmo sada primjer za **klasifikaciju**. Vratimo se opet na klasifikaciju kreditne sposobnosti, na temelju značajki prihoda i ušteđevine. Dakle, radimo klasifikaciju u 2-D ulaznom prostoru. Kreditno sposobni su oni koji imaju ili velika primanja ili veliku uštedu.

Recimo da se odlučimo da te dvije klase želimo odvojiti **pravcem**, kao u ranijem primjeru. Taj pravac možemo napisati pomoću **implicitne jednadžbe pravca**:

$$\theta_1 x_1 + \theta_2 x_2 + \theta_0 = 0$$

Općenito, primjeri će se nalaziti s jedne ili s druge strane tog pravca. Dakle, pravac dijeli ulazni prostor na dva poluprostora. Primjeri na “pozitivnoj strani” pravca su oni za koje vrijedi:

$$\theta_1 x_1 + \theta_2 x_2 + \theta_0 \geq 0$$

a primjeri na “negativnoj strani pravca” su oni za koje vrijedi:

$$\theta_1 x_1 + \theta_2 x_2 + \theta_0 < 0$$

(Primjeri koji leže točno na pravcu mogu se tretirati kako god; ovdje smo ih uključili u pozitivnu stranu.)

Za primjere koji pozitivnoj strani pravca želimo da hipoteza vrati 1, a za primjere koji su na negativnoj želimo da vrati 0 (ili -1 , već kako odlučimo, ali odlučimo se ovdje za 0). To možemo ostvariti ako hipotezu definiramo na sljedeći način:

$$h(x_1, x_2; \theta_0, \theta_1, \theta_2) = \mathbf{1}\{\theta_1 x_1 + \theta_2 x_2 + \theta_0 \geq 0\}$$

Ovdje smo upotrijebili funkciju $\mathbf{1} : \{\perp, \top\} \rightarrow \{0, 1\}$, koja samo pretvara Booleove vrijednosti u 0 ili 1. Ta je funkcija definirana ovako:

$$\mathbf{1}\{P\} = \begin{cases} 1 & \text{ako } P \equiv \top \\ 0 & \text{inače} \end{cases}$$

Ovime smo definirali hipotezu $h : \mathcal{X} \rightarrow \{0, 1\}$, koja interno provjerava s koje se strane pravca primjer našao te ovisno o tome daje 0 ili 1. Granica između ta dva slučaja, tj. između dviju klasa, zapravo odgovara pravcu definiranom implicitnom jednadžbom $\theta_1 x_1 + \theta_2 x_2 + \theta_0 = 0$.

Primijetimo ovdje da imamo tri parametra, $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2)$, dok smo u prethodnom primjeru s regresijom imali samo dva. No, prethodni primjer je imao jednodimenzionalni ulazni prostor (y -os odgovarala je oznaci), dok ovaj primjer ima dvodimenzionalni ulazni prostor (oznaka ne odgovara niti jednoj od osi već je implicitna u poziciji primjera u odnosu na pravac).

4

3.4 Model

U strojnog učenju neprestano pričamo o modelima. Sada možemo demistificirati što je to zapravo. **Model** nije ništa drugo nego **skup hipoteza**. Budući da su hipoteze funkcije, to zapravo znači da je model **skup funkcija**. Označit ćemo ga sa \mathcal{H} .

Dakle, model je jedna “vreća” koja sadrži različite hipoteze h . Na primjer, ako je h definiran kao u gornjem primjeru, onda je model skup svih mogućih ravnina u 3-D prostoru. Ili to može biti skup svih hiperravnina u 4-D prostoru. Ili, ako radimo regresiju, skup svih pravaca, ili skup svih krivulja, itd.

Formalno ćemo model definirati ovako:

$$\mathcal{H} = \{h(\mathbf{x}; \boldsymbol{\theta})\}_{\boldsymbol{\theta}}$$

Model \mathcal{H} je, dakle, skup hipoteza h , koje su parametrizirane sa $\boldsymbol{\theta}$, a supskript $\boldsymbol{\theta}$ znači da su elementi tog skupa indeksirani parametrom $\boldsymbol{\theta}$. To pak znači da svaki vektor $\boldsymbol{\theta}$ odgovara jednoj funkciji iz skupa \mathcal{H} . Drugim riječima, za zadani vektor parametara $\boldsymbol{\theta}$ možemo dohvatiti njemu odgovarajuću funkciju $h \in \mathcal{H}$. To jest, $\boldsymbol{\theta} \mapsto h$ (θ se preslikava u h , odnosno θ jednoznačno određuje h). Prema tome, model je skup funkcija **parametriziranih (tj. indeksiranih)** s $\boldsymbol{\theta}$.

Sada kada sve ovo znamo, evo zanimljivog (i istinitog) pogleda na strojno učenje: **učenje (treniranje modela)** nije ništa drugo nego **pretraživanje** skupa hipoteza \mathcal{H} u nastojanju da se nađe najbolja hipoteza $h \in \mathcal{H}$.

Ujedno se odmah prirodno postavlja sljedeće pitanje: što je nabolja hipoteza? Intuitivno je jasno da bi **najbolja hipoteza** bila ona koja najtočnije klasificira primjere (kod klasifikacije)

5

odnosno ona daje vrijednosti najbliže ciljnim brojčanim vrijednostima (kod regresije). Budući da se tu onda radi o traženju po nekom kvantitativnom kriteriju dobrote, sada vidimo da je učenje zapravo **optimizacijski problem**. Iz tog je razloga strojno učenje općento vrlo povezano s područjem optimizacije.

6

U stvarnim primjenama, skup \mathcal{H} će biti vrlo velik. Što to znači? To znači da u modelu \mathcal{H} postoji puno mogućih hipoteza h . Zbog toga će nam trebati nekakva **heuristička optimizacija**, koja će pametno pretraživati taj skup, umjesto da ga pretražujemo iscrpno.

Također, primjetite na ovom mjestu da je bilo vrlo mudro što smo parametrizirali funkciju h , odnosno što smo model definirali kao skup parametriziranih funkcija, $\mathcal{H} = \{h(\mathbf{x}; \boldsymbol{\theta})\}_{\boldsymbol{\theta}}$. Naime, da to nismo napravili, i da smo funkcije ostavili sasvim proizvoljnima te da smo rekli da je model skup baš svih mogućih funkcija, onda ne bi postojao način da pretražujemo po tom skupu, pa ne bismo imali po čemu optimizirati. Čak i da smo nekako drugačije ograničili skup mogućih funkcija, opet to ne bi bilo dobro ako funkcije ne bi bile parametrizirane. Ovako, kada su funkcije indeksirane vektorom parametara $\boldsymbol{\theta}$, možemo pretraživati po $\boldsymbol{\theta}$, jer nam svaka promjena u $\boldsymbol{\theta}$ može dati neku drugu funkciju. Zanimljivo je također primjetiti da to što smo parametrizirali funkcije znači da smo ograničili skup mogućih funkcija, ali da to ne znači nužno da je skup \mathcal{H} konačan (npr., možemo se ograničiti na skup hipoteza koje odgovaraju pravcima, no takvih funkcija i dalje može biti beskonačno mnogo).

4 Empirijska pogreška i funkcija gubitka

4.1 Empirijska pogreška

Vratimo se na pitanje određivanja “najbolje hipoteze”. Očito nam treba neka **numerička procjena** koliko je hipoteza dobra, a na temelju koje ćemo provoditi optimizaciju. Nas bi zapravo zanimalo koliko bi hipoteza radila dobro na **svim mogućim** primjerima. Međutim, nikada nećemo imati na raspolaganju sve moguće primjere, jer onda ne bismo trebali strojno učenje. Umjesto toga, mjerit ćemo koliko je hipoteza dobra na **skupu označenih primjera** koje imamo na raspolaganju. Ideja je onda da izmjerimo koliko dobro hipoteza radi na tom skupu, tj. u kojoj se mjeri izlaz hipoteze podudara s točnim oznakama. Mjera koliko je hipoteza dobra (odnosno loša) na označenom skupu podataka koji imamo na raspolagajnu naziva se **empirijska pogreška**.

Empirijska pogreška govori nam koliko hipoteza griješi kad klasificira primjere (klasifikacija) ili koliko su vrijednosti daleko od ciljnih vrijednosti (regresija). Pogrešku nazivamo **empirijska** jer je mjerimo na konkretnom skupu podataka (zapravo na statističkom uzorku koji imamo na raspolaganju). Nas zapravo zanima prava pogreška hipoteze na svim mogućim primjerima, ali, naravno, to ne možemo znati jer nemamo sve moguće primjere, pa dakle koristimo empirijsku pogrešku kao procjenu prave pogreške.

Empirijsku pogrešku hipoteze h na skupu označenih primjera \mathcal{D} označit ćemo sa:

$$E(h|\mathcal{D})$$

Ova notacija ima za svrhu naglasiti da je empirijska pogreška E funkcija hipoteze h za neki fiksirani skup označenih primjera \mathcal{D} . To jest, za fiksirani skup označenih primjera \mathcal{D} , različite hipoteze h mogu dati različite vrijednosti empirijske pogreške. (Oznaku ‘|’ koristit ćemo i inače kako bismo razdvojili argumente funkcije koji su varijable od onih za koje u nekom kontekstu prodrazumijevamo da su konstantni.) Puni naziv za ovu funkciju je **funkcija empirijske pogreške**.

Kako bismo točno definirali empirijsku pogrešku? Tu imamo više mogućnosti. Za klasifikaciju često koristimo **pogrešku klasifikacije** (engl. *misclassification error*):

$$E(h|\mathcal{D}) = \frac{1}{N} \sum_{i=1}^N \mathbf{1}\{h(\mathbf{x})^{(i)} \neq y^{(i)}\}$$

Ovo se svodi na to da jednostavno pobrojimo koliko je primjera, od njih ukupno N , za koje hipoteza h daje oznaku koja je drugačija od prave oznake $y^{(i)}$ (funkciju $\mathbf{1}\{\cdot\}$ definirali smo u prethodnom primjeru), a onda taj broj podijelimo s ukupnim brojem primjera kako bismo dobili udio netočno klasificiranih primjera.

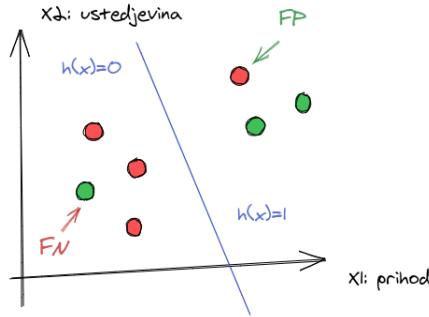
Specifično, za binarnu klasifikaciju s oznakama $\mathcal{Y} = \{0, 1\}$ funkciju pogreške mogli bismo definirati i ovako:

$$E(h|\mathcal{D}) = \frac{1}{N} \sum_{i=1}^N |h(\mathbf{x})^{(i)} - y^{(i)}|$$

Klasifikator će općenito grijesiti, tj. empirijska pogreška bit će veća od nule. Kod binarne klasifikacije, klasifikator može grijesiti na dva načina: može pozitivan primjer (onaj za koji je $y = 1$) klasificirati kao negativan, ili može negativan primjer (onaj za koji je $y = 0$) klasificirati kao pozitivan. Prvi slučaj, kada klasifikator pozitivan primjer klasificira kao negativan, zove se **lažno negativan** primjer (engl. *false negative*, *FN*). Drugi slučaj, kada klasifikator negativan primjer klasificira kao pozitivan, zove se **lažno pozitivan** primjer (engl. *false positive*, *FP*). Zbroj lažno pozitivnih i lažno negativnih primjera u skupu \mathcal{D} daje nam ukupan broj pogrešaka klasifikatora, a kada taj broj podijelimo sa ukupnim brojem primjera N , dobivamo empirijsku pogrešku, odnosno udio pogrešno klasificiranih primjera. Primjere na kojima klasifikator nije pogriješio također možemo stvrstati u dvije skupine: pozitivni primjeri koje je klasifikator ispravno klasificirao kao pozitivni su **istinito pozitivni** (engl. *true positive*, *TP*), a negativni primjeri koje je klasifikator klasificirao kao negativni su **istinito negativni** (engl. *true negative*, *TN*).

► PRIMJER

Razmotrimo opet primjer binarne klasifikacije kreditno sposobnih klijenata banke. Pozitivna klasa (ona za koju $y = 1$) neka odgovara kreditno sposobnim klijentima. Recimo da smo definirali linearan model (kao u ranijem primjeru), naučili ga na skupu \mathcal{D} koji se sastoji od 7 označenih primjera, te dobili ovakvu granicu između dvije klase u ulaznom prostoru:



Prisjetimo se, hipoteza $h(\mathbf{x})$ ovdje odgovara pravcu te naš dvodimenzionalni ulazni prostor dijeli na dva poluprostora, s granicom u točkama za koje $h(\mathbf{x}) = 0$. Zeleni primjeri neka su oni koji su u skupu \mathcal{D} označeni kao pozitivni ($y = 1$), a crveni neka su oni koji su označeni kao negativni ($y = 0$). Na slici vidimo da naš klasifikator na nekim primjerima grijesi. Konkretno, klasifikator je tri primjera klasificirao kao pozitivna (gornji desni dio ulaznog prostora), ali je jedan od njih zapravo negativan. Taj primjer je lažno pozitivan (FP). Također, klasifikator je četiri primjera klasificirao kao negativna (doljni lijevi dio ulaznog prostora), ali je jedan od tih primjera zapravo pozitivan, i to je onda lažno negativan primjer (FN). Ukupno je klasifikator od sedam primjera njih dva pogrešno klasificirao, pa je empirijska pogreška hipoteze h na skupu \mathcal{D} jednak:

$$E(h|\mathcal{D}) = \frac{1}{N} \sum_{i=1}^N \mathbf{1}\{h(\mathbf{x})^{(i)} \neq y^{(i)}\} = \frac{1}{7}(1 + 0 + 0 + 0 + 1 + 0 + 0) = \frac{2}{7} = 0.286$$

odnosno 28.6%. Posljedično, **točnost** klasifikatora je $1 - 0.286 = 0.714$ odnosno 71.4%. Usput,

primijetite da s modelom definiranim tako da odgovara pravcu na ovakovom skupu \mathcal{D} niti nismo mogli ostvariti $E(h|\mathcal{D}) = 0$. Više o tome kasnije.

4.2 Funkcija gubitka

Primijetimo da svaki pojedinačni primjer iz skupa označenih primjera može doprinijeti empirijskoj pogrešci. Npr., kod klasifikacije, svaki će primjer imati doprinos pogrešci koji će biti jednak 0 ili 1, ovisno je li dotični primjer ispravno klasificiran. Iznos pogreške načinjene na pojedinačnom primjeru (funkcija unutar sume) izračunava **funkcija gubitka** (engl. *loss function*). Za dani označeni primjer, funkcija gubitka govori nam koliko je model izgubio na točnosti (tj. dobio na ukupnoj pogrešci) na tom jednom primjeru. Ako je vrijednost funkcije gubitka za neki primjer jednaka nuli, to znači da model ispravno klasificira taj primjer i da taj primjer ne doprinosi ukupnoj pogrešci modela. U gornjoj definiciji empirijske pogreške koristili smo funkciju gubitka $\mathbf{1}\{h(\mathbf{x})^{(i)} \neq y^{(i)}\}$. Ta se funkcija gubitka zove se **gubitak nula-jedan** (engl. *zero-one loss*). Ima i mnogo drugih funkcija gubitaka, kao što ćemo vidjeti u narednim tjednima.

Formalno, funkciju gubitka definirat ćemo kao $L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+$. To je funkcija koja uzima dvije oznake, jednu točnu i jednu dobivenu kao izlaz hipoteze, te izračunava gubitak (koji je nenegativan realan broj). Konkretno, pisat ćemo $L(y, h(\mathbf{x}))$, gdje je y točna oznaka primjera \mathbf{x} , a $h(\mathbf{x})$ je oznaka koju daje za taj primjer daje hipoteza h .

Kako je funkcija gubitka povezana s empirijskom funkcijom pogreške? Veza je ta da ćemo empirijsku funkciju pogreške u većini slučajeva definirati indirektno, preko funkcije gubitka. Jednostavno ćemo izračunati srednju vrijednost funkcije gubitka na skupu označenih primjera. Malo općenitije, reći ćemo da je funkcija pogreške **očekivana vrijednost funkcije gubitka** na svim mogućim primjerima iz $\mathcal{X} \times \mathcal{Y}$, a mi to očekivanje aproksimiramo empirijski kao srednju vrijednost funkcije gubitka na označenom skupu primjera.

Da sažmemo: u modelu \mathcal{H} (koji je skup hipoteza) postoje različite funkcije h . Kada te funkcije primijenimo na označene primjere \mathcal{D} , neke će funkcije davati točniju a neke manje točnu klasifikaciju. Koliko je klasifikacija na skupu označenih primjera točna govori nam empirijska pogreška, koja je srednja vrijednost funkcije gubitka na skupu označenih primjera. Kada kažemo da učimo (treniramo) model, mi zapravo pretražujemo model \mathcal{H} kako bismo pronašli hipotezu s najmanjom empirijskom pogreškom na skupu označenih primjera \mathcal{D} .

5 Tri komponente algoritma strojnog učenja

Spojimo sada zajedno sve ovo što smo naučili. Bit će nam od velike koristi da svaki, baš svaki algoritam strojnog učenja analiziramo u smislu **tri glavne komponente**. Te komponente su:

1. Model

$$\mathcal{H} = \{h(\mathbf{x}; \boldsymbol{\theta})\}_{\boldsymbol{\theta}}$$

Prisjetimo se, model je skup funkcija (hipoteza) parametriziranih vektorom parametara $\boldsymbol{\theta}$.

2. Funkcija gubitka

$$E(h|\mathcal{D}) = \frac{1}{N} \sum_{i=1}^N L(y^{(i)}, h(\mathbf{x}^{(i)}))$$

S obzirom da parametri $\boldsymbol{\theta}$ jednoznačno određuju funkciju h , tj. $\boldsymbol{\theta} \mapsto h$, to znači da možemo pisati i:

$$E(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{N} \sum_{i=1}^N L(y^{(i)}, h(\mathbf{x}^{(i)}; \boldsymbol{\theta}))$$

tj. možemo govoriti o pogrešci hipoteze h ili o pogrešci parametara θ koji definiraju tu istu hipotezu. U drugom slučaju napisali smo $h(\mathbf{x}^{(i)}; \theta)$ kako bismo dodatno naglasili da je funkcija h parametrizirana sa θ (ona je to uvijek, no nekada to nije potrebno posebno istaknuti).

3. Optimizacijski postupak

To je postupak kojim unutar modela \mathcal{H} nalazimo hipotezu h^* koja minimizira empirijsku pogrešku:

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} E(h|\mathcal{D})$$

što je, zato što θ jednoznačno određuju h , istovjetno nalaženju optimalnih parametara θ^* :

$$\theta^* = \underset{\theta}{\operatorname{argmin}} E(\theta|\mathcal{D})$$

Za hipotezu h^* nekad ćemo koristiti naziv "naučeni/trenirani model". Slično, za parametre θ^* koristit ćemo naziv "parametri modela" (premda bi točnije bilo "vrijednosti parametara naučenog/treniranog modela").

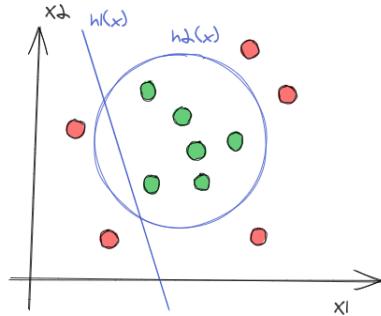
Primjetite da smo ove tri komponente definirali doista vrlo apstraktно. Model smo definirali kao skup hipoteza, ali nismo točno rekli kako su hipoteze parametrizirane. Pogrešku smo definirali kao srednju vrijednost funkcije gubitka, a nismo točno rekli kako je definirana funkcija gubitka. I optimizacijski postupak definirali smo najapstraktnije moguće, kao funkciju koja pronalazi hipotezu (odnosno, ekvivalentno, parametre) koji minimiziraju empirijsku pogrešku. To je namjerno, jer želimo da ove tri komponente budu okvir u koji možemo staviti svaki algoritam strojnog učenja. U tjednima koji slijede, kako budemo uvodili pojedine algoritme strojnog učenja, za svaki ćemo algoritam konkretno specificirati ove tri komponente. Takav unificirani tretman omogućiće nam da bolje uočimo sličnosti i razlike između svih tih algoritama. Zato svakako upamtite ove tri komponente!

6 Složenost modela

U idealnom slučaju, u modelu (skupu hipoteza) \mathcal{H} postoji hipoteza h čija je empirijska pogreška jednaka nula, $E(h|\mathcal{D}) = 0$. No, lako je moguće (i u praksi je to redovito slučaj) da takva h ne postoji, tj. $\forall h \in \mathcal{H}. E(h|\mathcal{D}) > 0$. Tada kažemo da model \mathcal{H} nije dovoljne **složenosti** ili **kapaciteta** za naš problem (definiran našim skupom označenih primjera \mathcal{D}).

► PRIMJER

Želimo naučiti (trenirati) binaran klasifikator za primjere iz skupa \mathcal{D} , koji su ulaznoma prostoru raspoređeni ovako (zeleni primjeri su pozitivni, crveni su negativni):



(Na primjer, ako je x_1 prihod, a x_2 ušteđevina, ovaj skup označenih primjera mogao bi odgovarati problemu u kojem banka, možda za neku svoju posebnu ponudu, želi identificirati klijente koji nemaju niti preveliki prihod niti preveliku ušteđevinu.) Razmatramo dva modela: \mathcal{H}_1 je skup svih hipoteza koje odgovaraju pravcima (kao u ranijem primjeru), a \mathcal{H}_2 je skup svih hipoteza koje odgovaraju kružnicama (s podesivim središtem i radiusom; neka je model definiran tako da su svi primjeri koji se nalaze unutar kružnice pozitivno klasificirani). Na skupu \mathcal{D} , niti jedna hipoteza h iz \mathcal{H}_1 ne može doseći $E(h|\mathcal{D}) = 0$ (hipoteza $h \in \mathcal{H}_1$ s najmanjom pogreškom ostvaruje $E(h|\mathcal{D}) = \frac{3}{11}$). To znači da model \mathcal{H}_1 nije dovoljne složenosti (kapaciteta) za problem definiran skupom \mathcal{D} . (Ovdje je to zato jer skup \mathcal{D} nije linearno odvojiv, pa linearan model, definiran pravcem, ne može odvojiti linearno neodvojive primjere). S druge strane, u modelu \mathcal{H}_2 postoje neke hipoteze koje mogu savršeno točno klasificirati primjere iz \mathcal{D} , tj. $\exists h \in \mathcal{H}_2. E(h|\mathcal{D}) = 0$. Jedna takva hipoteza ucrtana je u gornjoj slici. Stoga zaključujemo da je model \mathcal{H}_2 dovoljne složenosti (kapaciteta) za klasifikacijski problem definiran skupom označenih primjera \mathcal{D} .

Odmah se postavlja pitanje: je li ovo uopće problem? Zašto jednostavno ne definiramo složeniji model, takav da u \mathcal{H} postoji hipoteza h čija je empirijska pogreška jednaka nuli? Međutim, kao što možete pretpostaviti, u stvarnosti stvari ipak nisu tako jednostavne.

Naime, htjeli mi to ili ne, u našim podatcima (u skupu označenih primjera) uvijek (osim u idealiziranim akademskim primjerima) postoji **šum** (engl. *noise*). Šum je neželjena anomalija u podacima zbog koje dolazi do odstupanja opaženih vrijednosti od pravih vrijednosti, bilo u značajkama pojedinih primjera \mathbf{x} ili njihovim oznakama y . U oba slučaja rezultat će općenito biti taj da će oznake nekih primjera u našem skupu označenih primjera biti netočne. Naravno, problem je u tome što mi ne možemo sa sigurnošću utvrditi koji su to primjeri.

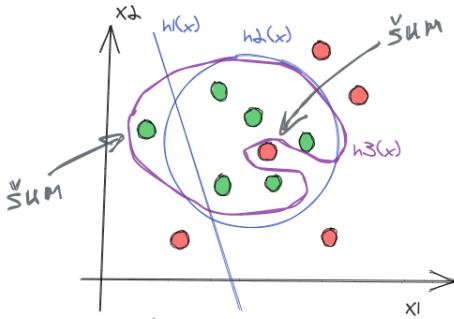
Mogući su razni uzroci šuma u podatcima:

- **Nepreciznost** pri mjerenu značajki (pogotovo ako su značajke neke fizikalne veličine, npr. visina, težina);
- **Pogreške u označavanju** – takozvani **teacher noise** – npr., zamolimo ljude da nam označe podatke, a oni su nešto krivo shvatili, pa krivo označavaju, ili tu i tamo nisu koncentrirani, npr. zato jer rade još nešto paralelno sa označavanjem (npr. prebacuju se između označavanja i Facebooka);
- Postojanje **skrivenih značajki** (latentnih varijabli) – npr., klasificiramo tko će preživjeti potonuće Titanica, ali iz nekog razloga zaboravili smo na značajku starosne dobi. Bez te značajke, izgledat će nam kao da su podatci puni šuma, a zapravo nisu, nego nam nedostaje jedna dimenzija;
- **Nejasne granice** između klasa (subjektivnost) – problem je inherentno subjektivan, i različiti ljudi imaju različito mišljenje koja je točna klasifikacija. (npr., klasifikacija govora mržnje u porukama na Twitteru).

Što god da je uzrok šuma, njegova je posljedica ta da će granica između pozitivnih i negativnih primjera (ili, općenito, granica između primjera iz različitih klasa) u ulaznom prostoru biti složenija nego što bi ona bila da šuma nema, odnosno složenija nego što stvarno jest.

► PRIMJER

Zamislimo da u skupu \mathcal{D} iz prethodnog primjera postoji šum takav da su neki primjeri pogrešno označeni. Npr., ovako:



Dva primjera su pogrešno označena: jedan pozitivan (zeleni) primjer označen je negativno (crveno), a jedan negativan primjer (crveni) označen je pozitivno (zeleno). U stvarnosti, oznake primjera su onakve kakve su bile u prethodnom primjeru, no sada su, zbog šuma, neke oznake pogrešne. Kao što vidimo, sada više niti model \mathcal{H}_2 nije dovoljne složenosti (kapaciteta) da na skupu \mathcal{D} ostvari empirijsku pogrešku jednaku nuli. Želimo li ostvariti savršenu klasifikaciju na \mathcal{D} , treba nam neki još složeniji model. Neka je to model \mathcal{H}_3 , koji odgovara proizvoljno zakriviljenim regijama u ulaznom prostoru. Nazovimo taj model kolokvijalno "model krumpira". Model krumpira je dovoljno složen za ovaj problem i sadrži hipotezu $h_3 \in \mathcal{H}_3$ koja na \mathcal{D} daje savršenu klasifikaciju (odnosno čija je empirijska pogreška jednaka nuli).

Pitanje je, naravno, želimo li koristiti tako složen model. Naime, u stvarnosti su pozitivni primjeri zapravo grupirani zajedno unutar kružnice, ali samo zbog šuma izgleda kao da to nije tako. U tom smislu, ispravna klasifikacija je zapravo ona koju daje hipoteza h_2 iz modela \mathcal{H}_2 (kružnica). S druge strane, model \mathcal{H}_3 (krumpir) je presložen, pa je zajedno s ispravnom klasifikacijom naučio i onu neispravnu uslijed šuma. Problem je, dakle, što se presložen model može previše prilagoditi šumu. To je problem jer takav model neće dobro generalizirati (v. idući primjer).

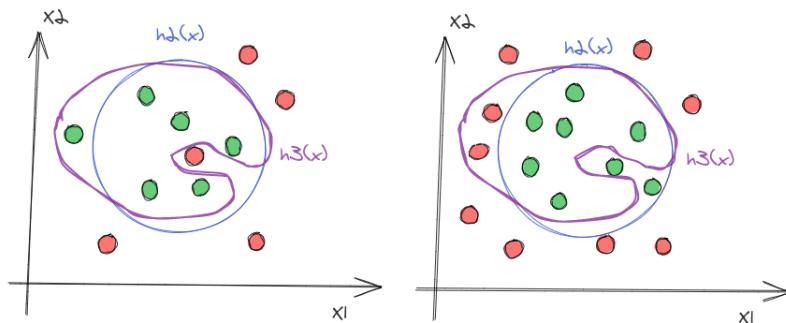
Vidimo da, ako postoji šum, jednostavan model (npr., pravci) ne može doseći $E(h|\mathcal{D}) = 0$. S druge strane, ako je model suviše složen, on će se previše prilagoditi podatcima, pa će naučiti i šum, a ne pravu klasifikaciju bez šuma. Ovdje smo razmatrali primjere u dvodimenzijskom ulaznom prostoru. Situacija je mnogo gadnija u ulaznim prostorima visoke dimenzije, gdje će složeni model imati priliku još se više prilagoditi podatcima odnosno šumu u njima.

Bit problema je dakle ova: prejednostavni modeli imat će previsoku empirijsku pogrešku, dok će presloženi modeli vjerojatno imati nisku empirijsku pogrešku, ali će se previše prilagoditi šumu. Kako god, rezultat je da će takvi modeli **loše generalizirati**. Prisjetimo se, generalizacija je ključno svojstvo modela strojnog učenja: želimo da model radi dobro na primjerima koje nikada nije video tijekom treniranja. Prejednostavan model ne može dobro generalizirati jer općenito radi loše (ima visoku empirijsku pogrešku već i na podatcima na kojima je treniran), dok presložen model ne može dobro generalizirati jer se previše prilagodio šumu, a šum će u neviđenim podatcima sigurno biti drugačiji od onoga u podatcima na kojima je model treniran. Situaciju kada za neki klasifikacijski ili regresijski problem koristimo prejednostavan model nazivamo **podnaučenost**. Obratno, situaciju u kojoj koristimo presložen model nazivamo **prenaučenost**.

- **Podnaučenost** (engl. *underfitting*) – model \mathcal{H} je prejednostavan u odnosu na stvarnu klasifikaciju/funkciju, što rezultira lošim predikcijama modela i na viđenim i na neviđenim primjerima;
- **Prenaučenost** (engl. *overfitting*) – model \mathcal{H} je previše složen u odnosu na stvarnu klasifikaciju/funkciju, što rezultira dobrom predikcijama na viđenim primjerima ali lošim predikcijama na neviđenim primjerima.

► PRIMJER

Vratimo se opet na primjer s pravcima, kružnicama i krumpirima. Za model pravaca (\mathcal{H}_1) očigledno je da je taj model prejednostavan, budući da niti na viđenim primjerima ne može ostvariti nisku empirijsku pogrešku (pravac naprosto ne možemo saviti oko kružno raspoređenih primjera). Dakle, model odgovara situaciji podnaučenosti. S druge strane, model krumpira (\mathcal{H}_3) je presložen, pa će ga biti lako prenaučiti. To znači da hipoteza iz \mathcal{H}_3 može točno klasificirati sve viđene primjere (i tako ostvariti empirijsku pogrešku jednaku nuli), ali neće točno klasificirati mnoge neviđene primjere. To je zato što se hipoteze iz \mathcal{H}_3 previđe prilagođavaju šumu, a šum na neviđenim primjerima neće biti isti. Razmotrimo sljedeću situaciju:



Na lijevoj strani je slika ulaznog prostora za naš označeni skup primjera \mathcal{D} , na kojem se model \mathcal{H}_3 prenaučio (prilagodio se dvama neispravno označenim primjerima). Na desnoj strani je slika istog ulaznog prostora, ali s nekim još neviđenim primjerima, na kojima modeli \mathcal{H}_2 i \mathcal{H}_3 nisu učeni. Budući da je šum stohastičke prirode, na tim neviđenim primjerima šum će vrlo vjerojatno biti drugačiji nego na skupu \mathcal{D} . To znači da će hipoteza h_3 , dobivena modelom \mathcal{H}_3 koji je presložen i zato sklon prenaučenosti, pogrešno klasificirati neke (moguće mnoge) neviđene primjere. S druge strane, hipoteza h_2 iz modela kružnice \mathcal{H}_2 , koji nije dovoljno složen da se previše prilagodi šumu na skupu \mathcal{D} , točnije će klasificirati neviđene primjere.

Zaključujemo, dakle, da je u ovom slučaju model \mathcal{H}_1 prejednostavan i rezultira podnaučenim hipotezama, model \mathcal{H}_3 je presložen i rezultira prenaučenim hipotezama, dok bi model \mathcal{H}_2 bio baš taman, ni prejednostavan ni presložen.

Očito, idealan model bio bi onaj koji nije niti prejednostavan (tako da se ne može podnaučiti) niti presložen (tako da se ne može prenaučiti). A ako pored ovoga možemo ili trebamo birati, dajemo prednost što jednostavnijim modelima. Naime, jednostavniji modeli imaju niz prednosti nad složenim modelima:

- **Bolja generalizacija** – jednostavniji model ima veću šansu da će dati bolje predviđanja na još neviđenim podatcima;
- **Lakše učenje/uporaba** – jednostavniji model ima manje parametara, pa ga je lakše (brže) naučiti, i lakše ga je (brže) koristiti za predikciju;
- **Lakše tumačenje** – jednostavniji je model lakše tumačiti (usporedite model s tri parametra koji opisuje implicitnu jednadžbu pravca i neuronsku mrežu sa 10k parametara). Ovo je bitno ako želimo razmumijeti podatke i moći objasniti klasifikacijsku odluku modela, a ne samo napraviti sustav koji radi vrlo dobru predikciju, ali zapravo funkcioniра као crna kutija.

Naravno, očito je pak da model ne smije biti prejednostavan, jer prejednostavan model ne može opisati ni podatke koje imamo, a kamoli još neviđene podatke.

Trebamo, dakle, odabrati model koji točno odgovara **pravoj složenosti** problema koji nastojimo naučiti. To nas dovodi do vrhunca današnjeg predavanja: kako, u neizbjježnom prisustvu šuma u podatcima, odabrati model optimalne složenosti, odnosno model s najmanjom pogreškom generalizacije? Ispostavlja se da je to sveprisutan problem u strojnom učenju, poznat

10

11

pod nazivom **odabira modela** (engl. *model selection*). Pogledajmo to malo detaljnije.

12

7 Odabir modela

Dakle, moramo odabrati optimalan model \mathcal{H} , optimalan u smislu da se neće ni podnaučiti ni prenaučiti. Taj odabir često radimo unutar neke **familije modela**. Budući da je model \mathcal{H} skup, znači familija modela je zapravo skup skupova, npr:

$$\{\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_k\}$$

Primijetite da smo ovdje na **jednoj razini više** nego što smo bili ranije, kada smo pričali o modelu: sada smo na razini skupa modela (skup skupova funkcija), dok smo prije bili na razini jednog modela (skup funkcija).

Razmotrimo neke primjere odabira modela. Na primjer, kod klasifikacije, možemo odabrati hoće li granice između klasa biti pravci, kružnice ili nekakve krivulje još veće nelinearnosti. Slično, kod regresije možemo birati stupanj polinoma: počevši od pravca preko parabole pa sve do neke visoko nelinearne krivulje. Zapravo, kada biramo model, često upravo biramo koliki će biti **stupanj nelinearnosti**. Onda modele možemo posložiti u linearni uređaj s obzirom na složenost: od manje složenih do vrlo složenih modela. Npr., kod regresije: pravac, parabola, polinom trećeg stupnja, polinom četvrtog stupnja, itd.

Ako složenost modela možemo nekako kvantificirati i dovesti u vezu s nekim parametrom koji upravlja tom složenošću, onda takav parametar nazivamo **hiperparametar modela**. Npr., stupanj nelinearnosti polinoma je hiperparametar regresijskog modela. Zašto je to hiperparametar a ne parametar? Zato što to nije običan parametar modela. Obični parametri su oni koji čine vektor θ , po kojemu algoritam strojnog učenja radi minimizaciju empirijske pogreške i koji određuju hipotezu h iz modela \mathcal{H} . S druge strane, hiperparametar određuje koji skup \mathcal{H}_i ćemo uopće koristiti pri optimizaciji parametara θ . Dakle, **hiperparametri određuju model**, dok **parametri određuju hipotezu**. U tom smislu hiperparametri su jednu razinu iznad običnih parametara, i zato ih tako zovemo (“hiper” u smislu “iznad”).

Ovdje primijetimo tri stvari. Prvo, budući da hiperparametri određuju model, kada kažemo **odabir modela**, to je potpuno isto kao da smo rekli **optimizacija hiperparametara**. Npr., pretpostavimo da želimo odabrati regresijski model optimalne složenosti iz familije modela koja sadrži polinoma prvog, drugog, trećeg, itd. stupnja. Odabratи optimalan model isto je kao i optimizirati (tj. odabratи optimalan) stupanj polinoma, koji je hiperparametar modela.

Drugo, primijetimo da kod odabira i treniranja modela postoji jasan vremenski slijed. Budući da hiperparametri određuju model, očito je da **prvo moramo optimirati hiperparametre**, čime odabiremo model. Tek nakon toga možemo optimizirati parametre, čime odabiremo hipotezu. Dakle, odabir modela uvijek prethodi treniranju modela.

Treća stvar koju treba primijetiti jest da postoji jasna podjela odgovornosti između algoritma strojnog učenja i onoga koji radi odabir modela. Naime, **odabir modela moramo napraviti mi** (čovjek), dok **treniranje modela radi algoritam strojnog učenja**. Ovo je važno, pa ponovimo: algoritam strojnog učenja nije zadužen za optimizaciju hiperparametara, nego samo za optimizaciju parametara. Odgovornost odabira optimalnog modela je na nama, a ne na algoritmu strojnog učenja. Ako algoritmu strojnog učenja damo suboptimalan model (dakle model koji je prejednostavan ili presložen), ne trebamo se čuditi što će naučen model (tj. hipoteza) raditi loše. Nažalost, ovo se u praksi često zaboravlja, pa se nerijetko događa (pa i u recenziranim znanstvenim radovima) da se prikazuju rezultati dobiveni modelom za koji nije provedena optimizacija hiperparametara.

Spomenimo još da, premda odabir modela moramo napraviti mi a ne algoritam strojnog učenja, to ne znači da ga baš uvijek moramo napraviti ručno: možemo automatski isprobavati više različitih modela (tj. vrijednosti hiperparametara, koristeći, na primjer, neki heuristički postupak optimizacije, npr. genetičke algoritme) te svaki od tih modela trenirati algoritmom

strojnog učenja, te u konačnici onda odabratи najbolji model (onaj koji daje hipotezu koja najbolje generalizira).

Možda već možete zaključiti da je odabir modela jedan sumnjiv biznis, budući da moramo naći neku zlatnu sredinu, a ne znamo ni sami gdje bi ta bila. Zbog toga je u strojnom učenju razvijeno mnogo teorije o odabiru modela. U praksi, međutim, najčešće koristimo jednostavan empirijski postupak koji se zove **unakrsna provjera** ili **križna validacija** (engl. *cross-validation*).

8 Unakrsna provjera

Unakrsna provjera metoda je za procjenu sposobnosti generalizacije modela. Osnovna ideja je sljedeća. Unutar familije modela koju razmatramo, želimo odabratи onaj model koji najbolje generalizira, odnosno koji dobro radi na **neviđenim primjerima**. Međutim, budući da nemamo neviđene primjere (jer inače očito ne bi bili neviđeni), poslužit ćemo se jednim zgodnim trikom: izdvojiti ćemo iz našeg skupa primjera za učenje dio primjera, i oni će glumiti neviđene primjere. Model ih neće vidjeti kod učenja, nego ćemo naučeni model ispitivati na tim primjerima, da vidimo kako dobro generalizira. Primijetite, naravno, da to nisu doista neviđeni primjeri, jer smo ih očito mi vidjeli, međutim ti primjeri nisu korišteni za učenje modela, pa su dakle to neviđeni primjeri iz perspektive algoritma strojnog učenja.

Unakrsnu provjeru provodimo tako da skup primjera dijelimo na **skup za učenje** i **skup za ispitivanje**:

$$\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}$$

Skupovi moraju biti međusobno **disjunktni** (dakle, nepreklapajući). Tipično radimo podjelu 70:30 ili 60:40 ili slično, ovisno o tome koliko podataka imamo i kako detaljnju analizu rada modela želimo napraviti. Ovdje očito postoji kompromis: za detaljniju analizu rada modela i bolju procjenu pogreške modela bilo bi dobro imati što više primjera u ispitnom skupu, međutim ako u skupu za učenje nemamo dovoljno primjera, model uopće nećemo moći dobro naučiti. Ovu podjelu obično radimo slučajnim odabirom primjera, kako bismo osigurali da oba skupa ostanu reprezentativna, osim ako postoje dobri razlozi za drugačiji pristup.

Nakon što smo napravili podjelu skupa primjera na skup za učenje $\mathcal{D}_{\text{train}}$ i skup za ispitivanje $\mathcal{D}_{\text{test}}$, sada računamo dvije pogreške za $h \in \mathcal{H}$ koristeći ta dva skupa:

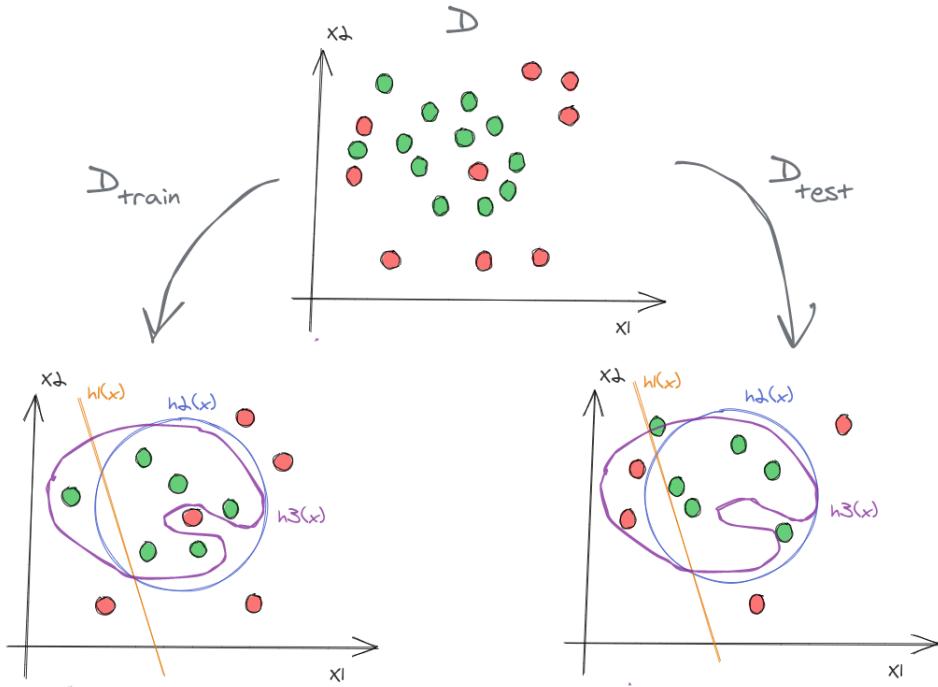
- **Empirijska pogreška učenja** (engl. *empirical training error*) – pogreška hipoteze h izračunata na skupu za učenje $\mathcal{D}_{\text{train}}$, tj. $E(h|\mathcal{D}_{\text{train}})$;
- **Empirijska ispitna pogreška** (engl. *empirical test error*) – pogreška hipoteze h izračunata na skupu za ispitivanje $\mathcal{D}_{\text{test}}$, tj. $E(h|\mathcal{D}_{\text{test}})$.

Ispitna pogreška, budući da je mjerena na skupu primjera koje algoritam nije “vidio” (tj. koji nisu korišteni za treniranje modela), odgovara pogrešci modela na neviđenim podatcima, odnosno kazuje nam koliko će model dobro generalizirati. Zato ispitnu pogrešku zovemo i **pogreška generalizacije**.

Primijetite da kod obje ove pogreške govorimo o **empirijskoj pogrešci**, tj. obje izračunavamo kao očekivanje (ili prosjek) funkcije gubitka na dotičnim skupovima primjera (za razliku od pravih pogrešaka, koje su nam nepoznate). Međutim, jednostavnosti radi, često se govorí samo o **pogrešci učenja** i **pogrešci ispitivanja** (dakle, bez pridjeva “empirijska”, koji se podrazumijeva).

► PRIMJER

Vratimo se opet na prethodni primjer s modelima pravaca (\mathcal{H}_1), kružnica (\mathcal{H}_2) i “krumpira” (\mathcal{H}_3). Pretpostavimo da smo naš skup označenih primjera \mathcal{D} razdijelili, slučajnim uzorkovanjem, na skup za učenje $\mathcal{D}_{\text{train}}$ i skup za ispitivanje $\mathcal{D}_{\text{test}}$, ovako:



U ovom slučaju, skup za učenje ima 11 primjera ($|D_{\text{train}}| = 11$), dok skup za ispitivanje ima 10 primjera ($|D_{\text{test}}| = 10$).

Pogledajmo najprije što se događa na skupu za učenje (lijeva slika). Na skupu za učenje treniramo naša tri modela, \mathcal{H}_1 , \mathcal{H}_2 i \mathcal{H}_3 . Algoritam strojnog učenja (koji god da je) dao nam je za svaki model optimalnu hipotezu, tj. onu koja minimizira empirijsku pogrešku na skupu za učenje. No, vidimo da na skupu za učenje hipoteza h_1 iz modela pravaca \mathcal{H}_1 loše klasificira primjere. Preciznije, njezina pogreška učenja je $E(h_1|D_{\text{train}}) = \frac{5}{11}$. Hipoteza h_2 iz modela kružnica \mathcal{H}_2 daje nešto točniju klasifikaciju na skupu skupu za učenje, i njezina je pogreška učenja jednaka $E(h_2|D_{\text{train}}) = \frac{2}{11}$. S druge strane, model krumpira \mathcal{H}_3 je visoke složenosti, i najbolja hipoteza h_3 iz tog modela može bez problema savršeno klasificirati sve primjere iz skupa za učenje, pa je njezina pogreška učenja jednaka $E(h_3|D_{\text{train}}) = 0$.

Pogledajmo sada kako će se te tri hipoteze, koje su sada naučene na skupu za učenje i fiksirane, provesti na ispitnome skupu (desni grafikon). Tu je situacija dosta drugačija. Zapravo, za hipotezu $h_1 \in \mathcal{H}_1$ situacija i nije toliko drugačija, jer je to podnaučena hipoteza iz prejednostavnog modela, pa ona loše klasificira primjere i na skupu za učenje i na skupu za ispitivanje. Njezina ispitna pogreška je $E(h_1|D_{\text{test}}) = \frac{2}{10}$. Ispitna pogreška hipoteze iz modela kružnica, $h_2 \in \mathcal{H}_2$, je $E(h_2|D_{\text{test}}) = \frac{1}{10}$. Konačno, ispitna pogreška hipoteze h_3 , koja je iz najsloženijeg modela, modela krumpira, je $E(h_3|D_{\text{test}}) = \frac{4}{10}$. Općenito, razumno je očekivati da će ispitna pogreška hipoteze biti veća nego njezina pogreška učenja. To je zato jer je hipoteza optimizirana na skupu za učenje, a ne na skupu za ispitivanje, što znači da će se hipoteza bolje prilagoditi skupu za učenje nego ispitnome skupu, koji algoritam nije ni vidio kod učenja. Međutim, to ne mora uvijek biti tako: zbog stohastičnosti uzorkovanja može se dogoditi da ispitna pogreška bude manja od pogreške učenja (ovdje nam se to dogodilo da je za hipotezu h_1).

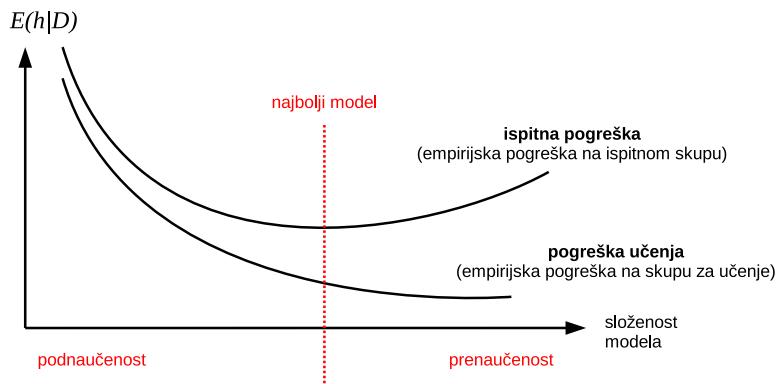
Razmotrimo sada odnose između pogreške učenja i pogreške ispitivanja za svaku od hipoteza. Što se ovdje dogodilo? Dogodilo se to da se hipoteza h_3 previše prilagodila šumu u skupu za učenje, ali takvog šuma nije više bilo u skupu za učenje (tamo je šum drugačiji). Zbog toga h_3 ima vrlo nisku pogrešku učenja, ali vrlo visoku ispitnu pogrešku. Drugim riječima, h_3 loše generalizira, jer loše klasificira neviđene primjere, odnosno model se prenaučio. S druge strane, hipoteza h_1 iz modela pravaca također loše generalizira (ispitna pogreška je visoka), ali ta hipoteza ima i visoku pogrešku učenja, što signalizira da se radi o podnaučenom modelu. Hipoteza h_2 iz modela kružnica najbolje prolazi u ovoj priči: ona doduše grijesi i na skupu za učenje i na ispitnom skupu, ali to je hipoteza koja najmanje grijesi na ispitnom skupu. Drugim riječima, hipoteza h_2 najbolje generalizira. To znači da je model \mathcal{H}_2 model optimalne složenosti za naš klasifikacijski problem definiran skupom

označenih primjera \mathcal{D} .

Sažmimo naša opežanja. Model koji je prejednostavan (\mathcal{H}_1) bit će podnaučen i dat će hipotezu koja ima visoku i pogrešku učenja i pogrešku ispitivanja. Model koji je presložen (\mathcal{H}_3) bit će prenaučen i dat će hipotezu koja doduše ima nisku pogrešku učenja, ali visoku ispitnu pogrešku. Model koji je optimalne složenosti (\mathcal{H}_2) dat će hipotezu čija je pogreška učenja između pogreške učenja podnaučenog i prenaučenog modela, dok će njezina ispitna pogreška biti manja i od one podnaučenog i prenaučenog modela.

Naša tri modela – \mathcal{H}_1 (pravci), \mathcal{H}_2 (kružnice) i \mathcal{H}_3 (krumpiri) – možemo postaviti u linearan (potpuni) uređaj prema njihovoj složenosti: model \mathcal{H}_1 je jednostavniji od modela \mathcal{H}_2 , koji je pak jednostavniji od modela \mathcal{H}_3 . Štoviše, mogli bismo definirati da model \mathcal{H}_2 uključuje u sebe i sve pravce (recimo da pravce smatramo degeneriranim kružnicama; dakle, to bi onda bio model kružnica i pravaca), dok bismo za model \mathcal{H}_3 mogli reći da uključuje kružnice i pravce kao posebne slučajevе. Tada imamo (budući da su modeli skupovi hipoteza): $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \mathcal{H}_3$. U tom slučaju možemo crtati grafikon pogreške učenja i pogreške ispitivanja kao funkcije složenosti modela: x -os tog grafikona odgovara složenosti modela (ili vrijednosti hiperparametra koji određuje složenost modela), a na y -osi je iznos funkcije pogreške učenja odnosno pogreške ispitivanja (v. tekst u nastavku).

Na temelju gornjeg primjera vidimo da pogreška na skupu za učenje, $E(h|\mathcal{D}_{\text{train}})$, pada sa složenošću modela, dok pogreška na ispitnom skupu, $E(h|\mathcal{D}_{\text{test}})$, tipično prvo pada pa zatim raste. Dakle, grafovi pogreške na skupu za učenje (pogreška učenja) i pogreške na skupu za ispitivanje (ispitna pogreška), kao funkcije složenosti modela, očekivano izgledaju ovako:



Optimalan model je upravo onaj koji minimizira ispitnu pogrešku $E(h|\mathcal{D}_{\text{test}})$. Naime, to je pogreška modela na neviđenim primjerima, pa nam ona govori koliko dobro model generalizira. Prema tome, između modela koje razmatramo (familija modela), želimo odabratи onaj model koji ima najmanju pogrešku na neviđenim primjerima jer to znači da taj model najbolje generalizira, tj. da ima najmanju pogrešku generalizacije. Modeli koji su manje složenosti od optimalnog modela (lijeva strana grafikona) nedovoljne su složenosti i oni su skloni podnaučenosti (visoka pogreška učenja i visoka ispitna pogreška). Suprotno, modeli složenosti veće od optimalnog modela (desna strana grafikona) jesu modeli prevelike složenosti i oni su skloni prenaučenosti (mala pogreška učenja ali velika ispitna pogreška).

Gornji graf vjerojatno je najvažniji graf u strojnom učenju. No, trebamo napomenuti da on predstavlja idealizirani slučaj: prikazuje **očekivano** ponašanje pogreške učenja i pogreške ispitivanja u ovisnosti o složenosti modela. U praksi, budući da uvijek radimo sa slučajnim uzorcima podataka, na našem konkretnom skupu primjera može doći do nekih odstupanja (npr., može se dogoditi da ispitna pogreška malo pada pa raste pa opet pada pa raste, ili da je pogreška učenja za model neke složenosti čak veća od ispitne pogreške). Međutim, kada bismo ponavljali uzorkovanje primjera, izračunavali vrijednosti pogrešaka za modele različitih složenosti, i zatim uprosječivali te vrijednosti, možemo očekivati da bismo dobili krivulje kakve smo prikazali na gornjem grafikonu.

Sažetak

- **Hipoteza** je funkcija koje primjere preslikavaju u oznake (klase ili brojeve), definirana do na parametre
- **Model** je skup hipoteza, indeksiran njihovim parametrima
- Učenje (treniranje) modela se svodi na **optimizaciju parametara** modela, što odgovara pretraživanju u skupu hipoteza
- Svaki algoritam strojnog učenja sastoji se od **modela, funkcije gubitka/pogreške i optimacijskog postupka**
- Modeli se općenito razlikuju po svojoj **složenosti** (sposobnosti da dosegnu nisku pogrešku učenja)
- Model koji je **podnaučen** ili **prenaučen** loše generalizira
- Odabir modela svodi se na **optimiranje hiperparametara** modela
- **Unakrsnom provjerom** može se procijeniti **pogreška generalizacije** i odabrati optimalan model

Bilješke

- [1] Ovo predavanje uglavnom slijedi strukturu drugog poglavlja iz ([Alpaydin, 2020](#)), uz neka proširenja. Predavanjem nije pokrivena tema **induktivne pristranosti** (engl. *inductive bias*) – tu temu trebate dodatno proučiti iz skripte (poglavlje 2.3), a proći ćemo ju zajednički na vježbama.
- [2] Pristupima temeljenima na dubokom strojnem učenju nastoji se zaobići ekstrakcija značajki, tako da neuronska mreža ne uči samo klasifikaciju/regresiju ulaznog primjera, već implicitno uči i ekstrakciju “značajki” iz sirovih podataka, odnosno **uči reprezentaciju** (engl. *representation learning*). Učenjem reprezentacija posao ekstrakcije značajki prebacuje se s čovjeka na algoritam strojnog učenja. Npr., umjesto da čovjek oblikuje i implementira algoritme za ekstrakciju relevantnih značajki iz slike (npr., detekcija kutova, segmentacija, tekstura i sl.), na ulaz neuronske mreže dovodi se slika u izvornome obliku (matrica trokomponentnih slikovnih elemenata) te se tijekom učenja početni slojevi neuronske mreže sami specijaliziraju za ekstrakciju značajki koje su relevantne za zadatak. Učenje reprezentacija pokazalo se posebno korisnim u računalnom vidu i u obradi prirodnog jezika, gdje je ono dovelo do značajnih napredaka. Za uvod u područje učenja reprezentacija, pogledajte ([Bengio et al., 2013](#)). Mi se na ovom predmetu nećemo baviti učenjem reprezentacija. Prepostavit ćemo da je primjer prikazan značajkama, međutim neće nas zanimati na koji smo točno način došli do tih značajki.
- [3] Redukcijom dimenzionalnosti nećemo se baviti na ovom predmetu. Međutim, tehnike za redukciju dimenzionalnosti izuzetno su korisne i u praksi je nužno da poznajete barem neke od njih. Dobar pregled tehnika redukcije dimenzionalnosti možete pronaći u ([Fodor, 2002](#)) i ([Burges, 2010](#)).
- [4] Možda izgleda da su tri parametra previše i da bi nam bila dosta dva jer imamo dvodimenzionalni prostor primjera, te da bismo mogli koristiti eksplisitni oblik jednadžbe pravca (podjeliti jednadžbu s θ_0) umjesto da koristimo implicitni oblik. No, to nije tako. S eksplisitnim oblikom gubimo mogućnost da definiramo “orientaciju” pravca (koja strana je pozitivna a koja negativna). Treba nam funkcija koja zapravo definira ravninu ugrađenu u 3D prostoru. Upravo to smo ovdje napravili. Na mjestu gdje ta ravnina siječe ravninu X-Y, tj. u točkama za koje $h(\mathbf{x}) = 0$, tu se nalazi granica između klasa. Općenito, za klasifikaciju u n -dimenzionalnom prostoru linearan klasifikacijski model treba imati $n + 1$ parametara.
- [5] Različite hipoteze imat će različite vektore parametara $\boldsymbol{\theta}$. Međutim, različiti vektori parametara $\boldsymbol{\theta}$ ne moraju nužno davati različite hipoteze. Na primjer, ako je ulazni prostor diskretan (npr. $\mathcal{X} = \mathbb{N}^n$), onda možemo imati pravce koji su malo različiti (dakle parametri $\boldsymbol{\theta}$ im se razlikuju), ali ipak daju identičnu klasifikaciju primjera u dvije klase, tj. funkcija h je jedna te ista

(kako je uobičajeno, jednakost funkcije ovdje definiramo **ekstenzionalno**: dvije funkcije su jednake ako jednako preslikavaju elemente iz domene u kodomenu, tj. $\forall x \in X. h_1(x) = h_2(x)$). To znači da različiti θ mogu dati identične funkcije $h(\mathbf{x}; \theta)$. Dakle, ne vrijedi nužno $h \mapsto \theta$. (Ako ste zainteresirani za ideju funkcijeske ekstenzionalnosti, pogledajte ovo: <https://ncatlab.org/nlab/show/function+extensionality>. Za one s još neiskorijenjenim filozofskim sklonostima, preporučam za početak pročitati o ekstenzionalnosti i odnosu prema intenzionalnosti, ovdje: <https://plato.stanford.edu/entries/logic-intensional/>).

6 Optimizacija i strojno učenje tijesno su povezani. S jedne strane, strojno učenje naveliko koristi algoritme optimizacije i gladno je za sve učinkovitim postupcima. S druge strane, strojno učenje nerijetko inspirira razvoj novih algoritama optimizacije. Vrlo dobar pregled optimizacije za strojno učenje možete pronaći u ([Sra et al., 2012](#)).

7 Empirijska pogreška hipoteze $E(h|\mathcal{D})$ na skupu označenih primjera \mathcal{D} je aproksimacija stvarne (ali nama nepoznate) pogreške hipoteze $E(h)$ na svim mogućim primjerima. Formalno:

$$E(h) = \mathbb{E}_{\mathbf{x}, y}[L(y, h(\mathbf{x}))] \approx \frac{1}{N} \sum_{i=1}^N L(y^{(i)}, h(\mathbf{x}^{(i)})) = E(h|\mathcal{D})$$

gdje je $\mathbb{E}_{\mathbf{x}, y}[L(y, h(\mathbf{x}))]$ **očekivanje funkcije gubitka** po svim označenim primjerima (\mathbf{x}, y) (par (\mathbf{x}, y) tretiramo kao slučajnu varijablu koja se pokorava nekoj nama nepoznatoj distribuciji). Ovo očekivanje aproksimiramo (procjenjujemo) na skupu označenih primjera \mathcal{D} kao srednju vrijednost funkcije gubitka na tom skupu, i to je naša empirijska pogreška $E(h|\mathcal{D})$. U statističkom smislu, empirijska pogreška je procjena prave pogreške hipoteze procijenjena na uzorku \mathcal{D} . Kako bi ova procjena bila dobra, skup označenih primjera \mathcal{D} treba biti **repräsentativan** (tj. vjerno predstavljati skup svih mogućih označenih primjera), što se u pravilu postiže ako je skup \mathcal{D} uzorkovan slučajno.

8 Tri komponente mogu se sažeti sljedećom “jednadžbom”: **učenje = reprezentacija + evaluacija + optimizacija**. Ovakva “trodioba vlasti” pokazala se kao vrlo koristan način gledanja na algoritme strojnog učenja jer nam omogućava da uočavamo sličnosti i razlike između algoritama. Stoga ćemo na ovom predmetu inzistirati na tome da svaki algoritam strojnog učenja promatramo kroz prizmu ovih triju komponenata. U retrospektivi, definicija strojnog učenja kroz ove tri komponente doimaju se gotovo banalnom, međutim netko se toga ipak trebao prvi dosjetiti. Ne znam tko je to bio. [Alpaydin \(2020\)](#) tri komponente spominje u drugom poglavju svoje knjige (prvo izdanje je još iz 2004. godine), dok se gornja jednadžba nalazi u [Domingos \(2012\)](#). Zanimljivo je da navedene tri komponente lijepo korespondiraju s različitim vrstama **induktivne pristranosti** algoritma strojnog učenja: tako model definira **pristranost jezikom** (pristranost ograničenja), dok funkcija gubitka i optimizacija definiraju **pristranost preferencijom** (pristranost pretraživanja).

9 Funkcija argmin daje minimizator funkcije, tj. $\text{argmin}_x f(x) = x_m$ akko $f(x_m) = \min_x f(x)$. Pazite da ne brkate funkciju min (minimum funkcije) i funkciju argmin (minimizator funkcije). Na primjer, $\min_x (x - 1)^2 = 0$, međutim $\text{argmin} (x - 1)^2 = 1$. Zašto ovdje koristimo funkciju argmin a ne funkciju min? Zato jer je glavni rezultat optimizacijskoj postupka hipoteza h^* (ili, ekvivalentno, parametri θ^*) koja minimiziraju pogrešku. Naravno, često će nas zanimati i koliko iznosi ta pogreška, međutim to uvijek možemo izračunati jednom kada smo pronašli h^* odnosno θ^* , dok obrat ne vrijedi (ako samo znamo koliko iznosi pogreška, iz toga ne možemo dobiti hipotezu).

10 Ideja da u načelu preferiramo jednostavne modele nad složenijima općenito je prihvaćena u znanosti te je poznata kao **načelo parsmonije** ili **Occamove britve**: ako su dva modela jednake prediktivne točnosti, ispravan model je vjerojatno onaj koji je jednostavniji. No to, naravno, ne znači da jednostavni modeli moraju dati veću predikcijsku točnost. Za raspravu o primjenjivosti načela Occamove britve u strojnom učenju pogledajte ([Domingos, 1999](#)).

11 U zadnjih nekoliko godina, a pogotovo s porastom primjene modela dubokih učenja, počelo se tematizirati pitanje **interpretabilnosti i pravednosti** predikcijskih modela. Složeni modeli, poput neuronskih mreža, u tom su smislu posebno problematični jer su netransparentni (*black-box*) te relativno podložni amplificiranju postojećih pristranosti u skupovima podataka. Ovakva razmatranja dovela su do povećanog interesa za tzv. **objašnjivom umjetnom inteligencijom** (engl. *explainable AI, XAI*). Pogledati, na primjer: <https://xaitutorial2020.github.io/>. Vrlo dobar uvod u problem pravednosti algoritama strojnog učenja daje (još nedovršena) knjiga [Barocas et al. \(2018\)](#), dostupna online

na <https://fairmlbook.org/>, te pregledni rad (Mehrabi et al., 2019). Poznata knjiga Cathy O'neil (O'neil, 2016) izvrsno tematizira posljedice netransparentnih i nepravednih algoritama na pojedinca i društvo.

- 12 Problem odabira modela izravno je povezan s tzv. **pretpostavkom induktivnog učenja**, kako ju je definirao Mitchell (1997):

"Svaka hipoteza koja koja na dovoljno velikom skupu primjera za učenje dobro aproksimira ciljnu funkciju također će dobro aproksimirati ciljnu funkciju na drugim, neopaženim primjerima."

Ova hipoteza se u suštini ne može dokazati a da se ne uvedu neke dodatne pretpostavke o ciljnoj funkciji (klasifikaciji). Također je potrebno precizno definirati što znači "dobro aproksimirati". Takvim pitanjima bavi se **teorija računalnog učenja** (engl. *computational learning theory, COLT*), u kojoj su razvijeni teorijski radni okviri za izračun gornje ografe pogreške generalizacije. Npr., radni okvir **vjerojatno približno točno** (engl. *probably approximately correct, PAC*), pomoću kojega možemo izračunati vjerojatnost da će se, za skup primjera određene veličine, pogreška generalizacije biti manja od nekog iznosa. Mitchellova hipoteza induktivnog učenja govori nam da će empirijska pogreška hipoteze, $E(h|\mathcal{D})$, konvergirati stvarnoj pogrešci hipoteze, $E(h)$, s porastom veličine skupa označenih primjera za učenje \mathcal{D} . Međutim, u praksi imamo konačan skup primjera za učenje. Nešto praktičniju verziju hipoteze induktivnog učenja dao je Sam Roweis (<https://cs.nyu.edu/~roweis/csc2515-2004/notes/lec1x.pdf>):

"Ako je pogreška hipoteze na dovoljno velikom skupu primjera za učenje mala i ako model nije suviše složen, hipoteza će vjerojatno dobro klasificirati i nove, slične primjere."

Ovako formulirana pretpostavka induktivnog učenja zapravo tvrdi da je generalizacija moguća, no pod trima uvjetima: (1) empirijska pogreška na skupu za učenje je mala (tj. model nije podnaučen), (2) model nije presložen (tj. model nije prenaučen) i (3) neviđeni primjeri slični su primjerima na kojima je model učen (tj. skup za učenje je reprezentativan uzorak čitave populacije primjera). Zainteresirane za teoriju računalnog strojnog učenja upućujem na izvrsnu knjigu (Shalev-Shwartz and Ben-David, 2014).

- 13 Automatska optimizacija hiperparametara, odnosno automatski odabir modela, zanimljiv je i netrivialan problem u strojnem učenju. Potpunom (ili djelomičnom) automatizacijom primjene algoritma strojnog učenja bavi se područje **automatiziranog strojnog učenja** (engl. *automated machine learning, AutoML*). Idealan cilj jest automatizirati sve korake primjene algoritma strojnog učenja (v. dio 2), uključivo i one koje sada još uvjek obavlja čovjek: od pripreme podataka preko ekstrakcije značajki do odabira modela, njegovog vrednovanja i dijagnostike. Više o automatiziranom strojnem učenju možete pronaći u (Hutter et al., 2019).
- 14 Za pregled različitih pristupa obabira modela, pogledajte (ne baš ažuriranu, ali temeljnu) bibliografiju na <http://www.modelselection.org>. Dobra referenca na tu temu je (Anderson and Burnham, 2004). Pregled postupaka za unakrsnu provjeru možete naći u (Arlot and Celisse, 2010).
- 15 Postupak je nešto složeniji kada, uz provjeru pogreške modela na ispitnome skupu, želimo također provesti **odabir modela**. U tom slučaju skup primjera dijelimo na tri međusobno disjunktna skupa: skup za učenje, skup za provjeru i skup za ispitivanje. Više o tome kasnije, u predavanju o vrednovanju modela.
- 16 Podjelu primjera u skup za učenje i skup za ispitivanje obično radimo slučajnim odabirom, kako bi oba skupa bila reprezentativni uzorci populacije svih primjera. Međutim, u nekim situacijama želimo više kontrole nad time koji će primjeri završiti u skupu za učenje a koji u skupu za ispitivanje. Tipične takve situacije jesu skupovi podataka s vremenskom komponentom, kod kojih u želimo da skup za ispitivanje sadrži novije primjere jer neviđeni primjeri će tipično biti novi primjeri, ili skupovi podataka s mnogo kategorija s neuravnoteženim brojem primjera, gdje želimo da skup za učenje i skup za ispitivanje vjerno zrcale distribuciju klasa, za što onda koristimo **stratificirano uzorkovanje** (engl. *stratified sampling*).

Literatura

- E. Alpaydin. *Introduction to machine learning*. MIT press, 2020.
- D. Anderson and K. Burnham. Model selection and multi-model inference. *Second. NY: Springer-Verlag*, 63(2020):10, 2004.
- S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.
- S. Barocas, M. Hardt, and A. Narayanan. Fairness and machine learning. fairmlbook.org, 2018.
- Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.
- C. J. Burges. *Dimension reduction: A guided tour*. Now Publishers Inc, 2010.
- P. Domingos. The role of Occam’s razor in knowledge discovery. *Data mining and knowledge discovery*, 3(4):409–425, 1999.
- P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.
- I. K. Fodor. A survey of dimension reduction techniques. Technical report, Lawrence Livermore National Lab., CA (US), 2002.
- F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.
- N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A survey on bias and fairness in machine learning. *arXiv preprint arXiv:1908.09635*, 2019.
- T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
- C. O’neil. *Weapons of math destruction: How big data increases inequality and threatens democracy*. Broadway Books, 2016.
- S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.
- S. Sra, S. Nowozin, and S. J. Wright. *Optimization for machine learning*. MIT Press, 2012.

3. Linearna regresija

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v1.12

Prošli puta uveli smo osnovne koncepte strojnog učenja: **hipotezu**, koja je funkcija koja preslikava iz ulaznih podataka u oznaće i definirana je s parametrima theta, i **model**, koji je skup hipoteza indeksiran parametrima theta. Rekli smo da se strojno učenje svodi na pronalaženje najbolje hipoteze u modelu, odnosno nalaženje optimalnih parametara. Pritom kao kriterij uzimamo pogrešku hipoteze, koju mjerimo na skupu označenih primjera, i tu mjeru zovemo **empirijska pogreška**. Zaključili smo da svaki algoritam SU ima **tri komponente**: model, funkciju gubitka (čije je očekivanje funkcija pogreške) i optimizacijski postupak.

Danas ćemo pogledati jedan konkretan algoritam strojnog učenja za **regresiju**. Prisjetimo se: regresija znači predviđanje numeričkih vrijednosti, dakle oznaće \mathcal{Y} su brojevi. Npr., predviđanje cijene stanova u Bostonu. Mi ćemo se fokusirati na jedan vrlo važan algoritam, a to je **linearan model regresije**, koji je vrlo važan i u strojnem učenju i u statistici.

U statistici, regresija primarno služi za **objašnjavanje odnosa** između dviju skupova varijabli te za statističko zaključivanje (npr., intervali pouzdanosti parametara, testovi statističke značajnosti prediktora). U strojnem učenju, regresija nas zanima prije svega zbog **predviđanja**, dok nas objašnjavanje u pravilu manje zanima. Posljedica toga je da su pogledi na regresiju iz kuta statistike i iz kuta strojnog učenja malo različiti, premda se radi se o istoj stvari.

Današnje predavanje je strukturirano tako da najprije pogledamo najjednostavniji scenarij, a onda ćemo polako ići prema složenijim scenarijima.

1 Jednostavna regresija

Kao motivirajući primjer, zamislimo da želimo napraviti model koji predviđa cijenu nekretnine na temelju njezinih značajki. Očito, to je regresijski problem, jer predviđamo brojčanu vrijednost. Budući da se radi o nadziranom strojnem učenju, trebaju nam i označeni podatci. Općenito, raspolaćemo označenim skupom podataka, $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}$, gdje su primjeri n -dimenzijski vektori značajki, $\mathbf{x} \in \mathbb{R}^n$, a oznaće su realni brojevi, $y \in \mathbb{R}$. U našem slučaju, oznaće y bile bi cijene nekretnina u Bostonu: za svaku nekretninu imamo niz značajki \mathbf{x} koje ju opisuju, i imamo njezinu cijenu y u USD. Želimo napraviti model koji za nove, još neviđene nekretnine može predvidjeti njihovu cijenu.

Kod regresije, hipoteza h preslikava primjere u oznaće, dakle $h : \mathbb{R}^n \rightarrow \mathbb{R}$, dakle preslikava iz ulaznog prostora (prostora primjera) u brojčane vrijednosti. Npr., hipoteza h za neku nekretninu opisanu značajkama \mathbf{x} izračunava njezinu cijenu, $h(\mathbf{x})$. Kod regresije, varijable koje čine vektor \mathbf{x} nazivamo **ulazne varijable** (ili **nezavisne varijable** ili **prediktorske varijale** ili **kovarijati**), a izlaz y nazivamo **izlazna varijabla** (ili **zavisna varijabla** ili **kriterijska varijabla**).

Danas ćemo se fokusirati na **linearu regresiju**. Kod linearne regresije, izlaz je linearna kombinacija ulaznih varijabli. Odabirom takvog modela, mi smo implicitno prepostavili da je izlazna varijabla linearno ovisna o ulaznim varijablama. Naš model (odnosno skup hipoteza definiranih do na parametre) definiran je ovako:

$$h(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Tipično, kod regresije vektor parametara modela umjesto sa θ označavamo sa \mathbf{w} . Parametri w_0, \dots, w_n određuju utjecaj svake značajke na vrijednost izlazne varijable. U strojnom učenju ti se parametri često nazivaju **težine**, dok se u statistici nazivaju **regresijskim koeficijentima** ili **beta-koeficijentima** i označavaju sa β_0, \dots, β_n .

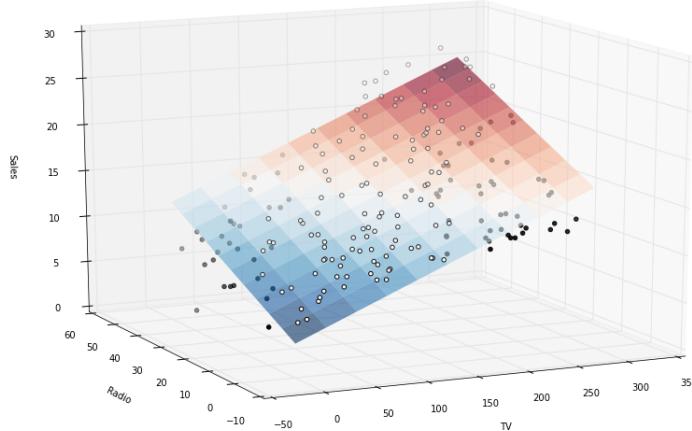
Za jednodimenzionalni ulazni prostor, $n = 1$, hipoteza će biti **pravac**. To je tako zvana **jednostavna regresija** (engl. *simple regression*). To je najjednostavniji linearni model regresije, kod kojega imamo samo jednu ulaznu varijablu:

$$h(x; w_0, w_1) = w_0 + w_1 x$$

► PRIMJER

- (1) Uštedjeljiva s obzirom na dob (raste više-manje linearno)
- (2) Cijena automobila s obzirom na kilometražu (naglo padajuća)

Ako je $n = 2$, onda hipoteza opisuje **ravninu** u vektorskome prostoru koji je definiran do-menom $\mathcal{X} = \mathbb{R}^2$ i kodomenom $\mathcal{Y} = \mathbb{R}$ (preciznije, vektorski prostor dobivamo kao kartezijev umnožak tih dvaju skupova). To izgleda ovako:



Ako $n > 2$, hipoteza h opisuje **hiperravninu** u prostoru $\mathbb{R}^n \times \mathbb{R}$. Položaj te hiperravnine određen je vektorom težina \mathbf{w} .

Pogledajmo sada malo detaljnije **algoritam linearne regresije**. Prisjetimo se najprije da je svaki algoritam strojnog učenja definiran trima komponentama: modelom, funkcijom gubitka i postupkom optimizacije. Radi jednostavnosti, ograničimo se na jednostavnu regresiju, $n = 1$. Model smo već definirali. Trebaju nam još funkcija gubitka i optimizacijski postupak.

Krenimo od **funkcije gubitka**. Prisjetimo se: to je pogreška koju hipoteza čini na svakom pojedinačnom primjeru x , ako je točna oznaka primjera y . Kod regresije, za funkciju gubitka (oznaka L) tipično se koristi **kvadratno odstupanje** između ciljne i predvidene vrijednosti:

$$L(y, h(x)) = (y - h(x))^2$$

U statistici, odstupanje ciljne vrijednosti od predviđene vrijednosti, $y - h(x)$, naziva se **rezidual**. Možemo onda reći da je funkcija gubitka jednaka **kvadratu reziduala**.

Ovdje se legitimno možemo zapitati: zašto bismo koristili baš kvadratno odstupanje kao funkciju gubitka? Zašto ne samo razliku između ciljne i predviđene vrijednosti? Zato što je

gubitak gubitak, neovisno o smjeru. Ako bismo računali samo razlike, pozitivni i negativni gubitci bi se međusobno poništavali, a to ne želimo. No, zašto onda ne koristimo absolutnu vrijednost te razlike? Odgovor se krije u potrebi da optimiziramo funkciju pogreške. Naime, funkcija absolutne vrijednosti nije derivabilna, dok funkcija kvadratnog odstupanja jest. Ako funkcija gubitka nije derivabilna, onda to sigurno neće biti i funkcija pogreške, koja je definirana kao njezino očekivanje. Zato dajemo prednost kvadratnom odstupanju. No, vidjet ćemo kasnije da ima i dubljih razloga zašto koristiti kvadratno odstupanje.

Što je s **funkcijom pogreške**? Prisjetimo se: gubitak i pogreška nisu isto. Rekli smo na prethodnom predavanju da je pogreška hipoteze zapravo **očekivana vrijednost gubitka hipoteze**, koju empirijski izračunavamo kao prosječnu vrijednost funkcije gubitka na skupu označenih primjera. Slično ćemo definirati i pogrešku hipoteze regresije:

$$E(h|\mathcal{D}) = \frac{1}{2} \sum_{i=1}^N L(y^{(i)}, h(x^{(i)})) = \frac{1}{2} \sum_{i=1}^N (y^{(i)} - h(x^{(i)}))^2$$

s tom razlikom da umjesto prosjeka, dakle umjesto $1/N$, koristimo $1/2$ zbog kasnije matematičke jednostavnosti. Ta sitna izmjena, međutim, nema baš nikakvog utjecaja na optimizaciju: minimizator (odnosno vrijednost za koju funkcija poprima minimum) bit će isti, neovisno o konstanti kojom množimo funkciju gubitka. Ovu funkciju pogreške nazivamo **funkcija kvadratne pogreške** (engl. *quadratic error function*). 4

Treća komponenta algoritma regresije jest **optimizacijski postupak**: potraga za onom hipotezom u modelu, $h \in \mathcal{H}$, koja minimizira empirijsku pogrešku. To jest:

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} E(h|\mathcal{D}) = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^N (y^{(i)} - h(x^{(i)}))^2$$

Budući da je hipoteza h indeksirana parametrima (w_0, w_1) , mi zapravo tražimo parametre modela koji minimiziraju pogrešku, tj.:

$$(w_0, w_1)^* = \underset{w_0, w_1}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^N (y^{(i)} - (w_1 x^{(i)} + w_0))^2$$

Razmotrimo ovaj optimizacijski problem s matematičkog stajališta. Imamo funkciju dviju varijabli i tražimo njezin minimum. Što bi nam prvo trebalo pasti na pamet? Pa, trebali bismo pronaći nultočke derivacije ove funkcije: derivirati funkciju, izjednačiti je s nulom te rješiti sustav jednadžbi. (Primijetimo i da je ovo konveksna funkcija, pa će derivacija prvog reda imati samo jednu nultočku, i ta je nultočka onda sigurno minimizator funkcije.) Pokažimo kako bi to izlgedalo. Deriviramo funkciju pogreške i izjednačavamo ih s nulom:

$$\nabla_{w_0, w_1} E(h|\mathcal{D}) = 0$$

Budući da se radi o funkciji dvije varijable, računamo parcijalne derivacije po prvom parametru (w_0) i po drugom parametru (w_1) i izjednačavamo ih s nulom:

$$\begin{aligned} \frac{\partial}{\partial w_0} \left[\frac{1}{2} \sum_i^N (y^{(i)} - (w_1 x^{(i)} + w_0))^2 \right] &= 0 \\ \frac{\partial}{\partial w_1} \left[\frac{1}{2} \sum_i^N (y^{(i)} - (w_1 x^{(i)} + w_0))^2 \right] &= 0 \end{aligned}$$

Nakon nekoliko koraka, koje ćemo ovdje preskočiti jer nam nisu naročito interesantni, došli bismo do sljedećeg rješenja: 5

$$\begin{aligned} w_0 &= \bar{y} - w_1 \bar{x} \\ w_1 &= \frac{\sum_i^N x^{(i)} y^{(i)} - N \bar{x} \bar{y}}{\sum_i^N (x^{(i)})^2 - N \bar{x}^2} \end{aligned}$$

gdje su \bar{x} i \bar{y} srednje vrijednosti značajke x odnosno oznake y kroz sve primjere u skupu \mathcal{D} .

Uočimo ovdje dvije stvari. Prva je da se naša optimizacija svela na jednostavno uvrštavanje brojeva u formulu: želim li pronaći parametre (w_0, w_1) koji minimiziraju funkciju kvadratne pogreške, sve što trebam napraviti jest primijeniti gornje formule. Za ovakve probleme, gdje je minimizator dan formulom i optimizacija se, umjesto nekog heurističkog postupka, svodi na uvrštavanje brojeva u tu formulu, kažemo da imaju **rješenje u zatvorenoj formi** (engl. *closed form solution*). To znači da je rješenje neki matematički izraz koji koristi “široko prihvaćene” funkcije i operatore (logaritam, korijen, trigonometrijske funkcije i sl.) te se može izračunati u konačnom broju koraka. U strojnom učenju jako volimo kada naš optimizacijski problem ima rješenje u zatvorenoj formi jer to u pravilu znači da je optimizacija jednostavna i egzaktna.

Drugu stvar koju želim da uočite jest da su formule za w_0 i w_1 zapravo različite, i da je formula za w_1 zapravo poprilično ružna. Ono što ovdje doduše ne možete uočiti, ali biste se u to lako mogli uvjeriti da to pokušate izvesti, jest da bismo za model s tri parametara, (w_0, w_1, w_2) , dobili složenije formule, za model sa četiri parametra još složenije, itd. To nije idealno, jer želimo općenito rješenje optimizacijskog postupka, u smislu da jednadžba rješenja ne ovisi o broju parametara (konkretno rješenje, naravno, mora ovisiti o konkretnom broju parametara kao što ovisi i o konkretnom skupu označenih primjera). Kao što (možda?) slutite, način da postignemo takvu općenitost bit će da rješenje optimizacijskog problema izvedemo u matričnoj formi (jer će to rješenje onda vrijediti za matrice dizajna i vektore oznaka proizvoljnih dimenzija). I, doista, to ćemo i napraviti (nekoliko stranica ispod).

2 Vrste regresije

Nakon što smo pogledali ovaj jednostavan slučaj regresije, pokušajmo poopćiti naš pristup. Bavili smo se jednostavnim regresijom, koja ima samo jednu ulaznu varijablu. To je ograničeno korisno: stvarno zanimljivi fenomeni iz stvarnog života uglavnom se ne mogu dobro opisati samo jednom nezavisnom varijablom. Srećom, jednostavna regresija je samo jedna vrsta regresije. Postoje i druge vrste regresije. Konkretno, prema broju **ulaznih (nezavisnih, prediktorskih)** varijabli, regresija može biti **jednostruka regresija** (ili **jednostavna regresija**), kod koje $n = 1$ (na ulazu je samo jedna značajka) ili **višestruka regresija** (ili **multipla regresija**), kod koje $n > 1$ (primjer je opisan s više značajki). S druge strane, prema broju **izlaznih (zavisnih, kriterijskih)** varijabli, regresija može biti **univarijatna regresija** (ili **jednoizlazna regresija**), kod koje $h(\mathbf{x}) = y$, ili **multivarijatna regresija** (ili **višeizlazna regresija**), kod koje $h(\mathbf{x}) = \mathbf{y}$ (izlazna varijabla također je vektor). Kada ukrižimo ove dvije podjele, dobivamo četiri vrste regresije, koje tablično možemo prikazati ovako:

	Jedan izlaz (y)	Više izlaza (\mathbf{y})
Jedan ulaz (x)	(Univarijatna) jednostavna	Multivarijatna jednostavna
Više ulaza (\mathbf{x})	(Univarijatna) višestruka	Multivarijatna višestruka

Tipično se podrazumijeva da je regresija univarijatna; ako to nije slučaj, onda je dobro da se eksplicitno kaže da je multivarijatna. Zbog toga se univarijatna višestruka regresija u literaturi tipično jednostavno naziva **višestruka regresija** (engl. *multiple regression*).

Mi ćemo se na ovom predmetu fokusirati upravo na **višestruku regresiju**, budući da se ona najčešće koristi, a i najbolje korespondira s klasifikacijskim modelima kojima ćemo se baviti kasnije. Dakle, fokusirat ćemo se na regresiju s više ulaza i s jednim izlazom, tj. za svaki primjer, opisan nizom značajki, predviđamo jedan broj. Npr., cijena nekretnina u Bostonu, gdje je nekretnina opisana nizom značajki (broj soba, udaljenost od glavne prometnice, ima li lift, jesu li susjedi podnošljivi, itd.). Univarijatna jednostavna regresija je, naravno, samo poseban slučaj univarijatne višestruke regresije gdje je $n = 1$.

6

7

3 Tri komponente algoritma linearne regresije

Pogledajmo sada opet tri osnovne komponente algoritma linearne regresije, i to višestruke, tj. one za koju $n > 1$.

1. Model

$$h(\mathbf{x}; \mathbf{w}) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = \sum_{j=1}^n w_jx_j + w_0$$

Kao što smo već sugerirali gore, izračun će biti mnogo jednostavniji ako pređemo na matrični račun. U tu svrhu svaki vektor primjera $\mathbf{x}^{(i)}$ proširit ćemo jednom tzv. **dummy značajkom**, $x_0^{(i)} = 1$. Vrijednost te značajke uvijek je jednaka 1, i služi tome da se pomnoži težinom w_0 , pa da sve težine možemo tretirati jednako. Model je onda:

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

gdje je $\mathbf{w}^T \mathbf{x}$ skalarni produkt vektora \mathbf{w} i \mathbf{x} , zapisan kao skalarni produkt vektor-retka \mathbf{w}^T (transponirani vektor-stupac) i vektor-stupca \mathbf{x} .

2. Funkcija gubitka (i pripadna funkcija pogreške)

Algoritam regresije kao gubitak koristi kvadratno odstupanje, koje nazivamo i **kvadratni gubitak** (engl. *quadratic loss*):

$$L(y^{(i)}, h(\mathbf{x}^{(i)})) = (y^{(i)} - h(\mathbf{x}^{(i)}))^2$$

Funkcija pogreške definirana je kao zbroj gubitaka kroz sve primjere iz označenog skupa \mathcal{D} , podijeljena s dva (i to je proporcionalno empirijskom očekivanju gubitka):

$$E(h|\mathcal{D}) = \frac{1}{2} \sum_{i=1}^N (y^{(i)} - h(\mathbf{x}^{(i)}))^2$$

3. Optimizacijski postupak

Kao i prošli tjedan, optimizacijski postupak definirat ćemo najprije vrlo apstraktno:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w}|\mathcal{D})$$

U kontekstu regresije, gdje je funkcija pogreške definirana preko kvadratnog gubitka, ovaj se postupak naziva postupak **najmanjih kvadrata** (engl. *least squares*) ili metoda **obični najmanji kvadrati** (engl. *ordinary least squares, OLS*). Naziv reflektira činjenicu da tražimo parametre \mathbf{w} za koje je zbroj kvadrata odstupanja (kvadrata reziduala) najmanji.

Dobra vijest je da, kod linearog modela regresije, neovisno o broju značajki n , rješenje ovog optimizacijskog problema uvijek postoji u **zatvorenoj formi**. Dakle, rješenje ćemo moći napisati kao matematički izraz koji će biti izračunljiv u konačnometu broju koraka. (Doduše, vidjet ćemo da taj izračun neće biti posve trivijalan, jer će uključivati izračun inverza matrice, a to kod velikih matrica nije trivijalno.)

4 Postupak najmanjih kvadrata

Razmotrimo sada detaljnije postupak najmanjih kvadrata, odnosno kako možemo izračunati optimalne parametre \mathbf{w} u smislu zbroja kvadratnih odstupanja, i to za višestruku linearu

regresiju, gdje $n > 1$. Raspolažemo skupom primjera (kod regresije to su ulazne varijable odnosno “nezavisne varijable”):

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \dots & x_n^{(2)} \\ \vdots & & & \\ 1 & x_1^{(N)} & x_2^{(N)} \dots & x_n^{(N)} \end{pmatrix}_{N \times (n+1)}$$

Prisjetimo se, matricu primjera \mathbf{X} nazivamo **matrica dizajna**. Imamo također i vektor izlaznih vrijednosti (tj. oznake primjera odnosno “zavisne varijable”):

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{pmatrix}_{N \times 1}$$

4.1 Krivi put: egzaktno rješenje

Pokušajmo prvo s nečim što nije postupak najmanjih kvadarta, a što će se pokazati kao loša ideja. Naime, na naš optimizacijski problem možemo gledati kao na rješavanje sustava jednadžbi. Idealno, mi bismo željeli pronaći rješenje za koje vrijedi

$$(\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D}. h(\mathbf{x}^{(i)}) = y^{(i)}$$

odnosno

$$(\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D}. \mathbf{w}^T \mathbf{x}^{(i)} = y^{(i)}$$

To bi bilo **egzaktno rješenje** našeg problema: rješenje kod kojega regresijski model za svaki ulazni primjer savršeno predviđa njegovu izlaznu vrijednost. Tako definiran problem zapravo je sustav od N jednadžbi s $(n + 1)$ nepoznanicom. Takav sustav jednadžbi možemo elegantno napisati kao **matričnu jednadžbu**:

$$\begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \dots & x_n^{(2)} \\ \vdots & & & \\ 1 & x_1^{(N)} & x_2^{(N)} \dots & x_n^{(N)} \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{pmatrix}$$

to jest

$$\mathbf{X}\mathbf{w} = \mathbf{y}$$

(uvjerite se da je to isto kao i sustav jednadžbi koji smo napisali ranije). Egzaktno rješenje ovog sustava jednadžbi dobivamo tako da jednadžbu slijeva pomnožimo sa \mathbf{X}^{-1} :

$$\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$$

Ovo je lijepo, međutim, neće uvijek biti rješivo. Naime, problem će često biti **loše definiran** (engl. *ill-defined*), što znači da gornji sustav linearnih jednadžbi nema rješenja (**nekonzistentan je**) ili da pak **nema jedinstvenog rješenja** (ima beskonačno mnogo rješenja). Prvi i najbanalniji problem jest taj što matrica dizajna \mathbf{X} , da bi imala inverz, nužno mora biti **kvadratna**. No, broj primjera općenito može biti veći od broja parametara, $N > n + 1$, ili može biti manji, $N < n + 1$. U prvom slučaju kažemo da je sustav jednadžbi **preodređen** (engl. *overdetermined*), a u drugom da je **pododređen** (engl. *underdetermined*). U oba slučaja matrica \mathbf{X} neće biti kvadratna i neće imati inverz. Istini za volju, zahtjev da je matrica \mathbf{X} kvadratna igra ulogu

samo ako rješenje sustava jednadžbi želimo riješiti pomoću inverza matrice. Možemo koristiti neku drugu metodu, npr., Gaussov u metodu eliminacije, koja je primjenjiva i na pravokutne matrice. No to, naravno, i dalje ne znači da će sustav jednadžbi imati rješenje niti da će ono biti jedinstveno. Konkretno, ako je **rang** matrice \mathbf{X} manji od ranga proširene matrice $\mathbf{X}|\mathbf{y}$, onda je sustav jednadžbi nekonzistentan. Nadalje, ako rang matrice \mathbf{X} nije jednak broju stupaca (odnosno broju parametara, koji je jednak $n + 1$), onda sustav ima beskonačno mnogo rješenja. Drugim riječima, da bi sustav imao jedno i jedinstveno rješenje, rangovi obaju matrica \mathbf{X} i $\mathbf{X}|\mathbf{y}$ moraju biti jednaki $n + 1$.

10

11

► PRIMJER

Evo nekoliko trivijalnih primjera koji demonstriraju da nas egzaktno rješenje sustava linearnih jednadžbi u praksi neće zadovoljiti.

Neka je skup primjera za učenje za jednostavnu regresiju $\mathcal{D} = \{(x^{(i)}, y^{(i)})\} = \{(1, 1), (1, 2)\}$. Matrica dizajna \mathbf{X} doduše jest kvadratna, ali je sustav nekonzistentan (matrica \mathbf{X} je ranga 1, dok je proširena matrica $\mathbf{X}|\mathbf{y}$ ranga 2). To je zato što za primjer $x = 1$ imamo dvije različite označke. Suprotno, za označeni skup primjera $\mathcal{D} = \{(1, 1), (1, 1)\}$ matrica \mathbf{X} je i dalje kvadratna, ali matrica $\mathbf{X}|\mathbf{y}$ ima rang 1, dok $n + 1 = 2$, pa sustav ima beskonačno mnogo rješenja. Intuitivno, to je zato što imamo samo jedan primjer u skupu za učenje, a kroz jednu točku možemo provući beskonačno mnogo pravaca.

Razmotrimo sada skup primjera $\mathcal{D} = \{(1, 1), (2, 2), (3, 3)\}$. Ove točke leže na pravcu i očekujemo naći egzaktno rješenje za regresijski pravac. Doduše, to rješenje ne možemo naći gornjom jednadžbom (inverzom matrice), jer matrica dizajna \mathbf{X} nije kvadratna (nego je dimenzija 3×2), odnosno sustav je preodređen. Međutim, sustav jest konzistentan (rang matrice \mathbf{X} jednak je 2 i to je jednako rangu matrice $\mathbf{X}|\mathbf{y}$) i ima jedinstveno rješenje (rang je jednak $n + 1 = 2$), pa rješenje ($w_0 = 0, w_1 = 1$) možemo dobiti, na primjer, Gaussovom metodom eliminacije. No, u stvarnosti podatci neće biti tako idealni. Prisjetimo se **šuma** o kojemu smo pričali prošli prošli put. Dovoljno je da se, zbog šuma, jedna od ovih triju točaka malo pomakne s pravca, i cijela stvar pada u vodu. Na primjer, ako je skup primjera za učenje $\mathcal{D} = \{(1, 1), (2, 2.1), (3, 3)\}$ (zbog šuma je označka drugog primjera 2.1 umjesto 2), već imamo nekonzistentan sustav jednadžbi (rang od \mathbf{X} je 2, a rang od $\mathbf{X}|\mathbf{y}$ je 3). Očito, to je zato što kroz ove tri točke ne možemo provući pravac, budući da one ne leže na istom pravcu.

Slično, razmotrimo skup primjera $\mathcal{D} = \{((1, 1), 1), ((2, 2), 2)\}$ za dvoulaznu ($n = 2$) višestruku regresiju. Odgovarajuća proširena matrica $\mathbf{X}|\mathbf{y}$ izgleda ovako:

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \end{array} \right)$$

Rang matrice \mathbf{X} i rang matrice $\mathbf{X}|\mathbf{y}$ jednaki su 2, što je manje od broja parametara ($n + 1 = 3$), pa zaključujemo da sustav jednadžbi ima beskonačno mnogo rješenja. To je zato jer raspolaćemo samo dvama primjerima: svaki primjer je jedna točka u 3d-prostoru definirana koordinatama (x_1, x_2, y) , a kroz dvije točke u 3d-prostoru možemo provući beskonačno mnogo ravnina.

Ovi primjeri nam ukazuju da, iz perspektive strojnog učenja, egzaktno rješenje nije dovoljno robusno. Čak i ako su podatci nastali nekom linearnom funkcijom, u skupu primjera za učenje postojat će šum, pa ne možemo očekivati da će sve točke savršeno ležati na pravcu. Nadalje, čak ako imamo manje primjera nego parametara, ipak bismo voljeli imati neko rješenje. Sve u svemu, egzaktno rješenje pretpostavlja da je svijet savršen, ali on to nije. U nesavršenom svijetu, približno rješenje je savršeno dobro.

4.2 Pravi put: rješenje najmanjih kvadrata

Zbog ograničenja koja smo upravo ustanovili, umjesto da tražimo egzaktno rješenje sustava $\mathbf{X}\mathbf{w} = \mathbf{y}$, tražit ćemo **približno** rješenje sustava. Što mislimo pod "približno"? Mogli bismo to definirati na razne načine, no kod postupka najmanjih kvadrata "približno" je definirano u smislu najmanjih kvadratnih odstupanja. To smo zapravo već i napravili, kada smo gledali onaj prvi primjer, s jednostavnom regresijom. Tamo smo funkciju gubitka definirali preko kvadratne funkcije gubitka, pa dakle ta funkcija upravo mjeri kvadratno odstupanje rješenja od označaka u

skupu podataka. Sada bismo takav pristup želi poopćiti na višestruku regresiju, dakle na slučaj s više značajki, $n > 1$.

Ovdje na scenu nastupa **matrični račun**. Naime, sada kada imamo više značajki, puno je jednostavnije da kod optimizacije pređemo na zapis gdje ćemo koristiti matrice, i da onda pokušamo izvesti rješenje s operacijama nad matricama.

Prisjetimo se, naša funkcija pogreške za regresiju je:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{2} \sum_{i=1}^N (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

Matrični oblik ove funkcije je:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{2} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y})$$

Ovdje si uzmite malo vremena da se uvjerite da je to doista tako. (Shvatili ste kada znate objasniti kamo je nestao kvadrat i kamo je nestala suma.) Sada idemo dalje raspisivati desnu stranu izraza. Transpoziciju primjenjujemo na pojedinačne pribrojниke (jer $(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$), zatim primjenjujemo distributivnost i nakon toga sljedeće dvije jednakosti iz matričnog računa:

$$\begin{aligned} (\mathbf{A}^T)^T &= \mathbf{A} \\ (\mathbf{AB})^T &= \mathbf{B}^T \mathbf{A}^T \end{aligned}$$

I tako dobivamo:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{2} (\mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{y}^T \mathbf{y})$$

Ovdje se sada trebamo prisjetiti da je $E(\mathbf{w}|\mathcal{D})$ zapravo skalar, što znači da svi izrazi koje ovdje zbrajamo zapravo moraju biti dimenzija 1×1 . Doista, na primjer za $\mathbf{w}^T \mathbf{X}^T \mathbf{y}$:

$$\underbrace{\mathbf{w}^T}_{1 \times (n+1)} \cdot \underbrace{(\mathbf{X}^T)_{(n+1) \times N}} \cdot \underbrace{\mathbf{y}}_{N \times 1} = 1 \times 1$$

Ako su to skaliari, onda ih možemo transponirati a da time ništa ne promijenimo, jer transpozicija skalarova daje isti taj skalar. A to onda znači da su srednja dva pribrojnika u gornjem izrazu jednakova, jer

$$\mathbf{w}^T \mathbf{X}^T \mathbf{y} = (\mathbf{w}^T \mathbf{X}^T \mathbf{y})^T = ((\mathbf{w}^T \mathbf{X}^T) \mathbf{y})^T = \mathbf{y}^T (\mathbf{w}^T \mathbf{X}^T)^T = \mathbf{y}^T (\mathbf{X} \mathbf{w})$$

gdje smo iskoristili gornje dvije jednakosti iz matričnog računa. Tako u konačnici dobivamo:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{2} (\mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{y}^T \mathbf{y})$$

Sada želimo naći minimizator ove funkcije. Za to nam trebaju pravila za **deriviranje matrica**. Postoji niz pravila, ali srećom nama će biti dovoljna samo ova dva:

$$\begin{aligned} \frac{d}{dx} \mathbf{Ax} &= \mathbf{A} \\ \frac{d}{dx} \mathbf{x}^T \mathbf{Ax} &= \mathbf{x}^T (\mathbf{A} + \mathbf{A}^T) \end{aligned}$$

Prije nego što nastavimo, da vidimo što mi sad zapravo želimo. Mi funkciju pogreške $E(\mathbf{w}|\mathcal{D})$ želimo derivirati po vektoru \mathbf{w} , jer tražimo minimum po parametrima \mathbf{w} . Funkcija E je funkcija vektora \mathbf{w} , dakle to je funkcija od $(n+1)$ varijabli. Što će biti derivacija funkcije

E po vektoru \mathbf{w} , tj. što je $\nabla_{\mathbf{w}}E$? To je **gradijent** funkcije E . Gradijent je opet funkcija, i to vektorska funkcija, koja za neku točku \mathbf{w} daje smjer (vektor) najbržeg rasta funkcije E . Dakle $\nabla_{\mathbf{w}}E$ je $(n+1)$ -dimenzijski vektor (tj. gradijent). Stacionarna točka funkcije je ona u kojoj je gradijent jednak nuli (preciznije, nul-vektor), a za funkciju E stacionarna točka će biti točka minimuma (prisjetimo se, funkcija E je konveksna, dakle njezina stacionarna točka sigurno je njezin minimum). U konačnici nas, dakle, zanima koja je to točka gdje je gradijent funkcije E jednak nuli. Eto, sada kada smo se svega ovoga prisjetili, možemo dalje...

Derivirajmo funkciju pogreške i izjednačimo je s nul-vektorom:

$$\nabla_{\mathbf{w}}E = \frac{1}{2} \left(\mathbf{w}^T (\mathbf{X}^T \mathbf{X} + (\mathbf{X}^T \mathbf{X})^T) - 2\mathbf{y}^T \mathbf{X} \right) = \mathbf{w}^T (\mathbf{X}^T \mathbf{X}) - \mathbf{y}^T \mathbf{X} = \mathbf{0}$$

iz čega slijedi:

$$\mathbf{w}^T (\mathbf{X}^T \mathbf{X}) = \mathbf{y}^T \mathbf{X}$$

odnosno, nakon transpozicije (uz primjenu gornjih pravila za transpoziciju umnoška matrica):

$$\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

Ovo je sustav tzv. **normalnih jednadžbi**. Rješenje sustava dobivamo množenjem s $(\mathbf{X}^T \mathbf{X})^{-1}$ slijeva:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \mathbf{X}^+ \mathbf{y}$$

i to je naše rješenje za vektor parametara \mathbf{w} koji minimizira kvadratno odstupanje.

Ovo zaslužuje nekoliko komentara. Prvo, vidimo da smo rješenje opet dobili u zatvorenoj formi, što je vrlo lijepo. Drugo, matrica koju smo označili sa \mathbf{X}^+ , definirana kao $\mathbf{X}^+ = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$, je takozvani **pseudoinverz** (točnije: Moore-Penroseov inverz) matrice \mathbf{X} . Što je pseudoinverz? Pseudoinverz je poopćenje koncepta inverza matrice. Za razliku od običnog inverza, koji postoji samo za kvadratne matrice punog ranga, pseudoinverz **postoji za baš svaku matricu**. Kao poseban slučaj, ako je matrica \mathbf{X} kvadratna i punog ranga, onda je pseudoinverz jednak običnom inverzu, $\mathbf{X}^+ = \mathbf{X}^{-1}$.

Ovdje je sada važno primijetiti da je pseudoinverz sam po sebi rješenje problema najmanjih kvadrata. Naime, pogreška kvadratnog odstupanja bila je otpočetka ugrađena u ovaj postupak, jer smo od nje krenuli. To znači da, kada izračunamo pseudoinverz, dobit ćemo parametre \mathbf{w} koji nam daju hiperravninu $h(\mathbf{x}; \mathbf{w})$ koja je optimalna u smislu najmanjih kvadratnih odstupanja od označenih podataka.

Preciznije, rješenje \mathbf{w} dano ovom jednadžbom je takvo da je vektor $\mathbf{X}\mathbf{w}$ u smislu kvadratnih odstupanja minimalno udaljen od vektora \mathbf{y} , odnosno to je rješenje koje minimizira L_2 -normu $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2$. To je ujedno i razlog zašto gornje jednadžbe nazivamo "normalnima".

Ako je sustav jednadžbi pododređen i ima više rješenja, onda pseudoinverz daje rješenje s najmanjom normom $\|\mathbf{w}\|_2$. Ako je sustav preodređen, pseudoinverz daje rješenje koje minimizira $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2$, no ako je i takvih više, opet daje ono s najmanjom normom $\|\mathbf{w}\|_2$.

4.3 Računalni aspekti postupka najmanjih kvadrata

Dakle, ono što je ovdje dobro je da naš optimizacijski postupak daje rješenje u zatvorenoj formi. Međutim, u praksi taj izračun, koji je u domeni **numeričke matematike**, ipak može biti poprilično računalno zahtjevan.

Naime, primijetite da mi ovdje računamo **inverz** umnoška matrica dizajna sa samom sobom: $\mathbf{X}^T \mathbf{X}$. Tu matricu nazivamo **Gramova matrica**. Gramova matrica može biti poprilično velika, pa izračun inverza može biti poprilično skup – tipično $\mathcal{O}(n^3)$ (npr., metodom LU-dekompozicije). Stoga je ključno pitanje: kojih je dimenzija matrica koju invertiramo, $(\mathbf{X}^T \mathbf{X})^{-1}$? Odgovor je: dimenzije su $(n+1) \times (n+1)$). Dakle, na složenost izračuna inverza matrice utječe samo broj značajki n , a ne i broj primjera N . To je dobro imati na umu.

Druga stvar koju valja primijetiti jest da, premda je Gramova matrica očigledno uvijek kvadratna (dimenzija $(n+1) \times (n+1)$), ona ne mora biti punog ranga. Naime, može se pokazati da je rang Gramove matrice jednak rangu matrice dizajna \mathbf{X} . Ako je taj rang jednak $n+1$, onda je Gramova matrica punog ranga i pseudoinverz možemo izračunati kao što smo ovdje napisali, dakle kao $\mathbf{X}^+ = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$. Međutim, ako je rang matrice dizajna manji od $n+1$ (a to će biti ako je matrica dizajna "plitka", odnosno kada $N < n+1$, tj. kada imamo manje primjera nego značajki), onda pseudoinverz ne možemo izračunati na ovaj način. Međutim, pseudoinverz tada možemo ga izračunati koristeći druge metode, tipično **rastavom na singularne vrijednosti (SVD)**. No, nećemo ovdje ići u detalje. Dovoljno je znati da pseudoinverz možemo uvijek izračunati, na ovaj ili onaj način. O računalnim aspektima izračuna pseudoinverza nastavit ćemo pričati idući put, kada ćemo se baviti prenaučenošću regresijskog modela i regularizacijom kao načinom da se ona spriječi.

5 Probabilistička interpretacija regresije

Gubitak smo definirali kao **kvadratno odstupanje**, odnosno kvadrat reziduala, a empirijsku pogrešku definirali smo kao sumu kvadrata reziduala. No, nekako smo nemušto opravdali uporabu baš kvadrata odstupanja. U strojnog učenju ne bismo se smjeli zadovoljiti takvim paušalnim objašnjenjima. Postoji li neki dublji razlog zašto bismo baš koristili kvadratni gubitak? Postoji! No da bismo to vidjeli, moramo o strojnog učenju pričati iz probabilističke perspektive. Mi ćemo tu perspektivu imati u drugoj polovici ovog predmeta, međutim već je sada korisno napraviti kratke izlete u svijet probabilističkog strojnog učenja. Vidjet ćemo da su svi ti pogledi isprepleteni, i da je korisno, a i nužno, neki algoritam razumijeti iz više perspektiva.

Pogledajmo, dakle, probabilističku interpretaciju linearne regresije. Naš model linearne regresije je:

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

Prepostavimo da su naši označeni primjeri u stvarnosti generirani nekom funkcijom $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Za tu funkciju često kažemo da je "**funkcija iza podataka**" (engl. *function underlying the data*). U stvarnosti nam, naravno, ta funkcija **nije poznata**.

Kao što već vrlo dobro znamo, u označenim podatcima neminovno postoji **šum**, koji se superponira na pravu vrijednost ishodišne funkcije. Dakle, ono što mi vidimo u oznakama jest vrijednost funkcije plus šum:

$$y^{(i)} = f(\mathbf{x}^{(i)}) + \varepsilon_i$$

I ovdje sada ulazimo u probabilističke vode. Modelirat ćemo šum kao slučajnu varijablu, distribuiranu po **normalnoj (Gaussovoj) distribuciji**, sa središtem u nuli i varijancom σ^2 . To pišemo kao:

$$\varepsilon \sim \mathcal{N}(0, \sigma^2)$$

Naime, šum se, kao i mnogi drugi fenomeni u prirodi, može dobro opisati normalnom distribucijom, tako da je ova pretpostavka posve razumna. Nadalje, prepostavljamo da je šum identičan za svaki pojedinačni primjer, $\varepsilon = \varepsilon_i$.

Prisjetimo se, Gaussova distribucija (ili normalna razdioba) ima sljedeću **funkciju gustoće vjerojatnosti** (engl. *probability density function, PDF*):

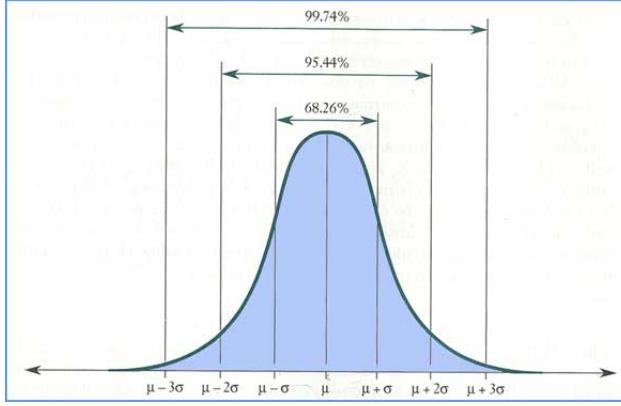
$$p(Y = y | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y - \mu)^2}{2\sigma^2}\right).$$

Reda radi, prisjetimo se i kako ona izgleda:

17

18

19



Sada možemo napisati da će, za zadani ulazni vrijednosti \mathbf{x} , izlazna vrijednost y biti distribuirana oko vrijednosti $f(\mathbf{x})$ (s odstupanjem uslijed šuma):

$$p(y|\mathbf{x}) = \mathcal{N}(f(\mathbf{x}), \sigma^2)$$

$p(y|\mathbf{x})$ je vjerojatnost da primjer \mathbf{x} ima oznaku y , ako znamo da su podatci generirani funkcijom $f(\mathbf{x})$. (Zapravo, \mathbf{x} je kontinuirana varijabla, pa je p gustoća vjerojatnosti oznake y , a ne vjerojatnost, ali to sad nije toliko važno, ideja stoji i dalje.)

Ovime smo modelirali vjerojatnost oznake za pojedinačni primjer. Idući korak je da modeliramo vjerojatnost svih oznaka u cijelom skupa označenih primjera \mathcal{D} . Prisjetimo se, skup označenih primjera \mathcal{D} možemo rastaviti na matricu dizajna \mathbf{X} , koja sadrži samo primjere, te na vektor oznaka \mathbf{y} , koji sadrži oznake za sve primjere. Vjerojatnost da je skup primjera \mathbf{X} označen oznakama \mathbf{y} je:

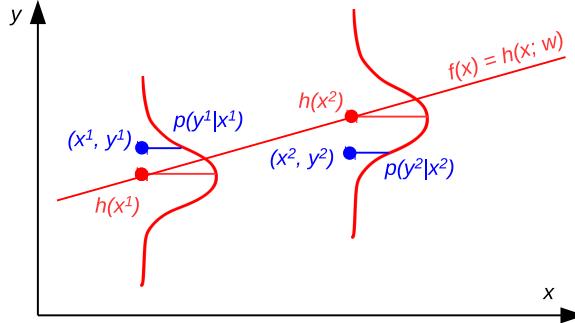
$$p(\mathbf{y}|\mathbf{X}) = \prod_{i=1}^N p(y^{(i)}|\mathbf{x}^{(i)}) = \prod_{i=1}^N \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - f(\mathbf{x}^{(i)}))^2}{2\sigma^2}\right).$$

Kako smo došli do toga? Pretpostavili smo da su oznake primjera međusobno **nezavisne** i da dolaze iz **iste distribucije**. Ako su je tako, onda je vjerojatnost nekog skupa oznaka jednostavno umnožak vjerojatnosti pojedinačnih oznaka. Ta je pretpostavka ovdje razumna, i ona se često koristi u strojnom učenju, pod nazivom **nezavisno i identično distribuirane** varijable (engl. *independently and identically distributed variables, IID*).

Gornja vjerojatnost govori nam dakle koliko je vjerojatno da skup primjera \mathbf{X} bude označen sa \mathbf{y} ako su primjeri generirani funkcijom $f(\mathbf{x})$. Međutim, prisjetimo se da je funkcija $f(\mathbf{x})$ nama nepoznata: mi ne znamo koja je funkcija generirala podatke. Zapravo, naš zadatak je upravo da pronađemo tu funkciju, i tu funkciju modeliramo našom hipotezom $h(\mathbf{x}|\mathbf{w})$. Dakle, tražimo hipotezu $h(\mathbf{x}; \mathbf{w})$ koja je generirala primjere, odnosno hipotezu za koju vrijedi $h(\mathbf{x}; \mathbf{w}) = f(\mathbf{x})$. To sad jednostavno znači da umjesto $f(\mathbf{x})$ možemo pisati $h(\mathbf{x}; \mathbf{w})$. Dakle, imamo:

$$p(\mathbf{y}|\mathbf{X}) = \prod_{i=1}^N p(y^{(i)}|\mathbf{x}^{(i)}) = \prod_{i=1}^N \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{w}))^2}{2\sigma^2}\right).$$

Ovo je, dakle, vjerojatnost da primjeri \mathbf{X} imaju oznake \mathbf{y} ako su nastali funkcijom $h(\mathbf{x}; \mathbf{w})$. Također, kažemo da je to "vjerojatnost skupa podataka pod modelom" (u smislu: ako je hipoteza h istinita, vjerojatnost ovako označenih primjera je ta-i-ta). Grafički, to izgleda ovako:



Dakle, u svakoj točki $\mathbf{x}^{(i)}$ imamo jednu normalnu distribuciju sa središtem u $f(\mathbf{x}^{(i)})$. Oznaka $y^{(i)}$ može odstupati od one koju predviđa hipoteza, $h(\mathbf{x}^{(i)}; \mathbf{w})$, i to odstupanje modelirano je normalnom distribucijom. Najvjerojatnije je da će oznaka $y^{(i)}$ biti baš u sredini normalne distribucije, tj. da je $y^{(i)} = h(\mathbf{x}^{(i)}; \mathbf{w})$, jer je tamo gustoća vjerojatnosti $p(y^{(i)}|\mathbf{x}^{(i)})$ najveća. Što je oznaka $\mathbf{x}^{(i)}$ dalje od tog središta, to je ona manje vjerojatna. Ukupna vjerojatnost (odnosno gustoća vjerojatnosti) cijelog skupa oznaka \mathbf{y} je umnožak gustoća vjerojatnosti $p(y^{(i)}|\mathbf{x}^{(i)})$ za sve točke $\mathbf{x}^{(i)}$. Umnožak će biti to veći (odnosno vjerojatnost skupa oznaka će biti to veća) što više oznaka $y^{(i)}$ bude nalazilo bliže središtu njima odgovarajućih normalnih razdioba. Analogno, umnožak će biti to veći (odnosno vjerojatnost skupa oznaka će biti to veća) što funkcija $h(\mathbf{x}; \mathbf{w})$ prolazi što bliže svakoj oznaci $y^{(i)}$.

Ovo zadnje opažanje je ključno: vjerojatnost skupa označenih primjera ovisi o funkciji $h(\mathbf{x}; \mathbf{w})$. Ta vjerojatnost će biti maksimalna ako hipoteza h prođe što bliže oznakama \mathbf{y} . Naš je cilj onda pronaći hipotezu h , odnosno težine \mathbf{w} , koje maksimiziraju vjerojatnost skupa primjera. Drugim riječima, želimo pronaći hipotezu takva da ona ove naše podatke koje imamo čini najvjerojatnijima. Na taj način mi zapravo prepostavljamo da su podatci \mathcal{D} kojima raspolažemo zapravo najvjerojatniji mogući: ako smo te podatke dobili, onda ćemo prepostaviti da su upravo ti podatci upravo najvjerojatniji mogući, jer inače je vjerojatnije da bismo dobili neke druge podatke.

Želimo, dakle, naći hipotezu koja **maksimizira vjerojatnost oznaka**. U nastavku će nam biti lakše prebaciti se u logaritamsku domenu (a vidjet ćemo uskoro i zašto). Također, umjesto hipoteze h možemo pisati težine \mathbf{w} , jer zapravo njih tražimo. Vjerojatnost oznaka \mathbf{y} za težine \mathbf{w} je:

$$\ln p(\mathbf{y}|\mathbf{w}) = \ln \prod_{i=1}^N p(y^{(i)}|x^{(i)}) = \ln \prod_{i=1}^N \frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{(y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{w}))^2}{2\sigma^2} \right\}$$

Sada možemo malo srediti izraz (logaritmom ući u umnožak), pa dobivamo:

$$\underbrace{-N \ln(\sqrt{2\pi}\sigma)}_{=\text{konst.}} - \frac{1}{2\sigma^2} \sum_{i=1}^N (y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{w}))^2$$

Kako nas zanima maksimizacija po \mathbf{w} , to ovaj prvi član te σ^2 možemo zanemariti jer su konstantni. Ostaje nam:

$$-\frac{1}{2} \sum_{i=1}^N (y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{w}))^2$$

Prisjetimo se, ovo je vjerojatnost koju želimo **maksimizirati**. To znači da izraz nakon znaka minus želimo **minimizirati**. A taj izraz je upravo **pogreška kvadratnog odstupanja!** Drugim riječima, uz prepostavku normalno distribuiranog šuma, hipoteza koja će maksimizirati vjerojatnost da su podatci označeni tako kako jesu jest upravo ona koja minimizira kvadratno odstupanje predviđenih i stvarnih oznaka. I to je, konačno, probabilističko opravdanje za uporabu kvadratne funkcije gubitka!

Postupak koji smo odradili ovdje može se sažeti u sljedećem: izrazili smo vjerojatnost označaka tako da smo svaku označku modelirali kao normalno distribuiranu varijablu sa srednjom vrijednošću jednakom predviđanju hipoteze, i zatim smo izrazili vjerojatnost skupa označaka, pretpostavivši IID. Zatim tražimo parametre hipoteze koji maksimiziraju tu vjerojatnost. To nas je dovelo do izraza koji želimo maksimizirati, za koji se ispostavio da je jednak negativnoj empirijskoj pogrešci, koju želimo minimizirati. Ovaj princip nije bezveze, nego je jedan univerzalan princip kako u strojnog učenju možemo definirati funkcije pogreške. O tome ćemo još govoriti u nekim narednim predavanjima.

Sažetak

- Regresija služi predviđanju **numeričkih vrijednosti** (zavisne varijable) na temelju ulaznih primjera (nezavisne varijable)
- Kod **linearne regresije**, izlazna vrijednost je linearna kombinacija ulaznih vrijednosti
- Funkcija gubitka je **kvadratni gubitak** (kvadratna razlika predviđene i stvarne označake)
- Optimizacija se provodi **postupkom najmanjih kvadrata**, koji izračunavamo pomoću **pseudoinverza** matrice dizajna
- Uz pretpostavku normalno distribuiranog šuma, rješenje najmanjih kvadrata istovjetno je maksimizaciji vjerojatnosti označaka, što daje **probabilističko opravdanje** za kvadratni gubitak

Bilješke

- [1] Ovo predavanje je lepršavija varijanta poglavlja o regresiji iz (nedovršene) skripte: http://www.fer.unizg.hr/_download/repository/SU-Regresija.pdf.
- [2] Regresija je jedan od osnovnih i nevjerojatno moćnih alata statističkog zaključivanja, koji se pogotovo često koristi u analizi podataka u društvenim znanostima i medicini. Želite li uroniti u regresiju iz aspekta primjenjene statistike, preporučam vam da krenete od ([Kutner et al., 2005](#)), a zatim da pročitate ([Harrell Jr, 2015](#)). Računajte na to kao višemjesečni projekt. Mi u strojnog učenju nećemo inati ništa za reći o statističkom pristupu regresiji.
- [3] U pogledu naziva regresijskih koeficijenata u statistici postoji terminološka nekonzistentnost: ponekad se ti koeficijenti nazivaju betama (beta-koeficijentima) i označavaju sa β , dok se nekad pod tim nazivom podrazumijevaju isključivo **standardizirani koeficijenti**: koeficijenti dobiveni nakon transformacije značajki tako da imaju srednju vrijednost nula i standardnu devijaciju jednaku 1, tj. transformacije na z-vrijednosti. Standardizirani koeficijenti omogućavaju nam da uspoređujemo relativnu važnost neke značajke u odnosu na sve druge značajke.
- [4] Vidjet ćemo da je ovo česta tema – često u strojnog učenju malo manipuliramo matematičke izraze kako bi se u optimizaciji nešto pokratilo ili pojednostavilo. Naravno, to je legitimno, dok god ne utječe na krajnje rješenje. Npr., množenje funkcije koju minimiziramo konstantnim faktorom nema utjecaja na minimizator funkcije. Primjena **monotone transformacije** (npr., logaritamske funkcije) na funkciju koju minimiziramo također nema utjecaja na minimizator funkcije, a može pojednostaviti algebarski oblik funkcije koju minimiziramo i povećati numeričku stabilnost rješenja.
- [5] U nastavku je puni izvod za parametre (w_0, w_1) jednostavne regresije. Pronađimo najprije minimum

s obzirom na parametar w_0 :

$$\begin{aligned} \frac{\partial}{\partial w_0} \left[\frac{1}{2} \sum_i^N (y^{(i)} - (w_1 x^{(i)} + w_0))^2 \right] = \\ \sum_i^N (-y^{(i)} + w_1 x^{(i)} + w_0) = -\sum_i^N y^{(i)} + w_1 \sum_i^N x^{(i)} + N w_0 = 0. \end{aligned} \quad (1)$$

Rješavanjem za w_0 i uvrštenjem $\bar{x} = \sum_i^N x^{(i)}/N$ i $\bar{y} = \sum_i^N y^{(i)}/N$ dobivamo

$$w_0 = \frac{1}{N} \left(\sum_i^N y^{(i)} - w_1 \sum_i^N x^{(i)} \right) = \frac{1}{N} \sum_i^N y^{(i)} - w_1 \frac{1}{N} \sum_i^N x^{(i)} = \bar{y} - w_1 \bar{x}. \quad (2)$$

Za w_1 dobivamo

$$\begin{aligned} \frac{\partial}{\partial w_1} \left[\frac{1}{2} \sum_i^N (y^{(i)} - (w_1 x^{(i)} + w_0))^2 \right] = \\ \sum_i^N -x^{(i)}(y^{(i)} - w_1 x^{(i)} - w_0) = -\sum_i^N x^{(i)}y^{(i)} + w_1 \sum_i^N (x^{(i)})^2 + w_0 \sum_i^N x^{(i)} = 0. \end{aligned} \quad (3)$$

Uvrštenjem (2) u (3) te zamjenom $\sum_i^N x^{(i)} = N\bar{x}$ dobivamo

$$-\sum_i^N x^{(i)}y^{(i)} + w_1 \sum_i^N (x^{(i)})^2 + (\bar{y} - w_1 \bar{x})N\bar{x} = w_1 \left(\sum_i^N (x^{(i)})^2 - N\bar{x}^2 \right) + N\bar{x}\bar{y} - \sum_i^N x^{(i)}y^{(i)} = 0.$$

iz čega slijedi

$$w_1 = \frac{\sum_i^N x^{(i)}y^{(i)} - N\bar{x}\bar{y}}{\sum_i^N (x^{(i)})^2 - N\bar{x}^2}.$$

6 Jednostavan opis što je to rješenje u zatvorenoj formi: http://riskencyclopedia.com/articles/closed_form_solution/

7 A što ako bismo željeli raditi multivarijatnu regresiju, gdje bismo za jedan primjer predviđali više brojeva (npr., cijenu nekretnine ali i energetsku potrošnju te iste nekretnine)? U principu, to uvijek možemo napraviti pomoću univarijatne regresije, tako da za svaku izlaznu varijablu trenirano zaseban model univarijatne regresije. Međutim, sa statističkog stajališta dekompozicija multivarijatne regresije u skup univarijatnih regresijskih modela nije uvijek zadovoljavajuća, jer između izlaznih varijabli postoje zavisnosti, pa će i pogreške univarijatnih regresijskih modela biti donekle korelirane, a to, ako želimo raditi propisno statističko zaključivanje, svakako treba uzeti u obzir. Više ovdje: <https://stats.stackexchange.com/q/254254/93766>

8 U literaturi iz strojnog učenja (npr., u Bishopovog knjizi, a također i u našoj skripti) ponekad se uvodi posebna notacija za tako proširene vektora \mathbf{x} i \mathbf{w} :

$$\begin{aligned} \tilde{\mathbf{x}} &= (1, \mathbf{x}) \\ \tilde{\mathbf{w}} &= (w_0, \mathbf{w}) \end{aligned}$$

Onda model možemo definirati kao: $h(\mathbf{x}; \tilde{\mathbf{w}}) = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}}$. Jednostavnosti radi, mi to u ovim materijalima nećemo raditi. Vjerujem da će iz konteksta biti jasno radi li se o proširenom vektoru ili ne.

9 Zapravo, postupak **obični najmanji kvadrati** (engl. *ordinary least squares*, *OLS*) samo je jedna vrsta postupka **najmanjih kvadrata** (engl. *least squares*, *LS*). Potonji uključuju linearne i nelinearne postupke, ovisno o tome ovise li reziduali o prediktorima linearno ili nelinearno. Postupak običnih kvadrata pripada skupini **linarnih najmanjih kvadrata** (engl. *linear least squares*, *LLS*), kojoj pripadaju i postupci **težinski najmanji kvadrati** (engl. *weighted least squares*, *WLS*) i poopćeni najmanji kvadrati (engl. *generalized least squares*, *GLS*).

10 Prisjetimo se, **rang** (engl. *rank*) matrice \mathbf{A} , označen kao $\text{rank}(\mathbf{A})$, je broj linearno nezavisnih stupaca matrice \mathbf{A} , i taj je broj jednak broju linearno nezavisnih redaka matrice \mathbf{A} . Kvadratna matrica \mathbf{A}

dimenzija $n \times n$ je **punog ranga**, $\text{rank}(\mathbf{A}) = n$, ako i samo ako nema linearno zavisnih redaka niti linearno zavisnih stupaca. Matrica ima puni rang ako i samo ako nije **singularna** i ako i samo ako ima **inverz**. Npr., ova matrica ima rang 3, dakle puni rang:

$$\begin{pmatrix} 1 & 5 & 1 \\ 2 & 0 & 4 \\ 3 & 2 & 6 \end{pmatrix}$$

dok ova matrica nema puni rang, već rang 2, jer je treći stupac linearna kombinacija prvog:

$$\begin{pmatrix} 1 & 5 & 2 \\ 2 & 0 & 4 \\ 3 & 2 & 6 \end{pmatrix}$$

Pravokutna matrica \mathbf{B} dimenzija $m \times n$ je punog ranga ako ima maksimalni mogući rang, tj. ako $\text{rank}(\mathbf{B}) = \min(m, n)$. Npr., ova matrica ima puni rang, jer ima rang 2, što je maksimalni mogući rang za matricu dimenzija 3×2 :

$$\begin{pmatrix} 1 & 5 & 1 \\ 2 & 0 & 4 \end{pmatrix}$$

dok ova matrica nema puni rang, jer ima rang jednak 1:

$$\begin{pmatrix} 1 & 5 & 2 \\ 2 & 10 & 4 \end{pmatrix}$$

- [11] Kronecker-Capellijev teorem: https://en.wikipedia.org/wiki/Rouch%C3%A9%E2%80%93Capelli_theorem
- [12] Ovo je dobro mjesto da napomenemo da kad kažemo "skup" označenih primjera, zapravo i ne mislimo doista na skup u smislu kolekcije elemenata koji se ne ponavljaju. Zapravo, u strojnom učenju skup elemenata je N -torka. Redoslijed uglavnom nije bitan, ali ponavljanja su moguća, i općenito je moguće da se jedan te isti primjer u skupu pojavljuje više puta te moguće s različitim oznakama.
- [13] Matrični račun (odnosno linearna algebra općenitije) jedan je od osnovnih matematičkih alata u strojnom učenju, premda ćemo se mi u ovom predmetu vrlo ograničeno koristiti matričnim računom (i linearom algebrrom). Dobar pregled matričnog računa u možete naći u <https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf>. Standardna referenca za algoritme za operacije nad matricama je (Van Loan and Golub, 1983). Standardna (i didaktički izvrsna) referenca za linearnu algebru je (Strang et al., 1993).
- [14] Pseudoinverz je prvi definirao Eliakim Moore 1921. godine, ponovo ga je otkrio Arne Bjerhammar 1951., a Roger Penrose ga je "ponovo ponovo otkrio" (?) 1955. godine. Roger Penrose je britanski matematičar, fizičar i filozof, poznat po raznim lijepim "Penroseov X" stvarima, na primjer "Penroseovom trokutu", nemogućem trokutu koji je inspiriran sličnim grafikama nizozemskog slikara Eschera, "Penroseovom uzorku", "Penroseovom procesu", itd. U AI krugovima poznat je po svojim radovima na temu filozofije uma, točnije povezanosti svijesti i fizike. Zainteresiranim preporučujem pročitati njegovu knjigu *Carev novi um* (Penrose, 1989), u kojoj argumentira protiv mogućnosti inteligencije današnjih računala kao algoritamski determinističkih sustava i time u osnovi protiv mogućnosti jake umjetne inteligencije.
- [15] **L_2 -norma** (druga norma, euklidska norma) vektora \mathbf{x} je $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^T \mathbf{x}}$. Funkciju pogreške $E(\mathbf{w}|\mathcal{D})$ definirali smo kao sumu kvadrata reziduala (razlike između predikcije modela i ciljne oznake), a to je upravo kvadrat L_2 -norme vektora $(\mathbf{X}\mathbf{w} - \mathbf{y})$:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{2} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

Kvadriranje norme je monotona transformacija, pa je rješenje koje minimizira $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$ isto ono koje minimizira $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2$. Zato možemo reći da je minimizacija $E(\mathbf{w}|\mathcal{D})$ jednaka minimizaciji L_2 -norme vektora $(\mathbf{X}\mathbf{w} - \mathbf{y})$, tj. druge norme vektora reziduala.

- [16] **Gramova matrica** skupa vektora je matrica unutarnjih produkata tih vektora (odnosno skalarnih produkata, kada govorimo o euklidskom prostoru). Ako je \mathbf{A} matrica dimenzija $m \times n$ sastavljena

od m vektor-stupaca dimenzija n , $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m]$, onda je Gramova matrica definirana kao $\mathbf{G} = \mathbf{A}^T \mathbf{A}$. To je matrica čiji je element $\mathbf{G}_{i,j}$ jednak skalarnom produktu vektora \mathbf{a}_i s vektorom \mathbf{a}_j , tj. $\mathbf{a}_i^T \mathbf{a}_j$. Gramova matrica je simetrična, te je pozitivno semidefinitna ($\mathbf{x}^T \mathbf{G} \mathbf{x} \geq 0$). Još važnije, svaka pozitivno semidefinitna matrica je Gramova matrica za neki skup vektora, što znači da svaka pozitivno semidefinitna matrica odgovara skalarnom produktu vektora u nekom vektorskom prostoru. Ovo će nam biti jako korisna spoznaja kada ćemo pričati o stroju potpornih vektora i jezgrenim strojevima. Općenito, koncept Gramove matrice često se pojavljuje u strojnom učenju.

- [17] Vrijedi $\text{rank}(\mathbf{A}^T \mathbf{A}) = \text{rank}(\mathbf{A})$. Dokaz: <https://math.stackexchange.com/q/349738/273854>
- [18] No, pogledajte zanimljivu diskusiju o sveprisutnosti i opravdanosti normalne distribucije: <https://stats.stackexchange.com/q/204471/93766>. Jedan od žešćih kritičara pretjeranog oslanjanja na normalnu distribuciju u pokušajima da se predviđi nepredvidivi ili rijetki događaji je Nassim Taleb; preporučam pročitati njegovog *Crnog labuda* (Taleb, 2007).
- [19] Ova se prepostavka u statistici naziva **homoskedastičnost** (engl. *homoscedasticity*) reziduala. To je jedna od osnovnih prepostavki modela linearne regresije (pored prepostavke o normalnosti reziduala, linearnosti zavisnosti varijable u odnosu na nezavisne i nepostojanje multikolinearnosti, uz temeljne prepostavke o uključenosti svih varijabli u model i nezavisnosti opažanja). Ako ova prepostavka nije ispoštovana, ne postoje garancije da je statistička analiza (intervali pouzdanosti, statistički testovi) valjana.
- [20] Prepostavka o **nezavisno i identično distribuiranim (IID)** primjerima i/ili njihovim oznakama je centralna prepostavka mnogih algoritama strojnog učenja, i zapravo svih algoritama koje ćemo raditi na ovom predmetu. Slučajne variable X i Y su IID ako su međusobno nezavisne i imaju istu distribuciju. Formalno, slučajne varijable X i Y su identično distribuirane akko $P_X(X) = P_Y(Y)$ i ako $P(X, Y) = P(X)P(Y)$.
- [21] Ovo je zapravo **funkcija izglednosti** (engl. *likelihood function*) hipoteze h na skupu primjera za učenje \mathcal{D} . Izglednost je jedan od središnjih koncepcija strojnog učenja i statistike, koji ćemo formalno uvesti malo kasnije. Ljudi obično buni razlike između vjerojatnosti i izglednosti. Razlika se svodi na to što je fiksirano a što varira. Izglednost hipoteze h uz fiksirane podatke \mathcal{D} je vjerojatnost podataka \mathcal{D} uz fiksiranu hipotezu h . Dakle, da izglednost od h jest vjerojatnost, samo što to nije vjerojatnost od h nego vjerojatnost od \mathcal{D} . Međutim, kod izglednosti hipoteza h je ona koja varira, dok je \mathcal{D} fiksirana varijabla.

Literatura

- F. E. Harrell Jr. *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer, 2015.
- M. H. Kutner, C. J. Nachtsheim, J. Neter, W. Li, et al. *Applied linear statistical models*, volume 5. McGraw-Hill Irwin New York, 2005.
- R. Penrose. *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*. Oxford University Press, 1989.
- G. Strang, G. Strang, G. Strang, and G. Strang. *Introduction to linear algebra*, volume 3. Wellesley-Cambridge Press Wellesley, MA, 1993.
- N. N. Taleb. *The black swan: The impact of the highly improbable*, volume 2. Random house, 2007.
- C. F. Van Loan and G. H. Golub. *Matrix computations*. Johns Hopkins University Press Baltimore, 1983.

4. Linearna regresija II

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v1.10

Prošli smo puta pričali o **linearnoj regresiji**: definirali smo linearan model, funkciju kvadratne pogreške i optimizaciju, koja se svela na izračun pseudoinverza matrice dizajna. Pogleđali smo i koja je probabilistička interpretacija regresije i zaključili smo da – ako prepostavimo da je **šum** u oznakama normalno distribuiran – onda je upravo **kvadratna pogreška** ona koja maksimizira vjerojatnost skupa označenih podataka. To nam je dalo probabilističko opravdanje za uporabu kvadratne pogreške i uspostavilo (našu prvu) vezu prema probabilističkom strojnem učenju.

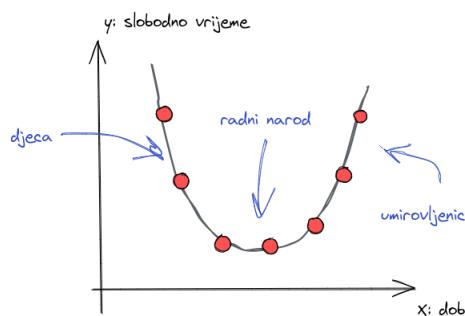
U ovom predavanju nastavljamo razmatrati regresiju. Konkretno, pričat ćemo o **preslikavanju u prostor značajki** i o **regularizaciji**. Ispostavit će se da su to dva vrlo važna koncepta u strojnem učenju, koja se koriste u mnogim algoritmima strojnog učenja, a ne samo kod regresije.

1 Nelinearna regresija

Za početak, prisjetimo se **modela linearne regresije**. Model je definiran kao linearna kombinacija značajki. Drugim riječima: zavisna varijabla je **linearna funkcija** nezavisnih varijabli:

$$h(\mathbf{x}; \mathbf{w}) = \sum_{j=0}^n w_j x_j = \mathbf{w}^T \mathbf{x}$$

Međutim, često će u praksi veza između nezavisnih varijabli i zavisne varijable biti **nelinearna**. Na primjer, slobodno vrijeme (kao zavisna varijabla) na temelju dobi neki osobe (kao nezavisna varijabla):



► PRIMJER

Još nekoliko primjera za koje nam treba nelinearna regresija:

- (1) Cijena automobila s obzirom na kilometražu – naglo opada pa onda stagnira (ovo smo prošli put uzeli kao primjer linearne ovisnosti, ali u stvarnosti to vjerojatno nije slučaj)
- (2) Box Office Revenue s obzirom na cijenu produkcije – raste pa stagnira
- (3) Prosjek ocjena s obzirom na broj sati provedenih na predavanjima – konkavna funkcija (osim za predmet strojno učenje!)

1.1 Model nelinearne regresije

Da bismo modelirali nelinearne ovisnosti, treba nam **nelinearan model**. Npr., recimo da radimo jednostavnu regresiju ($n = 1$). Umjesto linearног modela

$$h(x; w_0, w_1) = w_0 + w_1 x \in \mathcal{H}_1$$

možemo model definirati kao polinom drugog stupnja:

$$h(x; w_0, w_1, w_2) = w_0 + w_1 x + w_2 x^2 \in \mathcal{H}_2$$

Takvim modelom mogli bismo vrlo točno modelirati ove ranije primjere. Primijetite da je model \mathcal{H}_2 **složeniji** (odnosno “većeg kapaciteta”) od modela \mathcal{H}_1 : model \mathcal{H}_2 sadrži nelinearne hipoteze, a uključuje i sve linearne hipoteze koje sadrži model \mathcal{H}_1 . Konkretno, za $w_2 = 0$, isključujemo kvadratnu značajku, pa polinomijalni model degenerira na linearni. Dakle, vrijedi $\mathcal{H}_1 \subset \mathcal{H}_2$. Primijetimo također da ta povećana složenost dolazi s cijenom: model \mathcal{H}_2 ima jedan parametar više od modela \mathcal{H}_1 (naime, parametar w_2).

Pogledajmo drugi primjer. Radimo višestruku regresiju s $n = 2$. Na primjer, želimo predviđjeti cijene nekretnine na temelju površine (x_1) i starosti (x_2). Linearan model je:

$$h(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2$$

Nelinearan model mogao bi opet biti polinom drugog stupnja:

$$h(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2$$

Ovdje, osim kvadratnih značajki, imamo još nešto zanimljivo: značajka $x_1 x_2$. Ta se značajka naziva **interakcijska značajka** (ili **cross-term**) i modelira interakciju između dviju ulaznih varijabli. Npr., ako je nekretnina velike površine i vrlo stara, onda će ova značajka imati visoku vrijednost (to, na primjer, može indicirati veliku staru vilu... možda je to poželjna nekretnina s visokom cijenom?).

Korištenje polinoma drugog stupnja samo je jedan način – ne i jedini – kako dobiti nelinearan model. Veću nelinearnost mogli smo ostvariti da smo model definirali polinomom stupnja višeg od dva. Dodavanje interakcijskih značajki drugi je način da se model učini nelinearnim. Možemo, naravno, i kombinirati ova dva načina. Nadalje, nelinearnost se može ostvariti i drugim funkcijama koje nisu polinomi (premda mi u to nećemo ulaziti). 2

Evo nekoliko različitih regresijskih modela, temeljenih na polinomima, koji se razlikuju po načinu modeliranja nelinearnosti:

- Linearna višestruka regresija:

$$h(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Ovaj model ima više značajki (ukupno n), ali je posve linearan, dakle, ne modelira nikakvu nelinearnost.

- Jednostruka (jednostavna) polinomijalna regresija ($n = 1$):

$$h(x; \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_d x^d$$

Ovaj model ima samo jednu značajku ($n = 1$), no koristi polinom stupnja d kako bi modelirao nelinearnost. Primijetite da je d ovdje **hiperparametar**, budući da određuje složenost modela.

- Višestruka polinomijalna regresija ($n = 2, d = 2$):

$$h(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2$$

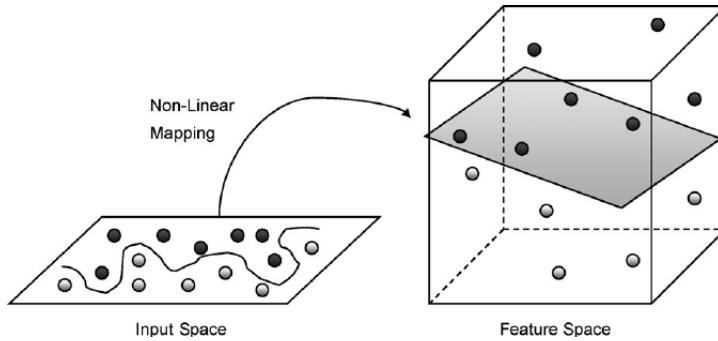
Ovaj model ima dvije značajke ($n = 2$) i koristi polinom drugog stupnja $d = 2$ kako bi modelirao nelinearnost. Osim kvadratnih značajki x_1^2 i x_2^2 , model sadrži i interakcijsku značajku $x_1 x_2$, kojom može modelirati interakciju između značajki x_1 i x_2 (u jeziku statistike: neaditivni efekt nezavisnih varijabli na zavisnu varijablu).

Ovo je samo nekoliko primjera; očito, možemo definirati model proizvojnog stupnja polinoma, možemo uključiti ili ne uključiti interakcijske članove za samo neke parove varijabli, možemo uključiti interakcijske članove za više od dvije varijable, itd. (I opet napomenimo da se nelinearnost može ostvariti i nekim drugim nelinearnim funkcijama, ne nužno polinomijalnima.)

1.2 Preslikavanje u prostor značajki

Premda su modeli koje smo upravo razmotrili zapravo različiti, mi bismo ih voljeli nekako **unificirati**, da ih možemo tretirati na isti način kada radimo optimizaciju. To nas dovodi do jedne poprilično moćne ideje u strojnom učenju: uvijek kada želimo s linearog modela preći na nelinearni, umjesto da mijenjamo model, mi ćemo promijeniti podatke! Način na koji ćemo promijeniti podatke jest da ih preslikamo iz **ulaznog prostora** (engl. *input space*) u neki drugi prostor. Taj drugi prostor zvat ćemo **prostor značajki** (engl. *feature space*). Postupak preslikavanja zvat ćemo **preslikavanje u prostor značajki** (engl. *feature mapping*). Ideja je da, ako odaberemo prikladno preslikavanje, onda će podatci – premda u ulaznom prostoru nisu bili linearni – u prostoru značajki to biti!

Ovakvo preslikavanje je sveprisutno u strojnom učenju, kako u regresiji, tako i u klasifikaciji. Npr., kod klasifikacije:



Prikladnim (nelinearnim) preslikavanjem iz ulaznog prostora u prostor značajki više dimenzije možemo ostvariti da primjeri, koji u ulaznom prostoru nisu bili linearno odvojivi, to budu u prostoru značajki. Na ovom primjeru vidimo da linearna granica (hiperravnina) u prostoru značajki, koja savršeno odjeljuje primjere dviju klasa, u ulaznom prostoru odgovara nelinearnoj granici. No, umjesto da pokušamo u ulaznom prostoru naći nelinearnu hipotezu, mi ćemo primjere nelinearnim preslikavanjem preslikati u prostor značajki veće dimenzije i tamo ih onda odvojiti linearnom hipotezom. U ulaznom prostoru činit će se kao da smo ih odvojili nelinearnom hipotezom.

Ideja će pritom biti uvijek ista: problem koji je ulaznom prostoru nelinearan, nelinearnim preslikavanjem preslikat ćemo u prostor više dimenzije gdje će on (nadamo se) biti linearan, i onda ćemo tamo jednostavno primijeniti linearne modele.

Definirajmo sada formalno kako se radi ovo preslikavanje. Uvest ćemo fiksani skup tzv. **baznih funkcija** (nelinearne funkcije ulaznih varijabli):

$$\{\phi_0, \phi_1, \phi_2, \dots, \phi_m\}$$

gdje $\phi_j : \mathbb{R}^n \rightarrow \mathbb{R}$. Dakle, svaka bazna funkcija ϕ_j uzima cijeli vektor primjera \mathbf{x} i preslikava ga u jedan broj. Taj broj odgovara jednoj od m značajki u m -dimenzijskom prostoru značajki. Baznih funkcija ukupno ima $m + 1$, po jedna za svaku dimenziju m -dimenzijskog prostora značajki, plus jedna dodatna bazna funkcija, ϕ_0 . Ta dodatna bazna funkcija dogovorno je definirana kao $\phi_0(\mathbf{x}) = 1$ i ona odgovara "dummy" jedinici kakvu smo imali i kod proširenog vektora \mathbf{x} . Vrijednost **funkcije preslikavanja** definirana je onda kao vektor dobiven primjenom

pojedinačnih baznih funkcija na primjer \mathbf{x} :

$$\begin{aligned}\phi : \mathbb{R}^n &\rightarrow \mathbb{R}^{m+1} \\ \phi(\mathbf{x}) &= (\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_m(\mathbf{x}))\end{aligned}$$

Funkcija preslikavanja značajki preslikava primjere iz n -dimenzijskog ulaznog prostora u m -dimenzijski prostor značajki. Tipično je $m > n$, odnosno preslikavamo u prostor više dimenzije od ulaznog prostora. Zašto? Zato jer tako imamo veću šansu da će nešto što je u ulaznom prostoru bilo nelinearno u prostoru značajki postati linearno.

Sada to preslikavanje možemo lako ugraditi u naš linearan model, jednostavno tako da vektor \mathbf{x} zamijenimo vektorom $\phi(\mathbf{x})$:

$$h(\mathbf{x}; \mathbf{w}) = \sum_{j=0}^m w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

gdje je, prisjetimo se, vektor $\phi(\mathbf{x})$ proširen “dummy” značajkom, tj. $\phi_0(\mathbf{x}) = 1$.

Ovime smo ostvarili naš cilj: primjeri iz ulaznog prostora preslikavaju se u neki prikladni prostor značajki, ali je sâm model ostao nepromijenjen – i dalje je to linearan model, jedina je razlika što umjesto značajki x_j koristimo preslikane značajke $\phi_j(\mathbf{x})$.

Ovakav model naziva se **model linearne regresije**. To je model koji je **linearan u parametrima** (w_j), ali koji – ovisno o funkciji preslikavanja – ne mora biti linearan u izvornim značajkama!

Ovdje treba paziti na suptilne razlike u nazivlju: **model linearne regresije** nije isto što i **linearna regresija**. Linearna regresija (koju smo radili prošli puta) poseban je slučaj modela linearne regresije kod kojega ne radimo baš nikakvo preslikavanje.

Sada kada smo općenito definirali funkciju preslikavanja, pogledajmo i neke konkretne funkcije preslikavanja. Uz odgovarajući odabir funkcije preslikavanja, lako možemo dobiti bilo koji od ranije razmatranih modela linearne regresije.

- Linearna višestruka regresija:

$$\phi(\mathbf{x}) = (1, x_1, x_2, \dots, x_n)$$

Ovdje zapravo nemamo nikakvog preslikavanja, tj. funkcija preslikavanja je zapravo funkcija identiteta (uz proširenje “dummy” jedinicom), $\phi(\mathbf{x}) = (1, \mathbf{x})$. S takvom funkcijom preslikavanja naš model linearne regresije je doista linearna regresija.

- Jednostruka (jednostavna) polinomijalna regresija ($n = 1$):

$$\phi(x) = (1, x, x^2, \dots, x^d)$$

Značajku x preslikavamo u polinom stupnja d , čime iz 1-dimenzijskog ulaznog prostora ($n = 1$) prelazimo u d -dimenzijski prostor značajki ($m = d$).

- Višestruka (bivarijatna) polinomijalna regresija drugog stupnja ($n = 2, d = 2$):

$$\phi(\mathbf{x}) = (1, x_1, x_2, x_1 x_2, x_1^2, x_2^2)$$

Ovdje iz dvodimenzijskog ulaznog prostora preslikavamo u 5-dimenzijski prostor značajki primjenom funkcije preslikavanja definirane kao polinom drugog stupnja s interakcijskom značajkom. Prostor značajki sastoji od izvornih značajki x_1 i x_2 , te dodatno od interakcijske značajke $x_1 x_2$ i dviju kvadratnih značajki, x_1^2 i x_2^2 .

1.3 Optimizacijski postupak

Ovime smo riješili pitanje modela. Funkcija gubitka (i, posljedično, funkcija pogreške), ista je kao i kod linearne regresije jer je izlaz identičan za oba modela: dakle, to je kvadratni gubitak odnosno funkcija kvadratne pogreške. Ostaje nam još razmotriti **optimizacijski postupak**. Kako izgleda optimizacijski postupak za model linearne regresije?

Dobra vijest je što se baš ništa suštinski ne mijenja u odnosu na ono što smo već izveli! Naime, model s funkcijom preslikavanja je i dalje linearan model. Jedino što ćemo, umjesto nad izvornom matricom dizajna \mathbf{X} , optimizaciju provesti nad matricom dizajna u kojoj smo proveli preslikavanje u prostor značajki. Tu matricu označit ćemo sa Φ .

Dakle, umjesto izvorne matrice dizajna (koja je definirana s izvornim značajkama):

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & & & & \\ 1 & x_1^{(N)} & x_2^{(N)} & \dots & x_n^{(N)} \end{pmatrix}_{N \times (n+1)}$$

sada ćemo imati matricu dizajna u kojoj smo na svakom primjeru primijenili funkciju preslikavanja $\phi(\mathbf{x})$:

$$\Phi = \begin{pmatrix} 1 & \phi_1(\mathbf{x}^{(1)}) & \dots & \phi_m(\mathbf{x}^{(1)}) \\ 1 & \phi_1(\mathbf{x}^{(2)}) & \dots & \phi_m(\mathbf{x}^{(2)}) \\ \vdots & & & \\ 1 & \phi_1(\mathbf{x}^{(N)}) & \dots & \phi_m(\mathbf{x}^{(N)}) \end{pmatrix}_{N \times (m+1)} = \begin{pmatrix} \phi(\mathbf{x}^{(1)})^T \\ \phi(\mathbf{x}^{(2)})^T \\ \vdots \\ \phi(\mathbf{x}^{(N)})^T \end{pmatrix}_{N \times (m+1)}$$

Pritom nemojmo zaboraviti na “dummy” jedinicu (prvi stupac u matrici Φ). Optimizacijskom postupku je, dakako, svejedno s kakvom matricom radi. Prije smo imali:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \mathbf{X}^+ \mathbf{y}$$

a sada jednostavno podmećemo matricu Φ umjesto \mathbf{X} :

$$\mathbf{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y} = \Phi^+ \mathbf{y}$$

gdje je $\Phi^+ = (\Phi^T \Phi)^{-1} \Phi^T$ pseudoinverz matrice dizajna Φ . Za pseudoinverz matrice Φ , vrijede, naravno, sve iste napomene kao i one koje smo prošli put dali za pseudoinverz matrice dizajna \mathbf{X} .

Ovime smo dobili ono što smo i željeli: jedan unificirani algoritam za regresiju, neovisno o tome koju funkciju preslikavanja ϕ odaberemo. Nije li to izvrsno?! No pitanje je sada: koju funkciju preslikavanja odabrat? To nas vodi do iduće teme: prenaučenost.

2 Prenaučenost

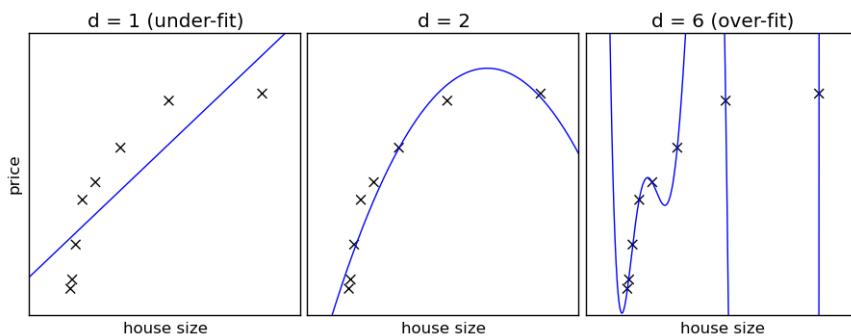
Prošli tjedan rekli smo da, ako na raspolaganju imamo familiju modela, onda trebamo odabrati optimalan model za naš problem. U suprotnom će model biti ili **prenaučen** ili **podnaučen**. Također smo rekli da su različiti modeli indeksirani njihovim **hiperparametrima**.

Kao što smo vidjeli, kod linearног modela regresije možemo ostvariti različite složenosti modela odabirom odgovarajuće funkcije preslikavanja. Što je onda hiperparametar kod linearног modela regresije? Pa, hiperparametar je upravo funkcija preslikavanja ϕ . Naime, ovisno o tome kakvu funkciju preslikavanja odabremo, model će biti više ili manje složen. Npr., model polinoma drugog stupnja je složeniji model od linearног modela. Dalje, model polinom trećeg stupnja je još složeniji, itd. Istdobro trebamo biti svjesni da, što je model složeniji, to on ima više parametara, i takav je model onda lakše prenaučiti.

► PRIMJER

Recimo da želimo naučiti model linearne regresije za predviđanje cijena nekretnina u Bostonu. Jednostavnosti radi, razmotrimo jednostavnu (tj. jednoulaznu) regresiju, $n = 1$. Neka je nezavisna varijabla stambena površina nekretnine. Razmatramo tri modela: polinom prvog stupnja (ilići pravac), polinom drugog stupnja (parabola) i polinom šestog stupnja (“sekstičan polinom”, u slučaju da vas je zanimalo). Kao što već znamo, sve te modele lako možemo definirati prikladnom funkcijom preslikavanja $\phi(\mathbf{x}) = (x_1, x_1^2, \dots, x_1^d)$, gdje je d željeni stupanj polinoma.

Neka se naš skup označenih primjera sastoji od šest primjera, tj. $|\mathcal{D}| = 6$. Nakon što so naučili tri spomenuta modela, dobivamo u ulaznom prostoru ovakve krivulje:



Što se ovdje dogodilo? Polinom prvog stupnja ne čini se baš prikladnim modelom, budući da predikcije tog modela znatno odstupaju od ciljnih vrijednosti. Taj model je podnaučen. Polinom drugog stupnja čini se vrlo dobrim na našem skupu podataka. Za taj model bismo na našem skupu \mathcal{D} rekli da je optimalne složenosti (premda nam zdrav razum govori da parabola vjerojatno nije najbolji način za modeliranje cijene stana u ovisnosti o površini; odnos je vjerojatno nelinearan, ali ne očekujemo baš da cijene za nekretnine veće površine počnu padati). Polinom šestog stupnja je apsolutni šampion u smislu da je regresijska krivulja doista prošla kroz svaku točku, međutim jasno je da je taj model prenaučen. Čini se razumnim pretpostaviti da cijena o površini nekretnine ovisi nekako nelinearno, no malo je vjerojatno da je ta ovisnost tako histerična kako ju modelira hipoteza definirana polinomom šestog stupnja. Ovaj se model naučio šumu u podatcima, pa je rezultat hipoteza koja žestoko oscilira i koja će očito loše generalizirati na neviđenim podatcima.

7

Kako sprječiti prenaučenost modela? Ovo je općenito pitanje, i koji god odgovor da smislimo on će vrijediti kako za algoritam regresije, tako i za većinu drugih algoritama strojnog učenja. Glavni načini za sprječavanje prenaučenosti su sljedeći:

- **Koristiti više primjera za učenje** – intuitivno, što imamo više primjera, to će regresijskoj krivulji biti teže da prođe baš kroz svaki od njih. Kada je primjera toliko da krivulja ne može proći kroz svaki od njih, kvadratni gubitak će se pobrinuti da parametri budu takvi da krivulja minimizira zbroj kvadratnog odstupanja, što znači da će krivulja manje oscilirati. Dakle, što više primjera imamo, to stabilnija hipoteza, odnosno to manje prenaučen model. Premda je ovaj recept za sprječavanje prenaučenosti jednostavan, u praksi je uglavnom teško izvediv: naime, u praksi je broj označenih primjera redovito ograničen, i ne možemo samo tako pribaviti nove primjere (ili ih nemamo, ili je njihovo označavanje skupo, ili je označavanje dugotrajno);
- **Odabrat model unakrsnom provjerom** – ovo nam je već poznato. Konkretno, kod modela linearne regresije, to bi značilo da na skupu za učenje treniramo modele s različitim funkcije preslikavanja, koje zatim ispitujemo na ispitnom skupu, te odabiremo onaj model koji ima najmanju pogrešku na ispitnom skupu, jer taj model najbolje generalizira na neviđene podatke. Ako je funkcija preslikavanja definirana kao polinom stupnja d , onda između takvih funkcija možemo uspostaviti potpuni uređaj po složenosti modela,

gdje složenost određuje hiperparametar d . Unakrsna provjera svodi se onda na ispitivanje hiperparametra d iz nekog unaprijed zadanog intervala, npr., $d \in \{1, 2, \dots, 6\}$;

- **Odabratи podskup značajki** – model je to složeniji što ima više značajki. Ako smanjimo broj značajki, dobit ćemo jednostavniji model. Naravno, ne bi bilo pametno ukloniti značajke koje su korisne; idealno, izbacili bismo iz modela one značajke koje su beskorisne. Postupak pronalaženja optimalnog podskupa značajki naziva se **odabir značajki** (engl. *feature selection*), i o njemu ćemo pričati na samom kraju predmeta. Budući da skup značajki zapravo definira model, na postupak odabira značajki možemo gledati kao na odabir modela;
- **Reducirati dimenzionalnost ulaznog prostora** – umjesto izbacivanja nepotrebnih značajki, možemo cijeli ulazi prostor preslikati u prostor značajki nižih dimenzija. Te nove značajke neće odgovarati izvornim značajkama, no nekad nam to nije ni bitno;
- **Regularizacija** – regularizacija je postupak kojim se implicitno sprječava odabir presloženih modela pri samom postupku optimizacije parametara modela. Regularizacija je univerzalna ideja u strojnem učenju, i o njoj ćemo pričati danas;
- **Bayesovska regresija** (odnosno, općenitije, **bayesovski odabir modela**) – bayesovski postupci u statistici i strojnem učenju oslanjaju se na ideju da se parametri modela, jednakо као и primjeri \mathbf{x} i oznake y , tretiraju kao slučajne varijable za koje je potrebno unaprijed definirati odgovarajuće distribucije (apriorne distribucije), a učenje se onda svodi na procjenu tih parametara i primjenu Bayesovog teorema kako bi se dobila aposteriorna distribucija oznake y za ulazni primjer \mathbf{x} . Mi na ovom predmetu (nažalost) nećemo pričati ništa o bayesovskim metodama.

Stanje s ovim tehnikama je sljedeće. Prva tehnika (ubaciti više primjera) nije uvijek izvediva. Iduće tri tehnike se često koriste. One striktno razdvajaju fazu odabira modela od faze učenja modela. S druge strane, zadnje dvije tehnike (regularizacija i bayesovska regresija) u nekoj mjeri spajaju faze odabira modela i učenje modela.

Mi ćemo se fokusirati na **regularizaciju**, kao jednu vrlo osnovnu, vrlo učinkovitu i često korištenu tehniku u strojnem učenju.

3 Regularizacija

Početak priče o regularizaciji kreće od jednog jednostavnog opažanja: kod linearnih modela, što je model složeniji, to ima veće vrijednosti parametara \mathbf{w} . Uzmimo kao primjer model polinoma drugog stupnja:

$$h(\mathbf{x}; \mathbf{w}) = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$$

Ako su težine w_3 , w_4 ili w_5 velike magnitude (bilo pozitivne ili negativne), hipoteza će biti izrazito nelinearna. S druge strane, što su težine w_3 , w_4 i w_5 bliže nuli, to je hipoteza manje nelinearna. U krajnosti, ako $w_3 = w_4 = w_5 = 0$, model efektivno degenerira u linearan model.

Ovaj primjer ukazuje na to da složenost linearног modela regresije izravno ovisi o težinama značajki, i da će prenaučeni modeli imati visoke vrijednosti za mnoge težine. Onda, kako bismo to spriječili, mi ćemo već pri učenju modela **ograničiti magnitudu parametara**. To ćemo raditi tako da ćemo **kažnjavati** hipoteze s visokim vrijednostima parametara. Opisani mehanizam nazivamo **regularizacija**.

U praksi, to izgleda tako da krenemo s relativno složenim modelom, kako bismo spriječili podnaučenost, ali onda koristimo regularizaciju kako bismo mu efektivno ograničili složenost. Dakle, krećemo od vrlo složenog, raskošnog modela, ali onda ga sputavamo, da se ne “razuzda” previše. Ako to sputavanje pametno napravimo, spriječit ćemo prenaučenost, odnosno ostvarit ćemo kompromis između jednostavnosti i složenosti modela.

Idealan cilj regularizacije bio bi što više parametara (težina) pritegnuti na nulu. Osim što ćemo time iz složenog modela dobiti jednostavniji model, ako mnoge težine pritegnemo baš na nulu, onda ćemo dobiti tzv. **rjetke modele** (engl. *sparse models*). U strojnom učenju volimo rijetke modele jer su (1) manje skloni prenaučenosti, (2) računalno jednostavniji kod predviđanja i (3) interpretabilniji – znamo koje značajke definitivno ne utječu na izlaz (što može biti važno u statističkoj analizi podataka).

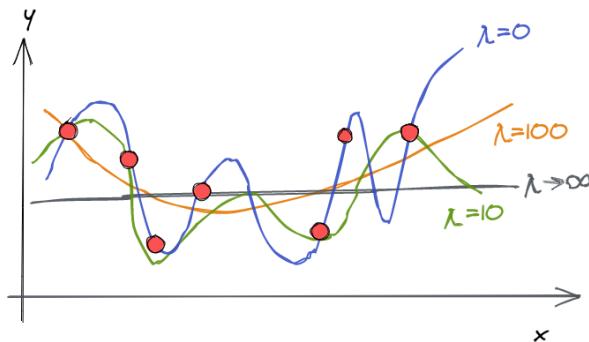
3.1 Regularizirana regresija

Pogledajmo sada kako zapravo ostvarujemo regularizaciju. Vrlo jednostavno: u funkciji pogreške (koju minimiziramo), pored empirijske pogreške, dodat ćemo još jedan član, koji karakterizira složenost modela u ovisnosti o njegovim težinama. Tako dobivamo **regulariziranu funkciju pogreške**:

$$E_R(\mathbf{w}|\mathcal{D}) = E(\mathbf{w}|\mathcal{D}) + \underbrace{\lambda\Omega(\mathbf{w})}_{\text{reg. izraz}}$$

U ovom izrazu, $\Omega(\mathbf{w})$ je **regularizacijski izraz**, a λ je tzv. **regularizacijski faktor**. Ako $\lambda = 0$, onda se vraćamo na neregulariziranu funkciju pogreške. Veća vrijednost regularizacijskog faktora λ više će kažnjavati složenost modela i imat će za posljedicu smanjenje efektivne složenosti modela.

► PRIMJER



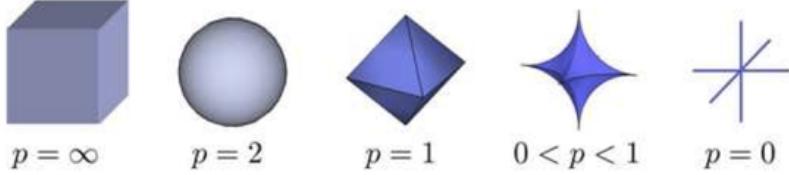
Treniramo model L2-regularizirane jednostavne regresije ($n = 1$) sa funkcijom preslikavanja definiranom kao polinom 10. stupnja. Razmotrimo četiri modela, koji se međusobno razlikuju po iznosu regularizacijskog faktora λ . Ako ne regulariziramo ($\lambda = 0$), polinom 10. stupnja na našem skupu od 7 primjera ostvaruje savršenu točnost. Kako povećavamo iznos regularizacijskog faktora λ , tako težine \mathbf{w} sve više pritežemo k nuli, što rezultira sve zaglađenijom regresijskom krivuljom u ulaznom prostoru. Npr., polinom 10. stupnja regulariziran sa $\lambda = 100$ mogao bi u našem slučaju efektivno odgovarati polinomu drugog stupnja (paraboli). U ekstremnom slučaju, kada $\lambda \rightarrow \infty$, sve težine osim težine w_0 (objasnit ćemo ispod zašto je w_0 izuzeta) pritežemo k nuli, pa dobivamo pravac nagiba nula koji os y siječe na mjestu srednje vrijednosti oznaka y . Naravno, tako snažna regularizacija nije ono što u praksi želimo.

Pogledajmo sada kako bismo konkretno definirali $\Omega(\mathbf{w})$. Vrijednost tog izraza ovisi o težinama \mathbf{w} . Treba nam neka funkcija težina \mathbf{w} , i to takva da je njezina vrijednost velika, kada su težine velike, a mala, kada su težine male. Pritom, imajmo na umu da je \mathbf{w} zapravo **vektor** težina. Kakva bi to bila funkcija? Najizravnija opcija je **norma vektora**. Naime, norma je funkcija koja daje duljinu vektora, i ona će biti to veća što su veće magnitude komponenata vektora.

U matematici postoji jedna općenita klasa vektorskih normi koja se zove **p -norma**. Općenito, dakle, regularizacijski izraz $\Omega(\mathbf{w})$ možemo definirati kao **p -normu vektora težina**:

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_p = \left(\sum_{j=1}^m |w_j|^p \right)^{\frac{1}{p}}$$

Različite p -norme možemo vizualizirati ovako:



Slike prikazuju gdje, u 3-dimenzijском простору, leže točke (odnosno vrhovi dogovarajućih vektora s hvatištem u ishodištu) koje su jednakoj udaljeno od ishodišta (odnosno vektori koji imaju istu normu). Drugim riječima, slike prikazuju kako izgleda (hiper)površina definirana funkcijom $\|\mathbf{w}\|_p = \text{konst}$. Za beskonačnu normu to je kocka, za 2-normu to je kugla, za 1-normu to je oktaedar, za 0-normu to su točke na vrhovima oktaedra (na osima). U višim dimenzijama imali bismo hiperkocku, hiperkuglu, hiperoktaedar odnosno točke u vrhovima hiperoktaedra.

U strojnog učenju tipično koristimo norme sa $p = 2$ i $p = 1$, a spomenut ćemo još i normu sa $p = 0$. Njih zovemo L2-norma, L1-norma, odnosno L0-norma. Njihove definicije slijede iz gornje definicije za općenitu p -normu:

- **L2-norma** ($p = 2$), poznata i kao **euklidska norma**:

$$\|\mathbf{w}\|_2 = \sqrt{\sum_{j=1}^m w_j^2} = \sqrt{\mathbf{w}^T \mathbf{w}}$$

- **L1-norma** ($p = 1$), poznata i kao **Manhattan-udaljenost**:

$$\|\mathbf{w}\|_1 = \sum_{j=1}^m |w_j|$$

- **L0-norma** ($p = 0$), koja je u stvari jednaka broju ne-nul elemenata vektora (tj. broju značajki koje nisu ukljonjene):

$$\|\mathbf{w}\|_0 = \sum_{j=1}^m \mathbf{1}\{w_j \neq 0\}$$

Primjetite – a to je vrlo važno – da se težina w_0 ne regularizira. Zašto? Zato što w_0 određuje odsječan pravca (odnosno općenito hiperravnine) na y -osi. Ako predviđamo veličinu stopala s obzirom na dob, onda sigurno ne želimo imati $w_0 = 0$, jer bi to značilo da novorođenče ima veličinu stopala jednaku nuli.

9

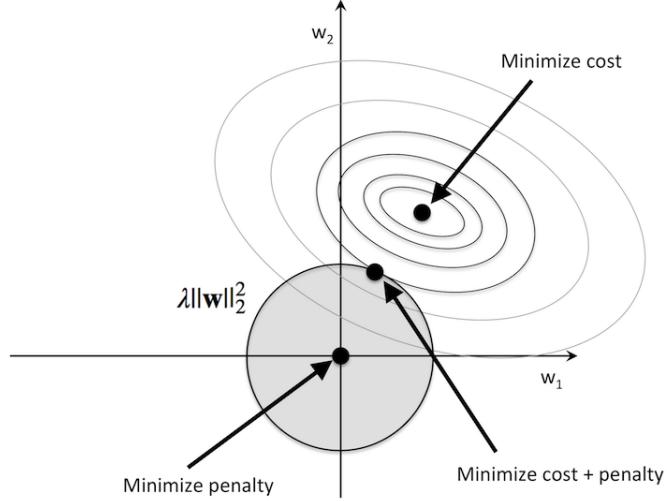
3.2 Regularizirani linearni model regresije

Pogledajmo sada kako regularizacija izgleda konkretno kod regresije. **L2-regularizacija** (ili Tikhonovljeva regularizacija) daje nam tzv. **hrbatnu regresiju** (engl. *ridge regression*):

$$E_R(\mathbf{w}|\mathcal{D}) = \frac{1}{2} \sum_{i=1}^N (\mathbf{w}^T \phi(\mathbf{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

Primijetite da koristimo L2-normu na kvadrat, i da imamo $\lambda/2$. To dvoje, zajedno sa $1/2$ ispred sume, koristimo zbog kasnije matematičke jednostavnosti. L2-regularizacija kažnjava hipotezu onoliko koliko njezine težine odstupaju od nule u smislu kvadratnog odstupanja. To znači da će veće težine biti kažnjenje više, a manje težine će biti kažnjene manje; ova spoznaja će nam biti od koristi nešto kasnije.

L2-regularizacija ima dosta intuitivnu geometrijsku interpretaciju. Kao što znamo, hipoteze su funkcije definirane parametrima \mathbf{w} . Posljedično, i pogreška $E(\mathbf{w}|\mathcal{D})$ je funkcija parametara \mathbf{w} . Možemo pogledati kao izgleda ta funkcija u prostoru parametara:



Slika prikazuje izokonture (skup točaka u kojima funkcija poprima konstantnu vrijednost) funkcije pogreške $E(\mathbf{w}|\mathcal{D})$ u prostoru parametara $w_1 \times w_2$ (parametar w_0 smo u prikazu radi preglednosti zanemarili). Eliptične konture u gornjem desnom dijelu slike odgovaraju izokonturama neregularizirane funkcije pogreške $E(\mathbf{w}|\mathcal{D})$ definirane kao zbroj kvadrata reziduala. Ta je funkcija konveksna, i njezin je minimum u središtu izokontura. Kružnica u donjem lijevom dijelu slike odgovara izokonturi L2-regularizacijskog izraza, čiji je minimum u ishodištu (jer ondje je norma vektora $\mathbf{w} = (0, 0)$ jednaka nuli). Regularizirana funkcija pogreške $E_R(\mathbf{w}|D)$ dobiva se zbrojem ovih dviju krivulja. Zbroj ovih konveksnih funkcija daje opet konveksnu funkciju. Izokonture te konveksne funkcije nisu ucrtane u gornjoj slici, ali je ucrtan njezin minimum, kao točka između minimuma neregularizirane funkcije pogreške i minimuma L2-regularizacijskog izraza (ishodišta). Na regularizaciju možemo stoga gledati kao na privlačnu silu koja minimizator funkcije pogreške primiče bliže ishodištu prostora parametara.

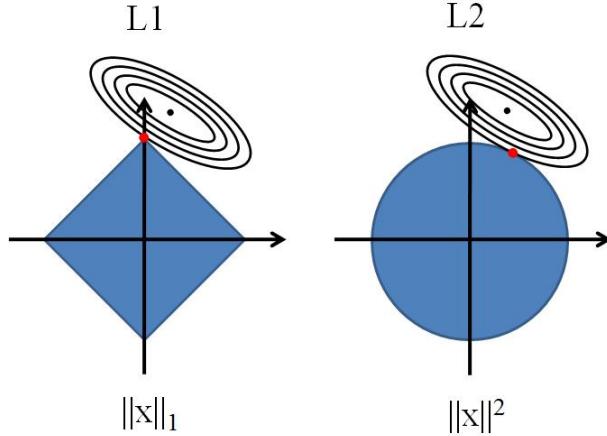
Umjesto L2-regularizacije, možemo odlučiti raditi **L1-regularizaciju** ili **LASSO regularizaciju** (engl. *least absolute shrinkage and selection operator*):

$$E_R(\mathbf{w}|\mathcal{D}) = \frac{1}{2} \sum_{i=1}^N (\mathbf{w}^T \phi(\mathbf{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2} \|\mathbf{w}\|_1$$

L1-regularizacija kažnjava hipotezu onoliko koliko njezine težine odstupaju od nule po apsolutnoj vrijednosti.

Koja je razlika između L2- i L1-regularizacije? Razlika se vidi kod malenih težina, koje su blizu nule. L1-regularizacija nema kvadrat, pa će te malene težine kažnjavati više od L2-regularizacije. S druge strane, L2-regularizacija će male težine kažnjavati vrlo malo, i zapravo će ih vrlo teško dotjerati skroz do nule. Zbog toga će L1-regularizacija općenito rezultirati modelima kod kojih je više težina pritegnuto baš na nulu, tj. L1-regularizacija rezultirat će **rjeđim modelima** nego L2-regularizacija.

Razlika između L2- i L1-regularizacije može se protumačiti i grafički, u prostoru parametara:



Slika prikazuje L1-regularizaciju (lijevo) i L2-regularizaciju (desno). Izokonture L1-regularizacijskog izraza su općenito hiperoktaedar, odnosno u dvodimenzijskome prostoru parametara to je kvadrat. S druge strane, izokonture L2-regularizacijskog izraza su hipersfere, odnosno u dvodimenzijskome prostoru parametara to je kružnica. Slika prikazuje da će se, kod L1-regularizacije, zato što izokonture regularizacijskog izraza nisu oble nego imaju vrhove, lakše dogoditi da se minimizator regularizirane funkcije pogreške nađe na nekoj od koordinatnih osi prostora parametara, tj. u jednom od vrhova kvadrata (jer vrhovi su na osima). Nadalje, ako se minimizator nalazi na nekoj od osi prostora parametara, to onda znači da je druga koordinata (tj. težina) jednaka nuli. U trodimenijskom prostoru parametara izokontura je oktaedar te, ako se minimum nađe u nekom od vrhova oktaedra, onda će druge dvije težine biti jednakne nuli. Ako se minimum nađe na bridu oktaedra, onda će jedna težina biti jednakna nula. U višedimenzijskom prostoru, svaki puta kada se minimum nađe na vrhu ili na bridu hiperoktaedra, određeni broj težina će biti jednak nuli. U višedimenzijskom prostoru vjerojatnost da se to dogodi raste s porastom brojem dimenzija, jer hiperoktaeder onda ima sve više vrhova i bridova. Iz toga onda zaključujemo da L1-regularizacija lakše dovodi do rijetkih modela (mnoge težine su pritegnute na nulu) nego L2-regularizacija.

Konačno, možemo razmišljati i o **L0-regularizaciji**, kod koje je regularizirana pogreška definirana ovako:

$$E_R(\mathbf{w}|\mathcal{D}) = \frac{1}{2} \sum_{i=1}^N (\mathbf{w}^T \phi(\mathbf{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^m \mathbf{1}\{w_j \neq 0\}$$

L0-regularizacija kažnjava hipotezu onoliko koliko ima ne-nul značajki. To znači da svaku značajku tretira kao da je ili uključena ili posve isključena. Time, efektivno, L0 regularizacija odabire najbolji podskup značajki, što znači da zapravo radi postupak **odabira značajki** (engl. *feature selection*).

Sada kada smo se upoznali sa L2, L1 i L0 regularizacijom, prirodno se nameće pitanje: koja regularizacija je najbolja? Odgovor je: u načelu preferiramo onu regularizaciju koja nam daje što rjeđe modele. To bi bila dakle L0, a onda L1, a onda tek L2. Međutim, problem u praksi predstavlja računalna složenost. Naime, L0-regularizacija je **NP-potpun problem** i nema traktabilno rješenje (treba isprobati sve kombinacije značajki, a njih je 2^m). S druge strane, L1-regularizacija je traktabilna, ali **nema rješenje u zatvorenoj formi**. Situacija je takva da, kod linearног modela regresije, jedino L2-regularizacija ima rješenje u zatvorenoj formi.

Mi ćemo u nastavku pogledati L2-regulariziranu regresiju (odnosno hrbatnu regresiju), baš zato što njezina optimizacija ima analitičko rješenje. Hrbatna regresija vrlo često koristi u praksi. U praksi se također koristi i L1-regularizacija (koja nema analitičko rješenje, pa se optimizacija radi iterativno), ili kombinacija L1-regularizacije i L2-regularizacije, koja se zove **elastična mreža** (engl. *elastic net*). Ali o tome nećemo.

3.3 Hrbatna regresija: optimizacija

Kao što smo rekli, L2-regularizirana regresija, odnosno hrbatna regresija, ima rješenje u zatvorenoj formi. Lako ga je izvesti. Pogledajmo kako.

Prisjetimo se: ovako smo izveli rješenje za težine \mathbf{w} za neregularizirani model linearne regresije:

$$\begin{aligned} E(\mathbf{w}|\mathcal{D}) &= \frac{1}{2}(\Phi\mathbf{w} - \mathbf{y})^T(\Phi\mathbf{w} - \mathbf{y}) \\ &= \frac{1}{2}(\mathbf{w}^T\Phi^T\Phi\mathbf{w} - 2\mathbf{y}^T\Phi\mathbf{w} + \mathbf{y}^T\mathbf{y}) \end{aligned}$$

Zatim, da bismo našli minimum, izračunali smo gradijent ove funkcije, izjednačili ga s nulom, i iskazali \mathbf{w} :

$$\begin{aligned} \nabla_{\mathbf{w}} E &= \Phi^T\Phi\mathbf{w} - \Phi^T\mathbf{y} \\ \mathbf{w} &= (\Phi^T\Phi)^{-1}\Phi^T\mathbf{y} \end{aligned}$$

Sada ćemo raditi isto kao i prije, samo ćemo funkciji pogreške dodati regularizacijski faktor (označeno crvenom):

$$\begin{aligned} E_R(\mathbf{w}|\mathcal{D}) &= \frac{1}{2}(\Phi\mathbf{w} - \mathbf{y})^T(\Phi\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \\ &= \frac{1}{2}(\mathbf{w}^T\Phi^T\Phi\mathbf{w} - 2\mathbf{y}^T\Phi\mathbf{w} + \mathbf{y}^T\mathbf{y} + \lambda\mathbf{w}^T\mathbf{w}) \end{aligned}$$

Sada računamo gradijent funkcije, izjednačavamo ga s nulom, i iskazujemo \mathbf{w} :

$$\begin{aligned} \nabla_{\mathbf{w}} E_R &= \Phi^T\Phi\mathbf{w} - \Phi^T\mathbf{y} + \lambda\mathbf{w} \\ &= (\Phi^T\Phi + \lambda\mathbf{I})\mathbf{w} - \Phi^T\mathbf{y} = 0 \\ \mathbf{w} &= (\Phi^T\Phi + \lambda\mathbf{I})^{-1}\Phi^T\mathbf{y} \end{aligned}$$

Ovdje smo iz matričnog diferencijalnog računa iskoristili jednakost $\frac{d}{dx}\mathbf{x}^T\mathbf{A}\mathbf{x} = \mathbf{x}^T(\mathbf{A} + \mathbf{A}^T)$, gdje je $\mathbf{A} = \mathbf{I}$, a zatim $\frac{d}{dw}\mathbf{w}^T\mathbf{w} = 2\mathbf{w}^T$.

Matrica $\lambda\mathbf{I}$ je matrica dimenzija $(m+1) \times (m+1)$ koja na dijagonalni ima λ , a izvan dijagonale nule. Dodavanjem te matrice na Gramovu matricu efektivno se ostvaruje regularizacija. Međutim, prisjetimo se da težinu w_0 želimo izuzeti iz regularizacije. To znači da je matrica $\lambda\mathbf{I}$ zapravo definirana tako da je, pored elemenata izvan dijagonale, i gornji lijevi element jednak nuli, tj. $\lambda\mathbf{I} = \text{diag}(0, \lambda, \dots, \lambda)$.

Vidimo da se regularizacija lijepo "ugradila" u rješenje najmanjih kvadrata. Rješenje doduše više nije izravno pseudoinverz Gramove matrice Φ , nego treba računati inverz zbroja Gramove matrice i matrice $\lambda\mathbf{I}$. Ovo je trivijalno proširenje u računalnom smislu. Međutim, zanimljivo je kako se regularizacija svela na dopunu vrijednosti na dijagonalni Gramove matrice. I tu možemo ispričati jednu zanimljivu priču...

4 Regularizacija i multikolinearnost

Vratimo se malo na računanje inverza matrice dizajna Φ , o kojemu smo kratko pričali prošli put. Rekli smo da inverz ne možemo računati direktno, jer se lako može dogoditi da broj primjera i broj parametara nisu jednaki, što znači da će sustav jednadžbi biti **preoodređen ili pododređen**, odnosno da matrica dizajna neće biti kvadratna, pa da nema inverz. Štoviše, rekli smo da čak i ako matrica dizajna kojim čudom bude kvadratna, i dalje može biti da nema inverz, ako odgovarajući sustav jednadžbi nije konzistentan ili ako ima višestruka rješenja. Rješenje

smo onda našli u postupku najmanjih kvadrata, koji nas je vrlo brzo doveo do **pseudoinverza** matrice dizajna:

$$\mathbf{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

Rekli smo i da pseudoinverz uvijek postoji. Stoga, čak i ako pseudoinverz Gramove matrice ne možemo izračunati pomoću inverza matrice, možemo ga izračunati postupkom rastava na singularne vrijednosti (SVD).

Međutim, to ne znači da to uvijek želimo. Objasnimo o čemu se radi.

Prvo, pogledajmo kada jest moguće izračunati pseudoinverz Gramove matrice pomoću inverza matrice. Gramova je matrica dimenzija $(m + 1) \times (m + 1)$, tj. sigurno je kvadratna. Ali, da bismo je mogli invertirati, Gramova matrica također mora biti punog ranga. Kada će Gramova matrica biti punog ranga? Kao što smo spomenuli prošli put, rang Gramove matrice jednak je rangu matrice dizajna. To znači da će Gramova matrica biti punog ranga ako i samo je rang matrice dizajna $(m + 1)$, tj. ako u matrici dizajna imamo $(m + 1)$ linearno nezavisnih stupaca, odnosno ako su sve značajke **linearno nezavisne**. Nažalost, u praksi je čest problem upravo taj da su stupci linearno zavisni. Zašto dolazi do toga? Zato što skup podataka često ima **redundantne značajke**. Pogledajmo primjer.

► PRIMJER

Radimo predviđanje osobnog prihoda, a kao značajke koristimo dob osobe u danima te kao drugu značajku dob osobe u godinama. No, te dvije značajke linearne su gotovo savršeno korelirane. Matrica dizajna Φ u tom slučaju je, npr.:

$$\Phi = \begin{pmatrix} 1 & 9511 & 26 \\ 1 & 11340 & 31 \\ 1 & 8201 & 22 \\ 1 & 18022 & 49 \end{pmatrix}$$

Prvi stupac je "dummy" značajka $x_0 = 1$, drugi stupac je značajka x_1 koja odgovara starosti osobe u broju dana, a treći stupac je značajka x_2 koja odgovara starosti osobe u godinama. Drugi i treći stupac su linearne zavisne: treći stupac možemo dobiti iz drugog tako da drugi podijelimo sa 365.25 i zaokružimo na niži cijeli broj. Ovo nije savršena linearna zavisnost (jer zaokružujemo, tj. zato što je broj dana preciznija informacija nego broj godina), ali je dovoljno blizu savršenoj linearnej zavisnosti da bi u praksi bila problematična. Ako ne bismo zaokruživali, već godine prikazali kao realan broj, imali bismo savršenu linearnu zavisnost drugog i trećeg stupca.

Ovakav fenomen zovemo **multikolinearnost**: dvije ili više ulaznih varijabli su korelirane, pa je neku varijablu moguće vrlo dobro predvidjeti kao linearnu kombinaciju jedne ili više drugih varijabli. Ekstremni slučaj multikolinearnosti je **savršena multikolinearnost**, kao u prethodnom primjeru (ako ne bismo zaokruživali broj godina). Savršena multikolinearnost, međutim, u praksi se rijetko događa. S druge strane, (nesavršena) multikolinarnost događa se razmjerno često. Npr., ako radimo predviđanje osobnog prihoda na temelju dobi i na temelju godina radnog staža, što su razumne značajke za ovaj problem, moglo bi se lako dogoditi da te dvije varijable budu visoko korelirane, što bi dovelo do multikolinearnosti.

Vratimo se sada na našu matricu dizajna. Što se događa ako je neka značajka savršena linearna kombinacija jedne ili više drugih značajki? To onda znači da matrica dizajna Φ nema rang $m + 1$. Dalje, Gramova matrica (matrica skalarnih produkata) $\Phi^T \Phi$ ima isti rang kao i matrica dizajna, što znači da ona neće imati puni rang. To onda znači da Gramova matrica nema inverz, tj. **singularna** je. Ako baš želimo, možemo izračunati pseudoinverz od Φ SVD-om, no problem je da će naše rješenje tada biti vrlo **nestabilno**. Što mislimo pod time? Mislimo da je rješenje za \mathbf{w} biti **vrlo osjetljivo na promjene vrijednosti ulaznih varijabli**. Vrlo mala promjena ulaznih varijabli može dati vrlo veliku promjenu u težinama.

Što ako su značajke **multikolinearne**, ali nisu savršeno multikolinearne? Onda je matrica dizajna punog ranga, i Gramova matrica također, pa dakle nije singularna, i možemo izračunati inverz Gramove matrice, dakle možemo izračunati pseudoinverz pomoću inverza matrice. Međutim, trebamo znati da, premda nije singularna, Gramova je matrica tada vrlo blizu tome da bude singularna, pa će naše rješenje opet biti vrlo **nestabilno**.

Takva nestabilnost rješenja je tipično ponašanje koje opažamo kod **prenučenih hipoteza**. Kod regresije, to znači da ćemo za male promjene u označenom skupu podataka imati radikalno različite izlaze. Npr., ako koristimo polinom 10. stupnja, imat ćemo vrlo različite hipoteze ovisno o malim promjenama u ulazu. Pogledajte opet primjer s predviđanjem cijena nekretnina u Bostonu. Tamo smo imali polinom 6. stupnja, čija krivulja će jako divljati za male promjene u ulaznim podatcima. To znači da će za male promjene u matrici dizajna hipoteza $h(x; \mathbf{w})$ imati vrlo različite parametre \mathbf{w} . To je znak prenaučenosti modela.

Srećom, postoji način da detektiramo ovaku nestabilnost rješenja. U numeričkoj matematici govorimo o **kondicijskom broju** matrice. Ovdje nemamo vremena ulaziti u detalje, ali je dovoljno da znate da kada je taj broj velik, onda je naše rješenje nestabilno i model je sigurno prenaučen. Za takvu matricu kažemo da je **loše kondicionirana** (engl. *ill-conditioned*).

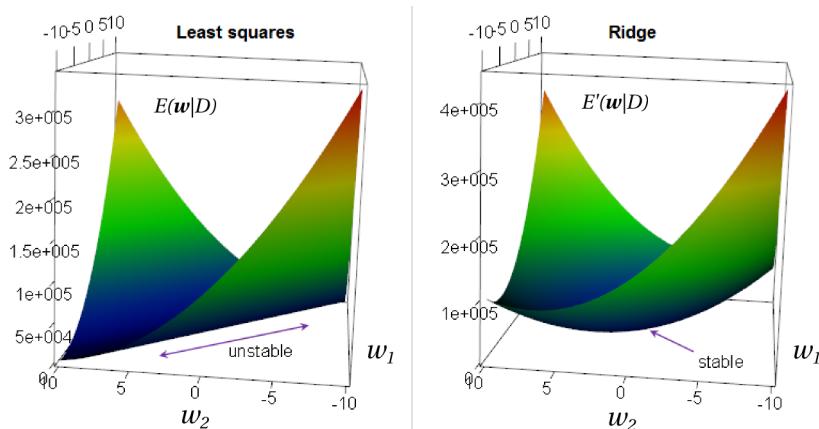
Mi svakako želimo izbjegći situaciju prenaučenosti modela. Primijetimo da, ako je model složen, a podatka je malo, dakle $m \geq N$, onda se lako dovodimo u situaciju prenaučenosti. Naime, onda će matrica dizajna biti “**široka i plitka**”, rang matrice će sigurno biti manji od $(m + 1)$, tj. bit će najviše N . To će onda ujedno biti i rang Gramove matrice, koja dakle neće biti punu ranga, pa sigurno imamo multikolinearnost!

Kako izbjegći multikolinearnost? Prvo, prije treniranja modela strojnog učenja uvijek je dobro analizirati korelacije između parova značajki te izbaciti one koje su redundantne. To je jedno rješenje. Drugo rješenje nam je već poznato: **regularizacija**! Pogledajmo, sada kada znamo da je problem pretreniranosti posljedica loše kondicioniranje Gramove matrice, kako regularizacija rješava taj problem. Konkretno, pogledajmo što regularizacija zapravo čini Gramovoj matrici:

$$\mathbf{w} = (\Phi^T \Phi + \lambda \mathbf{I})^{-1} \Phi^T \mathbf{y}$$

Vidimo da svakom stupcu Gramove matrice dodajemo vrijednost λ na drugom mjestu (tj. u drugom retku). Ako su stupci bili linearno zavisni ili skoro linearne zavisni, sada će biti manje linearne zavisne. Što je vrijednost λ veća, to je matrica koju invertiramo “sličnija” dijagonalnoj matrici, to smo dalje od singularne matrice, i to je manja multikolinearnost. U krajnosti, kada je matrica dominantno dijagonalna, sigurno nemamo multikolinearnosti. Dakle, regularizacija zapravo popravlja kondiciju Gramove matrice (smanjuje njezin kondicijski broj). Kažemo da smo matricu **rekondicionirali**.

Konačno, zanimljivo je pogledati koji je efekt rekondicioniranja matrice na oblik **funkcije pogreške** u prostoru parametara:



11

12

Slike prikazuju graf funkcije pogreške $E(\mathbf{w}|\mathcal{D})$ regresije (definirane, sjetimo se, preko kvadratne funkcije gubitka) u prostoru parametara $w_1 \times w_2$ (opet, radi preglednosti, tj. da bismo uopće mogli nacrtati graf, izostavili smo težinu w_0 , no znamo da po njoj također optimiziramo, premda ju ne regulariziramo). Kao i uvijek, tražimo vektor težina \mathbf{w} koji minimizira pogrešku. Na lijevoj slici je neregularizirana empirijska pogreška, tj. funkcija koju minimiziramo postupkom običnih najmanjih kvadrata (OLS), dok je na desnoj slici regularizirana empirijska pogreška, tj. funkcija koju minimiziramo kod hrbatne regresije (a to odgovara kriteriju najmanjih kvadrata plus regularizacijski izraz). Kod neregularizirane pogreške vidimo da se minimum nalazi u dugačkoj udolini. Zapravo, događa se to da sve točke u toj udolini, koja ovdje na slici odgovara pravcu, imaju jednaku vrijednost, dakle imamo beskonačno mnogo točaka u kojima funkcija pogreške poprima minimum. Tu udolinu zovemo "**hrbat**": područje nestabilnog rješenja, gdje je moguće beskonačno mnogo rješenja. Na desnoj slici vidimo regulariziranu funkciju pogreške. Efekt regularizacije, odnosno kondicioniranja matrice dizajna, jest taj da funkcija pogreške postaje "konveksnija". Time se smanjuje nestabilnost rješenja, i zapravo dovodimo se u situaciju da postoji samo jedan minimum. Također, intuitivno je jasno da, što je konveksnija funkcija pogreške, tj. što jača regularizacija, to će rješenje biti stabilnije. I upravo to je razlog zašto L2-regularizirana regresija nazivamo **hrbatna regresija**: jer može raditi sa takvima "hrptovima" (odnosno udolinama).

Što je poanta ove priče? Jedna vrlo konkretna poanta jest da na regularizaciju možemo gledati kao na "popravljanje" Gramove matrice kojim se odmičemo od nestabilnih rješenja (prenaučenih hipoteza). Druga, šira poanta priče jest da u strojnem učenju na jednu te istu stvar često možemo gledati na više različitih, međusobno povezanih načina. Jedna od glavnih svrha ovog predmeta jest da vam to osvijestimo.

5 Napomene

Završit ćemo s nekoliko općenitih napomena o linearном modelu regresije.

- Magnituda parametra w_i odgovara **važnosti značajke**, a predznak upućuje na njezin utjecaj (pozitivan ili negativan) na izlaznu vrijednost.
- Međutim, ako je model prenaučen (postoje multikolinearne značajke), onda magnituda parametra ne znači ništa. U tom slučaju, dobro je koristiti regularizaciju.
- Regularizacija **sprječava prenaučenost** na način da smanjuje složenost modela prigušujući težine pojedinih značajki, odnosno efektivno ih izbacuje (kada $w_j \rightarrow 0$).
- Ako je model nelinearan, regularizacijom smanjujemo nelinearnost.
- Težinu w_0 treba izuzeti iz regularizacijskog izraza (jer ona definira odsječak) ili treba centrirati podatke tako da $\bar{y} = 0$, jer onda $w_0 \rightarrow 0$.
- L2-regularizacija kažnjava težine proporcionalno njihovom iznosu (velike težine više, a manje težine manje). Zbog toga se teško događa da težine budu pritegnute baš na nulu, i zbog toga L2-regularizacija ne rezultira rijetkim modelima.
- L1-regularizirana regresija rezultira rijetkim modelima, ali nema rješenja u zatvorenoj formi (međutim, mogu se koristiti iterativni optimizacijski postupci).
- Regularizacija je korisna kod modela s puno parametara, jer je takve modele lako prenaučiti.
- Regularizacija smanjuje mogućnost prenaučenosti i multikolinearnosti, ali ostaje problem odabira hiperparametra λ . Kao što već sigurno pogađate, taj se odabir najčešće radi **unakrsnom provjerom**. Što bismo kao optimalnu vrijednost za λ kada bismo optimizaciju radili na skupu za učenje? Odgovor je da bismo bismo sigurno odabrali $\lambda = 0$, jer to daje najsloženiji model koji bi imao najmanju pogrešku na označenom skupu primjera za učenje. I to naravno ne bi bilo dobro. Zato ćemo λ odabrati unakrsnom provjerom.

Sažetak

- **Linearan model regresije** linearan je u parametrima
- Nelinearnost regresijske funkcije ostvaruje se preslikavanjem iz ulaznog prostora u **prostor značajki** pomoću nelinearnih **baznih funkcija**
- Parametri linearog modela uz kvadratnu funkciju gubitka imaju rješenje u zatvorenoj formi u obliku **pseudoinverza matrice dizajna**, neovisno o tome koje preslikavanju u prostor značajki koristimo
- **Regularizacija smanjuje prenaučenost** ugradnjom dodatnog izraza u funkciju pogreške kojim se kažnjava složenost modela
- **L2-regularizirana regresija** ima rješenje u zatvorenoj formi, te efektivno **rekondicionira** matricu dizajna odnosno **uklanja multikolinearnost** značajki

Bilješke

- 1 Ovo predavanje je lepršavija varijanta poglavlja o regresiji iz (nedovršene) skripte: http://www.fer.unizg.hr/_download/repository/SU-Regresija.pdf.
- 2 U jeziku numeričke matematike, uporaba polinomijalne funkcije za modeliranje nelienarnosti funkcije $h(\mathbf{x}; \mathbf{w})$ jest problem **polinomijalne interpolacije**. Jedna alternativa polinomijalnoj interpretaciji, a koja se vrlo često koristi u statistici, jest interpolacija pomoću **spline funkcija**. Spline funkcija je funkcija koja je definirana po dijelovima pomoću funkcije polinoma. Interpolacija pomoću spline funkcije daje slične rezultate kao i polinomijalna interpolacija, ali je rezultat stabilniji, u smislu da, za razliku od polinomijalne funkcije, spline funkcije ne dovode do oscilacija pri rubovima interpolacijskog intervala (tzv. **Rungeov fenomen**), što je u stvari manifestacija **prenaučenosti modela**. U statistici se dominantno koriste **ograničene (prirodne) kubne spline funkcije** (engl. *restricted (natural) cubic splines*), koje su po dijelovima definirane kao polinom trećeg stupnja, uz dodatnu ogragu da je spline funkcija linearna prije prvog segmenta i na kraju zadnjeg segmenta. Više ovdje: <http://fourier.eng.hmc.edu/e176/lectures/ch7/node6.html>.
- 3 Ideja da umjesto modela mijenjamo podatke podsjeća na priču o plavom i ružičastom slonu: Kako upucati plavog slona? Puškom za plave slonove. Kako upucati ružičastog slona? Zadaviti slona dok ne poplavi, onda ga upucati puškom za plavog slona. (Niti jedan slon nije nastradao u ovoj lošoj šali.) Manje nasilan primjer: utrkujete se, imate fiću i loš je u zavojima. Umjesto da pređete na bolid, ostanite u fići, ali izravnajte cestu.
- 4 Nameće se pitanje: možemo li preslikavati u prostor niže dimenzije, tj. da je $m < n$? Možemo, naravno. Metode **redukcije dimenzionalnosti** (koje nećemo raditi na ovom predmetu), poput PCA i MSD, zapravo preslikavaju u prostor niže dimenzije, s ciljem da u tom prostoru podatci budu bolje objašnjivi i/ili da model bude manje sklon prenaučenosti. Također, možemo preslikavati u prostor značajki koji je iste dimenzije kao i ulazni prostor (tj. $m = n$); takav primjer imat ćemo u zadatcima za vježbu na temu **jezgrenih funkcija** (V10, zadatak 2), koje služe za implicitno preslikavanje u prostor značajki. Međutim, ako je naš cilj ostvariti nelinearnost (tj. nelinearnu granicu između klasa kod klasifikacije, odnosno nelinearnu regresijsku funkciju kod regresije), premda je moguće da nelinearnim preslikavanjem u prostor iste ili manje dimenzije ostvarimo željenu nelinearnost, vjerojatnije je da ćemo to ostvariti ako preslikavamo u prostor veće dimenzije. Ova je intuicija iskazana **Coverovim teoremom**, v. https://en.wikipedia.org/wiki/Cover%27s_theorem. Ako vas zanimaju koji su drugi važni teoremi strojnog učenja, pogledajte ovu raspravu: <https://stats.stackexchange.com/q/321851/93766> (na ovom predmetu spomenut ćemo, bez ulaska u detalje, tri od ovdje navedenih teorema).
- 5 Dakle, pazite: **model linearne regresije \neq linearna regresija**. Vidjet ćemo tijekom semestra da je strojno učenje puno takvih terminoloških mina kojima je lako zbuniti protivnika. Svako znanstveno područje ima svojih terminoloških muka, a pogotovo ona koja u kojima se stapaju utjecaji više drugih

područja i istraživačkih zajednica, kao što je to slučaj sa strojnim učenjem. No, kad god postoji mogućnost terminološke zabune, mi ćemo to posebno istaknuti.

- 6 Primijetite da, kada govorimo o dimenziji ulaznog prostora ili prostora značajki, "dummy" značajku x_0 odnosno $\phi_0(\mathbf{x})$ ne brojimo kao zasebnu dimenziju. Tako, naprimjer, kažemo da preslikavanje $\phi(\mathbf{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$ inducira 5-dimenzijski (a ne 6-dimenzijski) prostor značajki. Ovo je pitanje konvencije. Naime, ako je model definiran kao $h(\mathbf{x}) = w_0 + w_1x_1 + \dots + w_nx_n$, onda je on zapravo **afina funkcija** u \mathbb{R}^n . No, svaka se afina funkcija u \mathbb{R}^n može napisati kao **homogena linearna funkcija** u \mathbb{R}^{n+1} primjenjena na transformaciju koja dodaje "dummy" jedinicu vektoru značajki \mathbf{x} , i onda imamo $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. O homogenim funkcijama: https://en.wikipedia.org/wiki/Homogeneous_function.
- 7 Prema **Lagrangeovom interpolacijskom teoremu**, za skup $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ od N primjera postoji jedinstven polinom $h(x)$ stupnja ne većeg od $N - 1$ za koji vrijedi $h(x^{(i)}) = y^{(i)}$ za $i = 1, \dots, N$, tj. taj polinom prolazi kroz svaku točku iz \mathcal{D} (v. https://en.wikipedia.org/wiki/Polynomial_interpolation#Interpolation_theorem). U primjeru s nekretninama iz Bostona, za ovaj konkretan skup od 8 točaka dovoljan je već polinom stupnja 6.
- 8 Dobar uvod u **bayesovsku regresiju** možete naći u Bishopovoj knjizi, poglavlju 3.3 (Bishop, 2006). Štovateljima R-a mogao bi se svidjeti i poglavlje o bayesovskoj regresiji iz online knjige (Clyde et al., 2018), dostupno na <https://statswithr.github.io/book/introduction-to-bayesian-regression.html>. Standardna referenca za bayesovske statističke metode općenito je (Gelman et al., 2013). Za kraći uvod u **bayesovski odabir modela**, pogledajte (Bishop, 2006), poglavlje 3.4, ili https://www.cse.wustl.edu/~garnett/cse515t/fall_2019/files/lecture_notes/7.pdf. Vrlo dobar pregled postupaka daju (Hooten and Hobbs, 2015).
- 9 Alternativno, možemo centrirati podatke (npr., standardizacijom), tako da ih dovedemo do ishodišta, i onda je $w_0 = 0$.
- 10 **Savršena multikolinearnost** događa se kada radimo na sirovim podatcima, koji često koriste redundantne značajke. Kada se redundantne značajke uklone, često se događa da su preostale varijable (nesavršeno) multikolinearne (zbog inherentnih korelacija u podatcima).
- 11 Matematički gledano, dodajemo lambdu na **svojstvene vrijednosti** Gramove matrice, pa će sve svojstvene vrijednosti biti različite od nule, pa onda matrica neće biti singularna. Tehnički gledano, može se dogoditi da matrica nije imala multikolinearnosti, ali da smo upravo dodavanjem lambdi na dijagonalu doveli matricu do pogrešne kondicioniranosti. To je moguće, ali u praksi dosta malo vjerojatno.
- 12 Uz manje izmjene, slika sam preuzeo sa <https://stats.stackexchange.com/a/151351/93766>.
- 13 U geologiji, hrbat (greben) je izbočeni i izduženi dio brda. Budući da mi tražimo minimum a ne maksimum, zapravo umjesto hrpta imamo "negativan hrbat" ili udolinu. Usput, ako vas muči kao što muči mene, ovdje je deklinacija riječi "hrbat": http://hjp.znanje.hr/index.php?show=search_by_id&id=fVxvWhQ%3D.
- 14 Pažljivi čitatelj ovdje će možda primijetiti moguću cirkularnost: regularizaciju smo predstavili kao tehniku za sprječavanje prenaučenosti koja je alternativa unakrsnoj provjeri, i kod koje se složenost modela prilagođava podatcima, no sada ispada da ipak treba unakrsna provjera da bismo optimizirali hiperparametar λ (regularizacijski faktor). Točno je da nam unakrsna provjera ipak treba. Međutim, to ne umanjuje korisnost regularizacije. Naime, optimizacijom hiperparametra λ mi modelu dopuštamo da svoju složenost više ili manje prilagodi podatcima, ali mu i dalje dajemo slobodu na koji način da se prilagodi (koje težine više a koje manje pritegnuti nuli). U tom smislu, odabir hiperparametra λ , za razliku od izravnog odabira modela, je na neki način "soft" odabir modela, gdje ostavljamo mogućnost da se složenost modela bolje prilagodi podatcima. Druga prednost regularizacije nad izravnom optimizacijom hiperparametara vidi se u situacijama kada model ima više hiperparametara: tada nam je lakše optimizirati samo jedan "master" hiperparametar (regularizacijski faktor λ) nego više hiperparametara odjednom.

Literatura

- C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
- M. Clyde, M. Q. Rundel, C. Rundel, D. Banks, C. Chai, and L. Huang. An introduction to bayesian thinking, 2018.
- A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian data analysis*. CRC press, 2013.
- M. B. Hooten and N. T. Hobbs. A guide to bayesian model selection for ecologists. *Ecological Monographs*, 85(1):3–28, 2015.

5. Linearni diskriminativni modeli

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v2.3

Prošli tjedan bavili smo se regresijom te smo razmatrali **linearan model regresije**, koji – uz odgovarajući odabir nelinearne funkcije preslikavanja u prostor značajki – zapravo postaje nelinearan model. Vidjeli smo da, neovisno o tome je li model linearan ili nelinearan, postupak najmanjih kvadrata daje rješenje u zatvorenoj formi, koje se svodi na izračun **pseudoinverza**. Također, pričali smo o **regularizaciji**, kojom se sprječava prenaučenost, i pokazali smo da i za L2-regularizirani linearan model regresije postoji rješenje u zatvorenoj formi.

Danas se više nećemo baviti regresijom, nego **klasifikacijom**. Zapravo, u nastavku predmeta uglavnom ćemo se baviti klasifikacijom. Prisjetimo se: klasifikacija je postupak predviđanja diskretnih oznaka pojedinačnih primjera, npr., klasifikacija elektroničke pošte u spam i ne-spam, ili klasifikacija poruka na Twitteru u pozitivne, negativne i neutralne s obzirom na sentiment.

Naravno, postoji više pristupa klasifikaciji. Mi ćemo se prvih nekoliko tjedana fokusirati na **linearne modele**. Konkretnije, bavit ćemo se modelima koji pripadaju grupi **linearnih diskriminativnih modela**. Kao što ćete vidjeti, to nije jedan model, već čitava familija, od kojih su neki vrlo učinkoviti (npr., logistička regresija i stroj potpornih vektora).

Naš današnji cilj jest objasniti osnovnu ideju linearnih diskriminativnih modela, a onda ćemo u naredna dva tjedna ovu temu detaljnije razraditi.

1 Linearni diskriminativni modeli

Krenimo od toga da naprije razjasnimo naslov današnjeg predavanja: što su **linearni diskriminativni** modeli? Usredotočimo se najprije na “linearan”. Da je model linearan znači da modelira linearu granicu između klase: npr., pravac (ako je ulazni prostor dvodimenzionalni) ili ravnina (ako je ulazni prostor trodimenzionalni) ili hiperravnina (za prostore s više od tri dimenzije).

Kako ćemo definirati linearan model? Pa, krenimo od modela regresije:

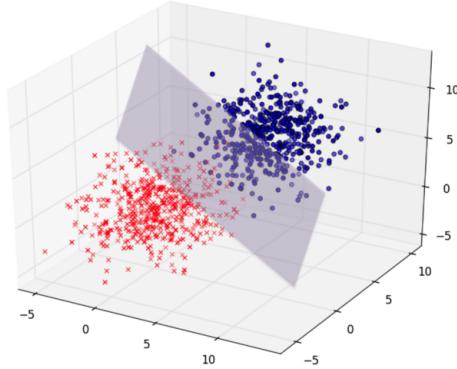
$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

gdje opet imamo onu “dummy” značajku $x_0 = 1$, kako bismo mogli koristiti ovaj jednostavniji vektorski zapis. Ovako definirana hipoteza (tj. model s fiksiranim težinama \mathbf{w}) $h(\mathbf{x}; \mathbf{w})$ daje neku vrijednost iz \mathbb{R} za svaki ulazni vektor \mathbf{x} iz \mathbb{R}^n , tj. $h(\mathbf{x}; \mathbf{w}) : \mathbb{R}^n \rightarrow \mathbb{R}$.

Kako možemo na ovaj model gledati kao na **binarni klasifikacijski model**? Tako da razmišljamo kako ovaj model zapravo dijeli ulazni prostor u dva **poluprostora**: jedan čine sve točke u \mathbb{R}^n za koje $h(\mathbf{x}) \geq 0$, a drugi sve točke u \mathbb{R}^n za koje $h(\mathbf{x}) < 0$. Binarni klasifikator onda bismo mogli definirati ovako:

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{1}\{\mathbf{w}^T \mathbf{x} \geq 0\}$$

gdje je $\mathbf{1}\{\cdot\} : \{\perp, \top\} \rightarrow \{0, 1\}$ indikacijska funkcija koju smo već bili definirali. Granica između klase je točno tamo gdje $h(\mathbf{x}) = 0$ (i te točke koje leže na granici također trebamo svrstati u jedan od dva poluprostora, svejedno je u koji). Za dvodimenzionalni ulazni prostor ta granica je pravac, za trodimenzionalni to je ravnina, a općenito to je $(n - 1)$ -dimenijska hiperravnina ugrađena u n -dimenijski ulazni prostor (“dummy” značajku x_0 ne računamo u dimenzije ulaznog prostora). Npr., za $n = 3$ to bi izgledalo ovako:

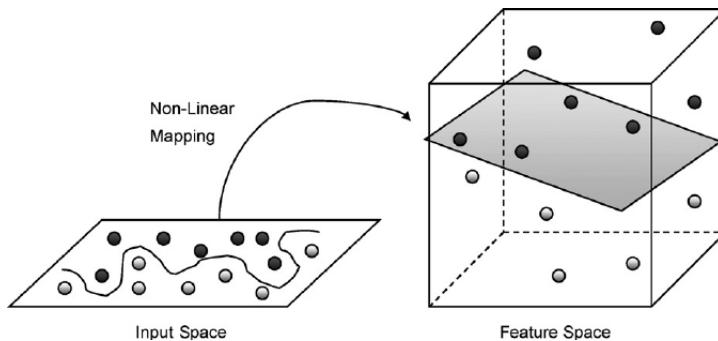


Općenito, dakle, granica je **potprostor** (engl. *subspace*) dimenzije $n - 1$ (uvijek za jedan manje od dimenzije ulaznog prostora) koji ulazni prostor dijeli na dva **poluprostora** (engl. *half-spaces*). Tu granicu zovemo **diskriminantna funkcija** (engl. *discriminative function*) ili **granica odluke** ili **decizijska granica** (engl. *decision boundary*), ili jednostavno kažemo **granica između klasa**.

No, što je s nelinearnošću? Znamo da su linearne granice između klasa u stvarnim problemima zapravo rijetkost; većina problema u strojnem učenju je takva da je granica između klasa nelinearna. Kako možemo dobiti nelinearnu granicu? Pa, ako želimo nelinearnu granicu, možemo iskoristiti trik koji smo naučili prošli put: možemo upotrijebiti **nelinearnu funkciju preslikavanja** kako bismo naše podatke iz ulaznog prostora preslikali u neki drugi prostor (prostor značajki), i tamo ih onda napali linearnim modelom. Dakle, uz funkciju preslikavanja $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^{m+1}$ dobivamo ovakav model:

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x})$$

Ovaj model još uvijek ima linearnu granicu u prostoru značajki, no – ako je preslikavanje ϕ nelinearno – granica u ulaznom prostoru bit će nelinearna. Tehnički, međutim, to je i dalje **linearan model**, jer je linearan u parametrima. Prisjetimo se primjera koji smo pokazali prošli put:



Ovdje preslikavamo iz ulaznog prostora dimenzije $n = 2$ u prostor značajki dimenzije $m = 3$. Ono što nije bilo linearno odvojivo u ulaznom prostoru sada je linearno odvojivo u prostoru značajki. Granica između klasa koja je linearna u prostoru značajki, nelinearna je u ulaznom prostoru.

► PRIMJER

Jednostavan primjer kako preslikavanje u prostor značajki više dimenzije od ulaznog prostora može problem koji je nelinearan u ulaznom prostoru učiniti linearnim u prostoru značajka jest poznati **XOR-problem** (problem “isključivog ili”). Riječ je o binarnom klasifikacijskom problemu definiranom u

dvodimenziskome ulaznom prostoru $\mathcal{X} = \{-1, +1\}^2$ sa sljedećim skupom označenih primjera:

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_i = \{((-1, -1), 0), ((+1, +1), 0), ((-1, +1), 1), ((+1, -1), 1)\}$$

Ovaj problem ne možemo riješiti linearnim modelom. Drugim riječima, ako je \mathcal{H} linearan model (skup pravaca), onda $\exists h \in \mathcal{H} : h(\mathcal{D}) = 0$. Međutim, ako primjere iz \mathcal{D} iz dvodimenziskoga ulaznog prostora preslikamo u trodimenziski prostor značajki pomoći sljedeće funkcije preslikavanja:

$$\phi(\mathbf{x}) = (1, x_1, x_2, x_1x_2)$$

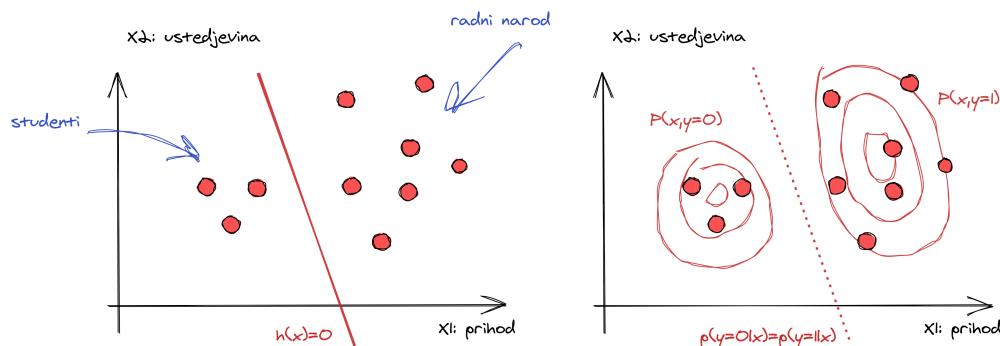
onda su primjeri iz \mathcal{D} u prostoru značajki odvojivi (postoji ravnina takva da su primjeri iz klase $y = 0$ iznad te ravnine, a primjeri iz klase $y = 1$ ispod nje).

Toliko o prvom dijelu naziva: "linearan" model. Pogledajmo sada što znači da je model "diskriminativan". **Diskriminativni modeli** su modeli koji modeliraju granicu koja diskriminira (razlikuje) između klasa, i to tako da parametri modela opisuju tu granicu i ništa više osim te granice. Drugim riječima, diskriminativni model imat će onoliko parametara koliko je potrebno da bi se definirala ta granica.

Ovo se možda čini očitim, no postaje zanimljivije ako razmotrimo što je alternativa. Diskriminativni modeli suprotstavljeni su jednoj drugoj velikoj familiji modela, koji se nazivaju **generativni modeli**. Generativni modeli modeliraju više od granice između klasa, i samu granicu zapravo modeliraju indirektno. Također ipično imaju mnogo više parametara nego diskriminativni modeli. Pomoći će da pogledamo jedan primjer.

► PRIMJER

Radimo klasifikaciju kreditno sposobnih klijenata banke: banka treba odlučiti kome će dati kredit. Kao što smo to već napravili u uvodnome predavanju, pojednostavimo problem i pretpostavimo da banke to rade na temelju svega dvije značajke: prihod i uštedjelina. Onda je ulazni prostor dvodimenziski ($x_1 \rightarrow$ prihod, $x_2 \rightarrow$ uštedjelina), a svaki klijent banke je jedan dvodimenziski vektor u tom prostoru. Neka to izgleda ovako:



Na lijevoj slici prikazana je (linearna) granica između klasa kakvu nalazi diskriminativni model. Granica će biti opisana parametrima tog modela. Na desnoj slici prikazan je generativni model. Generativni model modelirat će vjerojatnosne gustoće svake klase, koje su na slici prikazane kao izokonture funkcije gustoće vjerojatnosti. Granica između klasa sada se može izračunati indirektno na temelju tih distribucija, kao sve one točke za koje je vrijednost funkcije gustoće vjerojatnosti jednaka za objekklase (ta je granica prikazana crtanom linijom i ona je opet linear). To je više informacija nego što nudi diskriminativni model, ali te nam informacije možda nisu potrebne.

O generativnim modelima pričat ćemo detaljnije u drugom dijelu predmeta. Danas se fokusiramo na linearne diskriminativne modele. Sada kada smo objasnili pojmove "linearan" i

“diskriminativan”, pogledajmo malo detaljnije kako parametri linearne diskriminativne modela definiraju granicu u ulaznom prostoru, koji je zapravo euklidski vektorski prostor, tj. pogledajmo “geometriju” linearne modela.

2 Geometrija linearne modela

Već smo rekli da su kod linearnih diskriminativnih modela granica između klasa **hiperravnine**. Radi jednostavnosti, fokusirajmo se najprije na slučaj dviju klasa, $K = 2$, pa ćemo poslije proširiti na više klasa. Takođe, bez smanjenja općenitosti, fokusirajmo se na najjednostavniji model gdje nemamo preslikavanje u prostor značajki, dakle:

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

Zapravo, bit će nam lakše ako težinu w_0 izdvojimo iz vektora \mathbf{w} i tretiramo ju zasebno, pa imamo:

$$h(\mathbf{x}; w_0, \mathbf{w}) = \mathbf{w}^T \mathbf{x} + w_0$$

Granica između klasa dana je jednadžbom $h(\mathbf{x}) = 0$, što definira $(n - 1)$ -dimenzijsku hiperplaninu unutar n -dimenzijskog ulaznog prostora. Opet bez smanjenja općenitosti, u nastavku ćemo si pojednostaviti život i skicirati slučaj dvodimenzijskog ulaznog prostora, $n = 2$, u kojem je granica između klasa pravac. Dakle, model je onda:

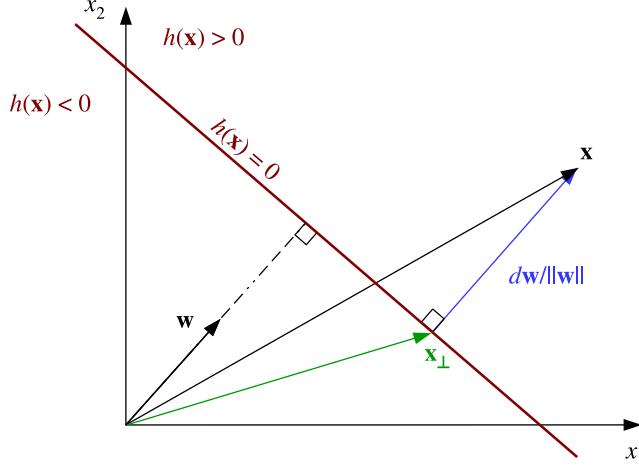
$$h(\mathbf{x}; w_0, \mathbf{w}) = w_1 x_1 + w_2 x_2 + w_0$$

a granica između klasa je:

$$w_1 x_1 + w_2 x_2 + w_0 = 0$$

što prepoznajemo kao implicitnu jednadžbu pravca. Međutim, u nastavku ćemo i dalje govoriti “hiperravnina”, jer će naši zaključci vrijediti općenito.

Za daljnja razmatranja poslužit ćemo se sljedećom slikom:



Slika prikazuje pravac (prikazan crveno) kao granicu u dvodimenzijskom ulaznom prostoru \mathbb{R}^2 sa značajkama x_1 i x_2 . Taj pravac odgovara jednadžbi $h(\mathbf{x}; w_0, \mathbf{w}) = 0$, tj. granicu čine točke (x_1, x_2) za koje hipoteza h daje vrijednost nula. Pravac dvodimenzijski prostor dijeli na dva poluprostora: jedan poluprostor su sve točke (vektori) \mathbf{x} za koje $h(\mathbf{x}; w_0, \mathbf{w}) > 0$, a drugi sve točke (vektori) \mathbf{x} za koje $h(\mathbf{x}; w_0, \mathbf{w}) < 0$. Normala pravca je vektor \mathbf{w} , koji je okomit na pravac, i koji pokazuje u smjeru poluprostora za koji $h(\mathbf{x}; w_0, \mathbf{w}) > 0$ (što bi, ako binarni klasifikator definiramo kao $h(\mathbf{x}; w_0, \mathbf{w}) = \mathbf{1}\{\mathbf{w}^T \mathbf{x} \geq 0\}$, bio poluprostor pozitivnih primjera). Ovdje je vektor normale \mathbf{w} prikazan kao vektor s početnom točkom u ishodištu koordinatnog sustava (tj. kao

“radijvektor”), no početna točka tog vektora je, naravno, proizvoljna. Na slici je još označen vektor nekog primjera \mathbf{x} , koji se nalazi negdje u poluprostoru pozitivnih primjera. Prisjetite se da je svaki primjer \mathbf{x} zapravo vektor, pa ga možemo prikazati kao radijvektor (vektor s početnom točkom u ishodištu koordinatnog sustava). Na slici je nadalje prikazan rastav vektora \mathbf{x} na zbroj dvaju vektora: $d \frac{\mathbf{w}}{\|\mathbf{w}\|}$ (prikazan plavo) i \mathbf{x}_\perp (prikazan zeleno); o tome više u nastavku.

Glavna stvar koja nas ovdje zanima jest s koje se strane hiperravnine nalazi neki primjer te koliko je od nje udaljen. Jasno je da nas zanima s koje se strane nalazi primjer, jer to određuje klasifikaciju primjera. No, zašto nam je zanimljiva udaljenost? Udaljenost nas zanima zato što je ona indikativna za **pouzdanost** klasifikacije primjera: koliko možemo biti sigurni da je primjer pozitivan ili negativan. Npr., kada se primjer nalazi daleko od hiperavnine u pozitivnoj strani, onda smo sigurniji da je primjer pozitivan, nego kada je vrlo blizu toj hiperravnini.

S ciljem, dakle, da utvrdimo stranu i udaljenost na kojoj se nalazi primjer, razmotrit ćemo dvije stvari: (1) što zapravo predstavlja vektor težina \mathbf{w} (bez w_0) i (2) kako izračunati udaljenost primjera od hiperravnine.

Prvo, što je zapravo vektor težina \mathbf{w} ? Razmotrimo dva primjera, odnosno dvije točke $\mathbf{x}^{(1)}$ i $\mathbf{x}^{(2)}$, koje leže na hiperravnini. Za njih vrijedi:

$$h(\mathbf{x}^{(1)}) = h(\mathbf{x}^{(2)}) = 0$$

Ako umjesto $h(\mathbf{x})$ uvrstimo definiciju modela, onda imamo:

$$\begin{aligned} \mathbf{w}^T \mathbf{x}^{(1)} + w_0 &= \mathbf{w}^T \mathbf{x}^{(2)} + w_0 \\ \mathbf{w}^T (\mathbf{x}^{(1)} - \mathbf{x}^{(2)}) + w_0 - w_0 &= 0 \\ \mathbf{w}^T (\mathbf{x}^{(1)} - \mathbf{x}^{(2)}) &= 0 \end{aligned}$$

Primijetimo da je $(\mathbf{x}^{(1)} - \mathbf{x}^{(2)})$ razlika dvaju vektora, što je opet vektor, i da taj vektor leži upravo na hiperravnini. Budući da smo upravo izveli $\mathbf{w}^T (\mathbf{x}^{(1)} - \mathbf{x}^{(2)}) = 0$, a znajući da skalarni produkt dvaju vektora iščezava kada su ti vektori međusobno **okomiti**, to znači da je vektor \mathbf{w} okomit na sve vektore koji leži na hiperravnini, pa zaključujemo da je vektor \mathbf{w} **normala hiperravnine**. Konkretno, na našoj prethodnoj slici \mathbf{w} je normala pravca, budući da imamo dvodimenzionalni ulazni prostor. [2]

Kao što smo rekli, druga stvar koja nas zanima jest koliko je **udaljenost** nekog primjera od hiperravnine. Promotrimo dakle neku točku \mathbf{x} koja je udaljena od hiperravnine. Ta točka ima svoju projekciju na hiperravninu – označimo tu točku sa \mathbf{x}_\perp . Sada vektor \mathbf{x} možemo rastaviti na zbroj dvaju vektora:

$$\mathbf{x} = \mathbf{x}_\perp + d \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

Ovaj drugi vektor je jedinični vektor normale, $\mathbf{w}/\|\mathbf{w}\|$, pomnožen sa d . Dakle, d je zapravo udaljenost točke \mathbf{x} od hiperravnine, i to je ono što nas zanima koliko iznosi.

A sad malo algebarske magije. Pomnožimo obje strane jednadžbe sa \mathbf{w}^T i dodajmo s obje strane w_0 :

$$\begin{aligned} \underbrace{\mathbf{w}^T \mathbf{x} + w_0}_{h(\mathbf{x})} &= \underbrace{\mathbf{w}^T \mathbf{x}_\perp + w_0}_{=h(\mathbf{x}_\perp)=0} + d \frac{\mathbf{w}^T \mathbf{w}}{\|\mathbf{w}\|} \\ h(\mathbf{x}) &= d \frac{\mathbf{w}^T \mathbf{w}}{\|\mathbf{w}\|} = d \|\mathbf{w}\| \end{aligned}$$

gdje smo iskoristili $\mathbf{w}^T \mathbf{w} = \|\mathbf{w}\|^2$. Iz ovoga dobivamo da je udaljenost d primjera \mathbf{x} od hiperavnine s vektorom normale \mathbf{w} jednaka:

$$d = \frac{h(\mathbf{x})}{\|\mathbf{w}\|}$$

Primijetite da ova udaljenost može biti pozitivna ili negativna, pa kažemo da je to **predznačena udaljenost** (engl. *signed distance*). Konkretno, imamo:

- $d > 0 \Rightarrow \mathbf{x}$ je na strani hiperravnine u smjeru normale \mathbf{w}
- $d < 0 \Rightarrow \mathbf{x}$ je na suprotnoj strani hiperravnine
- $d = 0 \Rightarrow \mathbf{x}$ je točno na hiperravnini

Sva ova naša razmatranja vrijede i za prostore dimenzije veće od dva, $n > 2$. Dakle, vektor \mathbf{w} (bez w_0) je normala $(n - 1)$ -dimenzijske ravnine u prostoru \mathbb{R}^n , i on pokazuje u smjeru poluprostora za koji $h(\mathbf{x}) > 0$. Predznačena udaljenost primjera \mathbf{w} od hiperravnine $h(\mathbf{x}) = 0$ jednaka je $h(\mathbf{x})/\|\mathbf{w}\|$. Također se može pokazati, no to ćemo preskočiti, da je udaljenost hiperravnine od ishodišta jednaka $-w_0/\|\mathbf{w}\|$.

Uočimo na ovom mjestu još i da se jedna te ista hiperravnina može definirati s različitim težinama \mathbf{w} : točnije, postoji beskonačno mnogo vektora težina (w_0, \mathbf{w}) koji definiraju identičnu hiperravninu. Naime, ako pomnožimo vektor (w_0, \mathbf{w}) s nekim faktorom α , onda će vektor normale \mathbf{w} biti α -puta veći, ali se neće promijeniti njegov smjer. Također, budući da će i težina w_0 biti α -puta veća, neće se promijeniti niti udaljenost hiperravnine od ishodišta niti udaljenost primjera od hiperravnine, budući da se obje te udaljenosti normiraju sa $\|\mathbf{w}\|$. Ova nam spoznaja sada nije od posebne koristi, ali će to biti za dva tjedna, kada ćemo pričati o modelu SVM.

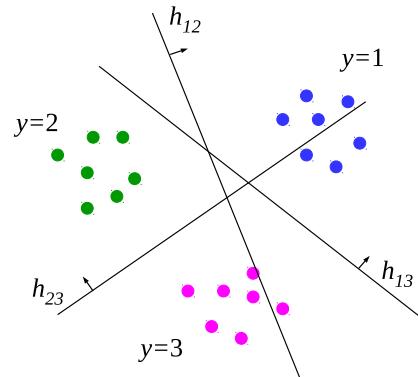
3 Višeklasna klasifikacija

Do sada smo se bavili binarnom klasifikacijom, tj. klasifikacijom u dvije klase. No, nisu svi klasifikacijski problemi s kojima ćemo se susretati binarni. Ponekad nam treba klasifikacija u više od dvije klase. Npr., klasifikacija poruka s Twittera u pozitivne, negativne ili neutralne. Ili klasifikacija novinskih tekstova u rubrike: sport, kultura, crna kronika, politika, itd.

Problem s višeklasnom klasifikacijom jest što su mnogi linearni diskriminativni modeli inherentno **binarni klasifikatori**. Onda se postavlja pitanje kako možemo iskoristiti takve modele kada imamo $K > 2$ klase? Voljeli bismo ne mijenjati model, nego radije promijeniti problem. Prisjetite se da smo sličan trik već bili primjenili, kada smo primjere preslikavali u prostor značajki, kako bismo mogli primjeniti linearan model ali imati nelinearnu granicu. Sada ćemo napraviti nešto slično: rastaviti ćemo višeklasni klasifikacijski problem na skup binarnih klasifikacijskih problema, i onda jednostavno više puta primjeniti binarni klasifikator. Konkretno, imamo dvije mogućnosti kako to napraviti: shema jedan-naspram-jedan i shema jedan-naspram-ostali. Pogledajmo ih redom.

Shema **jedan-naspram-jedan** (engl. *one-vs-one*, **OVO**) višeklasni problem svodi na $\binom{K}{2}$ nezavisnih binarnih klasifikacijskih problema, po jedan za svaki par klasa. Treniramo model h_{ij} tako da nauči razdvojiti primjere iz klase $y = i$ od primjera iz klase $y = j$, zanemarujući pritom primjere iz svih drugih klasa.

Na primjer, za $K = 3$ klase u dvodimenzionskom ulaznom prostoru to bi izgledalo ovako:



Budući da imamo tri klase, u shemi OVO treba nam $\binom{3}{2} = 3$ binarnih klasifikatora. Hipoteze koje odgovaraju tim trima klasifikatorima označene su na slici kao h_{12} , h_{23} , i h_{13} , gdje h_{ij} razdvaja primjere klase s oznakom $y = i$ od primjera klase s oznakom $y = j$. Na slici strelice pokazuju u smjeru pozitivne orientacije hiperravnina, tj. u smjeru gdje h_{ij} kao pozitivne klasificira primjere iz klase $y = i$. Na slici vidimo da svaka od triju hipoteza uvijek razdvaja samo dvije klase, dok treću klasu potpuno ignorira. Tako, na primjer, ravnina koja odgovara hipotezi $h_{23}(\mathbf{x}) = 0$ razdvaja klase $y = 2$ i $y = 3$, s orientacijom prema primjerima iz klase $y = 2$, dok međutim prolazi kroz primere iz klase $y = 1$, koje zanemaruje. To ostvarujemo tako da svaki od ovih binarnih klasifikatora h_{ij} treniramo na podskupu označenih primjera \mathcal{D} koji sadrži samo one primjere koji su iz klase $y = i$ i $y = j$.

Odluku o tome u koju klasu svrstati primjer sada dobivamo većinskim glasanjem. Drugim riječima, **višeklasni model u shemi OVO** definiramo ovako:

$$h(\mathbf{x}) = \operatorname{argmax}_i \sum_{i \neq j} \operatorname{sgn}(h_{ij}(\mathbf{x}))$$

pri čemu

$$h_{ij}(\mathbf{x}) = -h_{ji}(\mathbf{x})$$

Uvjerite se da ovakva definicija doista odgovara većinskoj odluci binarnih klasifikatora. Pogledajmo i jedan primjer.

► PRIMJER

Na prethodnoj slici, razmotrimo klasifikaciju nekog primjera \mathbf{x} iz klase $y = 2$. Za taj primjer imamo:

$$\begin{aligned} h_{12}(\mathbf{x}) &= -1 \Rightarrow h_{21}(\mathbf{x}) = 1 \\ h_{13}(\mathbf{x}) &= -1 \Rightarrow h_{31}(\mathbf{x}) = 1 \\ h_{23}(\mathbf{x}) &= 1 \Rightarrow h_{32}(\mathbf{x}) = -1 \end{aligned}$$

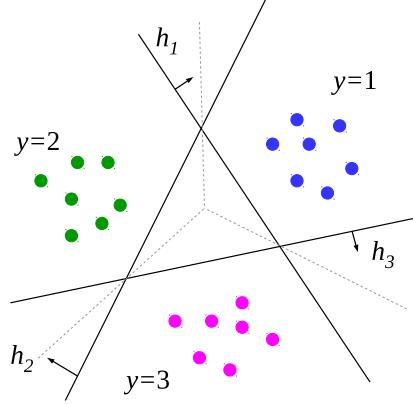
Glasovi hipoteza za pojedinačne klase su:

$$\begin{aligned} (i = 1) : \operatorname{sgn}(h_{12}(\mathbf{x})) + \operatorname{sgn}(h_{13}(\mathbf{x})) &= (-1) + (-1) = -2 \\ (i = 2) : \operatorname{sgn}(h_{21}(\mathbf{x})) + \operatorname{sgn}(h_{23}(\mathbf{x})) &= 1 + 1 = 2 \\ (i = 3) : \operatorname{sgn}(h_{31}(\mathbf{x})) + \operatorname{sgn}(h_{32}(\mathbf{x})) &= 1 + -1 = 0 \end{aligned}$$

pa je većinski izglasana klasa ona s oznakom $y = 2$, što je u skladu sa stvarnom oznakom primjera \mathbf{x} .

Shema OVO je jednostavna, no primijetimo da kod nje mogu postojati situacije kada odluka o klasifikaciji nije jednoznačna. To će biti slučaj sa svim onim primjerima koji padnu u dio ulaznog prostora (odnosno prostora značajki, ako koristimo preslikavanje) gdje je broj glasova za pobjedničku klasu izjednačen. Na našoj slici to bi bilo trokutasto područje u sredini ulaznog prostora. Međutim, u praksi se razmjerno rijetko događa da baš imamo popuno izjednačenje. A ako se i dogodi, situaciju možemo razriješiti tako da u obzir uzmemos pouzdanosti klasifikacije (na temelju predznačene udaljenosti primjera od hiperravnine, kako smo objasnili gore).

Druga shema za višeklasnu klasifikaciju pomoću dekompozicije na binarne klasifikacije jest shema **jedan-naspram-ostali** (engl. *one-vs-rest*, **OVR**) (koja se ponekad neispravno naziva *one-vs-all*). Kod te sheme imamo K nezavisnih binarnih klasifikatora, po jedan za svaku klasu. Svaki klasifikator h_i trenira se tako da razdjeljuje primjere klase $y = i$ od primjera svih ostalih klasa $y \neq i$, tj. klasifikator je naučen da raspoznaže radi li se o primjeru iz klase $y = i$ ili ne. Prethodni primjer u shemi OVR izgledao bi ovako:



Budući da imamo $K = 3$ klase, u shemi OVR imat ćemo 3 binarna klasifikatora. Hipoteze koje odgovaraju tim trima klasifikatorima su h_1 , h_2 i h_3 , gdje hipoteza h_i razdvaja primjere klase s oznakom $y = i$ od primjera svih drugih klasa.

Višeklasni model u shemi OVR odluku donosi na temelju pouzdanosti pojedinačnih klasifikatora tako da primjer svrstava u onu klasu za koju je klasifikacija najpouzdanija. To možemo vrlo jednostavno definirati ovako:

$$h(\mathbf{x}) = \operatorname{argmax}_j h_j(\mathbf{x})$$

dakle primjer se klasificira u klasu onog binarnog klasifikatora koji je najpouzdaniji u svoju pozitivnu klasifikaciju. Ovako definiran model zapravo implicitno definira granicu između dviju susjednih klasa $y = i$ i $y = j$ na mjestu gdje vrijedi $h_i(\mathbf{x}) = h_j(\mathbf{x})$. Na našoj slici takve granice odgovoraju simetralama kuteva (koje su prikazane kao iscrtkane linije).

Primijetite da je kod OVR sheme, za razliku od OVO sheme, mnogo manje područja gdje klasifikacijska odluka nije jednoznačna (može se, doduše, dogoditi da primjer sleti baš na pravac gdje je izlaz dvaju ili više najpouzdanijih klasifikatora izjednačen, ali to onda samo znači da smo urećeni i da bi bilo bolje da se klonimo strojnog učenja i svih koji ga prakticiraju).

Usporedimo sada ove dvije sheme. Očita prednost OVR nad OVO jest da imamo manje modela koje trebamo trenirati: K naspram $\binom{K}{2}$, što je linearna naspram kvadratna ovisnost o broju klasa. U ovom našem primjeru imali smo samo tri klase, pa razlika nije došla do izražaja. No, razlika je značajna ako imamo mnogo klasa i vrlo mnogo puno primjera, jer će onda treniranje mnogo modela vrlo dugo trajati, a i trajanje predikcije bi moglo biti nezanemarivo. Na primjer, ako imamo deset klasa (npr., za problem raspoznavanje znamenki), u shemi OVO trebat će nam 45 klasifikatora, a u shemi OVR samo njih deset.

S druge strane, problem sa shemom OVR jest taj da lako rezultira **neuravnoteženim** brojem primjera između parova klasa za koje treniramo model. Zašto? Zato što ćemo u pravilu za svaki model h_j imati puno više negativnih primjera (svi oni koji ne pripadaju klasi j) od pozitivnih primjera. To je već vidljivo u našem primjeru. Svaka od triju klasa ima po sedam primjera (v. prethodnu sliku). Ako treniramo binarni klasifikator da razdvaja primjere jedne klase od primjera ostalih dviju, onda će svaki takav klasifikator biti treniran sa 7 pozitivnih primjera i 14 negativnih. Situacija postaje to lošija što je veći broj klasa. Za deset klasa, odnos pozitivnih i negativnih primjera bit će 1:9. Općenito, ako su u skupu \mathcal{D} udjeli K klasa uravnoteženi, tj. ako imamo jednak broj primjera za svaku klasu, onda pozitivnih naspram negativnih primjera za pojedinačne binarne klasifikacijske probleme u shemi OVR bit će $1 : (K - 1)$. (Ako udjeli početno nisu uravnoteženi, onda će omjer za neke binarne probleme biti povoljniji, ali će za neke druge druge biti još lošiji.) Ovo je problematično jer se linearni diskriminativni modeli teško nose sa situacijom kada primjera iz neke klase ima puno više od primjera iz druge klase. Ono što se u takvim situacijama tipično događa jest da optimizacijski algoritam, u legitimnom nastajanju da smanji empirijsku pogrešku, to jednostavno napravi nauštrb primjera iz manje

klase, tako da ih sve klasificira u većinsku klasu. Riječ je o **problemu neuravnoteženosti klasa** (engl. *class imbalance problem*), koji je u strojnom učenju vrlo dobro poznat i dobro proučen. Problem je vrlo čest u praksi i predložena su neka rješenja, no mi se njima nećemo baviti.

5

Dakle, odabir između višeklasnih shema OVO i OVR svodi se u praksi na kompromis između broja klasifikatora s jedne i neuravnoteženosti klase s druge strane. Ako klasa nema baš jako puno i ako vremenska složenost nije prevelik problem, OVO je vjerojatno bolja opcija.

4 Klasifikacija regresijom

Do sada još nismo uveli niti jedan klasifikacijski algoritam, pa je vrijeme da to napokon učinimo. Zapravo, nećemo uvesti nov algoritam, nego ćemo pokušati iskoristiti ono što već znamo. Naime, prošli tjedan radili smo **linearan model regresije**. Razumno pitanje je možemo li taj algoritam nekako upotrijebiti za klasifikaciju. Pokazat će se da ne možemo, ali idemo ipak pokušati, jer ćemo iz toga nešto važno naučiti.

4.1 Naivan pristup

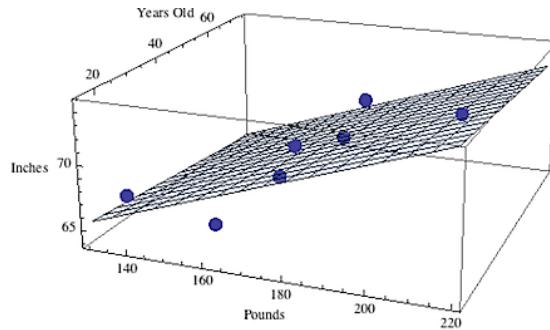
Prisjetimo se, funkcija pogreške (empirijsko očekivanje kvadratnog gubitka) linearog modela regresije jest:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{2} \sum_{i=1}^N (\mathbf{w}^T \phi(\mathbf{x}^{(i)}) - y^{(i)})^2 = \frac{1}{2} (\Phi \mathbf{w} - \mathbf{y})^T (\Phi \mathbf{w} - \mathbf{y})$$

Minimizator pogreške je:

$$\mathbf{w}^* = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y} = \Phi^+ \mathbf{y}$$

Kao primjer, razmotrimo regresiju s dvije značajke ($n = 2$). Dakle ulazni prostor je dvodimenzijski i regresijska funkcija je ravnina:



Ova ravnina svakom primjeru $(x_1, x_2) \in \mathbb{R}^2$ pridjeljuje jedan broj, od minus beskonačno do plus beskonačno. Pitanje je: kako bismo to mogli iskoristiti za klasifikaciju? Pa, najjednostavnija stvar koju možemo napraviti jest da klasifikaciju tretiramo kao regresiju: da primjerima iz dviju klasa dodijelimo brojčane oznaće iz skupa $\{0, 1\}$ i da na takvom skupu treniramo regresijski model. Drugim riječima, želimo naučiti regresijski model $h(\mathbf{x})$ koji će za primjere iz klase $y = 1$ davati $h(\mathbf{x}) = 1$, a za primjere iz klase $y = 0$ će davati $h(\mathbf{x}) = 0$. Kod predikcije, onda jednostavno gledamo je li $h(\mathbf{x})$ veći ili manji od 0.5. U prvom slučaju primjer klasificiramo u pozitivnu klasu, inače u negativnu. To jest:

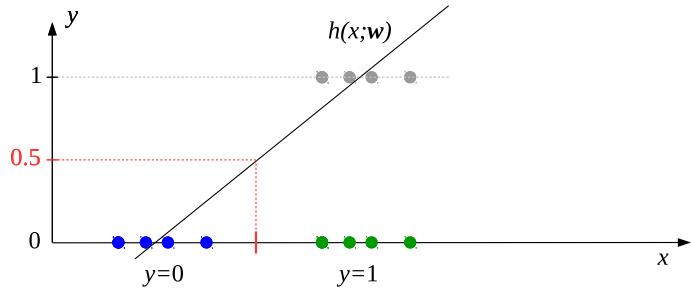
$$h(\mathbf{x}; \mathbf{w}) = \mathbf{1}\{\mathbf{w}^T \mathbf{x} \geq 0.5\}$$

Dakle, granicu između klase $y = 1$ i $y = 0$ čini pravac definiran jednadžbom $h(\mathbf{x}) = 0.5$. Alternativno, model možemo trenirati tako da za primjere iz negativne klase ciljna vrijednost bude $y = -1$. Granica između klase onda je definirana s $h(\mathbf{x}) = 0$.

Pogledajmo jedan primjer.

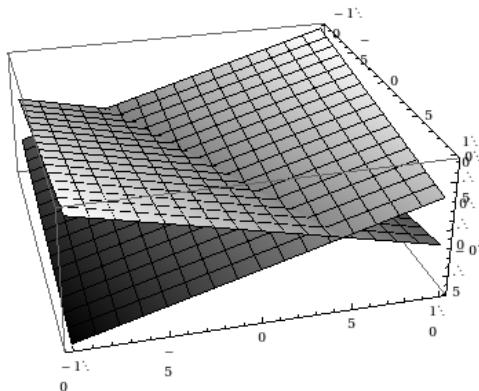
► PRIMJER

Najjednostavniji primjer je za jednodimenzionalni ulazni prostor ($n = 1$). To nije realan primjer, ali poslužit će da objasnimo ideju. Recimo da raspoložemo sa 4 primjera pozitivne klase ($y = 1$) i 4 primjera negativne klase ($y = 0$), i da se oni u jednodimenzionalnom ulaznom prostoru (dakle na osi x) mogu razdvojiti u te dvije klase. Treniramo regresijski model tako da za primjere iz klase $y = 1$ postavljamo da je ciljna vrijednost $y = 1$, a primjerima iz klase $y = 0$ da je ciljna vrijednost $y = 0$. To onda izgleda ovako:

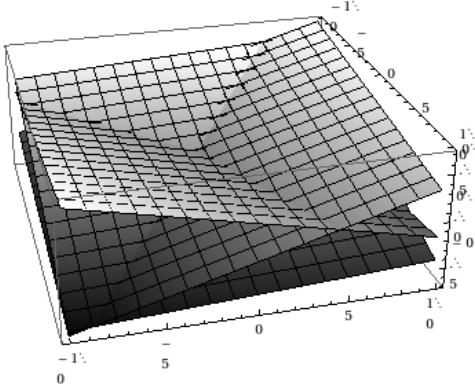


Dobivamo regresijski pravac $h(x; \mathbf{w})$ koji, kao i uvijek, minimizira sumu kvadratnog odstupanja predviđenih vrijednosti od ciljnih vrijednosti. Točka na osi x za koju je $h(x; \mathbf{w}) = 0.5$ predstavlja granicu između klasa u jednodimenzionalnom ulaznom prostoru. Za primjere lijevo od te točke vrijedi $h(x; \mathbf{w}) < 0.5$, pa ih klasificiramo kao negativne, a za primjere desno od te točke vrijedi $h(x; \mathbf{w}) \geq 0.5$, pa ih klasificiramo kao pozitivne. Ovo zasada izgleda kao da bi moglo raditi, ali vidjet ćemo da ipak postoji problem. (Možda ga već vidite?)

Uvjek, umjesto jednog modela za dvije klase, mogli smo trenirati dva modela, po jedan za svaku klasu, pa bi granica između klasa bila ondje gdje vrijedi $h_i(\mathbf{x}) = h_j(\mathbf{x})$. Npr., u dvodimenzionalnom ulaznom prostoru:



Ako pak želimo klasificirati u više od dvije klase, možemo npr. primijeniti shemu OVR, pa trenirati po jedan model za svaku od K klasa. Granica između susjednih klasa bit će tamo gdje $h_i(\mathbf{x}) = h_j(\mathbf{x})$, gdje su h_i i h_j dvije hipoteze s najvećim vrijednostima za \mathbf{x} . Na primjera, za $n = 2$ i $K = 3$:



4.2 Problemi

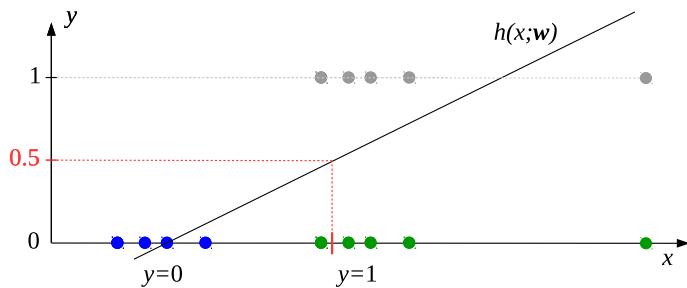
Ovo na prvi pogled izgleda izvrsno! Čini se da smo uspjeli riješiti problem klasifikacije pomoću linearog modela regresije. Prednosti tog pristupa jesu da je on vrlo jednostavan i postoji rješenje u zatvorenoj formi. Međutim, izgled vara. Postoje nedostatci klasifikacije pomoću linearog modela regresije, od kojih su neki poprilično problematični. Konkretno, ti nedostatci su:

1. Izlazi modela nemaju vjerojatnosnu interpretaciju, budući da domena hipoteze $h(\mathbf{x})$ nije ograničena na interval $[0, 1]$;
2. Model je nerobusan (neotporan), u smislu da je vrlo osjetljiv na vrijednosti koje odskaču. Konkretno, događa se to da algoritam kažnjava “pretočno” klasificirane primjere te zbog toga, u nekim slučajevima, pogrešno klasificira primjere čak i kada su oni linearno odvojivi.

Pogledajmo detaljnije što mislimo kad kažemo da je model nerobusan.

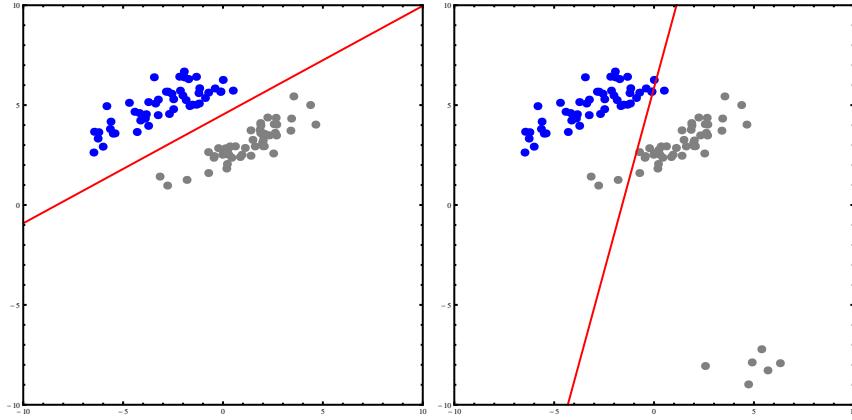
► PRIMJER

Razmotrimo istu situaciju kao u prethodnom primjeru, dakle klasifikaciju u jednodimenziskom ulaznom prostoru, ali dodajmo jedan pozitivan primjer ($y = 1$). Neka je taj primjer “duboko” u području pozitivnih primjera, dakle vrlo udesno na osi x . Ovako:



Kakav će to imati efekt na regresijski pravac? Budući da algoritam regresije minimizira kvadratno odstupanje, dodavanje novog primjera imat će efekt da će se cijeli pravac približiti tom novom primjeru, jer bi u protivnom kvadrat reziduala za taj primjer bio vrlo velik. Pravac $h(x; \mathbf{w})$ prikazan na slici bi otprilike bio pravac koji minimizira kvadratno odstupanje (ne baš, bio bi malo većeg nagiba, čini se, ali nema veze, razumijete poantu). S obzirom da je pravac dodavanjem novog primjera promijenio nagib, to se promijenila i vrijednost za koju $h(x; \mathbf{w}) = 0.5$, koja određuje granicu između klasa. Vidimo da se granica između klasa pomakla udesno, i sada je jedan pozitivan primjer pogrešno klasificiran (lažno negativan). Dogodilo se, dakle, da je dodavanje jednog primjera, koji je, primijetite, već bio na ispravnoj strani granice, dovelo do primicanja granice tom primjeru i pogrešne klasifikacije nekog drugog primjera koji je bio blizu granice.

Isti se problem događa i kada je ulazni prostor više dimenzije. Npr., za $n = 2$:



Na lijevoj slici prikazan je dvodimenzionalni ulazni prostor s primjerima iz dviju klasa (plave i sive točke) te granica između njih dobivena kao $h(\mathbf{x}; \mathbf{w}) = 0.5$ (crvena linija). Ta je granica dobivena tako da smo učili regresijski model koji primjerima iz jedne klase dodjeljuje vrijednost 1 (npr. plavi primjeri) a primjerima iz druge klase vrijednost 0 (npr. sivi primjeri). Hipoteza odgovara ravnini u trodimenzionskom prostoru $\mathbb{R}^2 \times \mathbb{R}$, a mjesto gdje ta ravnina sijeće ravninu $x_1 \times x_2$ jest upravo pravac opisan jednadžbom $h(\mathbf{x}; \mathbf{w}) = 0.5$. Na desnoj slici vidimo što se događa kada skupu za učenje dodamo primjere koji su daleko od granice, a to su sivo obojani primjeri dolje desno. Budući da je model sada treniran tako da i tim primjerima treba dodijeliti vrijednost 0, optimizacijski postupak najmanjih kvadrata nalazi novu ravninu koja minimizira empirijsku pogrešku. Kao što se vidi na slici, granica koju dobivamo kao $h(\mathbf{x}; \mathbf{w} = 0.5)$ sada je značajno primaknuta novododanim primjerima, te model pogrešno klasificira neke primjere iz pozitivne i neke primjere iz negativne klase. Slično kao i u ranijem primjeru, premda je problem i dalje linearno odvojiv, primjeri koji se nalaze daleko od granice imaju velik utjecaj na položaj granice i smanjuju točnost klasifikatora.

Problem koji smo upravo identificirali – da klasifikacija ne radi dobro ako su neki primjeri daleko od granice – zapravo nam ukazuje na to da linearan model regresije nije dobar klasifikator. Ako su primjeri linearno odvojivi u ulaznom prostoru, očekujemo da će nam algoritam strojnog učenja dati hipotezu koja savršeno točno klasificira sve primjere. Međutim, kod linearne regresije to očito nije slučaj. To je velik problem. Ako želimo raditi klasifikaciju, to moramo nekako riješiti.

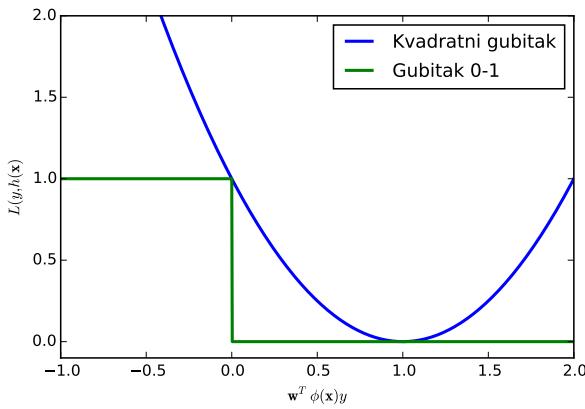
4.3 Tko je kriv?

U čemu je zapravo problem s klasifikacijom pomoću regresije? Prisjetimo se da se svaki algoritam strojnog učenja sastoji od tri komponente: modela, funkcije gubitka (iz koje izvodimo funkciju empirijske pogreške) i optimizacijskog algoritma. Ako postoji problem, on leži u nekoj od ovih triju komponenti. Kojoj? Optimizacijski algoritam definitivno je oslobođen krivnje, jer taj doista samo radi svoj posao na temelju modela i funkcije gubitka koje mu zadamo. Problem je, dakle, u modelu ili funkciji gubitka. Zapravo, možemo reći da je problem *ili* u modelu *ili* u funkciji gubitka. Naime, problem je u modelu, jer za primjere koji su daleko od granice daje vrlo visoke izlazne vrijednosti, koje onda daju veliko kvadratno odstupanje. S druge strane, problem je u funkciji gubitka definiranoj kao kvadratno odstupanje, je ona kažnjava i dobro klasificirane primjere.

Mi ćemo se u nastavku koncentrirati na funkciju gubitka kao krivca za ovaj problem. Cilj nam je da pokažemo zašto funkcija gubitka koju koristimo u algoritmu regresije nije dobra za klasifikaciju. Prisjetimo se, regresija ima **funkciju kvadratnog gubitka**:

$$L(y, h(\mathbf{x})) = (y - \mathbf{w}^T \phi(\mathbf{x}))^2$$

Funkcija gubitka L je tipa $L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_0^+$, tj. to je funkcija dvije varijable. Bit će nam lakše analizirati ju i grafički prikazati ako je nekako svedemo na funkciju jedne varijable. Srećom, to možemo lako napraviti. Najprije, prebacimo se iz skupa oznaka $y \in \{0, 1\}$ u skup oznaka $y \in \{-1, +1\}$. Sada primijetimo da, ako je klasifikacija nekog primjera (\mathbf{x}, y) ispravna, onda će predznak skalarnog umnoška $\mathbf{w}^T \mathbf{x}$ biti jednak predznaku oznake y , i to bilo da je primjer pozitivan ili negativan. To znači da će umnožak $\mathbf{w}^T \phi(\mathbf{x}) \cdot y$ uvijek biti pozitivan za točnu klasifikaciju, a negativan za netočnu klasifikaciju (slučaj $\mathbf{w}^T \phi(\mathbf{x})y = 0$ možemo uključiti u točnu klasifikaciju, ako je $h(\mathbf{x}) = \mathbf{1}\{\mathbf{w}^T \mathbf{x} \geq 0\}$). Ideja je onda da funkciju kvadratnog gubitka L skiciramo kao funkciju jednog argumenta, $\mathbf{w}^T \phi(\mathbf{x})y$. Evo kako bi izgledao takav graf za **gubitak 0-1** i za **kvadratni gubitak**:



Ovaj grafikon je važan, pa ćemo ga detaljno objasniti. Na x -osi grafa je vrijednost $\mathbf{w}^T \phi(\mathbf{x})y$, koja će, kao što smo već ustanovili, biti pozitivna za točnu klasifikaciju i negativna za netočnu klasifikaciju. Tu vrijednost možemo shvatiti kao "mjeru točnosti klasifikacije". Dakle, pozitivan dio x -osi odgovara slučajevima kada je klasifikacija točna, a negativan dio x -osi slučajevima kada je klasifikacija netočna, a što smo dalje od ishodišta udesno ili ulijevo to je klasifikacija više točna odnosno netočna. Na y -osi je vrijednost funkcije gubitka L . Zelenom je prikazana funkcija gubitka 0-1. Kao što je vidljivo iz grafa, ta funkcija ima vrijednost 1 ako je klasifikacija netočna ($\mathbf{w}^T \phi(\mathbf{x})y < 0$), a vrijednost 0 ako je klasifikacija točna ($\mathbf{w}^T \phi(\mathbf{x})y \geq 0$). Pogledajmo sada kako izgleda graf funkcije kvadratnog gubitka. Gubitak će biti jednak nuli kada je vrijednost predikcije jednaka oznaci y , a to će biti kada $\mathbf{w}^T \phi(\mathbf{x})y = 1$. Kada je predikcija jednaka nuli, onda je gubitak jednak 1. Također, kada je predikcija jednak 2 ili -2 , onda će gubitak isto biti 1. Dakle, imamo parabolu, koja je u grafikonu ucrtana plavom bojom.

Razmotrimo što smo dobili. Vidimo da je kvadratni gubitak jednak nuli samo za primjere za koje je $\mathbf{w}^T \phi(\mathbf{x})y = 1$, tj. samo za one primjere za koje je izlaz hipoteze jednak točno 1 (za pozitivne primjere) odnosno točno -1 (za negativne primjere). U svim ostalim slučajima kvadratni gubitak je veći od nule. Pogrešno klasificirani primjeri (negativna strana x -osi) nanose gubitak koji raste kvadratno s udaljenosću primjera od granice. Međutim, vidimo da kvadratni gubitak **kažnjava i točno klasificirane primjere** (pozitivna strana x -osi), i to pogotovo primjere koji su vrlo, vrlo točno klasificirani: primjeri koji su duboko u području ispravne klase području bit će jako kažnjeni. To je i uzrok nerobustnosti rješenja: ako postoji makar jedan primjer koji je duboko u ispravnom području (s koje god strane), gubitak će biti ogroman, i optimizacijski će ga algoritam nastojati smanjiti, pomicanjem hiperravnine tako da se taj gubitak smanji.

Očito, ako ovo želimo popraviti – a želimo – onda trebamo drugačije definirati funkciju gubitka. Postavlja se pitanje: kakvu funkciju gubitka bismo mi zapravo željeli? Idealno, željeli bismo kažnavati samo netočnu klasifikaciju, i to svaku netočnu klasifikaciju jednako, neovisno o tome koliko je netočna. Drugim riječima, mi želimo **gubitak 0-1** (označen zelenom na prethodnoj skici).

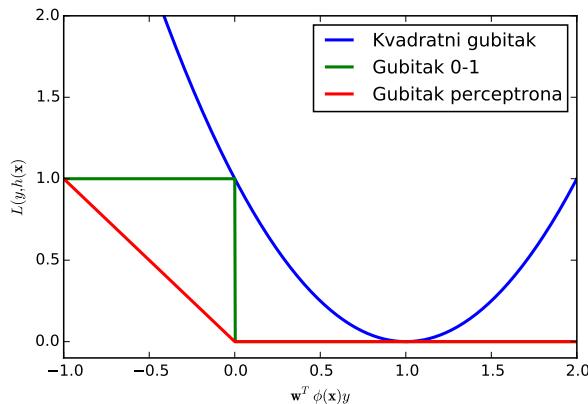
Naravno, problem je što ne možemo samo promijeniti funkciju gubitka i opet raditi linearnu

6

regresiju, jer je to onda više ne bi bila regresija. Naime, funkcija kvadratnog gubitka jedna je od komponenti algoritma linearne regresije. Ako bismo je zamijenili nekom drugom funkcijom gubitka, onda bismo morali promijeniti i optimizacijski postupak (jer postupak najmanjih kvadrata funkcionira samo s kvadratnim gubitkom), i to više ne bi bio algoritam linearne regresije. Dakle, da bismo radili klasifikaciju, trebat će nam ipak neki skroz drugi algoritam. Ne možemo samo tako upotrijebiti linearnu regresiju za klasifikaciju!

Jedna ideja koja odmah pada na pamet jest da za klasifikaciju koristimo algoritam koji kao funkciju gubitka ima gubitak 0-1. To je dobra ideja, ali nažalost nije ostvariva. Naime, funkciju pogreške definiranu kao očekivanje funkcije gubitka 0-1 ne bismo mogli koristiti za optimizaciju. Zašto? Iz dva razloga. Prvo, ta funkcija nije derivabilna, pa ne možemo dobiti rješenje u zatvorenoj formi. Drugo, ako pokušamo optimirati na neki drugi način, npr. gradijentnim spustom, problem s ovom funkcijom jest da je uglavnom (osim za vrijednost 0) konstantna, pa nema nagiba po kojem bismo se spuštali.

No, evo onda alternativne ideje: umjesto da koristimo gubitak 0-1, zašto ne bismo iskoristili neku funkciju koja joj je što sličnija, ali je derivabilna i ima nagib? Npr., možemo definirati funkciju gubitka tako da je gubitak jednak nuli za točno klasificirane primjere, ali da je gubitak za netočno klasificirane primjere proporcionalan tome koliko je primjer netočan. Na našem grafikonu funkcija gubitaka, takva bi funkcija gubitka izgleda ovako (crvena linija):



Dakle, funkcija gubitka na pozitivnoj strani x -osi je nula, a na negativnoj strani x -osi linearno raste. Taj dio funkcije, kada su primjeri netočno klasificirani, ima derivaciju koja nije nula, što znači da bismo mogli provesti optimizaciju, npr. iterativnom metodom, konkretno npr. **gradijentnim spustom**.

Ispostavlja se već postoji algoritam strojnog učenja koji ima upravo ovakav gubitak. Radi se o algoritmu **perceptronu**, s kojim su mnogi od vas vjerojatno već upoznati. Pogledajmo taj algoritam malo detaljnije, s posebnim naglaskom na funkciju gubitka.

5 Perceptron

Algoritam perceptrona jedan je od najranijih algoritama strojnog učenja i prva uzdanica konekcionističkog (neuronskog) pristupa umjetnoj inteligenciji. U osnovi, riječ o algoritmu za binarnu klasifikaciju. Krenimo od njegovog modela. Osnovna razlika u odnosu na linearan model regresije jest u tome što su kod perceptrona izlazi modela ograničeni na vrijednosti -1 i $+1$. To je ostvareno tako da je skalarni produkt vektora težina i vektora značajki omotan u odgovarajuće definiranu funkciju:

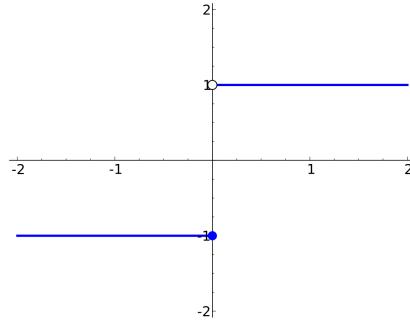
$$h(\mathbf{x}; \mathbf{w}) = f(\mathbf{w}^T \phi(\mathbf{x}))$$

7

8

gdje je f definirana kao **funkcija praga** (step-funkcija):

$$f(\alpha) = \begin{cases} +1 & \text{ako } \alpha \geq 0 \\ -1 & \text{inače} \end{cases}$$



pa je granica između klase $y = 1$ i $y = -1$ pravac definiran sa $h(\mathbf{x}; \mathbf{w}) = 0$. Funkciju f u kontekstu perceptronova (i, općenitije, neuronskih mreža) nazivamo **aktivacijska funkcija** (engl. *activation function*) ili **prijenosna funkcija** (engl. *transfer function*).

To je, dakle, perceptronov model. Pogledajmo sada njegovu funkciju gubitka. Funkciju gubitka perceptronova već smo bili ucrtali u grafikon funkciju gubitaka (v. gore, crvena linija). Tu funkciju možemo definirati ovako:

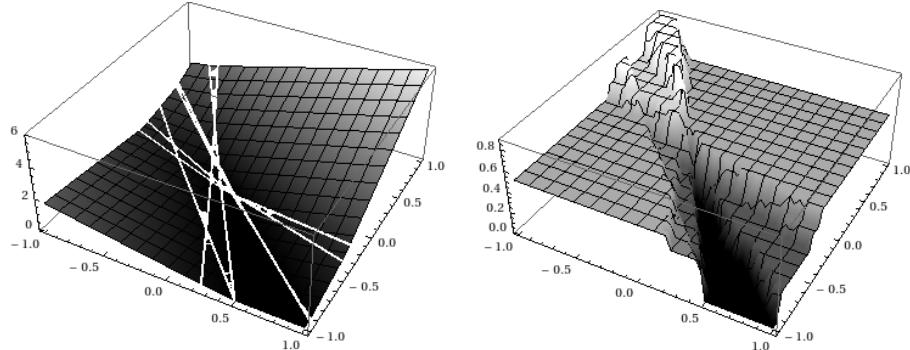
$$\max(0, -\mathbf{w}^T \phi(\mathbf{x})y)$$

Primijetite da koristimo izraz $\max(0, \cdot)$ kako bismo vrijednost funkcije gubitka odozdo ograničili na nulu, budući da ne želimo da gubitak bude negativan (negativan gubitak značio bi nagrada za hipotezu, a mi u nadziranom strojnom učenju nikoga ne želimo nagrađivati). Funkcija pogreške je očekivanje funkcije gubitka, odnosno funkcija empirijske pogreške je prosjek funkcije gubitka na skupu za učenje, no ovdje ćemo zanemariti član $1/N$ (Zašto? Zato jer možemo!), pa je to onda:

$$E(\mathbf{w}|\mathcal{D}) = \sum_{i=1}^N \max(0, -\mathbf{w}^T \phi(\mathbf{x}^{(i)})y^{(i)}) = - \sum_{i: f(\mathbf{w}^T \phi(\mathbf{x}^{(i)})) \neq y^{(i)}} \mathbf{w}^T \phi(\mathbf{x}^{(i)})y^{(i)}$$

Prvi izraz za pogrešku je jednostavno zbroj gubitka po svim primjerima iz \mathcal{D} . Drugi izraz za pogrešku je alternativan način da definiramo isto, ali bez funkcije max: suma samo po netočno klasificiranim primjerima. U svakom slučaju, kažnjavamo samo netočno klasificirane primjere, i to proporcionalo tome koliko su oni pogrešno klasificirani.

Sada bi bilo zanimljivo da pogledamo kako izgleda funkcija pogreške $E(\mathbf{w}|\mathcal{D})$ u prostoru parametara \mathbf{w} (tj. u prostoru težina). Za dvodimenzionalni prostor parametara ($w_1 \times w_2$), funkcija pogreške izgleda ovako (lijeva slika):



Vidimo da je funkcija po dijelovima linearna te da je konveksna. Funkcija mora biti konveksna jer je definirana kao zbroj konveksnih funkcija gubitaka. Na ovome grafu, svakoj točki (w_1, w_2) odgovara jedna konkretna hipoteza, tj. jedan pravac definiran parametrima \mathbf{w} kao $h(\mathbf{x}; \mathbf{w}) = 0$. Ta hipoteza akumulira određenu pogrešku, koju izračunavamo tako da pozbrajamo funkciju gubitka za svaki primjer iz \mathcal{D} . Promjenom parametara (w_1, w_2) dobivamo različite pravce. Kada pravac pređe preko nekog primjera iz skupa \mathcal{D} , klasifikacija tog primjera se mijenja iz $+1$ u -1 , ili obrnuto, pa se u toj točki diskretno mijenja vrijednost funkcije pogreške. Zbog toga je površina funkcije pogreške po dijelovima linearna. Na desnoj slici prikazana je funkcija pogreške koju bismo dobili s gubitkom 0-1, što odgovara udjelu pogrešno klasificiranih primjera. Vidimo da je problem s takvom funkcijom pogreške taj što je ona gotovo uvijek konstantna. Nagi pad ima samo u točkama gdje pravac prelazi preko nekog primjera, pa dolazi do promjene udio pogrešno klasificiranih primjera. Takvu funkciju ne možemo minimizirati niti analitički niti numerički. Suprotno tome, pogreška perceptronu (lijeva slika) uglavnom nije konstantna, tj. postoji nagib koji nas vodi prema minimumu. Također možemo vidjeti da je funkcija pogreške perceptronu zapravo aproksimacija udjela pogrešno klasificiranih primjera (tj. funkcija na lijevoj slici je aproksimacija funkcije na desnoj slici).

I, konačno, pogledajmo optimizaciju. Naša definicija funkcije pogreške je vrlo lijepa, no postoji cijena koju moramo platiti što koristimo ovu novu funkciju gubitka, a to je da ne postoji rješenje za minimum funkcije u zatvorenoj formi jer funkcija nije neprekidna. Alternativa je da u primjenimo **gradijentni spust**, i to tako da izračunamo derivaciju funkcije gubitka za svaki primjer koji je netočno klasificiran, i onda za svaki takav primjer promjenimo težine u smjeru suprotnom od porasta gubitka. To će nas postepeno dovesti do minimuma empirijske pogreške.

Gradijent gubitka za netočno klasificirane primjere je:

$$\nabla_{\mathbf{w}} (-\mathbf{w}^T \phi(\mathbf{x})y) = -\phi(\mathbf{x})y$$

Primijetite da nas zanima samo gradijent gubitka kada je primjer pogrešno klasificiran. Kada je primjer točno klasificiran, gradijent je nula. Tako smo izbjegli da računamo gradijent po dijelovima linearne funkcije $\max(0, -\mathbf{w}^T \phi(\mathbf{x})y)$.

Dobili smo vektor koji pokazuje u kojem smjeru će gubitak rasti (specifično za primjer \mathbf{x}). Ako želimo smanjiti gubitak, trebamo ići u suprotnome smjeru. To nam onda daje sljedeće **pravilo ažuriranja težina** (engl. *weight update rule*):

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \phi(\mathbf{x})y$$

gdje vrijednost η određuje tzv. **stopu učenja** (engl. *learning rate*), odnosno veličinu koraka kojim se spuštamo k minimumu. Ovo se pravilo naziva **Widrow-Hoffovo delta-pravilo**.

To sve sada možemo spojiti u vrlo jednostavan algoritam – algoritam perceptronu:

► Algoritam perceptronu

- 1: inicijaliziraj $\mathbf{w} \leftarrow (0, \dots, 0)$
- 2: **ponavljam** do konvergencije
- 3: za $i = 1, \dots, N$
- 4: ako $f(\mathbf{w}^T \phi(\mathbf{x}^{(i)})) \neq y^{(i)}$ onda $\mathbf{w} \leftarrow \mathbf{w} + \eta \phi(\mathbf{x}^{(i)})y^{(i)}$

Može se dokazati da, ako su primjeri linearno odvojivi, algoritam perceptronu nalazi rješenje (težine koje odgovaraju hipotezi koja savršeno klasificira sve primjere iz \mathcal{D}) u konačnom broju koraka. Međutim, ako primjeri nisu linearno odvojivi – a redovito nisu, unatoč funkciji preslikavanja $\phi(\mathbf{x})$ – onda algoritam perceptronu ne konvergira.

Sažmimo sada prednosti i nedostatke algoritma perceptronu. Prednosti je da je taj algoritam robustniji od modela linearne regresije, u smislu da točno klasificirani primjeri koji su daleko od

12

13

granice između klasa nemaju nikakvog utjecaja na tu granicu (ako je primjer točno klasificiran, gradijent gubitka za taj primjer je nula i težine \mathbf{w} za taj primjer se ne ažuriraju). Druga je prednost da je to očigledno jednostavan algoritam (optimizacijski postupak je jednostavniji od, npr., izračuna pseudoinverza matrice dizajna). No, perceptron ima i niz ozbiljnih nedostataka. Prvo, slično kao i kod regresije, izlazi modela nemaju vjerojatnosnu interpretaciju ($h(\mathbf{x}^{(i)})$) nije ograničena na interval $[0, 1]$). Drugi je problem da rezultat (hipoteza) ovisi o početnim težinama i redoslijedu kojim se provodi ažuriranje težina. Treći, i najveći problem jest što algoritam perceptrona ne konvergira ako primjeri nisu linearne odvojivi.

Prisjetimo se, glavni problem klasifikacije regresijom jest **nerobusnost**: “kažnjavanje” pretočno klasificiranih primjera. Perceptron nema taj problem. Međutim, perceptron ne konvergira ako primjeri nisu linearne odvojivi. Možemo li nekako kombinirati ova dva postupka, i dobiti najbolje od oba? Želimo i robusnost i konvergenciju. Možemo li k tome još dodati probabilistički izlaz? Pokazat će se da možemo. No, to ćemo ostaviti za idući put, kada ćemo govoriti o algoritmu **logističke regresije**.

Sažetak

- **Diskriminativni linearne modeli** daju linearnu granicu između klasa
- Granica između klasa je $(n - 1)$ -dimenzijska **hiperravnina** u n -dimenzijskom prostoru, definirana preko vektora težina (w_0, \mathbf{w})
- Svaki višeklasni klasifikacijski problem može se dekomponirati u skup binarnih klasifikacijskih problema shemom OVO ili OVR, koje imaju svoje prednosti i nedostatke
- **Klasifikacija regresijom** je jednostavna, ali nije robusna, pa se ne koristi
- **Perceptron** je linearan klasifikacijski model koji gradijentnim spustom minimizira funkciju pogreške koja aproksimira broj pogrešnih klasifikacija
- Perceptron **ne konvergira** za linearne nedvojive probleme, dok za linearne odvojive rješenje ovisi o inicijalizaciji i redoslijedu primjera
- Niti regresija niti perceptron ne daju probabilistički izlaz

Bilješke

- 1 Ako se prebacimo na zapis s “dummy” jedinicom, $x_0 = 1$, tj. ako težinu w_0 uključimo u vektor \mathbf{w} te model definirao kao $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$, onda je granica n -dimenzijska hiperravnina koja prolazi kroz ishodište u $(n + 1)$ -dimenzijskom proširenom ulaznom prostoru. Kako smo komentirali u bilješci za temu 4 (procitajte je ako još niste!), ova se razlika svodi na to je li model $h(\mathbf{x})$ definiran kao **afina funkcija** u \mathbb{R}^n (bez “dummy značajke”) ili kao **homogena funkcija** u \mathbb{R}^{n+1} (sa “dummy značajkom”). Oba pogleda su, naravno, valjana. Mi ćemo u matričnim izračunima tipično koristiti “dummy” značajku, ali kod skica i interpretacija tipično govorimo o prostoru dimenzije n (odnosno m , ako koristimo preslikavanje), a ne dimenzije $n + 1$ (odnosno $m + 1$).
- 2 Prisjetimo se: **vektor normale** na površinu u točki \mathbf{x} je vektor koji je okomit na tangencijalnu ravninu površine u točki \mathbf{x} . Kod pravca ili ravnine, normala je okomita na pravac odnosno ravninu. Isto vrijedi i u n -dimenzijskom prostoru: za $(n - 1)$ -dimenzijsku hiperravninu u \mathbb{R}^n normala je svaki vektor koji je okomit na sve vektore na hiperravnini. Koncept normale može se proširiti i na $(n - 1)$ -dimenzijske hiperpovršine u \mathbb{R}^n : normala hiperpovršine jednaka je vektoru gradijenta (i taj vektor ovisi o točki u kojoj računamo normalu). Normale hiperpovršina nam za sada neće trebati.
- 3 Pokažimo da je udaljenost $(n - 1)$ -dimenzijske hiperravnine definirane u \mathbb{R}^n sa $h(\mathbf{x}) = 0$ jednaka $-w_0/\|\mathbf{w}\|$. Neka je \mathbf{x} točka na hiperravnini. Za tu točku onda vrijedi $h(\mathbf{x}) = 0$, tj. vrijedi:

$$\mathbf{w}^T \mathbf{x} + w_0 = 0$$

Ako podijelimo obje strane jednadžbe s normom vektora \mathbf{w} i presložimo, dobivamo:

$$\frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|} = -\frac{w_0}{\|\mathbf{w}\|}.$$

Što smo dobili na lijevoj strani? Ljeva strana je **projekcija** vektora \mathbf{x} na vektor normale \mathbf{w} . Naime, prisjetimo se definicije skalarног produkta preko kuta između vektora:

$$\mathbf{w}^T \mathbf{x} = \|\mathbf{w}\| \|\mathbf{x}\| \cos \alpha$$

Ako podijelimo sa $\|\mathbf{w}\|$, dobivamo

$$\frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|} = \|\mathbf{x}\| \cos \alpha$$

gdje je $\|\mathbf{x}\| \cos \alpha$ upravo projekcija vektora \mathbf{x} na vektor \mathbf{w} . (Općenito, $\mathbf{a}^T \mathbf{b} / \|\mathbf{a}\|$ je projekcija vektora \mathbf{b} na vektor \mathbf{a} .) Iz glavne skice možemo vidjeti da je projekcija vektora \mathbf{x} na \mathbf{w} zapravo udaljenost hiperravnine od ishodišta. Dakle, zaključujemo da je udaljenost hiperravnine od ishodišta u prostoru \mathbb{R}^n jednaka $-w_0 / \|\mathbf{w}\|$ (u prostoru homogene funkcije \mathbb{R}^{n+1} udaljenost hiperravnine od ishodišta jednaka je nuli; vidi gornju bilješku). Primijetimo da je to **predznačena udaljenost**: može biti pozitivna ili negativna, ovisno o orijentaciji hiperravnine.

- 4 Da množenje vektora težina (w_0, \mathbf{w}) skalarom ne utječe na položaj hiperravnine definirane s $h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x} + w_0 = 0$ slijedi iz definicije homogenosti linearног preslikavanja. Naime, ako $h(\mathbf{x})$ definiramo kao homogenu funkciju (uključimo w_0 u vektor \mathbf{w}), onda $h(\mathbf{x}; \alpha \mathbf{w}) = h(\alpha \mathbf{x}; \mathbf{w})$, a po definiciji homogenosti $h(\alpha \mathbf{x}) = \alpha h(\mathbf{x})$ te, posljedično, $h(\alpha \mathbf{x}) = 0$ je isti skup točaka kao i $h(\mathbf{x}) = 0$.
- 5 **Problem neuravnoteženosti klase** nije specifičan samo za višeklasnu klasifikaciju, nego se može dogoditi i kod binarne klasifikacije. Na primjer, kod klasifikatora za dijagnostiku tumora većina će primjera (srećom) biti negativna. Kod klasifikatora za prepoznavanje sentimenta u pritužbama koje korisnici pišu telekom operatorima većina će primjera (nažalost) također biti negativna. Zapravo, u praksi se vrlo rijetko događa da imamo uravnoteženost klasa. O strategijama rješavanja problema neuravnoteženosti klase možete pročitati u ([Japkowicz, 2000](#)) i ([Japkowicz and Stephen, 2002](#)). Tipične strategije su uzorkovanje (naduzorkovanje i poduzorkovanje) ili primjena funkcije gubitka kod koje kazna za pogrešnu klasifikaciju ovisi o tome iz koje je klase primjer (tako da pogrešnu klasifikaciju primjera iz manjinskih klasa više kažnjavamo), što se naziva **klasifikacija osjetljiva na cijenu** (engl. *cost-sensitive classification*).
- 6 U nekim slučajevima ne želimo svaku netočnu klasifikaciju kažnjavati jednakom, već gubitak želimo definirati asimetrično (npr. drugačije kažnjavati lažno pozitivne slučajeve od lažno negativnih). Također, kako je objašnjeno u gornjoj bilješci, nekada gubitak želimo računati drugačije za različite klase. Međutim, i u takvim slučajevima gubitak se može definirati preko gubitka 0-1.
- 7 **Algoritam perceptron** osmislio je 1958. godine američki psiholog Frank Rosenblatt ([Rosenblatt, 1958](#)), kombiniravši matematički model umjetnog neurona, koji su 1943. godine predozili američki neurofiziolog Warren McCulloch i logičar Walter Pitts ([McCulloch and Pitts, 1943](#)), i teoriju o jačanju i slabljenju veza između neurona (sinaptičkoj plastičnosti) kao temelju za učenje, koju je 1949. godine predložio kanadski fiziolog Donald Hebb ([Hebb, 1949](#)). Algoritam je postavio temelj za razvoj neuronskih mreža te je bio zaslužan za buđenje velikog interesa i entuzijazma za konekcionistički pristup umjetnoj inteligenciji u šezdesetim godinama prošlog stoljeća, ali i za veliko razočaranje i stagnaciju koji su uslijedili u sedamdesetima, i to ponajviše zahvaljujući kritici u knjizi *Perceptrons* Marvin Minskog i Seymoura Paperta ([Minsky and Papert, 1969](#)). Kontroverza koja je u to vrijeme nastala oko perceptona te s time povezane debata o tome kako bi se područje umjetne inteligencije imalo razvijati lijepo su opisane ([Olazaran, 1996](#)). Kasnije modificirane varijante perceptronova našle su širu primjenu u strojnom učenju, npr. **perceptron s glasanjem** (engl. *voted perceptron*) ([Freund and Schapire, 1999](#)).
- 8 Ovako definiran model perceptronova čini se istim (do na razliku u pragovima i izlaznim vrijednostima) kao i naš (neuspјeo) model za klasifikaciju pomoću regresije, gdje smo model definirali kao $h(\mathbf{x}; \mathbf{w}) = 1\{\mathbf{w}^T \mathbf{x} \geq 0.5\}$. Doista, modeli su zapravo identični. Međutim, algoritmi će se razlikovati po funkciji

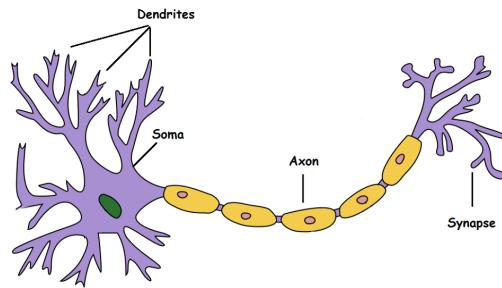
gubitka i optimizacijskome postupku.

- 9 Alternativno, aktivacijska funkcija perceptronu može se definirati kao:

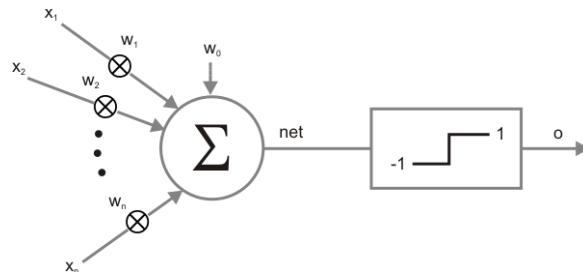
$$f(\alpha) = \begin{cases} 1 & \text{ako } \alpha \geq 0 \\ 0 & \text{inače} \end{cases}$$

što je tzv. **Heavisideova funkcija praga**. Mi koristimo aktivacijsku funkciju s kodomenom $\{-1, +1\}$ jer nam to omogućava lakšu usporedbu različitih funkcija gubitaka (naime, uz tako definiranu funkciju, $y \in \{-1, +1\}$, pa umnožak $\mathbf{w}^T \phi(\mathbf{x})y$ odgovara "točnosti" klasifikacije, što ne bi bio slučaj s Heavisideovom funkcijom).

- 10 Naziv **aktivacijska funkcija** (odnosno nešto stariji naziv **prijenosna funkcija**) inspiran je načinom rada biološkog neurona. Pojednostavljeni, biološki neuron preko svojih produžetaka (dendrita) prikuplja u tijelu stanice (somi) elektrokemijsku stimulaciju (preko neuroprijenosnika koji prelaze između sinapsi dendrita) od s njime povezanih ulaznih neurona te se, kada prikupljeni električni potencijal pređe određeni prag, neuron *aktivira* te duž svog živčanog vlakna (aksona) *prenosi* električni impuls prema s njime povaznim izlaznim neuronima:



Analogno, model perceptronu na izlazu daje $+1$ kada vrijednost linearne kombinacija težina i ulaznih značajki ($\mathbf{w}^T \mathbf{x}$) nadmaši vrijednost nula (ili, ekvivalentno, kada iznos $net = \sum_{j=1}^n w_j x_i$ nadmaši prag $-w_0$):



Aktivacijsku funkciju nemojte brkati s funkcijom gubitka: aktivacijska funkcija definira preslikavanje skalarног produkta $\mathbf{w}^T \mathbf{x}$ u izlazne oznake, dok funkcija gubitka definira kaznu uslijed pogrešne predikcije za pojedinačni primjer, gdje je predikcija dobivena kao izlaz aktivacijske funkcije. Aktivacijske funkcije i funkcije gubitka u načelu se mogu proizvoljno kombinirati, no vidjet ćemo da su iz teorijskih ili praktičnih razloga neke kombinacije smislenije od drugih.

- 11 Poneki pažljivi čitatelj možda će primijetiti da, premda govorimo o funkciji gubitka, nismo eksplisitno napisali da je funkcija gubitka jednaka ovom izrazu, tj. nismo napisali:

$$L(y, h(\mathbf{x})) = \max(0, -\mathbf{w}^T \phi(\mathbf{x})y)$$

Zašto? Zato što izraz na desnoj strani nije funkcija od $h(\mathbf{x})$, nego od $\mathbf{w}^T \phi(\mathbf{x})$. Formalno, dakle, izraz koji smo napisali nije funkcija gubitka. Međutim, kao što smo već pojasnili, izravna usporedba

funkcija gubitaka različitih algoritama mnogo je lakša ako funkciju gubitka promatramo kao funkciju od y i $\mathbf{w}^T \phi(\mathbf{x})y$ umjesto kao funkciju od y i $h(\mathbf{x})$.

- [12] **Gradijentni spust** (engl. *gradient descent*) jednostavna je tehnika optimizacije koja je sveprisutna u strojnom učenju. Ideja gradijentnog spusta jest da u nekoj početnoj točki (vektoru parametara) računamo gradijent funkcije pogreške (smjer porasta) i onda se malo po malo spuštamo u suprotnom smjeru, iterativno, računajući nanovo gradijent u svakoj iteraciji spusta, sve dok gradijent nije jednak nuli, ili vrlo blizu tome. Točka u kojoj je gradijent funkcije jednak nul-vektoru jest ekstrem funkcije (bilo minimumu ili maksimumu). Ako je funkcija konveksna, kao što je to često slučaj s našim funkcijama pogreške, onda je to globalni minimum funkcije. Više o gradijentnom spustu idući put.
- [13] To je **teorem o konvergenciji perceptrona** (Novikoff, 1963). Pristupačniji dokaz možete naći u <https://www.cse.iitb.ac.in/~shivaram/teaching/old/cs344+386-s2017/resources/classnote-1.pdf>

Literatura

- Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.
- D. O. Hebb. *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.
- N. Japkowicz. The class imbalance problem: Significance and strategies. In *Proc. of the Int'l Conf. on Artificial Intelligence*, volume 56. Citeseer, 2000.
- N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.
- W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- M. Minsky and S. Papert. *An introduction to computational geometry*. The MIT Press, 1969.
- A. B. Novikoff. On convergence proofs for perceptrons. Technical report, Stanford Research Institute, 1963.
- M. Olazaran. A sociological study of the official history of the perceptrons controversy. *Social Studies of Science*, 26(3):611–659, 1996.
- F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

6. Logistička regresija

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v2.3

Prošli put počeli smo razmatrati problem klasifikacije. Dali smo jedan općeniti uvod u **linearne diskriminativne modelle**. Analizirali smo geometriju tih modela te naučili kako napasti višeklasni problem, dekompozicijom na niz binarnih klasifikacija. Nakon toga probali smo, ponešto oportunistički, iskoristiti linearnu regresiju za klasifikaciju. Vidjeli smo da to nije dobro završilo, jer algoritam linearne regresije nije robustan na primjere koji su daleko od granice. Identificirali smo da problem leži ili u modelu ili u funkciji kvadratnog gubitka. Ako je model linearan, onda kvadratni gubitak nije dobra funkcija gubitka za klasifikaciju.

Zatim smo opisali **perceptron**, čija je funkcija gubitka doduše oblikovana za klasifikaciju, pa nemamo problema kao s regresijom, međutim perceptron ima drugih boljki: ne konvergira kod linearne neodvojivih problema i daje različite hipoteze ovisno o početnoj inicijalizaciji težina.

Danas ćemo napokon govoriti o algoritmu koji poštano obavlja klasifikaciju: to je **algoritam logističke regresije**. Nazivu unatoč, algoritam logističke regresije je klasifikacijski algoritam. Štoviše, radi se o vrlo dobrom klasifikacijskom algoritmu koji se često koristi u praksi, i zato ga je važno znati i razumijeti. [1]

1 Model logističke regresije

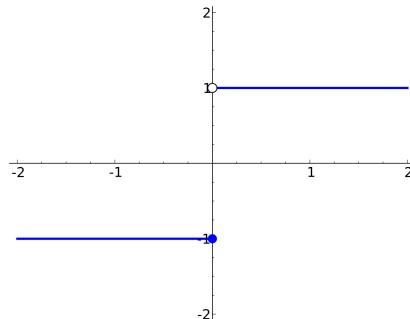
Naš pokušaj primjene regresije na klasifikaciju bio je neuspješan, ali poučan: problem je u tome što model (koji je hiperravninna) primjerima daje to veće vrijednosti što su oni dalje od granice, a funkcija gubitka to onda promptno (i drastično – kvadratno!) kažnjava.

Kako bismo to popravili, krenut ćemo najprije od toga da promjenimo model. To će nas onda automatski dovesti do drugačije funkcije gubitka (jer je gubitak povezan s modelom, budući da je gubitak definiran u odnosu na izlaz modela). Posljedično, to će nas dovesti i do drugačijeg optimizacijskog postupka (jer je optimizacijski postupak povezan s gubitkom, a koji je opet povezan s modelom). Krenimo, dakle, od modela.

Naprije, prisjetimo se modela perceptrona. Taj model definirali smo ovako:

$$h(\mathbf{x}; \mathbf{w}) = f(\mathbf{w}^T \phi(\mathbf{x}))$$

gdje je $f : \mathbb{R} \rightarrow \{-1, +1\}$ **funkcija praga** (step-funkcija):

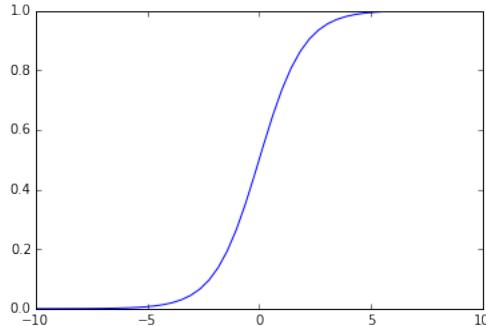


Ovakvi modeli, gdje je neka nelinearna funkcija omotana oko skalarnog umnoška vektora težina i vektora značajki, nazivaju se **poopćeni linearne modeli** (engl. *generalized linear models*, *GLM*). 2

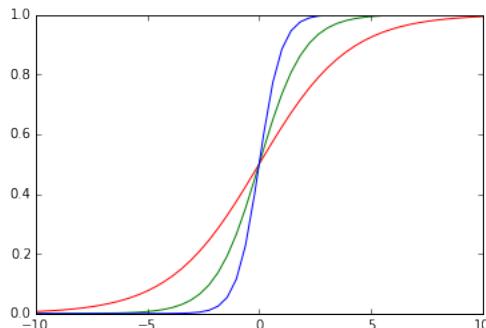
Već smo rekli da se funkcija f naziva **aktivacijska funkcija**. Tipično je to nelinearna funkcija definirana tako da vrijednost skalarnog umnoška $\mathbf{w}^T \mathbf{x}$ preslikava na neki ograničeni interval, npr. $f : \mathbb{R} \rightarrow [0, 1]$ ili $f : \mathbb{R} \rightarrow [-1, +1]$ (to također može biti i otvoreni interval $(0, 1)$ ili $(-1, 1)$). Kako je vrijednost skalarnog produkta u intervalu od minus beskonačno do plus beskonačno, vidimo da ova funkcija zapravo "stijeće" izlaz modela u ograničen interval. Zbog toga ovu funkciju ponekad zovemo i **funkcija gnječilica** (engl. *squashing function*). Koja je motivacija za to? Pa, želimo da izlaz ima ograničen raspon, kako bismo ga mogli tumačiti kao **vjerojatnost** (u slučaju intervala $[0, 1]$). Također, ne želimo da primjeri koji su jako duboko u pozitivnom ili negativnom području imaju velik izlaz modela. To je baš bio problem s linearom regresijom! Problem je bio u tome što tamo nismo koristili nikakvu aktivacijsku funkciju.

Vratimo se sada na logističku regresiju. Ideja je da upotrijebimo neku aktivacijsku funkciju, ali ne funkciju praga, kao kod perceptronu. Naime, željeli bismo neku funkciju koja je slična funkciji praga, ali da je glatka, tako da je funkcija derivabilna na cijeloj svojoj domeni. Jedna funkcija koja je za to idealna je tzv. **logistička funkcija**, također poznata kao **sigmoidna funkcija** ili, za prijatelje, **sigmoide**. Ta je funkcija definirana ovako: 3

$$\sigma(\alpha) = \frac{1}{1 + \exp(-\alpha)}$$



Nagib sigmoide može se regulirati množenjem ulaza određenim faktorom: 4



Zelenom krivuljom prikazan je graf funkcije $\sigma(\alpha)$, crvenom graf funkcije $\sigma(0.5\alpha)$, a plavom graf funkcije $\sigma(2\alpha)$. Vidimo da se množenjem ulaza sigmoide konstantom većom od 1 nagib sigmoide povećava. Primite ovo na znanje, jer će nam trebati malo kasnije.

Sigmoide ima **tri važne karakteristike** koje je čine dobrom odabirom za ovu našu rabotu. Prvo, vrijednosti gniyeći na (otvoreni) interval $(0, 1)$. Drugo, oblikom je slična funkciji praga, što znači da će davati vrijednost blizu 1 primjerima iz jedne klase, a vrijednost blizu 0 primjerima iz druge klase. Treće, funkcija je derivabilna, što nam je važno za optimizaciju. Konkretno,

derivacija sigmoidne funkcije jest (uvjerite se u ovo!):

$$\frac{d\sigma(\alpha)}{d\alpha} = \frac{d}{d\alpha} (1 + \exp(-\alpha))^{-1} = \sigma(\alpha)(1 - \sigma(\alpha))$$

Sada možemo definirati **model** logističke regresije:

$$h(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T \phi(\mathbf{x})) = \frac{1}{1 + \exp(-\mathbf{w}^T \phi(\mathbf{x}))}$$

gdje smo odmah uvrstili funkciju preslikavanja iz ulaznog prostora u prostor značajki, ϕ , kako bismo mogli ostvariti nelinearnost granice između klasa.

Ovo je binaran klasifikacijski model. Granica između dviju klasa definirana je hiperravninom u prostoru značajki za koju vrijedi $h(\mathbf{x}) = 0.5$. U tom smislu, možemo model definirati i kao:

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{1}\{\sigma(\mathbf{w}^T \phi(\mathbf{x})) \geq 0.5\}$$

Međutim, ovakva definicija nam je manje draga. Zašto? Zato što s njom gubimo informaciju o pouzdanosti klasifikacije primjera u klasu, a to je nešto što nam sigmoidna funkcija daje. Štoviše, izlaz sigmoidne funkcije je u intervalu $(0, 1)$, pa onda $h(\mathbf{x})$ možemo tumačiti kao **vjerojatnost** da primjer \mathbf{x} pripada klasi s označkom $y = 1$. Napišimo to eksplicitno:

$$h(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T \phi(\mathbf{x})) = P(y = 1 | \mathbf{x})$$

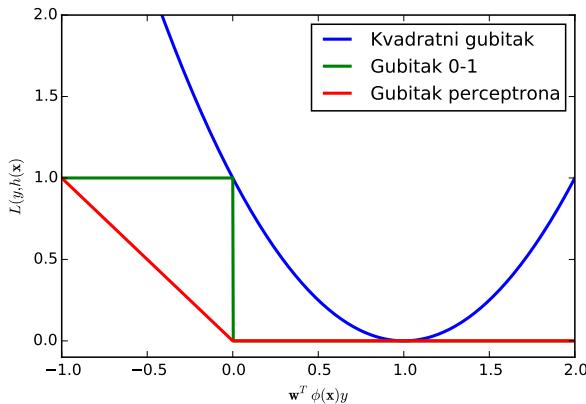
gdje je $P(y = 1 | \mathbf{x})$ uvjetna vjerojatnost označe $y = 1$ ako je primjer \mathbf{x} , tj. vjerojatnost da je označka primjera \mathbf{x} jednaka 1.

Da sažmemo: model logističke regresije svim primjerima dodjeljuju vrijednosti iz intervala $(0, 1)$, koje se dobivaju primjenom logističke funkcije na skalarni umnožak težina i značajki. Primjeri koji su daleko od granice imat će vrijednosti blizu 0 odnosno blizu 1, ovisno na kojoj su strani hiperravnine, ali nikada izvan tog intervala. Vrijednost hipoteze $h(\mathbf{x})$ možemo tumačiti kao vjerojatnost da primjer pripada pozitivnoj klasi (klasi za koju je $y = 1$).

Ovime smo definirali model logističke regresije. Sljedeće što trebamo napraviti jest definirati funkciju pogreške i optimizacijski postupak. Krenimo od funkcije pogreške.

2 Pogreška unakrsne entropije

Već znamo da je funkcija pogreške očekivanje funkcije gubitka. Sjetimo se i naše skice različitih funkcija gubitaka, koju smo bili nacrtali prošli put:



Za klasifikaciju bismo željeli gubitak koji je što sličniji gubitku 0-1, ali da je derivabilan. Trebala bi nam, dakle, funkcija koja daje relativno velik gubitak za primjere koji su pogrešno klasificirani, a malen i što manji gubitak za primjere koji su ispravno klasificirani i za koje je vjerojatnost $P(y = 1|\mathbf{x})$, tj. vrijednost hipoteze $h(\mathbf{x})$, što bliže jedinici. Pored toga, želimo da je ta funkcija derivabilna. Primijetite da niti jedan od triju funkcija gubitka iz gornjeg grafa ne ispunjava sve ove uvjete.

2.1 Izvod funkcije pogreške

Mogli bismo pokušati konstruirati “ručno” takvu funkciju gubitka, ali izvlačenje funkcije gubitka iz malog prsta ne zvuči baš obećavajuće. Naime, voljeli bismo da postoji neka dublja povezanost između funkcije gubitka i modela.

Ovdje nam može pomoći da se prisjetimo kako je to bilo kod linearne regresije: pokazali smo – doduše naknadno – da su parametri koji maksimiziraju vjerojatnosti oznaka iz označenog skupa primjera isti oni parametri koji minimiziraju kvadratnu pogrešku. Tehnički, to smo izveli tako da smo najprije pretpostavili da je na oznake koje imamo u skupu označenih primjera nadodan šum koji je normalno distribuiran, gdje je srednja vrijednost te distribucije jednaka predikciji našeg linearog modela (tj. $\mu = \mathbf{w}^T \phi(\mathbf{x})$):

$$P(y|\mathbf{x}) = \mathcal{N}(\mathbf{w}^T \phi(\mathbf{x}), \sigma^2)$$

Zatim smo napisali koja je ukupna (zajednička) vjerojatnost skupa oznaka:

$$P(\mathbf{y}|\mathbf{X}) = \prod_{i=1}^N p(y^{(i)}|\mathbf{x}^{(i)})$$

pri čemu je y oznaka jednog primjera, a \mathbf{y} vektor oznaka svih primjera iz skupa \mathcal{D} , te smo pretpostavili da su oznake primjera nezavise i da dolaze iz iste distribucije, tj. da vrijedi pretpostavka **IID**. Vjerojatnost $P(\mathbf{y}|\mathbf{X})$ smo zatim tretirali kao funkciju parametara \mathbf{w} te smo, kako bismo pojednostavili izračun, uzeli logaritam te funkcije. Na koncu smo zaključili da je negativan logaritam te funkcije proporcionalan kvadratnoj pogrešci regresije, što možemo napisati ovako:

$$E(\mathbf{w}|\mathcal{D}) \propto -\ln P(\mathbf{y}|\mathbf{X})$$

Iz toga je onda “prirodno” izašao kvadratni gubitak, pa smo zaključili da je **minimizacija empirijske pogreške** jednaka **maksimizaciji (logaritma) vjerojatnosti oznaka**.

Postupak koji smo upravo opisali nije bezvezan i nije slučajan. Na isti ovakav način možemo izvesti funkciju pogreške, odnosno gubitka, mnogih algoritama strojnog učenja. Ideja je, dakle, da krenemo od vjerojatnosti podataka (vektora svih oznaka \mathbf{y} iz skupa \mathcal{D}), da napišemo tu vjerojatnost kao funkciju parametara modela \mathbf{w} te da maksimiziramo logaritam te funkcije po parametrima modela. Negacija te funkcije je upravo empirijska pogreška $E(\mathbf{w}|\mathcal{D})$ koju želimo minimizirati.

Isti recept primjenit ćemo i sada: izvest ćemo funkciju pogreške logističke regresije kao negativan logaritam vjerojatnosti oznaka. Tu funkciju ćemo onda minimizirati po parametrima modela. Također, iz te će funkcije izaći funkcija gubitka, koju ćemo moći ucrtati u naš graf kao novu funkciju gubitka.

Prvi korak jest da odaberemo distribuciju kojom bismo modelirali distribuciju oznaka y za zadani primjer \mathbf{x} uslijed postojanja šuma, tj. distribuciju za $P(y|\mathbf{x})$. Kod linearne smo regresije za to koristili normalnu distribuciju. To je imalo smisla, jer je kod linearne regresije oznaka y broj (numerička varijabla), pa je šum normalno distribuiran. Međutim, kod logističke regresije, budući da radimo klasifikaciju, y nije broj, nego je diskretna vrijednost (0 ili 1). U vjerojatnosnom smislu, to je onda **binarna slučajna varijabla**: varijabla koja može poprimiti vrijednost

6

$y = 1$ (primjer pripada klasi) ili $y = 0$ (primjer ne pripada klasi) s nekom vjerojatnošću μ odnosno $1 - \mu$. U teoriji vjerojatnosti, takva se varijabla naziva **Bernoullijeva varijabla**. Drugim riječima, kod logističke regresije oznaku y za primjer \mathbf{x} modelirat ćemo Bernoullijevom distribucijom. Modeliranjem oznake kao slučajne varijable uvažavamo činjenicu da na oznaku utječe šum, tj. da je oznaka koju u skupu \mathcal{D} opažamo samo jedna moguća realizacija slučajne varijable.

Prije nego što idemo u detalje, prisjetimo se što je to Bernoullijeva varijabla. Bernoullijeva varijabla je slučajna varijabla koja ima dva moguća ishoda: događaj je ili nastupio ($y = 1$) ili nije ($y = 0$). Distribucija te varijable definirana je ovako:

$$P(y|\mu) = \begin{cases} \mu & \text{ako } y = 1 \\ 1 - \mu & \text{inače} \end{cases}$$

gdje parametar μ definira vjerojatnost nastupanja događaja, tj. vjerojatnost $P(y = 1)$. Distribucija Bernoullijeve varijable definirana izrazom s dvije grane (ako–inače) vrlo je jasna, međutim za naše potrebe nije prikladna. Naime, mi ćemo u nastavku trebati računati derivacije izraza koje sadrže vjerojatnosti Bernoullijeve varijable, a ako–inače izraze ne znamo derivirati. Ideja je onda da ako–inače izraz napišemo kao njemu ekvivalentan algebarski izraz, ali s operacijama koje znamo derivirati. To možemo lako napraviti, ovako:

$$P(y|\mu) = \mu^y (1 - \mu)^{1-y}$$

Uzmite si malo vremena da se uvjerite da je ovo isto kao i prethodni izraz, tj. da je ovo sažet način zapisa $P(1) = \mu$ i $P(0) = 1 - \mu$.

Vratimo se sada na modeliranje oznaka y . Oznake ćemo, dakle, modelirati Bernoullijevom varijablom. Sada je pitanje: odakle nam vrijednost parametra μ ? Primijetimo da je, po definiciji Bernoullijeve varijable, vrijednost μ zapravo vjerojatnost da je primjer klasificiran u klasu $y = 1$. Odakle nam ta vjerojatnost? Pa, tu vjerojatnost nam daje hipoteza, tj. to je upravo izlaz modela, odnosno izlaz sigmoidne funkcije. Drugim riječima, vjerojatnost da je primjer \mathbf{x} označen kao $y = 1$ je:

$$P(y = 1|\mathbf{x}) = \mu = h(\mathbf{x}; \mathbf{w})$$

Općenito, za zadani primjer \mathbf{x} , vjerojatnost njegove oznake (bilo $y = 1$ ili $y = 0$) prema definiciji za Bernoullijevu distribuciju jednaka je:

$$P(y|\mathbf{x}) = h(\mathbf{x}; \mathbf{w})^y (1 - h(\mathbf{x}; \mathbf{w}))^{1-y}$$

Dakle, samo smo umjesto μ uvrstili izlaz modela $h(\mathbf{x}; \mathbf{w})$, jer je izlaz modela jednak vjerojatnosti da je primjer klasificiran u klasu $y = 1$.

Sada ćemo ponoviti postupak po potpuno istom receptu koji smo koristili kod linearog modela regresije: definirat ćemo logaritam vjerojatnosti oznake, ovog puta uz pretpostavku Bernoullijeve distribucije za varijablu oznake y . Tu vjerojatnost tretirat ćemo kao funkciju parametra \mathbf{w} , i nju po tom parametru želimo **maksimizirati**. Nadalje, negativna vrijednost te funkcije jednaka je funkciji pogreške, koju onda želimo **minimizirati**. Pa, krenimo...

Ukupna (zajednička) vjerojatnost skupa oznaka je:

$$P(\mathbf{y}|\mathbf{X}) = \prod_{i=1}^N p(y^{(i)}|\mathbf{x}^{(i)})$$

Logaritam toga je:

$$\ln P(\mathbf{y}|\mathbf{X}) = \ln \prod_{i=1}^N p(y^{(i)}|\mathbf{x}^{(i)}) = \sum_{i=1}^N \ln p(y^{(i)}|\mathbf{x}^{(i)})$$

Uvrstimo Bernoullijevu distribuciju, uz $\mu = h(\mathbf{x}; \mathbf{w})$:

$$\begin{aligned}\ln P(\mathbf{y}|\mathbf{X}) &= \sum_{i=1}^N \ln \left(h(\mathbf{x}^{(i)}; \mathbf{w})^{y^{(i)}} (1 - h(\mathbf{x}^{(i)}; \mathbf{w}))^{1-y^{(i)}} \right) \\ &= \sum_{i=1}^N \left(y^{(i)} \ln h(\mathbf{x}^{(i)}; \mathbf{w}) + (1 - y^{(i)}) \ln (1 - h(\mathbf{x}^{(i)}; \mathbf{w})) \right)\end{aligned}$$

Emprijska pogreška jednake je negativnoj vrijednosti ove funkcije:

$$E(\mathbf{w}|\mathcal{D}) = \sum_{i=1}^N \left(-y^{(i)} \ln h(\mathbf{x}^{(i)}; \mathbf{w}) - (1 - y^{(i)}) \ln (1 - h(\mathbf{x}^{(i)}; \mathbf{w})) \right)$$

Dodatno, kako ne bi ovisila o broju primjera, empirijsku pogrešku skalirat ćemo sa $1/N$:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{N} \sum_{i=1}^N \left(-y^{(i)} \ln h(\mathbf{x}^{(i)}; \mathbf{w}) - (1 - y^{(i)}) \ln (1 - h(\mathbf{x}^{(i)}; \mathbf{w})) \right)$$

I to je to – izveli smo funkciju empirijske pogreške logističke regresije! Ova se funkcija pogreške naziva **pogreška unakrsne entropije** (engl. *cross-entropy error*). [8]

2.2 Gubitak unakrsne entropije

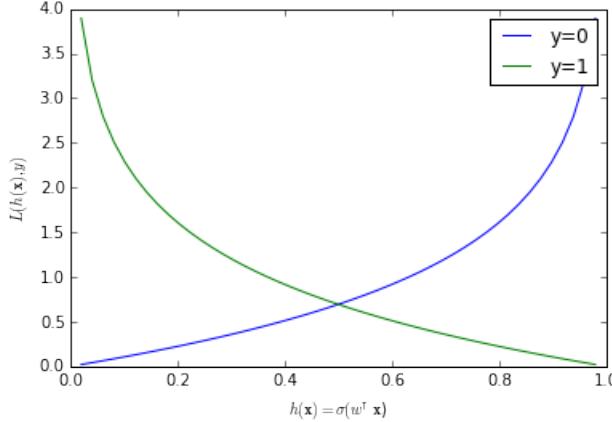
Ako je ovo što smo upravo izveli funkcija empirijske pogreške, što je onda **funkcija gubitka**? Pa, sjetimo se da je funkcija pogreške očekivanje funkcije gubitka, procijenjena kao prosjek funkcije gubitka na skupu primjera. To znači da je funkcija gubitka ono što se u izrazu za funkciju pogreške nalazi unutar izraza $\frac{1}{N} \sum_{i=1}^N$. Konkretno, dakle, funkcija gubitka je:

$$L(y, h(\mathbf{x})) = -y \ln h(\mathbf{x}) - (1 - y) \ln (1 - h(\mathbf{x}))$$

Ova se funkcija naziva **gubitak unakrsne entropije** (engl. *cross-entropy loss*).

Uvjerimo se da gubitak unakrsne entropije ima smisla. Izraz se sastoji od dva pribrojnika. Ako je oznaka primjera \mathbf{x} pozitivna, $y = 1$, onda je drugi pribrojnik jednak nuli te samo prvi pribrojnik igra ulogu. U tom slučaju, gubitak je jednak $-\ln h(\mathbf{x})$. Ako model primjer \mathbf{x} klasificira pozitivno, $h(\mathbf{x}) = 1$, onda je gubitak jednak $-\ln 1 = 0$, tj. gubitka nema. To je u redu, jer ako je $y = 1$ i $h(\mathbf{x}) = 1$, onda je klasifikacija tog pozitivnog primjera točna (istinito pozitivan primjer). S druge strane, ako model primjer \mathbf{x} klasificira negativno, $h(\mathbf{x}) = 0$, onda je gubitak jednak $-\ln(0) = \infty$, tj. gubitak je maksimalan jer je pozitivan primjer klasificiran netočno (lažno negativan primjer). Slično vrijedi i za obrnutu situaciju, ako je oznaka primjera negativna, $y = 0$. Tada je prvi pribrojnik jednak nuli, a drugi pribrojnik jednak je $-\ln(1-h(\mathbf{x}))$. Ako model primjer \mathbf{x} klasificira negativno, $h(\mathbf{x}) = 0$, onda je gubitak jednak nuli. S druge strane, ako model primjer \mathbf{x} klasificira pozitivno, $h(\mathbf{x}) = 1$, onda je gubitak maksimalan, $-\ln(0) = \infty$. Vidimo, dakle, da je gubitak nula ako je klasifikacija točna, a da je maksimalan (beskonačan) ako je klasifikacija netočna. Sada još samo trebamo primijetiti da izlaz modela $h(\mathbf{x})$ nikada neće biti 0 niti 1, jer je kodomena sigmoidne funkcije otvoreni interval $(0, 1)$. Posljedično, gubitak za točno klasificirani primjer nikada neće biti baš nula, nego blizu nule, kao što ni gubitak za netočno klasificirani primjer neće biti beskonačno velik, nego će biti neki velik ali konačan broj. Koliko će gubitak biti blizu nuli u slučaju točne klasifikacije, odnosno koliko će biti velik u slučaju netočne klasifikacije ovisi o tome koliko je izlaz modela (sigmoidne funkcije) blizu nuli odnosno jedinici.

Ovo postaje još jasnije ako funkciju unakrsnog gubitka $L(y, h(\mathbf{x}))$ prikažemo grafički. Budući da se radi o funkciji dviju varijabli, prikazat ćemo ju s dvije krivulje, jednom za $y = 0$ i drugom za $y = 1$:

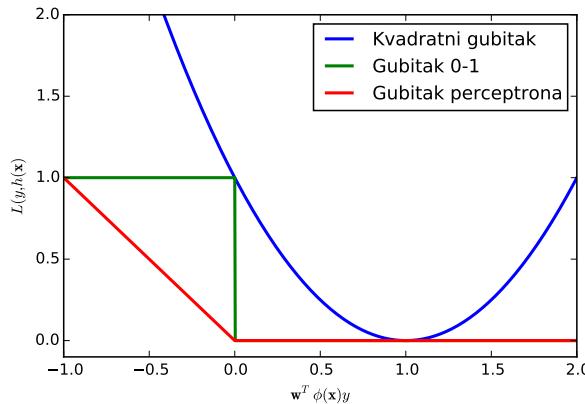


Iz grafa je opet lijepo vidljivo da gubitak to bliže nuli što je $h(\mathbf{x})$ bliži y , te da je gubitak to veći što je $h(\mathbf{x})$ dalje od y . Naravno, izlaz modela $h(\mathbf{x})$ može biti i bilo koja vrijednost između 0 i 1. Npr., koliki je gubitak na primjeru \mathbf{x} za koji model daje $h(\mathbf{x}) = P(y = 1|\mathbf{x}) = 0.7$, ako je stvarna oznaka primjera $y = 0$? Iz grafa vidimo da je gubitak onda negdje između 1 i 1.5. S druge strane, ako bi izlaz modela bio isti, a oznaka primjera $y = 1$, onda bi gubitak bio manji od 0.5.

Budući da gubitka nema jedino onda kada je primjer savršeno točno klasificiran ($h(\mathbf{x}) = 1$ za $y = 1$ odnosno $h(\mathbf{x}) = 0$ za $y = 0$), a da se to ne može dogoditi jer izlaz sigmoide nikada nije 1 niti 0, zaključujemo da uvijek postoji neki gubitak. Dakle, čak i ako je primjer točno klasificiran (nalazi se na ispravnoj strani granice), ipak nanosi malen gubitak, ovisno o pouzdanosti klasifikacije. No, ono što je bitno jest da primjeri na ispravnoj strani granice ($h(\mathbf{x}) \geq 0.5$ za $y = 1$ odnosno $h(\mathbf{x}) < 0.5$ za $y = 0$) nanose puno manji gubitak od primjera na pogrešnoj strani granice.

2.3 Usporedba funkcija gubitaka

Sada kada smo izveli funkciju gubitka unakrsne entropije, vratimo se na grafikon funkcija gubitaka i usporedimo tu funkciju s drugim funkcijama gubitka s kojima smo se već upoznali. Prisjetimo se, taj grafikon izgleda ovako:



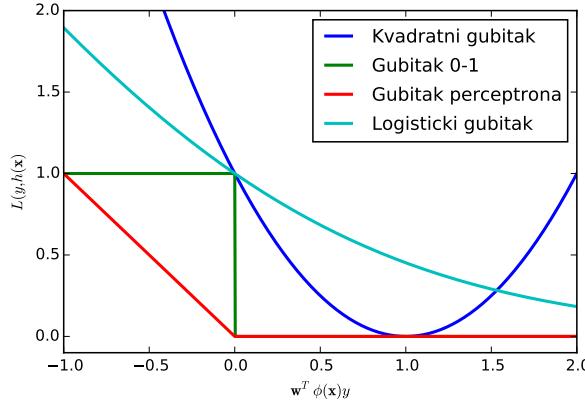
Međutim, da bismo to mogli napraviti, funkciju gubitka unakrsne entropije, koja je funkcija dvije varijable, trebamo preformulirati tako da to bude funkciju jedne varijable. Konkretno, trebamo napraviti isto što smo napravili i za funkciju kvadratnog gubitka: prebaciti se sa skupa oznaka $\{0, 1\}$ na skup oznaka $\{-1, +1\}$ te zatim funkciju gubitka preformulirati u funkciju od $\mathbf{w}^T \phi(\mathbf{x})y$. To možemo lako napraviti, no ovdje ćemo to preskočiti i umjesto toga odmah dati konačan rezultat:

$$L(\mathbf{w}^T \phi(\mathbf{x})y) = \ln(1 + \exp(-\mathbf{w}^T \phi(\mathbf{x})y))$$

Ovo se sada zove **funkcija logističkog gubitka** (engl. *logistic loss*). Mala nezgodancija s ovom funkcijom jest da, za primjer koji se nalazi točno na granici, $\mathbf{w}^T \phi(\mathbf{x}) = 0$, vrijednost funkcije je $\ln(1 + \exp(0)) = \ln 2$. To nam ne odgovara za naš grafikon, u kojemu sve funkcije gubitka (osim za gubitak perceptronu) vrijedi $L(0) = 1$. Zato ćemo napravili malu korekciju, ovako:

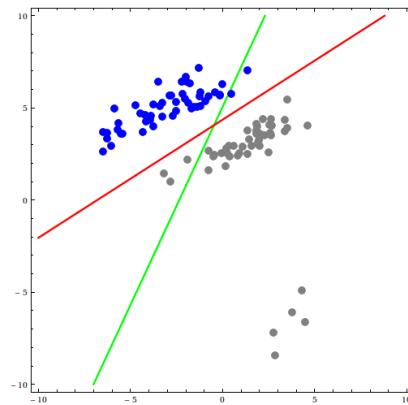
$$L(\mathbf{w}^T \phi(\mathbf{x})y) = \frac{1}{\ln 2} \ln (1 + \exp(-\mathbf{w}^T \phi(\mathbf{x})y))$$

Ovime smo samo malo (za konstantan faktor $\frac{1}{\ln 2}$) skalirali vrijednost funkcije gubitka, kako bi vrijedilo $L(0) = 1$. Time se u suštini ništa ne mijenja, samo što graf izgleda urednije. I sada, napokon, funkciju logističkog gubitka možemo ucrtati u grafikon funkcija gubitaka:



Vidimo da funkcija logističkog gubitka kažnjava netočno klasificirane primjere (negativan dio x -osi) iznosom većim od jedan, i da taj iznos raste što je primjer manje točno klasificiran (tj. što je dalje od granice na njezinoj pogrešnoj strani). S druge strane, gubitak za točno klasificirane primjere (pozitivan dio x -osi) je manji od onih za netočno klasificirane primjere, te pada što je primjer točnije klasificiran (tj. što je dalje od granice na ispravnoj strani). Međutim, primjetimo da gubitak nikada nije nula – čak i točno klasificirani primjeri nanose gubitak. Kao što smo već rekli, to je zato što izlaz modela $h(\mathbf{x})$ nikada nije točno 0 niti 1. Isto možemo zaključiti iz definicije funkcije logističkog gubitka kao funkcije jedne varijable: što je točno klasificirani primjer dalje od granice, to je $\mathbf{w}^T \phi(\mathbf{x})y$ većeg iznosa, i to je $\ln(1 + \exp(-\mathbf{w}^T \phi(\mathbf{x})y))$ bliži nuli, međutim nikada nije jednak nuli.

Sve ovo znači da, za razliku od linearne regresije, logistička regresija neće imati problem s primjerima koji su daleko od granice, a na njezinoj ispravnoj strani. Ilustrirajmo to na binarnoj klasifikaciji u dvodimenzijskom ulaznom prostoru:



Ovaj smo primjer već bili razmotrili prošli put. Zelenom linijom prikazana je granica između dviju klasa (plavi i sivi primjeri) koju dobivamo linearnom regresijom treniranom tako da primjerima jedne klase pridjeljuje vrijednost 1 a drugima vrijednost 0 (ili -1). Problem je što

kod kvadratnog gubitka primjeri koji su daleko od granice (grupa sivih primjera dolje desno) nanose velik gubitak, neovisno o tome jesu li točno ili netočno klasificirani. Zbog toga dobivamo hipotezu koja netočno klasificira neke od primjera iz skupa označenih primjera, premda je taj skup linearne odvojiv. Crvenom linijom prikazana je granica između klase koju dobivamo algoritmom logističke regresije. Primjeri koji su daleko od granice a točno klasificirani (grupa sivih primjera dolje desno) ne nanose velik gubitak (kao što smo vidjeli, gubitak teži k nuli što je točno klasificirani primjer pozicioniran dalje od granice), pa stoga nemaju velik utjecaj na položaj granice. Naravno, kad bi ti primjeri bili primjeri druge klase (plavi), onda bi oni nanosili značajan gubitak, budući da bi se u tom slučaju ti netočno klasificirani primjeri nalazili daleko od granice i to na njezinoj pogrešnoj strani.

2.4 Sažetak

Sažmimo do sada učinjeno. Model logističke regresije definirali smo ovako:

$$h(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T \phi(\mathbf{x})) = \frac{1}{1 + \exp(-\mathbf{w}^T \phi(\mathbf{x}))} = P(y = 1 | \mathbf{x})$$

zadnji izraz nas treba podsjetiti da je izlaz logističke regresije zapravo jednak vjerojatnosti da je primjer označen pozitivno. Zato kažemo da je logistička regresija **probabilistički klasifikator**. Logistička regresija je ujedno **diskriminativan klasifikacijski model**, jer modelira isključivo granicu između klasa (pomoću težina \mathbf{w}), a ne i distribuciju klasa, kao što bi to radili generativni modeli.

Funkciju pogreške zatim smo izveli kao negativan logaritam vjerojatnosti oznaka iz skupa \mathcal{D} , ovako:

$$E(\mathbf{w} | \mathcal{D}) = \frac{1}{N} \sum_{i=1}^N \left(-y^{(i)} \ln h(\mathbf{x}^{(i)}; \mathbf{w}) - (1 - y^{(i)}) \ln (1 - h(\mathbf{x}^{(i)}; \mathbf{w})) \right)$$

iz čega smo iščitali funkciju gubitka unakrsne entropije:

$$L(y, h(\mathbf{x}; \mathbf{w})) = -y \ln h(\mathbf{x}; \mathbf{w}) - (1 - y) \ln (1 - h(\mathbf{x}; \mathbf{w}))$$

Ovdje želim da primijetite da smo funkciju gubitka izveli iz funkcije pogreške, a ne obrnuto. To možda izgleda neobično: znamo da je funkcija pogreške očekivanje funkcije gubitka, pa se možda čini smislenijim da prvo definiramo funkciju gubitka, a onda funkciju pogreške jednostavno definiramo kao njezino očekivanje. Međutim, oba su pristupa jednako legitimna: budući da su pogreška i gubitak povezani, možemo krenuti od pogreške i izvesti gubitak, ili obrnuto. Ovisno o vrsti algoritama, nekad je smisleniji jedan a nekad drugi pristup. Kod probabilističkih modela (modela koji modeliraju vjerojatnost da primjer pripada klasi), kao što je to slučaj s logističkom regresijom, smislenije je krenuti od funkcije pogreške, odnosno negativnog logaritma vjerojatnosti oznaka, a onda iz toga iščitati funkciju gubitka.

11

3 Gradijentni spust

Definirali smo dvije komponente algoritma logističke regresije: model i funkciju gubitka (i pripadnu funkciju pogreške). Nedostaje nam još samo treća komponenta, a to je postupak optimizacije. Kao i uvijek, postupak optimizacije treba nam dati težine \mathbf{w} koje minimiziraju empirijsku pogrešku na skupu za učenje:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w} | \mathcal{D})$$

Kao što smo radili do sada, možemo pokušati izraziti gradijent za $E(\mathbf{w} | \mathcal{D})$ po \mathbf{w} , izjednačiti to s nulom i tako analitički pronaći minimizator \mathbf{w}^* . Nažalost, kod logističke regresije to neće

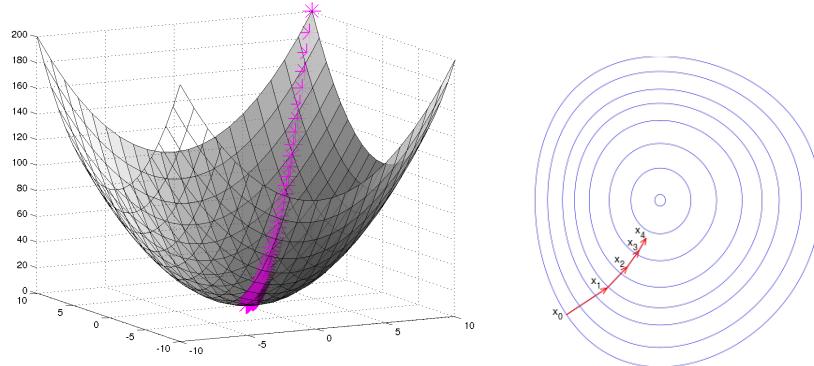
funkcionirati jer u ovom slučaju izlaz modela nelinearno ovisi o težinama \mathbf{w} (zbog nelinearne aktivacijske funkcije σ). Zbog toga za ovaj optimizacijski problem nažalost **ne postoji rješenje u zatvorenoj formi!**

Sličan problem imali smo i kod perceptronu. Kako smo ga tamo riješili? Radili smo **gradijentni spust**. Kod perceptronu je dodatan problem bio taj što funkcija praga nije neprekidna, ali to smo riješili tako da smo derivirali samo za primjere koji su netočno klasificirani (za primjere koji su točno klasificirani ionako je derivacija bila jednaka nuli).

I kod logističke regresije ćemo također raditi gradijentni spust. To nije jedina mogućnost, ali je najjednostavnija. Prisjetimo se, ideja gradijentnog spusta jest da se, krenuvši od neke početne točke (početnog vektora \mathbf{w}), postepeno “spuštamo” niz površinu funkcije $E(\mathbf{w}|\mathcal{D})$ u smjeru suprotnome od gradijenta u točki \mathbf{w} . To ponavljamo dok se ne spustimo u točku u kojoj je gradijent jednak nuli ili je barem dovoljno blizu nuli. Formalno, težine \mathbf{w} u svakoj iteraciji gradijentnog spusta ažuriramo na ovaj način:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w}|\mathcal{D})$$

gdje je η **stopa učenja** koja određuje veličinu koraka koje radimo spuštajući se prema minimumu. Ako radimo korake u dotičnom smjeru, spustit ćemo se do minimuma. Npr., u dvodimenzijском prostoru parametara to bi moglo izgledati ovako:

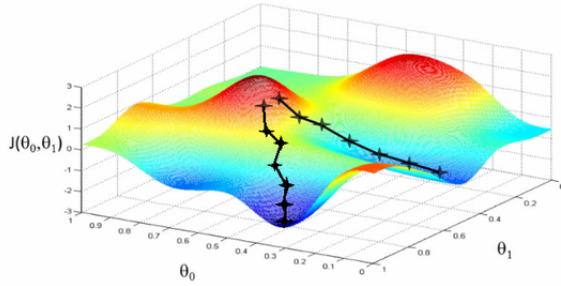


Ljeva slika prikazuje površinu funkcije pogreške $E(\mathbf{w}|\mathcal{D})$ niz koju se gradijentnim spustom iterativno spuštamo do minimuma (ljubičasti trag). Na desnoj su slici prikazane izokonture te funkcije pogreške i crvenim je vektorima naznačena staza kojom se gradijentim spustom spuštamo do minimuma funkcije (u sredini izokontura). Vektori odgovaraju negativnom gradijentu u točki funkcije pogreške u ishodištu vektora. Vektori su uvijek okomiti na izokonturu u točki ishodišta vektora, jer je smjer koji je okomit na izokonturu smjer najbržeg smanjivanja vrijednosti funkcije. Također možemo vidjeti da su vektori to manji što smo se spustili bliže minimumu, jer je nagib površine funkcije pogreške sve manji kako se primičemo točki minimuma.

Da bismo, dakle, algoritmom gradijentnog spusta pronašli minimum funkcije empirijske pogreške logističke regresije, trebat ćemo izračunati gradijent te funkcije. No, prije nego što to učinimo, pogledajmo nekoliko detalja.

3.1 Konveksnost

Gradijentni spust dovest će nas to točke za koju vrijedi $\nabla E(\mathbf{x}; \mathbf{w}) = 0$. Je li ta točka nužno minimum funkcije pogreške? Minimum da, ali to ne mora nužno biti globalni minimum. Nameće, funkcija pogreške općenito može imati jedan ili više **lokalnih minimuma**. U tom slučaju gradijentni spust, ovisno o početnoj točki, lako može zaglaviti u nekom od lokalnih minimuma. Na primjer:



Slika prikazuje funkciju pogreške koje ima nekoliko lokalnih minimuma. Ovisno o početnoj točki iz koje kreće gradijentni spust, možemo završiti u nekom od tih lokalnih minimuma.

Iz navedenog razloga u strojnog učenju posebno volimo funkcije koje su **konveksne**. Takve funkcije imaju samo jedan minimum, dakle, taj jedan minimum onda je sigurno i globalni minimum jer lokalnih minimuma nema. Optimizacijom konveksnih funkcija bavi se **konveksna optimizacija**, koja je jako važna za strojno učenje (možete o njoj razmišljati kao o starijoj sestri strojnog učenja; druga starija sestra je statistika).

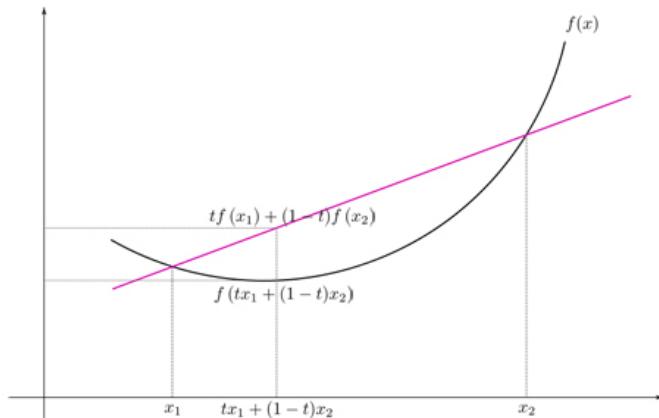
Neformalno, konveksna funkcija je ona koja je oblika "zdjele". Malo formalnije, funkcija $f : \mathbb{R}^n \rightarrow \mathbb{R}$ je **konveksna** akko: (1) domena of f , $\text{dom}(f)$, je konveksan skup i (2) svaka točka na svakoj sekanti funkcije (spojnici između dvije točke na krivulji) uvijek je veća ili jednaka od vrijednosti funkcije. Još formalnije, funkcija f je konveksna ako i samo ako:

1. Njezina domena $\text{dom}(f)$ je **konveksan skup**:

$$\forall \mathbf{x}_1, \dots, \mathbf{x}_n \in \text{dom}(f), \forall \alpha_1, \dots, \alpha_n. \sum_i \alpha_i = 1 \quad \text{vrijedi} \quad \sum_{i=1}^n \alpha_i \mathbf{x}_i \in \text{dom}(f)$$

2. Za svaki $\mathbf{x}_1, \mathbf{x}_2 \in \text{dom}(f)$ i svaki $t \in [0, 1]$ vrijedi:

$$f(\mathbf{x}) = f(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) \leq tf(\mathbf{x}_1) + (1-t)f(\mathbf{x}_2)$$



Kada će funkcija empirijske pogreške biti konveksna funkcija? Pa, prvo trebamo primijetiti da je empirijska pogreška definirana kao zbroj gubitaka. Zatim trebamo znati da je svojstvo konveksnosti aditivno: zbroj konveksnih funkcija je također konveksna funkcija. Dakle, konveksnost funkcije pogreške će ovisiti o funkciji gubitka: funkcija pogreške bit će konveksna ako i samo ako je funkcija gubitka konveksna. Mnoge funkcije pogreške, koje su nama zanimljive u strojnog učenju, jesu konveksne funkcije. Npr., funkcija pogreška regresije (suma reziduala) je konveksna funkcija. Također, pogreška unakrsne entropije je konveksna funkcija, budući da je i gubitak unakrsne entropije konveksna funkcija, u što se možete uvjeriti ako pogledate graf te funkcije.

14
15

16

3.2 Stopa učenja i globalna konvergencija

Rekli smo da konstanta η određuje **stopu učenja**, odnosno koliko velike korake radimo dok se spuštamo. Stopa učenja treba biti odabrana tako da nije ni prevelika ni premalena. Naime, ako odaberemo preveliku stopu učenja, može nam se dogoditi da postupak ne konvergira već da **divergira**. S druge strane, ako odaberemo premalenu stopu učenja, postupak će doduše konvergirati, ali će konvergencija biti spora:



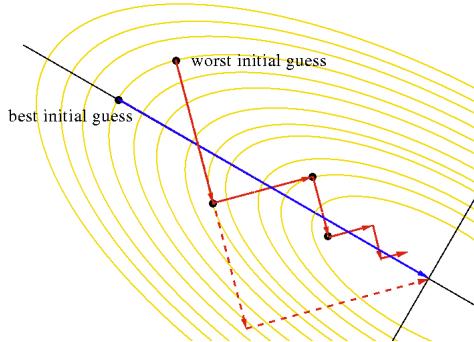
Na lijevoj slici prikazana je situacija kada je stopa učenja prevelika, pa postupak divergira. Na desnoj slici prikazana je obrnuta situacija: kada je stopa učenja premala, pa postupak sporo konvergira. U sredini je prikazana situacija za optimalnu stopu učenja.

Kako odrediti optimalnu stopu učenja? Za početak, željeli bismo imati jamstvo da će gradijentni spust konvergirati, neovisno o tome odakle započinje spust. To se svojstvo naziva **globalna konvergencija**. Divergenciju možemo sprječiti – i time ostvariti globalnu konvergenciju – tako da u svakoj iteraciji spusta odaberemo optimalnu stopu učenja. Ideja je da, nakon što izračunamo gradijent, pretražujemo u smjeru suprotnome od smjera gradijenta sve dok ne dodemo do mjesta minimuma funkcije u tom smjeru, i da napravimo korak točno do tog mjesta. Ta se tehnika naziva **linijsko pretraživanje** (engl. *line search*). Dakle, kod linijskog pretraživanja, ako je $\Delta\mathbf{x} = -\nabla f(\mathbf{x})$ smjer spuštanja, onda optimiramo funkciju:

$$g(\eta) = f(\mathbf{x} + \eta\Delta\mathbf{x})$$

odnosno uzimamo η koji minimizira g . Primjetite da se linijsko pretraživanje svodi na optimizaciju, odnosno nalaženje minimuma, funkcije jedne varijable (dakle u jednoj dimenziji, tj. duž pravca). Za pretraživanje duž pravca tipično se koristi pretraživanje koje (velik) početni interval rekurzivno dijeli na podintervale ili se koristi algoritam linijskog pretraživanja s povratkom.

Pogledajmo primjer linijskog pretraživanja u dvodimenziskom prostoru parametara:



Slika prikazuje izokonture funkcije pogreške u prostoru parametara. Plavom je linijom označen trag gradijentnog spusta u optimalnom slučaju, kada je gradijent izračunat u početnoj točki direktno vodi do minimuma funkcije. Crvenom je označen trag spusta za najgori slučaj, kada je gradijent u početnoj točki pod kutem od 45° u odnosu na gradijent optimalnog slučaja. Od početne točke, linijskim pretraživanjem došli bismo do točke koja je minimum duž smjera spusta, a to je točka u kojoj je smjer spusta tangencijalan na izokonturu (na slici to je crna točka neposredno prije iscrtkane crvene linije). U toj točki ponovo računamo gradijent, koji će

biti okomit na izokonturu, te nastavljamo s linijskim pretraživanjem u smjeru suprotnome od gradijenta, itd.

Alternativa linijskom pretraživanju jest da koristimo neku od varijanti gradijentnog spusta s **adaptivnom (prilagodivom) stopom učenja**. Mi se tim postupcima nećemo baviti.

3.3 Stohastički gradijentni spust i online učenje

Znamo već da je funkcija pogreške očekivanje funkcije gubitka, a da to očekivanje aproksimiramo prosječnom vrijednošću funkcije gubitka na skupu za učenje. Prema tome, funkcija pogreške zapravo je zbroj funkcija gubitka za svaki primjer (do na faktor $\frac{1}{N}$):

$$E(\mathbf{w}|\mathcal{D}) \propto \sum_{i=1}^N L(y^{(i)}, h(\mathbf{x}^{(i)}; \mathbf{w}))$$

To nam onda daje dvije mogućnosti kako ažurirati težine kroz iteracije gradijentnog spusta: ili ćemo (1) napraviti derivaciju gubitaka za sve primjere iz skupa za učenje i onda zakoraknuti prema minimumu ili ćemo (2) za svaki primjer pojedinačno napraviti korak prema minimumu. Prvi pristup nazivamo **standardni gradijentni spust** ili **grupni gradijentni spust** (engl. *batch gradient descent*). Drugi pristup nazivamo **stohastički gradijentni spust** (engl. *stochastic gradient descent*, **SGD**), i kod njega težine ažuriramo za svaki primjer zasebno. Formalno, kod standardnog gradijentnog spusta težine ažuriramo na sljedeći način:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \sum_{i=1}^N \nabla_{\mathbf{w}} L(y^{(i)}, h(\mathbf{x}^{(i)}; \mathbf{w}))$$

dok kod stohastičkog gradijentnog spusta to radimo ovako:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L(y^{(i)}, h(\mathbf{x}^{(i)}; \mathbf{w}))$$

Primijetite da smo u oba slučaja zanemarili konstantu $\frac{1}{N}$. Ona se može apsorbirati u stopu učenja η (koja onda, doduše, ovisi o broju primjera N).

Stohastički gradijentni spust ima neke prednosti pred standardnim gradijentnim spustom. Npr., stohastički gradijentni spust manje je računalno zahtjevan od standardnog gradijentnog spusta, koji, prije nego što napravi korak, mora pozbrajati gradijent za svaki primjer. Razlika pogotovo dolazi do izražaja kod velikih skupova podataka (engl. *big data*), a problem u tom slučaju može biti i da sve primjere ne možemo učitati u memoriju. Stohastičkim gradijentnim spustom možemo u istom vremenu napraviti više koraka, što često znači da možemo doseći točku bližu globalnom optimumu. Nadalje, stohastički gradijentni spust u pravilu radi bolje kada funkcije pogreške nije konveksna, već ima mnoge lokalne minimume. Naime, u tom slučaju stohastičko krivudanje pri spustu može pomoći da algoritam ne zaglavi u lokalnom optimumu.

Kompromisno rješenje između standardnog i stohastičkog gradijentnog spusta, a koje se pokazalo da u praksi radi najbolje, jest da se gradijent izračunava za manje, slučajno uzorkovane grupe primjera, tzv. **minigrupe** (engl. *minibatches*). Gradijentni spust s minigrupama u pravilu radi bolje od standardnog i stohastičkog gradijentnog spusta s pojedinačnim primjerima. To je zato jer uvodi dovoljno stohastičnosti da pretraga ne zapinje tako lako u lokalnom optimumu, ali je opet dovoljno usmjerena da pretraga ne krivuda previše i time završi u suboptimalnoj točki.

U praksi, implementacije algoritma logističke regresije tipično koriste stohastički gradijentni spust. Motivacija za to nije izbjegavanje lokalnih optimuma (njih nema jer je funkcija pogreške unakrsne entropije konveksna), već brzina konvergencije i stabilnost rješenja. Napomenimo još da se linijsko pretraživanje u tom slučaju ne koristi. Naime, kod stohastičnog gradijentnog spusta (također i kod gradijentnog spusta s minigrupama) smjer spusta ionako je aproksimacija pravog gradijenta u toj točki, pa nema smisla trošiti računalno vrijeme na odabir optimalne stope učenja.

4 Gradijentni spust za logističku regresiju

Sada kada sve ovo znamo o gradijentnom spustu, možemo izvesti gradijentni spust specifično za logističku regresiju, tj. za pogrešku unakrsne entropije. Prisjetimo se, pogreška unakrsne entropije je:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{N} \sum_{i=1}^N \left(-y^{(i)} \ln h(\mathbf{x}^{(i)}; \mathbf{w}) - (1 - y^{(i)}) \ln (1 - h(\mathbf{x}^{(i)}; \mathbf{w})) \right)$$

Znamo da je funkcija pogreške prosjek funkcija gubitka po svim primjerima, pa je onda gradijent funkcije pogreške jednostavno prosjek gradijenata funkcije gubitka:

$$\nabla_{\mathbf{w}} E(\mathbf{w}|\mathcal{D}) = \frac{1}{N} \sum_{i=1}^N \nabla_{\mathbf{w}} L(y^{(i)}, h(\mathbf{x}^{(i)}; \mathbf{w}))$$

Za stohastički gradijentni spust trebat će nam samo ∇L , dok ćemo za standardni (grupni) gradijentni spust ukupni gradijent dobiti zbrajanjem gradijenata za pojedinačne primjere, prema gornjoj formuli. Izračunajmo onda najprije gradijent funkcije gubitka. Funkcija gubitka je:

$$L(y, h(\mathbf{x})) = -y \ln h(\mathbf{x}) - (1 - y) \ln (1 - h(\mathbf{x}))$$

Gradijent te funkcije po težinama \mathbf{w} je:

$$\begin{aligned} \nabla_{\mathbf{w}} L(y, h(\mathbf{x})) &= \left(-\frac{y}{h(\mathbf{x})} + \frac{1-y}{1-h(\mathbf{x})} \right) h(\mathbf{x})(1-h(\mathbf{x}))\phi(\mathbf{x}) \\ &= (h(\mathbf{x}) - y)\phi(\mathbf{x}) \end{aligned}$$

(Uvjerite se da možete sami napraviti ovaj izvod.) Ovdje smo, budući da je funkcija L kompozicija nekoliko funkcija, za deriviranje primjenili pravilo ulančavanja te iskoristili već ranije spomenuto činjenicu da je derivacija sigmoidne funkcije (a ovdje je to funkcija $h(\mathbf{x}; \mathbf{w})$) jednaka $\sigma(\alpha)(1 - \sigma(\alpha))$. Taj se izraz, međutim, pokratio, pa smo u konačnici dobili prilično elegantan izraz za gradijent funkcije gubitka. Vidimo da je gradijent gubitka za primjer \mathbf{x} zapravo jednak vektoru $\phi(\mathbf{x})$ (vektor primjera \mathbf{x} u prostoru značajki) pomnoženom sa skalarom $h(\mathbf{x}) - y$, što je razlika predikcije modela i točne oznake za primjer. To intuitivno ima smisla, jer što se predikcija modela $h(\mathbf{x})$ i točna oznaka y za neki primjer više razlikuju, to je veći nagib funkcije gubitka (pogledajte graf funkcije gubitka unakrsne entropije), odnosno to je veći utjecaj promjene težina na gubitak za dotični primjer.

Ovo nam je već dovoljno za stohastički gradijentni spust. Ako želimo raditi standardni (grupni) gradijentni spust, onda gradijent funkcije pogreške po težinama \mathbf{w} izračunavamo kao:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \sum_{i=1}^N (h(\mathbf{x}^{(i)}) - y^{(i)})\phi(\mathbf{x}^{(i)})$$

gdje smo faktor $1/N$ ispustili, jer ga se može apsorbirati u stopu učenja η .

Napokon, sve ovo sada možemo složiti u algoritam optimizacije težina logističke regresije gradijentnim spustom:

► Logistička regresija – standardni (grupni) gradijentni spust

```

1:  $\mathbf{w} \leftarrow (0, 0, \dots, 0)$ 
2: ponavljam do konvergencije
3:  $\Delta\mathbf{w} \leftarrow (0, 0, \dots, 0)$ 
4: za  $i = 1, \dots, N$ 
5:  $h \leftarrow \sigma(\mathbf{w}^T \phi(\mathbf{x}^{(i)}))$ 
6:  $\Delta\mathbf{w} \leftarrow \Delta\mathbf{w} - (h - y^{(i)}) \phi(\mathbf{x}^{(i)})$ 
7:  $\eta \leftarrow$  optimum linijskim pretraživanjem u smjeru  $\Delta\mathbf{w}$ 
8:  $\mathbf{w} \leftarrow \mathbf{w} + \eta \Delta\mathbf{w}$ 

```

Stohastički gradijentni spust razlikuje se od standardnog po tome što (1) ažuriranje težina radimo unutar petlje koja iterira po svim primjerima, (2) ne koristimo linijsko pretraživanje i (3) prije svakog prolaska kroz sve primjere (tzv. **epocha**) slučajno permutiramo primjere, kako se ažuriranje težina ne bi u svakoj epohi radilo na primjerima u istome redoslijedu, što bi smanjilo stohastičnost gradijentnog spusta.

27

► Logistička regresija – stohastički gradijentni spust

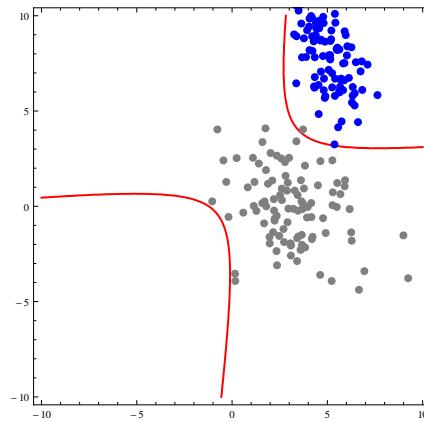
```

1:  $\mathbf{w} \leftarrow (0, 0, \dots, 0)$ 
2: ponavljam do konvergencije
3: slučajno permutiraj primjere u  $\mathcal{D}$ 
4: za  $i = 1, \dots, N$ 
5:  $h \leftarrow \sigma(\mathbf{w}^T \phi(\mathbf{x}^{(i)}))$ 
6:  $\Delta\mathbf{w} \leftarrow -(h - y^{(i)}) \phi(\mathbf{x}^{(i)})$ 
7:  $\mathbf{w} \leftarrow \mathbf{w} + \eta \Delta\mathbf{w}$ 

```

5 Regularizirana regresija

Već znamo da strojno učenje ima problema sa presloženim modelima. Ni algoritam logističke regresije nije imun na taj problem. Evo primjera presloženog modela logističke regresije:



Na slici je prikazana granica između klasa u dvodimenziskom ulaznom prostoru (crvene krivulje) dobivena algoritmom logističke regresije s funkcijom preslikavanja $\phi(\mathbf{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$. Ovaj je model očito presložen, jer dobivena hipoteza (skoro) savršeno klasificira sve primjere iz skupa za učenje, međutim granica je više nelinearna nego što bi to trebala biti. Konkretno, donji desni dio ulaznoga prostora proglašen je regijom plavih primjera, premda u tom prostoru nema niti jedan plavi primjer.

Znamo da je jedan od načina da spriječimo prenaučenost, ili barem smanjimo mogućnost prenaučenosti, **regularizacija**. Regularizacijom težine modela potiskujemo prema nuli, što ostvarujemo dodavanjem regularizacijskog izraza u empirijsku pogrešku definiranog tako da kažnjava hipoteze s velikim težinama.

Kod logističke regresije, možemo pričati o “trostrukom učinku” regularizacije:

1. Ako je model nelinearan, regularizacijom sprečavamo prenaučenost uslijed pretjerane ne-linearnosti (gornji primjer);
2. Ako imamo puno značajki, regularizacijom efektivno smanjujemo broj značajki jer težine potiskujemo prema nuli, čime sprečavamo prenaučenost uslijed previše značajki (ovo vrijedi za L_1 -regularizaciju);
3. Specifično za logističku regresiju: Ako je problem linearno odvojiv, sprječavamo “otvrđnjivanje” sigmoide (povećanje njezina nagiba), koje bi inače opet dovelo do prenaučenosti modela.

28

Posljednja točka zasluguje dodatno objašnjenje. Zašto porast težina \mathbf{w} uzrokuje otvrđnjavanje sigmoide, zašto bi težine uopće rasle, i zašto je to slučaj samo s linearno odvojivim problemima? Prisjetimo se, najprije, da s porastom magnituda težina, $\|\mathbf{w}\|$, raste i skalarni umnožak $\mathbf{w}^T \phi(\mathbf{x})$. To znači da raste argument sigmoidne funkcije, a već smo bili rekli da je efekt toga to da je sigmoida strmija. Ovo se događa samo ako su primjeri linearno odvojivi, jer samo u tom slučaju otvrđnjavanje sigmoide dovodi do smanjenja pogreške unakrsne entropije. U to se možete uvjeriti ako pogledate graf funkcije logističkog gubitka. Što je umnožak $\mathbf{w}^T \phi(\mathbf{x})y$ veći, to je gubitak $L(y, h(\mathbf{x}))$ manji. Prema tome, optimizacijskom je algoritmu u interesu da povećava magnitudu težina \mathbf{w} . Dapače, optimizacijski algoritam će htjeti težine \mathbf{w} povećati do beskonačnosti, jer bi tek tada gubitak bio jednak nuli. Također, prisjetite se s našeg prošlog predavanja da množenje vektora težina \mathbf{w} (koji uključuje težinu w_0) nema efekta na položaj i orijentaciju hiperravnine. To znači da optimizacijski algoritam može proizvoljno povećavati težine \mathbf{w} , u nastojanju da smanji empirijsku pogrešku, a da to nema nikakvog utjecaja na položaj i orijentaciju hiperravnine. Jedini efekt toga jest da sigmoida otvrđnjava, tj. postaje sličnija funkciji praga. No, tako nešto ne želimo, jer takav model loše generalizira. Također primijetite da ovaj problem nastupa samo kada su primjeri linearno odvojivi; ako oni to nisu, onda povećanje težina \mathbf{w} uzrokuje smanjenje gubitka na točno klasificiranim primjerima, ali istodobno i povećanje gubitka na netočno klasificiranim primjera, što znači da je minimum empirijske pogreške negdje između toga i da algoritam optimizacije neće beskonačno povećavati težine.

U nastavku ćemo se baviti L_2 -regularizacijom, jer je nju lako ugraditi u algoritam gradijentnog spusta koji smo već objasnili. Pogledajmo kako. Izraz za empirijsku pogrešku proširujemo s L_2 -regularizacijskim izrazom (označen crveno):

$$E_R(\mathbf{w} | \mathcal{D}) = \sum_{i=1}^N \left(-y^{(i)} \ln h(\mathbf{x}^{(i)}) - (1 - y^{(i)}) \ln (1 - h(\mathbf{x}^{(i)})) \right) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

29

Gradijent ove funkcije po vektoru težina \mathbf{w} je:

$$\nabla_{\mathbf{w}} E_R(\mathbf{w} | \mathcal{D}) = \sum_{i=1}^N (h(\mathbf{x}^{(i)}) - y^{(i)}) \phi(\mathbf{x}^{(i)}) + \lambda \mathbf{w}$$

gdje smo iskoristili jednakost $\frac{d}{d\mathbf{w}} \mathbf{w}^T \mathbf{w} = 2\mathbf{w}$ iz matričnog diferencijalnog računa. Ažuriranje težina u standardnom (grupnom) gradijentnom spustu onda je:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \left(\sum_{i=1}^N (h(\phi(\mathbf{x}^{(i)})) - y^{(i)}) \phi(\mathbf{x}^{(i)}) + \lambda \mathbf{w} \right)$$

30

Ako regularizacijski izraz izlučimo iz sume, dobivamo sljedeći alternativni zapis pravila za ažuriranje težina:

$$\mathbf{w} \leftarrow \mathbf{w}(1 - \eta\lambda) - \eta \sum_{i=1}^N (h(\mathbf{x}^{(i)}) - y^{(i)}) \mathbf{x}^{(i)}$$

gdje pribrojnik $\mathbf{w}(1 - \eta\lambda)$ dovodi do efekta **propadanja težina** (engl. *weight decay*). Naime, primijetite da, ako bi drugi pribrojnik bio konstantan, težine bi se u svakom koraku smanjivale proporcionalno s $(1 - \eta\lambda)$. Također primijetite da promjena težina ovisi ne samo o faktoru η , već i o broju primjera N : što je N veći, to je veća promjena težina. Zbog toga stopu učenja η treba korigirati u ovisnosti o broju primjera. Za stopu učenja može se koristiti i vrijednost η/N (što bismo ionako dobili da smo funkciju pogreške definirali kao očekivanje funkcije gubitka).

Posljednja napomena ovdje jest da nikako ne smijemo zaboraviti da težinu w_0 ne regulariziramo! Naime, ako bismo regularizirali težinu w_0 , ona bi težina w_0 težila prema nuli. Međutim, budući da w_0 određuje udaljenost hiperravnine od ishodišta (ta udaljenost je $-w_0/\|\mathbf{w}\|$), to znači da bi hiperravnina uvijek morala prolaziti kroz ishodište i da ne bismo mogli dobro razdvojiti dvije klase u slučajevima kada granica ne prolazi baš kroz ishodište (a općenito granica ne mora prolaziti kroz ishodište).

Konačno, definirajmo algoritme L_2 -regularizirane logističke regresije. Budući da težinu w_0 moramo tretirati posebno, u pseudokodu ćemo oznaku \mathbf{w} koristiti za težine bez w_0 , a oznaku $\tilde{\mathbf{w}}$ za proširen vektor težina koji uključuje w_0 . Također, koristimo $\tilde{\mathbf{x}}$ za vektor proširen sa "dummy značajkom" $x_0 = 1$. Algoritam L_2 -regularizirane logističke regresije optimiziran standardnim gradijentnim spustom je:

► L2-regularizirana logistička regresija – standardni (grupni) gradijentni spust

- 1: $\tilde{\mathbf{w}} \leftarrow (0, 0, \dots, 0)$
- 2: **ponavljam** do konvergencije
- 3: $(\Delta w_0, \Delta \mathbf{w}) \leftarrow (0, 0, \dots, 0)$
- 4: **za** $i = 1, \dots, N$
- 5: $h \leftarrow \sigma(\tilde{\mathbf{w}}^T \phi(\tilde{\mathbf{x}}^{(i)}))$
- 6: $(\Delta w_0, \Delta \mathbf{w}) \leftarrow \Delta \mathbf{w} - (h - y^{(i)}) \phi(\mathbf{x})$
- 7: $\eta \leftarrow$ optimum linijskim pretraživanjem u smjeru $\Delta \tilde{\mathbf{w}}$
- 8: $w_0 \leftarrow w_0 + \eta \Delta w_0$
- 9: $\mathbf{w} \leftarrow \mathbf{w}(1 - \eta\lambda) + \eta \Delta \mathbf{w}$

Jedina suštinska izmjena u odnosu na algoritam neregularizirane logističke regresije je u retku 9, gdje kod ažuriranja težina imamo efekt propadanja težina.

Algoritam L_2 -regularizirane logističke regresije optimiziran stohastičkim gradijentnim spustom je sljedeći:

► L2-regularizirana logistička regresija (stohastički gradijentni spust)

- 1: $\tilde{\mathbf{w}} \leftarrow (0, 0, \dots, 0)$
- 2: **ponavljam** do konvergencije:
- 3: slučajno permutiraj primjere u \mathcal{D}
- 4: za $i = 1, \dots, N$
- 5: $h \leftarrow \sigma(\tilde{\mathbf{w}}^T \phi(\tilde{\mathbf{x}}^{(i)}))$
- 6: $(\Delta w_0, \Delta \mathbf{w}) \leftarrow \Delta \mathbf{w} - (h - y^{(i)}) \phi(\mathbf{x})$
- 7: $w_0 \leftarrow w_0 + \eta \Delta w_0$
- 8: $\mathbf{w} \leftarrow \mathbf{w}(1 - \eta\lambda) + \eta \Delta \mathbf{w}$

Kao i kod neregularizirane varijante, razlika između standardnog gradijentnog spusta i stohastičkog gradijentnog spusta jest što se ažuiranje težina provodi unutar petlje koja iterira po svim primjerima, što slučajno permutiramo primjere u svakoj epohi, i što ne radimo linijsko pretraživanje.

Sažetak

- Logistička regresija je **diskriminativan klasifikacijski model** s probabilističkim izlazom
- Aktivacijska funkcija je **sigmoidna (logistička) funkcija**
- Algoritam koristi **pogrešku unakrsne entropije**, čija minimizacija odgovara maksimizaciji vjerojatnosti oznaka na skupu označenih primjera
- Optimizacija se provodi **gradijentnim spustom**, a prenaučenost se može spriječiti **regularizacijom**
- Logistička regresija vrlo je dobar klasifikacijski algoritam koji nema nedostatke koje imaju klasifikacija linearnom regresijom i perceptron

Bilješke

[1] Ovo je terminološka mina: **logistička regresija**, naime, nije algoritam regresije, nego klasifikacijski algoritam. Vjerojatno se pitate tko je tu lud i zašto je to tako. Odgovor je da ova terminološka nekonzistentnost postoji samo iz perspektive strojnog učenja, gdje se logistička regresija doista dominantno koristi kao algoritam za binarnu klasifikaciju, s izlazima 0 i 1. Međutim, logistička je regresija prvenstveno statistički model: u statistici je logistička regresija tek jedan u nizu modela za analizu zavisnosti između nezavisnih varijabli i zavisne varijable. U statistici nema terminološke nekonzistentnost jer tamo logistička regresija doista *jest* regresijski model s izlazima u intervalu (0, 1), gdje se onda tek dodatnim uvođenjem praga (tipično postavljenim na 0.5) dobiva klasifikacijski algoritam. Ovaj odgovor daje više detalja: <https://stats.stackexchange.com/a/127044/93766>. Standardne reference iz primijenjene statistike koje se bave logističkom regresijom su (Harrell Jr, 2015) i (Hosmer Jr et al., 2013).

[2] Ovdje imamo još jednu terminološku minu: **poopćeni linearni modeli** (engl. *generalized linear models*) nisu isto što i **opći linearni model** (engl. *general linear model*). Poopćeni linearni modeli poopćenje su linearne regresije kod kojega je skalarni umnožak $\mathbf{w}^T \mathbf{x}$ omotan nekom aktivacijskom funkcijom f . U statističkom smislu, to su modeli kod kojega je pogreška zavisne varijable poopćena sa normalne distribucije na neku drugu distribuciju (npr. Poissonovu, Bernoullijevu), gdje $f(\mathbf{w}^T \mathbf{x})$ određuje parametar μ srednje vrijednosti te distribucije. Logistička regresija, kojom se bavimo danas, je poopćeni linearni model gdje je pogreška zavisne varijable modelirana Bernoullijevom distribucijom.

S druge strane, opći linearan model je sažet način zapisa više različitih statističkih modela temeljenih na višestrukoj linearnoj regresiji (linearna regresija, ANOVA, ANCOVA, MANOVA, MANCOVA, t-test i F-test). Mi se nećemo baviti općim linearnim modelom.

U literaturi se za poopćene linearne modele ponekad umjesto kratica GLM koristi kratica GZLM ili GLiM, kako bi ih se razlikovalo od općih linearnih modela za koje se isto koristi kratica GLM. Poopćene linearne modele kao unifikaciju različitih statističkih modela predložili su 1972. godine statističari John Nelder i Rober Wedderburn (Nelder and Wedderburn, 1972). U intervjuu 2003. godine za časopis *Statistical Science*, Nelder je priznao da ime “poopćeni linearni modeli” možda i nije bila najsjretnija opcija zbog konfuzije s općim linearnim modelima, no sada je gotovo. Standardne reference za poopćene linearne modele su (McCullagh and Nelder, 1989) i (Dobson and Barnett, 2018).

[3] Priznajem da sam ovaj prijevod izmislio.

[4] **Logistička funkcija** ili **logistička krivulja** ili **S-krivulja** predložena je prvi puta još 1838. godine

kao model rasta populacije u radovima belgijskog matematičara Pierra Françoisa Verhulsta. Naziv “logistički” osmislio je upravo Verhulst, kao kontrast od “logaritamski”. Naziv, inače, nema nikakve veze sa “logistikom” (u smislu znanja i sredstava za strateško vojno ili slično organizirano djelovanje). Logistička je krivulja zapravo vrlo značajna jer dobro opisuje mnoge prijelazne pojave o prirodi, od aktivacije neurona i magnetizacije željeza do topljenja leda i promjena znanstvenih paradigm. Lijep opis toga možete naći u knjizi Pedra Domingosa ([Domingos, 2015](#)), koju smo bili spomenuli u uvodnome predavanju, i to u čevrtome poglavlju naslovjenome “Najvažnija krivulja na svijetu”. Domingos povlači zgodnu paralelu između logističke krivulje i poznatog citata iz Hemingwayevog romana *I sunce se rađa* (*The Sun Also Rises*), u kojem vječito pijan Mike Cambell odgovara na pitanje o tome kako je bankrotirao: “Dva načina. Postupno, a onda iznenada.” (“*Two ways. Gradually and then suddenly*”). Doista, mnoge fazne tranzicije u našim životima odvijaju se upravo takvom dinamikom kakvu opisuje logistička krivulja.

- 5 Naravno, činjenica da je izlaz modela u intervalu $[0, 1]$ (ili intervalu $(0, 1)$) ne znači automatski da ga možemo tumačiti kao vjerojatnost. Možemo li takav izlaz legitimno interpretirati kao vjerojatnost ovisi o tome kako definiramo vjerojatnost i koje distribucije prepostavke su ugrađene u model. O tome više u drugom dijelu predmeta.
- 6 Već smo rekli da je vjerojatnost $P(y|\mathbf{X})$ zapravo **izglednost** (engl. *likelihood*) parametara \mathbf{w} , koji preko modela $h(\mathbf{x}^{(i)}; \mathbf{w})$ definiraju srednju vrijednost distribucije oznaka $y^{(i)}$. No, nećemo ovdje uvoditi taj pojam, već to ostavljamo za temu broj 14.
- 7 U izrazu $P(y|\mu)$, y je slučajna varijabla, a μ je parametar distribucije i on nije slučajna varijabla. Ovo može zbunjavati jer, npr., u izrazu $P(y|x)$ su i y i x slučajne varijable. Moguće su i mješovite situacije, npr. u izrazu $P(y|\mathbf{x}, \mathbf{w})$ izrazi y i \mathbf{x} su slučajne varijable, a \mathbf{w} je parametar. U tom smislu, bilo bi bolje pisati $P(y; \mu)$ odnosno $P(y|\mathbf{x}; \mathbf{w})$, kako bi se parametri jasno razdvojili od slučajnih varijabli. Nažalost, to nije praksa u literaturi iz strojnog učenja i najbolje je razviti imunost na ovu notacijsku nedosljednost. Dobra je disciplina dati si zadatak za svaki izraz pojasniti tipove svih izraza koji se u njemu javljaju, tako da za svaki izraz znate reći radi li se o slučajnoj varijabli, konstanti ili parametru.
- 8 Naziv reflektira činjenicu da pogreška doista odgovara **unakrsnoj entropiji** iz teorije informacije, pri čemu je prava distribucija distribucija oznaka y , a procijenjena distribucija ona koju daje model $h(\mathbf{x})$. Naime, unakrsna entropija (za diskretne vjerojatnosne distribucije) definirana je kao:

$$H(p, q) = - \sum p(x) \ln q(x)$$

te ona odgovara prosječnom broju bitova potrebih za kodiranje događaja čija je vjerojatnost nastupanja određena distribucijom $p(x)$, a shema kodiranja optimirana je za distribuciju $q(x)$.

- 9 Naime, vrijedi:

$$h(\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \phi(\mathbf{x}))} \quad 1 - h(\mathbf{x}) = \frac{1}{1 + \exp(\mathbf{w}^T \phi(\mathbf{x}))}$$

pa onda:

$$\begin{aligned} L(y, h(\mathbf{x})) &= -y \ln h(\mathbf{x}) - (1 - y) \ln(1 - h(\mathbf{x})) \\ &= y \ln(1 + \exp(-\mathbf{w}^T \phi(\mathbf{x}))) + (1 - y) \ln(1 + \exp(\mathbf{w}^T \phi(\mathbf{x}))) \end{aligned}$$

Sada želimo da lijevi izraz bude jednak nuli za $y = -1$, a desni izraz da bude jednak nuli za $y = +1$. Zatim uočavamo da se lijevi i desni izraz razlikuju jedino po predznaku. To nam onda omogućava da ih stopimo u jedan izraz, ovako:

$$\ln(1 + \exp(-\mathbf{w}^T \phi(\mathbf{x})y))$$

i to je upravo ono što smo željeli: gubitak iskazan kao funkcija od $\mathbf{w}^T \phi(\mathbf{x})y$.

- 10 Dakle, formalno gledano, **gubitak unakrsne entropije** (engl. *cross-entropy loss*) i **logistički gubitak** (engl. *logistic loss*) nije jedno te isto. Gubitak unakrsne entropije je funkcija dviju varijabli, $L(y, h(\mathbf{x}))$, definirana za $y = \{0, 1\}$, dok je logistički gubitak funkcija jedne varijable, $L(\mathbf{w}^T \phi(\mathbf{x})y)$,

definirana za $y = \{-1, +1\}$. Međutim, obje ove funkcije jesu funkcije gubitka algoritma logističke regresije i razlika je samo pitanje konvencije. Uglavnom je iz konteksta uvijek jasno na koju se varijantu misli, i čini se da se ljudi u literaturi oko toga ne uzbuduju previše.

- [11] Kod nekih algoritama ima čak smisla krenuti najprije od optimizacijskog postupka, a tek onda iz njega izvesti funkciju gubitka i funkciju pogreške. Tako ćemo raditi kod stroja potpornih vektora (SVM), gdje ćemo najprije definirati otpimizacijski problem maksimalne margine kao problem kvadratnog programiranja, a tek onda iz toga izvesti funkciju pogreške i funkciju gubitka SVM-a.
- [12] Gradijent funkcije pogreške unakrsne entropije je

$$\nabla E(\mathbf{w}|\mathcal{D}) = \sum_{i=1}^N (h(\mathbf{x}^{(i)}) - y^{(i)}) \phi(\mathbf{x}^{(i)})$$

U ovom izrazu težine \mathbf{w} su u nelinearnom odnosu sa E , i to dovodi do toga da rješenje jednadžbe

$$\nabla E(\mathbf{w}|\mathcal{D}) = \sum_{i=1}^N (h(\mathbf{x}^{(i)}) - y^{(i)}) \phi(\mathbf{x}^{(i)}) = 0$$

ne postoji zatvorenoj formi. Prisjetimo se da kod linearne regresije nemamo taj problem: tamo rješenje postoji u zatvorenoj formi, no to je upravo zato jer ne postoji nelinearnost (nema aktivacijske funkcije). Čini se da je postojanje rješenja optimizacijskoj problema poopćenog linearног modela u zatvorenoj formi više iznimka nego pravilo, v. <https://stats.stackexchange.com/a/37640/93766>.

- [13] Kod logističke regresije, glavna alternativa gradijentnom spustu i iz njega izvedenih varijanti jest optimizacija drugog reda, konkretno **Newtonov postupak** i njegova aproksimativna varijanta, **kvazi-Newtonov postupak**, a posebice računalno učinkovita implementacije kvazi-Newtonovog postupka pod nazivom **LM-BFSG** (engl. *limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm*). O svemu tome pričat ćemo (doduše bez previše detalja) idući put.
- [14] Standardna referenca za **konveksnu optimizaciju** je (Boyd et al., 2004). Općenito, konveksna se optimizacija bavi problemima koji se mogu formalizirati kao:

$$\begin{aligned} & \text{minimiziraj } f_0(\mathbf{x}) \\ & \text{tako da } f_i(\mathbf{x}) \leq b_i, \quad i = 1, \dots, m \end{aligned}$$

gdje su $f_0, \dots, f_m : \mathbb{R}^n \rightarrow \mathbb{R}$ konveksne funkcije. Funkcija $f(\mathbf{x})$ je **funkcija cilja** (ili **kriterijska funkcija**), a $f_i(\mathbf{x}) \leq b_i$ su ograničenja jednakosti ili ograničenja nejednakosti. Ako imamo ograničenja, onda govorimo o **optimizaciji s ograničenjima** (engl. *constrained optimization*). Više o tome u predavanju o SVM-u.

- [15] Po nekima, područje strojnog učenja previše je, gotovo religiozno, privrženo konveksnoj optimizaciji, premda su mnogi stvarni problemi u strojnem učenju nekonveksi optimizacijski problemi. To je sasvim sigurno slučaj u **dubokom učenju**, kod kojega više slojeva nelinearnih aktivacijskih funkcija dovode do toga da je funkcija pogreško visoko nelinearna i nekonveksna. Za pregled nekonveksnih pristupa optimizaciji u strojnem učenju, pogledajte (Jain and Kar, 2017).
- [16] **Konveksnost funkcije** se, osim izravno na temelju definicije, što je u nekim slučajevima teško, može dokazati na nekoliko načina. Npr., dvostruko diferencijabilna funkcija je konveksna u zadanom intervalu ako i samo ako je njezina druga derivacija u tom intervalu nenegativna. U praksi je, međutim, konveksnost lakše dokazati tako da se funkcija napiše pomoću funkcija za koje je već dokazano da su konveksne, korištenjem operacija za koje je dokazano da čuvaju konveksnost (engl. *convexity preserving operations*). Vidjeti, npr., poglavlje 3 u (Boyd et al., 2004). U poglavljju 9 iste knjige opisuje se gradijentni spust u kontekstu optimizacije konveksnih funkcija.

- [17] Nemojte brkati **globalnu konvergenciju** i **globalni minimum**. Globalna konvergencija znači samo da se pretraga zaustavlja nakon konačnog broja koraka, ali ne mora se nužno zaustaviti u točki globalnog minimuma.

- [18] Malo više o **linijskom pretraživanju** (ili **pretraživanju po pravcu**) možete pročitati na https://en.wikipedia.org/wiki/Line_search. Linijsko pretraživanje može biti egzaktno (ekplicitna minimizacija funkcije $F(\mathbf{x} + \eta \Delta \mathbf{x})$) ili približno. Algoritam za približno pretraživanje je **linijsko pretraživanje**

s povratkom (engl. *backtracking line search*). Riječ je o iterativnom algoritmu koji u svakoj iteraciji smanjuje korak na temelju lokalnog gradijenta funkcije koja se minimizira, na temelju tzv. Armijo-Goldsteinovog uvjeta (pravila). Više na https://en.wikipedia.org/wiki/Backtracking_line_search. Detaljni opis linijskog pretraživanja možete naći u poglavlju 3 knjige (Nocedal and Wright, 2006).

- [19] Preuzeto sa <http://fourier.eng.hmc.edu/e176/lectures/NumericalMethods/node18.html>.
- [20] Pregled optimizacijskih postupaka temeljenih na gradijentnom spustu s prilagodivom stopom učenja možete naći u (Ruder, 2016) i (Goodfellow et al., 2016) (poglavlje 8).
- [21] Ovakvi se problemi u teoriji optimizacije nazivaju **problemi konačne sume** (engl. *finite sum problems*). V. http://www.stat.cmu.edu/~ryantibs/convexopt-F18/scribes/Lecture_9.pdf
- [22] Više o **stohastičkom gradijentnom spustu** možete pročitati u poglavlju 7 iz (Deisenroth et al., 2020). Matematički rigorozniji tretman možete naći u poglavlju 13 u (Sra et al., 2012) i u (Bottou and Bousquet, 2008).
- [23] Situacija u kojoj ćemo sigurno koristiti stohastički a ne standardni gradijentni spust jest ona kada nemamo na raspolaganju odmah sve primjere, nego učimo postepeno, kako dolaze novi primjeri. Takav način učenja nazivamo **online učenje**. Kod online učenja postoji **tok podataka** i trebamo dinamički trenirati klasifikator kako dolaze novi podatci. Pregled metoda za online učenje možete naći u (Gama, 2012) i (Benczúr et al., 2018).
- [24] U dubokom učenju površina funkcije pogreške redovito je vrlo složena i ima mnogo lokalnih minimuma. Također, redovito raspolažemo povećom količinom označenih primjera. To dvoje u kombinaciji govori u prilog korištenja gradijentnog spusta s minigrupama umjesto standardnog gradijentnog spusta. Optimizacijski postupci u dubokom strojnom učenju uglavnom su varijante gradijentnog spusta s adaptivnom (prilagodivom) stopom učenja. Izvrstan pregled raznih varijanti algoritma gradijentnog spusta, od kojih se mnoge standardno koriste u dubokom učenju, možete naći u (Ruder, 2016).
- [25] Moderne implementacija algoritma logističke regresije tipično koriste algoritme brzog **stohastičkog inkrementalnog gradijentnog spusta**, npr. SAG (engl. *stochastic average gradient*) (Schmidt et al., 2017), kod koje se konvergencija ubrzava pamćenjem gradijenta iz prethodne iteracije, i naprednija varijanta tog algoritma pod nazivom SAGA (Defazio et al., 2014). Pregled metoda inkrementalanog gradijentnog spusta možete naći u http://niaohe.ise.illinois.edu/ie598/IE598_BigDataOpt_lecturenotes_fall2016.pdf (lekciјa 23).
- [26] Vidi <https://stats.stackexchange.com/q/321592/93766>.
- [27] Empirijski je utvrđeno da stohastički gradijentni spust sa slučajnom permutacijom redoslijeda primjera u svakoj epohi radi bolje nego stohastički gradijentni spust s fiksnim poretkom primjera (Bottou, 2009). Međutim, razlozi za to dugo nisu bili jasni. Tek su nedavno napravljene teorijske analize konvergencije i ograda pogreške stohastičkog gradijentnog spusta koje daju teorijsko opravdanje za učinkovitost slučajne permutacije kod stohastičkog gradijentnog spusta (Ying et al., 2017; Gürbüzbalaban et al., 2019; Safran and Shamir, 2020).
- [28] Ovaj fenomen (da linearne odvojivoj problemi mogu dovesti do parametara čije vrijednosti teže k beskonačnosti, tj. da minimizator pogreške ne postoji) u statistici je poznat pod nazivom **Hauck-Donnerov efekt**. Taj je efekt relevantan kada se želi napraviti statističko zaključivanje o intervalima pouzdanosti parametara ili veličini učinka. Više ovdje: <https://stats.stackexchange.com/q/254124/93766>.
- [29] O algoritmu optimizacije za L_1 -regulariziranu logističku regresiju možete pročitati u (Lee et al., 2006) i (Koh et al., 2007). Već spomenuti optimizacijski algoritam SAGA (Defazio et al., 2014), koji se

danasm koristi za logističku regresiju, podržava L_2 - i L_1 -regularizaciju.

- 30 Ne baš. Jednakost iz matričnog diferencijalnog računa na koju se ovdje pozivamo glasi:

$$\frac{d}{dx} \mathbf{x}^T \mathbf{A} \mathbf{x} = \mathbf{x}^T (\mathbf{A} + \mathbf{A}^T)$$

pa uz $\mathbf{A} = \mathbf{I}$ imamo $\frac{d}{dw} \mathbf{w}^T \mathbf{w} = 2\mathbf{w}^T$. Stoga bi gradijent regularizacijskog izraza trebao biti $\lambda \mathbf{w}^T$ a ne $\lambda \mathbf{w}$. Međutim, pišemo $\lambda \mathbf{w}$, jer je i prvi pribrojnik zapravo već transponiran (naime, izravna primjena matričnog diferencijalnog računa na gradijent funkcije logističkog gubitka zapravo daje $(h(\mathbf{x}) - y)\phi(\mathbf{x})^T$). Na koncu, mi uvijek baratamo vektor stupcima, pa i gradijent treba biti vektor stupac, makar to značilo naknadno transponiranje nakon deriviranja.

- 31 Da je udaljenost hiperravnine od ishodiška jednaka $-w_0/\|\mathbf{w}\|$ pokazali smo u bilješci 3 u predavanju 5.

Literatura

- A. A. Benczúr, L. Kocsis, and R. Pálovics. Online machine learning in big data streams. *arXiv preprint arXiv:1802.05872*, 2018.
- L. Bottou. Curiously fast convergence of some stochastic gradient descent algorithms. In *Proceedings of the symposium on learning and data science, Paris*, 2009.
- L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168, 2008.
- S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- A. Defazio, F. Bach, and S. Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.
- M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.
- A. J. Dobson and A. G. Barnett. *An introduction to generalized linear models*. CRC press, 2018.
- P. Domingos. *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Basic Books, 2015.
- J. Gama. A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, 1(1):45–55, 2012.
- I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- M. Gürbüzbala, A. Ozdaglar, and P. Parrilo. Why random reshuffling beats stochastic gradient descent. *Mathematical Programming*, pages 1–36, 2019.
- F. E. Harrell Jr. *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer, 2015.
- D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

- P. Jain and P. Kar. Non-convex optimization for machine learning. *arXiv preprint arXiv:1712.07897*, 2017.
- K. Koh, S.-J. Kim, and S. Boyd. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine learning research*, 8(Jul):1519–1555, 2007.
- S.-I. Lee, H. Lee, P. Abbeel, and A. Y. Ng. Efficient ℓ^1 regularized logistic regression. In *Aaai*, volume 6, pages 401–408, 2006.
- P. McCullagh and J. A. Nelder. *Generalized Linear Models*. Chapman and Hall, 1989.
- J. A. Nelder and R. W. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384, 1972.
- J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
- S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- I. Safran and O. Shamir. How good is sgd with random shuffling? In *Conference on Learning Theory*, pages 3250–3284. PMLR, 2020.
- M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.
- S. Sra, S. Nowozin, and S. J. Wright. *Optimization for machine learning*. MIT Press, 2012.
- B. Ying, K. Yuan, S. Vlaski, and A. H. Sayed. On the performance of random reshuffling in stochastic learning. In *2017 Information Theory and Applications Workshop (ITA)*, pages 1–5. IEEE, 2017.

7. Logistička regresija II

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v2.1

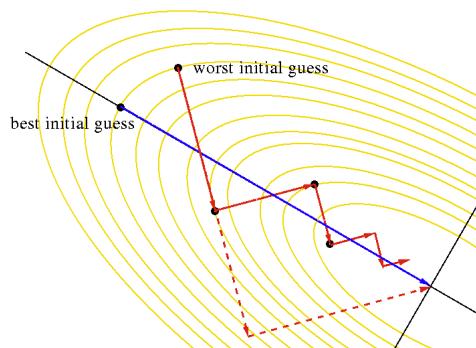
Prošli put upoznali smo se s algoritmom **logističke regresije**. Definirali smo model i zatim izveli funkciju pogreške unakrsne entropije kao negativan logaritam vjerojatnosti oznaka u skupu za učenje. Ustanovili smo da minimizacija te pogreške nema rješenje u zatvorenoj formi, pa smo se okrenuli iterativnim postupcima. Razmotrili smo najjednostavniji takav postupak, algoritam **gradijentnog spusta**, te smo ga primijenili na logističku regresiju, i to u standardnoj (grupnoj) i stohastičkoj izvedbi. Na koncu smo pričali o **regularizaciji**, i to konkretno L_2 -regularizaciji, koju smo vrlo jednostavno ugradili u optimizacijski postupak.

Danas ćemo još malo pričati o logističkoj regresiji. Prvo, razmotrit ćemo neke alternative gradijentnom spustu, koje su učinkovitije (čitaj: brže) od gradijentnog spusta. Drugo, razmotrit ćemo proširenje binarne logističke regresije na **višeklasnu logističku regresiju**. Treće, malo ćemo se osvrnuti na dosada naučeno i pogledati što svi ti modeli imaju zajedničko i kako ih možemo poopćiti. Konačno, pričat ćemo o **adaptivnim baznim funkcijama** kao načinu da iz podataka naučimo funkciju preslikavanja iz ulaznog prostora u prostor značajki, umjesto da tu funkciju definiramo ručno.

Za razliku od prethodnih predavanja, ovo će se predavanje uglavnom zadržati na razini upoznavanja, jer za više od toga nemamo vremena. Cilj mi je dati vam dovoljno informacija da znate samostalno gdje kopati dalje, ako će vam to trebati.

1 Alternative gradijentnom spustu

Prošli smo put ustanovili da gradijentni spust treba kombinirati s linijskim pretraživanjem, jer u suprotnom nemamo zajamčenu **globalnu konvergenciju**. To znači da se, ovisno o početnoj točki pretraživanja, može dogoditi da gradijentni spust ne konvergira već da divergira (efektivno se umjesto spuštanja počinje podizati). Linijsko pretraživanje sprječava da se to dogodi. Međutim, linijsko pretraživanje može rezultirati krivudavim (“cik-cak”) spustom. Prisjetimo se slike koju smo bili komentirali prošli put:



Plava trajektorija odgovara najboljem scenariju, a crvena najgorem scenariju za gradijentni spust s linijskim pretraživanjem. Scenarij ovisi o odabiru početne točke pretraživanja. Vidimo da se može dogoditi da spust bude brlo krivudav, što znači da će optimizacija konzumirati puno iteracija. Očito, problem nastaje zbog toga što smjer spuštanja nije toliko dobar koliko bi

mogao biti. Zamislite da se spuštate u neku jamu iz početne točke za crvenu trajektoriju. Teško da biste se spuštali baš po označenom pravcu. Sila teža bi vas vukla da se spuštate strmijim smjerom (tj. pod manjim kutom u odnosu na plavu liniju). Da je to tako zaključujemo na temelju zakriviljenosti izokontura (odnosno površine niz koju se spuštamo). Drugim riječima, zakriviljenosti površine daje nam, pored gradijenta, dodatnu informaciju o tome gdje se bi se mogao nalaziti minimum (barem kada je riječ o konveksnim funkcijama).

Gornja opažanja vrijede za standardni (grupni) gradijent. Kod stohastičkog gradijentnog spusta tipično ne koristimo linijsko pretraživanje. No, tamo će spust ionako dosta krivudati, budući da se svaki korak spusta radi na temelju gradijenta izračunatog za jedan pojedinačni primjer. U nastavku se fokusiramo na nestohastički, dakle grupni gradijentni spust.

Na temelju gornjeg razmatranja možemo zaključiti da bismo grupni gradijentni spust mogli unaprijediti, ako bismo u obzir uzeli ne samo nagib (gradijent) nego i zakriviljenost (promjenu gradijenta, tj. drugu derivaciju) funkcije pogreške. Takvi optimizacijski postupci zovu se **optimizacija drugog reda**, za razliku od optimizacijskih postupaka prvog reda, kao što je to gradijentni spust. Osnovni postupak drugog reda jest **Newtonov postupak**.

1.1 Newtonov postupak

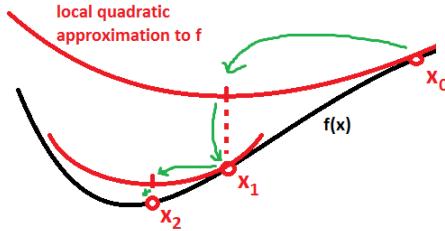
Razmotrimo minimizaciju funkcije $f(\mathbf{x})$. Ažuriranje parametara kod gradijentnog spusta bilo je:

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \nabla f(\mathbf{x})$$

Ako uvedemo indeks za interacije, onda to možemo napisati kao jednadžbu:

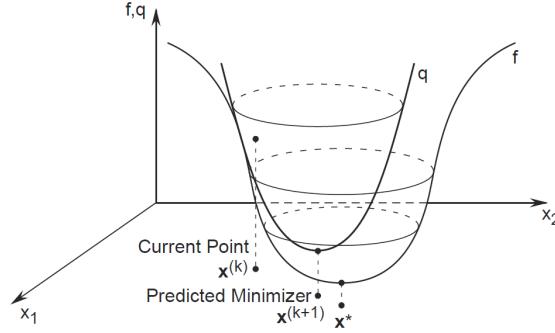
$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t)$$

Ideja kod Newtonovog algoritma jest da u točki \mathbf{x}_t (trenutačni minimum) napravimo kvadratnu aproksimaciju funkcije $f(\mathbf{x})$, i da onda skočimo u minimizator te kvadratne aproksimacije (taj minimizator je analitički poznat). Ako je f funkcija jedne varijable, to izgleda ovako:



Crna krivulja je funkcija $f(x)$ koju minimiziramo. Krećemo iz točke x_0 . U toj točki radimo kvadratnu aproksimaciju funkcije f , čime dobivamo parabolu koja je tangencijalna s funkcijom f u točki x_0 . Zatim se pretraga pomiče u točku koja minimizira kvadratnu aproksimaciju od f . Na slici je to točka x_1 . U toj točki opet radimo kvadratnu aproksimaciju funkcije f te se pomičemo u točku koja minimizira tu aproksimaciju. Postupak se tako iterativno ponavlja dok ažuriranje nije dovoljno malo.

Postupak funkcioniра identično za funkciju više varijabli, tj. u višedimenzijskome prostoru parametara. U dvodimenzijskome prostoru to izgleda ovako:



Dakle, ideja je da korak napravimo točno tako da sletimo u minimum kvadratne aproksimacije funkcije. Ovo će raditi dobro ako je $f(\mathbf{x})$ **konveksna funkcija**, a to kod nas (kod logističke regresije) jest slučaj.

Pozabavimo se sada tehnikalijama. Trebamo napraviti kvadratnu aproksimaciju funkcije u nekoj točki. Kako to napraviti? Prisjetimo se, iz matematičke analize, da svaku diferencijabilnu funkciju $f(x)$ možemo u nekoj zadanoj točki a napisati u obliku reda potencija. Preciznije, svaku diferencijabilnu funkciju možemo razviti u **Taylorov red** u okolini točke a , ovako:

$$\begin{aligned} f(x) &= f(a) + \frac{f'(a)}{1!}(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \frac{f'''(a)}{3!}(x - a)^3 + \dots \\ &= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x - a)^n \end{aligned}$$

Budući da nama treba samo kvadratna aproksimacija, uzeti ćemo samo prva tri člana Taylorovog reda. To je onda **Taylorova aproksimacija drugog reda**: [2]

$$f(x) \approx f(a) + \frac{f'(a)}{1!}(x - a) + \frac{f''(a)}{2!}(x - a)^2$$

Primijetite da je ovo sada aproksimacija; jednakost između $f(x)$ na lijevoj strani i reda na desnoj strani vrijedi samo kada je red beskonačan.

Naša funkcije pogreške, $E(\mathbf{w}|\mathcal{D})$, je funkcije više varijabli (vektora \mathbf{w}). Za funkciju više varijabli $f(\mathbf{x})$, kvadratni razvoj oko točke \mathbf{x}_t je:

$$f(\mathbf{x}) \approx f_{\text{quad}}(\mathbf{x}) = f(\mathbf{x}_t) + \nabla f(\mathbf{x}_t)^T (\mathbf{x} - \mathbf{x}_t) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_t)^T \mathbf{H}_f(\mathbf{x}_t) (\mathbf{x} - \mathbf{x}_t)$$

gdje je $\mathbf{H}_f(\mathbf{x}_t)$ **Hesseova matrica** (engl. *Hessian matrix*) funkcije $f(\mathbf{x})$ u točki \mathbf{x}_t . Hesseova matrica je kvadratna matrica dimenzija $n \times n$ parcijalnih derivacija drugog reda funkcije $f : \mathbb{R}^n \rightarrow \mathbb{R}$, odnosno funkcije koja n -dimenzijske vektore preslikava u skalare. Hesseova matrica definirana je ovako: [3]

$$\mathbf{H}_f = \nabla \nabla f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

to jest, element Hesseove matrice definiran je s: [4]

$$(\mathbf{H}_f)_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

Primijetite da je Hesseova matrica **simetrična matrica**, budući da redoslijed parcijalnih derivacija ne utječe na rezultat (komutativnost).

Iz ovog raspisa kvadratne funkcije f_{quad} sada se može izvesti **minimum** te funkcije, i to je upravo korak koji želimo napraviti pri spustu (izvod preskačemo). Ažuriranje parametara onda je:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \mathbf{H}_f(\mathbf{x}_t)^{-1} \nabla f(\mathbf{x}_t)$$

Ako je Hesseova matrica egzaktno izračunata (tj. nije aproksimacija), onda možemo staviti $\eta = 1$, jer ćemo tako napraviti korak točno u minimum kvadratne aproksimacije.

Vidimo da će za Newtonov optimizacijski postupak biti potrebno računati **inverz Hesseove matrice** funkcije $f(\mathbf{x}_t)$ u točki \mathbf{x}_t . Općenito, matrica $\mathbf{H}_f(\mathbf{x}_t)$ je **pozitivno semidefinitna** ($\mathbf{x}^T \mathbf{H}_f(\mathbf{x}_t) \mathbf{x} \geq 0$ za svaki ne-nul vektor \mathbf{x}) ako i samo ako je funkcija $f(\mathbf{x})$ **konveksna**. Međutim, da bi imala inverz, matrica $\mathbf{H}_f(\mathbf{x}_t)$ trebala bi biti **pozitivno definitna** ($\mathbf{x}^T \mathbf{H}_f(\mathbf{x}_t) \mathbf{x} > 0$ za svaki ne-nul vektor \mathbf{x}). Ako je matrica $\mathbf{H}_f(\mathbf{x}_t)$ pozitivno semidefinitna, ali nije pozitivno definitna, onda nema inverz i tada ne možemo primijeniti Newtonov postupak.

1.2 Newtonov postupak za logističku regresiju

Primijenimo sada Newtonov postupak na logističku regresiju. Pravilo za ažuriranje težina je:

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}_E^{-1}(\mathbf{w}) \nabla_{\mathbf{w}} E(\mathbf{w} | \mathcal{D}) \quad (\text{uz } \eta = 1)$$

Lako se može izvesti da je Hesseova matrica za pogrešku unakrsne entropije E sljedeća:

$$\mathbf{H}_E = \boldsymbol{\Phi}^T \mathbf{S} \boldsymbol{\Phi}$$

gdje je $\mathbf{S} = \text{diag}(h(\mathbf{x}^{(i)})(1 - h(\mathbf{x}^{(i)})))$, tj. dijagonalna matrica koja na dijagonali ima derivacije prvog reda logističkog izlaza $h(\mathbf{x})$ za svaki od N primjera iz skupa za učenje. Za svaki slučaj, provjerimo kompatibilnost matrica koje ovdje množimo: $((m+1) \times N) \cdot (N \times N) \cdot (N \times (m+1))$. Sve je u redu: dimenzija matrice \mathbf{H}_E je $(m+1) \times (m+1)$, gdje je m broj značajki u prostoru značajki (dakle, nakon preslikavanja).

Vratimo se nakratko na problem izračunavanja inverza Hesseove matrice $\mathbf{H}_E(\mathbf{w})$. Kako smo već rekli, $\mathbf{H}_E(\mathbf{w})$ je pozitivno semidefinitna ako i samo ako je funkcija E konveksna. Pogrešna unakrsna entropija je konveksna, pa će dakle matrica $\mathbf{H}_E(\mathbf{w})$ za logističku regresiju uvijek (u svakoj točki \mathbf{w}) biti pozitivno semidefinitna. Ali to i dalje ne znači da uvijek ima inverz. Srećom, može se pokazati da, budući da vrijedi rastav $\mathbf{H}_E = \boldsymbol{\Phi}^T \mathbf{S} \boldsymbol{\Phi}$, to $\mathbf{H}_E(\mathbf{w})$ mora biti pozitivno definitna. Drugim riječima, Hesseova matrica unakrsne entropije imati će inverz u svakoj točki \mathbf{w} , pa ćemo dakle za optimizaciju parametara logističke regresije uvijek moći primijeniti Newtonovu metodu. Ipak, zbog multikolinearnosti, $\mathbf{H}_E(\mathbf{w})$ može biti loše kondicionirana i rješenje će tada biti nestabilno, što znači da trebamo ili odabrati linearno nezavisani podskup značajki ili provesti regularizaciju (vidjet ćemo uskoro kako).

Ako bismo sada uvrstili $\boldsymbol{\Phi}^T \mathbf{S} \boldsymbol{\Phi}$ umjesto \mathbf{H}_E u gornje pravilo za ažuriranje težina i malo presložili izraz, dobili bismo **algoritam najmanjih kvadrata s iterativnim ažuriranjem težina** (engl. *iteratively reweighted least squares, IRLS*). Nećemo u detalje; dovoljno je da znate da se taj algoritam koristi za brzu optimizaciju za logističku regresiju, i da je to zapravo primjena Newtonovog postupka, koji je optimizacija drugog reda.

1.3 Kvazi-Newtonov postupak

Problem s Newtonovim postupkom (a tako i s IRLS-om) jest što izračun Hesseove matrice (a pogotovo njezina inverza) može biti skup (pogotovo ako je m , broj dimenzija prostora značajki, velik). Primjetite da u svakom koraku optimizacije moramo izračunati Hesseovu matricu i njezin inverz. Lako može biti da se više isplati raditi gradijentni spust, pa makar taj krivudao!

Alternativa jest da se, umjesto egzaktne Hesseove matrice, izračunava njezina aproksimacija. To rade tzv. **kvazi-Newtonovi postupci**. Ti postupci u svakom koraku spusta aproksimiraju Hesseovu matricu (ili njezin inverz) pomoću vektora gradijenta (iz trenutačnog i prethodnog

koraka). Najpoznatija takva metoda jest algoritam **BFSG**. Opet, nećemo u detalje, trebate samo znati da to postoji.

Drugi problem je što, čak i ako aproksimiramo Hesseovu matricu, njezino pohranjivanje u memoriju može biti problematično, jer je ona dimenzija $(m+1) \times (m+1)$. U tom slučaju može se napraviti “komprimiranje“ matrice metodom **aproksimacije matricom nižeg ranga**. Algoritam koji tako funkcionira jest **ograničen BFSG** (engl. *limited BFSG*, **L-BFSG**). Opet, dovoljno je samo da znate da takav algoritam postoji.

1.4 Newtonov postupak s regularizacijom

Znamo već koje su prednosti regularizacije, a prošli smo puta pričali i o dodatnom značaju regularizacije kod logističke regresije (kod optimizacije parametara za linearne odvojive probleme imamo $\|\mathbf{w}\| \rightarrow \infty$ i logistička se regresija lako prenauči). Proširenje Newtonovog postupka na regulariziranu pogrešku unakrsne entropije je jednostavno. Pogledajmo **L_2 -regularizaciju**. Dodavanje L_2 -regularizacijskog izraza na gradijent smo već imali:

$$\nabla E_R(\mathbf{w}|\mathcal{D}) = \nabla E(\mathbf{w}|\mathcal{D}) + \lambda \mathbf{w}$$

Slično imamo i za Hesseovu matricu:

$$\mathbf{H}_R = \mathbf{H} + \lambda \mathbf{I}$$

gdje je $\lambda \mathbf{I}$ dijagonalna matrica s regularizacijskim faktorom λ na dijagonali (osim na gornjoj lijevoj poziciji, jer težinu w_0 ne regulariziramo).

Vidimo, dakle, da se L_2 -regularizacija lako ugradi u postupak gradijentnog spusta ili u Newtonov postupak, kao što se uostalom i lako ugradila u optimizacijski postupak najmanjih kvadrata kod linearne regresije. Sve u svemu, L_2 -regularizacija je jedna pitoma zvjerka.

Što je s L_1 -regularizacijom? S njome se do sada uopće nismo bavili, a nećemo ni sada. Čisto informativno, trebamo znati da se tu stvari očekivano komplificiraju zbog toga što L_1 -norma nije diferencijabilna, pa ne možemo računati gradijent. Umjesto toga računamo **podgradijent** (engl. *subgradient*). Zatim tipično koristimo **koordinatni spust** (engl. *coordinate descent*), gdje optimiramo varijablu po varijablu (dimenziju po dimenziju), ili koristimo **proksimalne ili projekcijske** optimizacijske postupke. Za sada samo trebate znati da je optimizacija L_1 -regularizirane logističke regresije moguća, da za to postoje algoritmi i da su implementirani u standardnim alatima.

2 Višeklasna logistička regresija

Prošli put bavili smo se **binarnom logističkom regresijom**: klasificirali smo u klase $y = 1$ i $y = 0$. Kako bismo dobili vjerojatnosti, za aktivacijsku funkciju koristili smo sigmoidu:

$$h(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T \phi(\mathbf{x})) = \frac{1}{1 + \exp(-\mathbf{w}^T \phi(\mathbf{x}))} = P(y = 1|\mathbf{x})$$

No, što ako imamo više od dvije klase, tj. $K > 2$? Mogli bismo primijeniti dekompozicijsku shemu OVO ili OVR, ali problem je što se vjerojatnosti za pojedinačne klase ne bi zbrajale u 1. Također, u statističkom smislu procjene za parametre pojedinačnih modela bile manje pouzdane nego procjene za parametre modela koji u obzir uzima sve klase. Umjesto toga, bolje je raditi **multinomijalnu logističku regresiju** (MNR, nekad MLR), koju također zovemo i **klasifikator maksimalne entropije** (engl. *maximum entropy classifier*).

2.1 Model

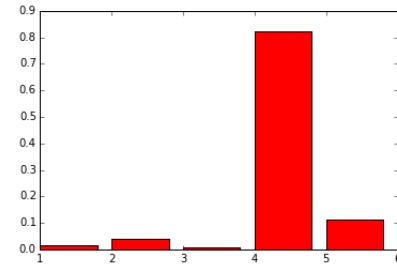
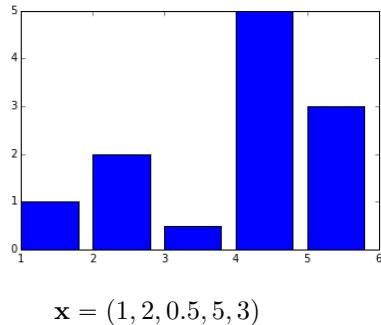
Ideja je zapravo vrlo jednostavna: koristiti zaseban vektor težina \mathbf{w}_k za svaku od K klase, ali onda skalarni umnožak $\mathbf{w}_k^T \mathbf{x}$ propustiti kroz prikladnu aktivacijsku funkciju koja će se pobrinuti da se vjerojatnosti svih klasa zbrajaju na ukupno 1. Funkcija koja radi baš to naziva se **funkcija softmax**. Za neki ulazni primjer \mathbf{x} , funkcija softmax uzima vrijednosti $\mathbf{w}_k^T \mathbf{x}$ za svaku od K klase, dakle K -dimenzijski vektor, te ih preslikava u K -dimenzijski vektor čije se komponente zbrajaju u 1. Formalno, softmax : $\mathbb{R}^n \rightarrow \mathbb{R}^n$, gdje je k -ta komponenta izlaznog vektora jednaka:

$$\text{softmax}_k(x_1, \dots, x_n) = \frac{\exp(x_k)}{\sum_j \exp(x_j)}$$

Funkcija softmax radi dvije stvari: **normalizira** sve vrijednosti tako da u zbroju budu 1, ali također **pojačava** veće vrijednosti i **smanjuje** manje vrijednosti. Funkcija se zove softmax jer odgovara funkciji max, ali je "meka" (u smislu da je, za razliku od funkcije max, neprekidna i diferencijabilna). Pogledajmo jedan primjer.

14

► PRIMJER



15

Model h multinomijalne logističke regresije definirat ćemo kao skup modela $\{h_k\}_k$, gdje je svaki model h_k zadužen za k -tu od ukupno K klase. Svaki model h_k definirat ćemo tako da daje vjerojatnost da primjer \mathbf{x} pripada klasi k , pomoću funkcije softmax:

$$h_k(\mathbf{x}; \mathbf{W}) = \frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}))}{\sum_j \exp(\mathbf{w}_j^T \phi(\mathbf{x}))} = P(y = k | \mathbf{x}, \mathbf{W})$$

gdje je $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K)$ matrica sastavljena od K vektora težina \mathbf{w}_k . Primijetite da model h_k za klasu k kroz funkciju softmax uzima u obzir izlaze svih ostalih $(k - 1)$ modela za preostale klase.

Time je definiran model. Sada ćemo izvesti funkciju pogreške.

2.2 Funkcija pogreške

Kod binarne logističke regresije funkciju pogreške izveli smo krenuvši od definicije negativnog logaritma vjerojatnosti oznaka. Oznake su bile binarne, $y \in \{0, 1\}$, odnosno to su bile **Bernoullijeve varijable**. Međutim, kako izlaz kod višeklasne regresije može poprimiti više od dvije vrijednosti ($K > 2$), prelazimo na **kategoričku varijablu**, koja se također naziva i **multinomialna**, ili, možda bolje, **multinoullijska** varijabla. Takvu varijablu prikazujemo kao **vektor indikatorskih (binarnih) varijabli**.

16

$$\mathbf{y} = (y_1, y_2, \dots, y_K)^T$$

gdje je $y_k = 1$, ako je ishod varijable k , a inače $y_k = 0$. Npr., $\mathbf{y} = (0, 0, 1, 0)$ označava da je multinomijalna varijabla poprimila treće stanje od četiri mogućih stanja. Pritom vrijedi $\sum_k y_k = 1$ (ishodi su međusobno isključivi i potpuni). Označimo vjerojatnost $P(y_k = 1)$ sa μ_k .

Sada ćemo definirati **distribuciju** te varijable. Prisjetimo se, distribucija Bernoullijeve varijable, koja ima samo dvije vrijednosti, $y = 0$ i $y = 1$, definirana je preko parametra μ na sljedeći način:

$$P(y|\mu) = \mu^y(1-\mu)^{1-y}$$

Ovo možemo poopćiti na $K \geq 2$ vrijednosti na sljedeći način. Najprije, treba nam K parametara, pa ćemo definirati vektor parametara:

$$\boldsymbol{\mu} = (\mu_1, \dots, \mu_K)$$

pri čemu za parametre μ_k vrijedi $\sum_k \mu_k = 1$ i $\mu_k \geq 0$, budući da predstavljaju vjerojatnosti.

Sada, po analogiji s Bernoullijevom distribucijom, distribuciju za kategoričku varijablu možemo definirati kao:

$$P(\mathbf{y}|\boldsymbol{\mu}) = \prod_{k=1}^K \mu_k^{y_k}$$

Primijetimo: kao i kod binarne logističke regresije, **vjerojatnost da primjer x pripada klasi k** je upravo ono što nam daje model:

$$h_k(\mathbf{x}; \mathbf{W}) = \mu_k = P(y = k|\mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_k^T \boldsymbol{\phi}(\mathbf{x}))}{\sum_j \exp(\mathbf{w}_j^T \boldsymbol{\phi}(\mathbf{x}))}$$

Sada konačno možemo napisati logaritam vjerojatnosti oznaka iz \mathcal{D} kao:

$$\begin{aligned} \ln P(\mathbf{y}|\mathbf{X}) &= \ln \prod_{i=1}^N P(\mathbf{y}^i|\mathbf{x}) \\ &= \ln \prod_{i=1}^N \prod_{k=1}^K \mu_k^{y_k^i} = \ln \prod_{i=1}^N \prod_{k=1}^K h_k(\mathbf{x}^i; \mathbf{W})^{y_k^i} = \sum_{i=1}^N \sum_{k=1}^K y_k^i \ln h_k(\mathbf{x}^i; \mathbf{W}) \end{aligned}$$

Funkcija pogreške koju želimo minimizirati je **negativan logaritam vjerojatnosti oznaka**:

$$E(\mathbf{W}|\mathcal{D}) = - \sum_{i=1}^N \sum_{k=1}^K y_k^i \ln h_k(\mathbf{x}^i; \mathbf{W})$$

Vidimo da smo došli do **poopćenja pogreške unakrsne entropije** na K klase. Također, iz ovoga možemo iščitati da je funkcija gubitka jednaka:

$$L(\mathbf{y}, h_k(\mathbf{x})) = - \sum_{k=1}^K y_k^i \ln h_k(\mathbf{x}^i; \mathbf{W})$$

Logika je ista kao i kod binarne logističke regresije: ako je oznaka y_k^i nekog primjera i za klasu k jednaka 1, onda želimo da predikcija modela (izlaz softmaxa) bude visoka vjerojatnost blizu 1, jer će tada $\log h(\mathbf{x}) \approx 0$ i gubitak će biti nula. Inače, ako model za primjer čija je oznaka jednaka 1 daje vrijednost blizu 0, onda će logaritam biti velik negativan broj, njegova negacija bit će velik broj, pa će gubitak biti velik.

2.3 Optimizacija

Kao i kod binarne logističke regresije, ne možemo minimizirati $E(\mathbf{W}|\mathcal{D})$ u zatvorenoj formi, već trebamo raditi **iterativnu optimizaciju**. Za gradijentni spust, možemo se pokazati (doduše malo je nespretno) da je **gradijent funkcije pogreške** jednak:

$$\nabla_{\mathbf{w}_k} E(\mathbf{W}|\mathcal{D}) = \sum_{i=1}^N (h_k(\mathbf{x}^{(i)}; \mathbf{W}) - y_k^{(i)}) \phi(\mathbf{x}^{(i)})$$

Ovo je gradijent po težinama posebno za klasu k . Ideja je da možemo ažurirati težine za svaku klasu posebno. Iz ovoga možemo direktno izvesti stohastički gradijentni spust (ažuriramo težine za svaki primjer, za svaku klasu). Možemo izvesti i standardni (grupni) gradijentni spust, gdje akumuliramo ažuriranja za sve primjere, za svaku klasu posebno. Možemo također izvesti i Newtonov postupak (Hesseovu matricu), ali to ćemo preskočiti.

2.4 “Online” učenje

Možda ste primijetili da je gradijent gubitka koju smo dobili za multinomijalnu logističku regresiju zapravo isti kao i onaj za binarnu logističku regresiju: **“pogreška modela puta vektor primjera”**. Kada to koristimo za učenje gradijentnim spustom, težine ažuriramo ovako:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta(h(\mathbf{x}^{(i)}; \mathbf{w}) - y^{(i)}) \phi(\mathbf{x}^{(i)})$$

Ovo pravilo smo već (u kontekstu perceptronu, ako se sjećate) rekli da se zove **Widrow-Hoffovo pravilo**, a drugi naziv je **least-mean-squares (LMS) algoritam** (ne brkati s least-squares, premda je očito povezano s time).

Algoritam LMS, odnosno ovakvo učenje gdje na temelju **stohastičkog gradijentnog spusta** minimiziramo pogrešku ažurirajući težine modela na gore definirani način, omogućava nam da radimo **online učenje**. O tome smo nešto rekli prošli puta. Podsjetimo se: online učenje je način učenja kod kojega ne moraju svi primjeri za učenje biti odmah dostupni, nego oni mogu dolaziti jedan po jedan, a težine modela će se ažuirati kako dolaze novi primjeri.

Vratimo se nakratko nekoliko tjedana unazad, na **linearnu regresiju**. Kada smo radili linearnu regresiju, pričali smo zapravo samo o **grupnoj optimizaciji**, koja se ostvaruje izračunom **pseudoinverza matrice dizajna**. Međutim, i tamo smo već mogli napraviti stohastičko (tj. online) ažuriranje težina. Naime, umjesto da analitički nalazimo minimum funkcije pogreške, možemo izračunati gradijent funkcije pogreške, pa primijeniti gradijentni spust. (Mi to tada nismo napravili, vjerojatno zato što smo bili zaslijepljeni čistom elegancijom rješenja u zatvorenoj formi). Pa, napravimo to sada. Funkcija kvadratne pogreške linearne regresije jest:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{2} \sum_{i=1}^N (\mathbf{w}^T \phi(\mathbf{x}^{(i)}) - y^{(i)})^2$$

Gradijent (za jedan primjer $\mathbf{x}^{(i)}$) jest:

$$\frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}|\mathcal{D}) = (\mathbf{w}^T \phi(\mathbf{x}^{(i)}) - y^{(i)}) \phi(\mathbf{x}^{(i)})$$

Dakle, pravilo za ažuriranje težina jest:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta(\mathbf{w}^T \phi(\mathbf{x}^{(i)}) - y^{(i)}) \phi(\mathbf{x}^{(i)})$$

što je opet pravilo LMS! Izgleda, dakle, da tu postoji neka zajedničkost – sva tri algoritma regresije za online učenje (stohastički gradijentni spust) koriste isto pravilo (LMS). Kako to? Pa, to je zato što svi ovi modeli koje smo radili – linearna regresija, logistička regresija i multinomijalna logistička regresija – pripadaju porodici **poopćenih linearnih modela**.

Sada je vrijeme da zaokružimo priču i damo jedan **unificirani pogled** na ta tri algoritma.

3 Poopćeni linearni modeli i eksponencijalna familija

Najprije, prisjetimo se s prošloga predavanja da su poopćeni modeli linearni modeli koji oko skalaranog umnoška vektora težina i vektora značajki imaju omotanu **aktivacijsku funkciju** f . Dakle:

$$h(\mathbf{x}; \mathbf{w}) = f(\mathbf{w}^T \phi(\mathbf{x}))$$

Pogledajmo tri poopćena linearna modela kojima smo se bavili, i pogledajmo za svaki od njih četiri stvari: (1) kako je definiran **model**, (2) kojoj **vjerojatnosnoj distribuciji** odgovara njihov izlaz, (3) kako je definiran **gubitak** i (4) kako glasi **gradijent gubitka**, koji se koristi za gradijentni spust.

Prvo, pogledajmo algoritam **linearne regresije**:

$$\begin{aligned} h(\mathbf{x}; \mathbf{w}) &= f(\mathbf{w}^T \phi(\mathbf{x})) = \mathbf{w}^T \phi(\mathbf{x}) \\ P(y|\mathbf{x}, \mathbf{w}) &= \mathcal{N}(\mu, \sigma^2) = \mathcal{N}(h(\mathbf{x}), \sigma^2) \\ L(y, h(\mathbf{x})) &= (h(\mathbf{x}) - y)^2 \\ \nabla_{\mathbf{w}} L(y, h(\mathbf{x})) &= (h(\mathbf{x}) - y) \phi(\mathbf{x}) \end{aligned}$$

Za grupno učenje koristimo postupak najmanjih kvadrata (pseudoinverz), a za online učenje koristimo pravilo LMS.

Pogledajmo algoritam **logističke regresije**:

$$\begin{aligned} h(\mathbf{x}; \mathbf{w}) &= f(\mathbf{w}^T \phi(\mathbf{x})) = \frac{1}{1 + \exp(-\mathbf{w}^T \phi(\mathbf{x}))} = P(y = 1|\mathbf{x}, \mathbf{w}) \\ P(y|\mathbf{x}, \mathbf{w}) &= \mu^y (1 - \mu)^{(1-y)} = h(\mathbf{x})^y (1 - h(\mathbf{x}))^{(1-y)} \\ L(y, h(\mathbf{x})) &= -y \ln h(\mathbf{x}) - (1 - y) \ln (1 - h(\mathbf{x})) \\ \nabla_{\mathbf{w}} L(y, h(\mathbf{x})) &= (h(\mathbf{x}) - y) \phi(\mathbf{x}) \end{aligned}$$

Za grupno učenje koristimo gradijentni spust, Newtonov postupak (IRLS) ili kvazi-Newtonov postupak (BFSG, L-BFSG). Za online učenje koristimo pravilo LMS.

Konačno, pogledajmo **multinomijalnu logističku regresiju**:

$$\begin{aligned} h_k(\mathbf{x}; \mathbf{W}) &= \text{softmax}(\mathbf{w}^T \phi(\mathbf{x})) = \frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}))}{\sum_j \exp(\mathbf{w}_j^T \phi(\mathbf{x}))} = P(y = k|\mathbf{x}, \mathbf{w}) \\ P(\mathbf{y}|\mathbf{x}, \mathbf{w}) &= \prod_{k=1}^K \mu_k^{y_k} = \prod_{k=1}^K h_k(\mathbf{x})^{y_k} \\ L(\mathbf{y}, h_k(\mathbf{x})) &= - \sum_{k=1}^K y_k \ln h_k(\mathbf{x}; \mathbf{W}) \\ \nabla_{\mathbf{w}_k} L(y_k, h_k(\mathbf{x})) &= (h_k(\mathbf{x}) - y_k) \phi(\mathbf{x}) \end{aligned}$$

Za učenje modela koristimo iste (mutatis mutandis) optimizacijske postupke kao i za logističku regresiju.

Uočimo zajedničkosti. Kod sva tri algoritma funkciju gubitka izveli smo iz izraza za negativan logaritam vjerojatnosti oznaka primjera iz skupa primjera. Pritom smo za linearnu, binarnu logističku i multinomijalnu logističku regresiju koristili normalnu, Bernoullijevu odnosno multinoullijevu razdiobu. Nadalje, za sva tri algoritma izveli smo identično pravilo (LMS) za online ažuriranje težina.

Pitanje je: kako to da smo dobili uvijek isto pravilo ažuiranja težina? Također, koja je veza između logističke funkcije i Bernoullijeve varijable, te između funkcije softmax i multinoullijeve razdiobe? Čini se da je to povezano, jer smo u oba slučaja dobili pogrešku unakrsne entropije. Odgovor leži u svojstvima distribucija koje smo koristili za modeliranje oznaka $y^{(i)}$.

3.1 Eksponencijalna familija

Distribucije koje smo do sada susreli (Gaussova, Bernoullijeva, multinoullijeva), ali i neke druge koje se često koriste u strojnom učenju (binomna, multinomijalna, Studentova t-distribucija, uniformna, beta-distribucija, gamma-distribucija, Dirichletova) pripadaju **eksponencijalnoj familiji** (engl. *exponential family*). Što je eksponencijalna familija? Eksponencijalna familija je jedna široka grupa distribucija koje se mogu napisati u ovakovom obliku:

$$p(\mathbf{x}|\boldsymbol{\theta}) = h(\mathbf{x}) \exp(\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}) - A(\boldsymbol{\theta}))$$

Eksponencijalna familija distribucija ima mnoga svojstva koja su važna za strojno učenje, ali uglavnom više za probabilističke pristupe, pa sada ovdje nećemo uopće dalje o tome.

Ono što nam je ovdje zanimljivo uočiti jest da je eksponencijalna familija ključna za poopcene linearne modele. Konkretno, za distribucije koje pripadaju eksponencijalnoj familiji (uključivo Gaussova, Bernoullijeva i multinoullijeva), postoji veza između distribucije i njezine (moguće nelinearne) aktivacijske funkcije f . Ta se funkcija u tom kontekstu naziva **funkcija sredine** (engl. *mean function*), zato što definira parametar μ distribucije, a to je parametar srednje vrijednosti distribucije. Aktivacijska funkcija f dakle definira μ kao funkciju od $\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$. Pogađate, za Gaussovu distribuciju aktivacijska funkcija je funkcija identiteta, budući da $\mu = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$. Za Bernoullijevu distribuciju to je logistička funkcija, a za multinoullijevu distribuciju to je funkcija softmax.

Nakon ove inspirativne teme, pogledajmo još jednu ne manje inspirativnu stvar... 18

4 Adaptivne bazne funkcije

Kod poopcenih linearnih modela (i za regresiju i za klasifikaciju) imali smo mogućnost primjere preslikati u prostor značajki pomoću **funkcije preslikavanja značajki**:

$$\begin{aligned}\boldsymbol{\phi} : \mathbb{R}^n &\rightarrow \mathbb{R}^{m+1} \\ \boldsymbol{\phi}(\mathbf{x}) &= (1, \phi_1(\mathbf{x}), \dots, \phi_m(\mathbf{x}))\end{aligned}$$

gdje je $\{\phi_1, \phi_2, \dots, \phi_m\}$ skup od m **baznih funkcija** (nelinearnih funkcija ulaznih varijabli): $\phi_j : \mathbb{R}^n \rightarrow \mathbb{R}$. Npr., polinomijalno preslikavanje za $n = 2$ i $d = 2$:

$$\boldsymbol{\phi}(\mathbf{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$$

Takvo preslikavanje smo onda vrlo lako ugradili u bilo koji poopceni linearni model:

$$h(\mathbf{x}; \mathbf{w}) = f(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})) = f\left(\sum_{j=0}^m w_j \phi_j(\mathbf{x})\right)$$

gdje je f neka odabrana aktivacijska funkcija (odnosno funkcija sredine, da odmah iskoristimo netom naučen pojам).

Premda mi to nismo isprobavali, bazne funkcije ϕ_j ne moraju nužno biti potencije ili faktori ulaznih značajki, već to mogu biti raznorazne funkcije. Jedna zanimljiva mogućnost su funkcije koje mjere sličnost primjera s nekim prototipnim primjerima u ulaznome prostoru. To se onda zove **jezgreni stroj**, i o tome ćemo pričati za dva tjedna.

Kako god, ograničavajuće je to što su ovo **fiksne bazne funkcije**: njihov broj i oblik je unaprijed određen. To je problem zato što mi u većini slučajeva ne znamo unaprijed koje su bazne funkcije dobre za naš problem. Drugim riječima, općenito ne znamo koje preslikavanje u prostor značajki će naš problem u tom prostoru učiniti linearno odvojivim.

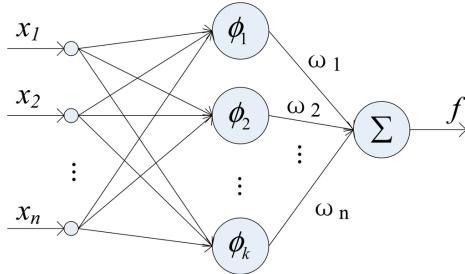
Problem možemo riješiti tako da bazne funkcije **prilagodimo** našim podatcima (primjerima za učenje). Tu imamo dvije mogućnosti. Prva mogućnost, koju koriste spomenuti jezgreni strojevi, jest da odaberemo neke primjere iz skupa za učenje kao prototipe, a onda bazne funkcije mjere sličnost ulaznog primjera s tim prototipima. Tu se ukupan broj baznih funkcija prilagođava podatcima. Druga mogućnost je da ipak koristimo fiksni broj baznih funkcija, ali da dozvolimo da se svaka od njih prilagodi podatcima. Pogledajmo to malo detaljnije.

Ideja prilagodivih baznih funkcija jest da ih definiramo do na neke parametre, koje onda možemo prilagoditi na podatcima. Dakle, imat ćemo **parametrizirane bazne funkcije**. Zvuči li to poznato? Dakako. To je upravo kako učimo modele u strojnem učenju: definiramo funkciju do na neke parametre, a parametre odredimo optimizacijom empirijske pogreške na skupu za učenje. No, pogledajmo prvo kako bismo parametrizirali bazne funkcije. Jedna mogućnost jest da kažemo da je svaka bazna funkcija jedan malen poopćeni linearni model za sebe! Dakle, imat ćemo poopćeni linearni model i unutar njega imat ćemo poopćene linearne modele kao njegove bazne funkcije:

$$h(\mathbf{x}; \mathbf{w}) = f\left(\sum_{j=0}^m w_j^{(2)} f\left(\underbrace{\sum_{i=0}^n w_{ji}^{(1)} x_i}_{=\phi_j(\mathbf{x})} \right) \right) = f(\mathbf{w}^{(2)\top} f(\mathbf{W}^{(1)} \mathbf{x}))$$

Primijetite da svaka bazna funkcija našeg modela treba imati svoj vektor težina, stoga težine w_{ji} u unutarnjoj sumi imaju dva indeksa: j je indeks bazne funkcije ϕ_j , a i je indeks težine u vektoru težina bazne funkcije ϕ_j . Sa superskriptom ⁽¹⁾ odnosno ⁽²⁾ označili smo koje se težine prve koriste u izračunu predikcije modela, a koje se koriste druge. Izraz na desnoj strani je samo matrični zapis modela, gdje smo se uspjeli rješiti suma i težine smo objedinili u vektor težina i matricu težina. Uzmite si malo vremena da vidite da je taj zapis ekvivalentan zapisu sa sumama.

Sada kada smo ovo metabolizirali, vrijeme je za veliko otkrivenje. Kakav je zapravo ovaj model? Za svaku baznu funkciju ϕ_j imamo težine iz matrice $\mathbf{W}^{(1)}$, koje množimo sa svim značajkama ulaznog vektora i zbrajamo. Zatim te vrijednosti opet množimo sa težinama $\mathbf{w}^{(2)}$ i zbrajamo. Dobili smo nešto što je većini vas već poznato: **neuronsku mrežu!**



Ova naša mreža je dvoslojna, međutim ništa nas ne sprječava da idemo dublje: da su bazne funkcije i same kombinacije drugih baznih funkcija, a one opet kombinacija trećih baznih funkcija, itd.

Očito, neuronske mreže su složeniji model od poopćenih linearnih modela. To, naravno, ima i svoju cijenu: složeniji postupak optimizacije (zbog **nekonveksnosti funkcije pogreške**, budući da gubitak u izlaznom sloju – koji i dalje može biti kvadratni gubitak ili gubitak unakrsne entropije – sada ima vrlo složenu ovisnost o težinama prethodnih slojeva) te veća mogućnost **pretreniranosti modela**. Naravno, za sve to postoje razni prijedlozi rješenja, posebice u okviru

danasa popularne paradigme dubokog učenja. Nećemo ići dalje, međutim, budući da se ova tema poprilično detaljno pokriva u drugim predmetima.

Nama je ovdje samo bitno da uočite poveznicu: neuronska mreža je proširenje poopćenog linearog modela kod kojega su bazne funkcije prilagodive, odnosno funkcija preslikavanja značajki također se uči iz podataka.

Sažetak

- **Newtonov postupak** je optimizacija drugog reda koja konvergira brže od gradijentnog spusta, a temelji se na izračunu **Hesseove matrice**. Varijanta za logističku regresiju zove se **IRLS**
- Izračun Hesseove matrice je vremenski i prostorno skup, pa možemo koristiti kvazi-Newtonov postupak, kao što je **L-BFGS**
- **Multinomijalna logistička regresija** je poopćenje logističke regresije na više od dvije klase, sa **softmax** funkcijom kao aktivacijskom funkcijom
- Zajedničko **poopćenim linearnim modelima** jest da su njihovi izlazi varijable iz **eksponentijalne familije** distribucija
- Umjesto fiksnih, možemo koristiti parametrizirane **adaptivne bazne funkcije**, što nas dovodi do **neuronskih mreža**

Bilješke

[1] **Newtonov optimizacijski postupak** naziva se nekad i **Newton-Raphsonov postupak**, premda to nije skroz točno. Newton-Raphsonov postupak je iterativan postupak za nalaženje nultočaka funkcije. Newtonov optimizacijski postupak koristi Newton-Raphsonov postupak za nalaženje nultočke prve derivacije funkcije koja se optimizira. Dakle, Newtonov postupak optimizacije koristi Newton-Raphsonov postupak. Inače, Newton-Raphsonov postupak osmislio je Isaac Newton 1685. godine, a Joseph Raphson, također engleski matematičar, predložio je sličnu metodu 1690. godine u ponešto simplificiranoj varijanti, i s nekim referencama na Newtonov rad. Postupak, dakle, nisu razvili zajednički, a čini se i da su slabo komunicirali (to možda ne čudi, jer je Newton navodno bio teška i agresivna osoba). O povijesti razvoja Newton-Raphsonovog postupka možete pročitati u ([Ypma, 1995](#)). Usput, osim što je poznat po Newton-Raphsonovom postupku, Joseph Raphson poznat je i po tome što je prvi uveo naziv "panteizam" za filozofski pravac koji je utemeljio prosvjetiteljski filozof Baruch Spinoza (pojednostavljen: Bog=priroda), a dvjesto godina kasnije tematizirao vrlo utjecajan američki filozof i književnik Ralph Waldo Emerson.

[2] Ovdje se prirodno nameće pitanje: zašto radimo aproksimaciju samo drugog reda? Zašto ne trećeg ili višeg reda? Ne bi li to optimizaciju učinilo još učinkovitijom? Odgovor je: ne bi. Kvadratna aproksimacija je dovoljno dobra, u smislu da nije računalno presložena, a konvergira brže od optimizacije prvog reda. V. ovdje: <https://stats.stackexchange.com/q/320082/93766>.

[3] U literaturi na hrvatskome jeziku ponekad se, čini mi se pogrešno, koristi naziv "Hessova matrica". Naime, **Hesseovu matricu** uveo je njemački matematičar Ludwig Otto Hesse (a ne "Hess"!), rođen u Königsbergu u tadašnjoj Prusiji, a današnjem Kalinjingradu, i to nekoliko godina nakon smrti Immanuela Kanta, koji je također živio u Königsbergu, kojega, navodno, nikada nije napustio. Ferovcima će možda biti zanimljivo znati da je Hesse bio doktorskim mentorom Gustavu Kirchoffu, koji je osmislio poznate Kirchoffove zakone, i to dok je još bio student. Konačno, napomenimo, za svaki slučaj, da Ludwig Hessse i Hesseova matrica nemaju nikakve veze s književnikom Hermannom Hesseom, rođenom nakon smrti Ludwiga Hessea (osim ako ne postoji neka veza između druge derivacije vektorske funkcije i spiritualnog samootkrivenja koje Hesse opisuje u Siddharthi).

[4] **Hesseovu matricu** nemojte brkati s **Jakobiјevom matricom** ("Jakobijanom"). Za razliku od Hesseove matrice, koja je matrica parcijalnih derivacija **drugog reda** funkcije $f : \mathbb{R}^n \rightarrow \mathbb{R}$, tj. vektorske funkcije

koja vraća skalar, Jakobijska matrica je matrica parcijalnih derivacija **prvog reda** vektorske funkcije koja vraća vektor, $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

gdje je f_i i -ta komponenta izlaza funkcije f . Drugim riječima, element Jakobijske matrice je:

$$\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}$$

Zanimljivo, Jakobijanu je uveo Carl Gustav Jacob Jacobi, koji je bio doktorski mentor upravo Ludwigu Hesseu. Eto kako je svijet malen.

Premda se i Hesseova matrica **H** i Jakobijska matrica **J** redovito se koriste u strojnem učenju, mi ćemo na ovom predmetu trebati samo Hesseovu (Jakobijska će vam trebati za izračun gradijentata kod neuronskih mreža u dubokom učenju).

- 5 Izvod za minimum od f_{quad} možete naći u ([Murphy, 2012](#)) (dio 8.3.3).
- 6 Dokaz pogledajte ovdje: <https://math.stackexchange.com/q/2083629/273854>.
- 7 Vidi ([Bishop, 2006](#)) (dio 4.3.3) ili ([Murphy, 2012](#)) (dio 8.3.1).
- 8 Dokaz pogledajte ovdje: <https://math.stackexchange.com/q/3189729/273854>
- 9 Izvod **algoritma najmanjih kvadrata s iterativnim ažuriranjem težina (IRLS)** možete naći u ([Bishop, 2006](#)) (dio 4.3.3) i ([Murphy, 2012](#)) (dio 8.3.4). Naziv algoritma reflektira činjenicu da pravilo za ažuriranje težina (nismo ga napisali; v. citiranu literaturu) nalikuje jednadžbi za izračun težina Moore-Penroseovim pseudoinverzom, kakvu smo imali kod optimizacijskog postupka najmanjih kvadrata kod linearne regresije, uz tu razliku da je izračun potrebno obavljati iterativno.
- 10 Algoritam **BFSG** nazvan je prema matematičarima Broydenu, Fletcheru, Goldfarbu i Shannou, koji su ga nezavisno osmislili 1970. godine. Čini se da poredak imena ne odgovara kronološkom slijedu objavljivanja članaka, pa je vjerojatno dogovorno uzet abecedan poredak. Kako god, lijepo je da su spomenuti svi izumitelji. Nažalost, rijetki su te sreće; v. **Stiglerov zakon eponimije**: https://en.wikipedia.org/wiki/Stigler%27s_law_of_eponymy.
- 11 Metode **aproksimacije matricom nižeg ranga** (engl. *low-rank matrix approximations*) intenzivno se koriste u nekim dijelovima strojnog učenja, pogotovo u tehnikama za smanjenje dimenzionalnosti i u jezgrenim metodama. Začudo, nama u ovom predmetu te metode neće trebati. Ako želite pročitati nešto o tim metodama, predlažem krenuti od ([Kishore Kumar and Schneider, 2017](#)).
- 12 Algoritam L-BFSG osmislio je američki matematičar i računarac Jorge Nocedal. Osnovno o algoritmu L-BFSG možete pročitati na https://en.wikipedia.org/wiki/Limited-memory_BFGS. Detaljniji opis možete naći 9. poglavljju Nocedalove knjige (Nocedal and Wright, 2006). Ovo je također dobar opis: <https://aria42.com/blog/2014/12/understanding-lbfgs>, unatoč snipetima Java koda. Na dotičnoj stranici možete naći i fotografiju na kojoj kvartet BFSG piye pivu. Nocedal je algoritam L-BFSG izvorno napisao u Fortranu; ako želite vidjeti kako taj kod izgleda, pogledajte ga ovdje: <http://users.iems.northwestern.edu/~nocedal/lbfgs.html>
- 13 **Podgradijent** (engl. *subgradient*) je iterativan optimizacijski postupak za minimizaciju nediferencijabilne konveksne funkcije. U strojnem učenju tipično se koristi za minimizaciju L_1 -regulariziranih pogrešaka ili pogrešaka izvedenih iz nediferencijabilne funkcije gubitka. Detaljnije u ([Boyd et al., 2003](#)).
- 14 Zapravo, funkcija softmax prije "mekoj" verziji funkcije argmax nego max. Naime, funkciju argmax možemo interpretirati kao funkciju koja na izlazu daje tzv. **one-hot vektor** – vektor binarnih indikacijskih varijabli koje su sve jednake nula osim jedne, koja je jednaka jedinici. Ta jedna jedinica odgovara indeksu ulaznog vektora na kojem se nalazi maksimalna vrijednost. Npr., $\text{argmax}(1, 4, 2, 3) =$

$(0, 1, 0, 0)$. Funkcija softmax onda se može shvatiti kao “meka” varijanta funkcije argmax jer na izlazu daje “zaglađeni” one-hot vektor. Npr., $\text{softmax}(1, 4, 3, 2) = (0.03, 0.64, 0.24, 0.09)$. Više ovdje: <https://medium.com/@u39kun/is-the-term-softmax-driving-you-nuts-ee232ab4f6bd>.

- [15] Ovdje se možda da naslutiti da je aktivacijska funkcija softmax zapravo poopćenje sigmoidne funkcije na više od dvije klase. Degenerirani slučaj funkcije softmax je slučaj kada $K = 2$, što će biti isto kao i izlaz sigmoidne funkcije, ali sa dvodimenziskim vektorom kao izlazom: $(\sigma(\mathbf{w}^T \mathbf{x}), 1 - \sigma(\mathbf{w}^T \mathbf{x}))$. V. <https://stats.stackexchange.com/a/254071/93766>.
- [16] Ne stižem ovdje napisati propisnu bilješku o nazivu **multinoullijseva varijabla**. Kao privremenu kompenzaciju, nudim ovaj tekst:
<https://geekyisawesome.blogspot.com/2016/12/bernoulli-vs-binomial-vs-multinoulli-vs.html>. Naziv “multinoullijseva varijabla” (kao stopljenicu od “Bernoullijseva” i “multinomijalna”) predložio je Murphy u ([Murphy, 2012](#)), ali nisam siguran da se uhvatila. Najsigurnije je koristiti “kategorička varijabla”. Naziv “multinomijalna varijabla” je u ovom kontekstu, zapravo, pogrešan.
- [17] Za izvod, pogledati <https://math.stackexchange.com/q/2852620/273854>
- [18] Inverz funkcije sredine naziva se **vezna funkcija** (engl. *link function*). Za funkciju identiteta to je funkcija identiteta, a za logističku/softmax funkciju to je funkcija **logit**. Kolokvijalno, “logiti” su vrijednosti koje ulaze u softmax funkciju (dakle vrijednost $\mathbf{w}^T \phi(\mathbf{x})$ kod logističke regresije, odnosno vrijednosti $\mathbf{w}_k^T \phi(\mathbf{x})$ za svaku klasu kod multinomijalne regresije).

Literatura

- C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
- S. Boyd, L. Xiao, and A. Mutapcic. Subgradient methods. *lecture notes of EE392o, Stanford University, Autumn Quarter, 2004:2004–2005*, 2003.
- N. Kishore Kumar and J. Schneider. Literature survey on low rank approximation of matrices. *Linear and Multilinear Algebra*, 65(11):2212–2244, 2017.
- K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.
- J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
- T. J. Ypma. Historical development of the Newton–Raphson method. *SIAM review*, 37(4):531–551, 1995.

8. Stroj potpornih vektora

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v3.1

Prošla tri tjedna bavili smo se **poopćenim linearnim modelima** za regresiju i klasifikaciju. Danas krećemo u nešto skroz novo i drugačije: govorit ćemo o algoritmu koji se zove **stroj potpornih vektora** (engl. *support vector machine*, **SVM**). Radi se o vrlo učinkovitom klasifikacijskom i regresijskom algoritmu koji je, od devedesetih godina kada je osmišljen, dugo vremena dominirao na sceni strojnog učenja i pobudio velik interes za tzv. **jezgrene metode**. I danas, gotovo trideset godina kasnije, to je i dalje jedan od omiljenih algoritama u teoriji i praksi.

Današnja je tema poprilično sofisticirana i može se ispričati na nekoliko načina. Mi ćemo ići “klasičnim” putem: formalizirat ćemo najprije problem **maksimalne margine**, i onda doći do tzv. **problema kvadratnog programiranja**. U tom trenutku preći ćemo iz tzv. **primarne** formulacije problema u **dualnu** formulaciju. Sve ovo oslanja se na teoriju iz **konveksne optimizacije**, u koju ćemo danas malo dublje ulaziti, i, nadam se, izroniti. Sve u svemu, danas će nam biti dosta intenzivno i zanimljivo.

1 Problem maksimalne margine

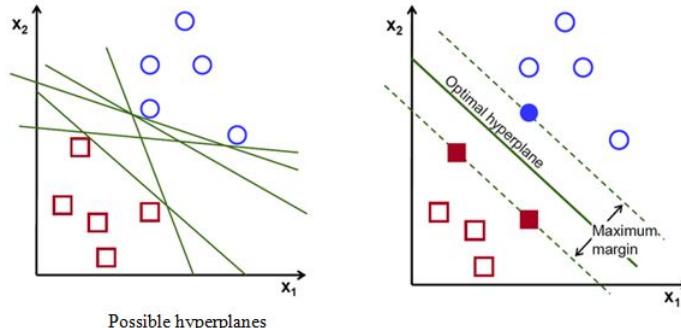
Krenimo od najjednostavnije stvari: **modela**. Model SVM-a običan je **linearan model**:

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

Primijetite da ovdje nemamo (nelinarnu) aktivacijsku funkciju f . No primijetite da, kao i do sada, možemo upotrijebiti trik s funkcijom preslikavanja ϕ kako bismo ostvarili nelinearnost u prostoru primjera.

U nastavku ćemo pretpostaviti da su primjeri **linearno odvojivi**, bilo u ulaznom prostoru ili u prostoru značajki (nakon preslikavanja), pa nećemo pisati funkciju ϕ , radi jednostavnosti. Ovo je samo naša radna pretpostavka; ona nije realna, i kasnije ćemo je relaksirati. No, za sada, bit će lakše objasniti ideju SVM-a ako pretpostavimo da su primjeri linearno odvojivi.

SVM se temelji na ideji **maksimalne margine**. Objasnjimo na primjeru o čemu se radi:

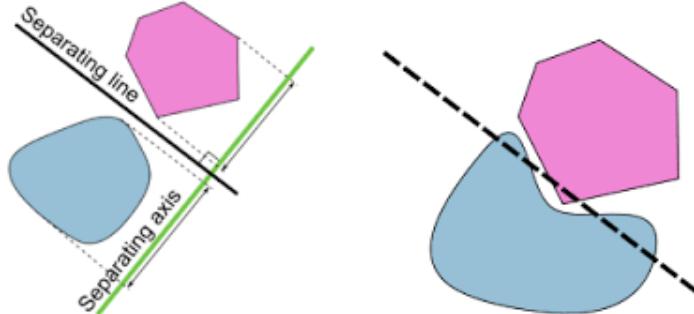


Na lijevoj je slici prikazan dvodimenzionalni ulazni prostor sa $N = 10$ primjera iz dvije klase. Kao što smo se dogovorili, primjeri su linearno odvojivi. Zeleni pravci su moguće granice između

klasa, tj. hipoteze koje daju savršenu klasifikaciju na skupu \mathcal{D} . Koliko takvih hipoteza postoji? Ako je ulazni prostor $\mathcal{X} = \mathbb{R}^2$, onda takvih hipoteza ima beskonačno mnogo hipoteza. Drugim riječima, beskonačno je mnogo pravaca koje možemo ucrtati između primjera iz ovih dviju klasa. Sada se možemo pitati: u nedostatku bilo kakve druge informacije, koji od tih beskonačno mnogo pravaca bismo trebali preferirati, ako želimo da model dobro generalizira? Kako pozicionirati pravac između primjera iz dvije klase, a da naš model najbolje radi na neviđenim primjerima? Ili, rečeno drugačije: gdje postaviti pravac, a da budemo što manje pristrani? Intuitivno znamo da bi najpametnije bilo da pravac postavimo točno u sredinu, tako da je maksimalno udaljen i od primjera jedne klase i od primjera druge klase, budući da bismo u suprotnom bili pristrani prema jednoj od dviju klasa.

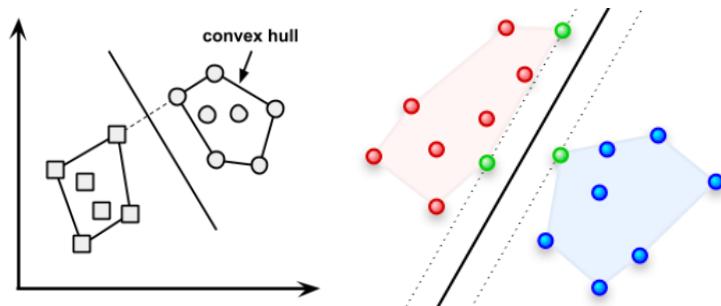
Upravo to je ideja SVM-a: postaviti hiperravninu tako da bude najviše udaljena od primjera iz dviju klasa. Udaljenost od hiperravnine do najbližeg primjera sa svake strane zvat ćemo **margina**. Mi dakle želimo takvu hiperravninu koja **maksimizira marginu**. Intuitivno nam je jasno da će hiperravnina koja maksimizira marginu dobro generalizirati. No to se može pokazati i formalno, u okviru **teorije računalnog učenja** (engl. *computational learning theory*, *COLT*), što mi, međutim, nećemo raditi.

Jasno je da, ako su primjeri linearno odvojivi, onda mora postoji razdvajajuća hiperravnina. No, korisno je to razmotriti sa stajališta geometrije. U geometriji, koncept linearne odvojivosti lijepo odgovara konceptu **konveksnosti skupova**. Konkretno, prema **teoremu o odvajanju hiperravninom** (engl. *hyperplane separation theorem*) za dva **disjunktna i konveksna podskupa** u \mathbb{R}^n postoji hiperravnina koja ih razdvaja. Ako takva hiperravnina ne postoji, onda to znači da barem jedan od ta dva podskupa nije konveksan.



Ljeva slika prikazuje dva konveksna skupa. Između njih je moguće provući pravac koji ih razdvaja, tj. koji ne prolazi kroz niti jedan od ta dva skupa. Na desnoj slici jedan skup nije konveksan, pa takav razdvajajući pravac ne mora nužno postojati (može, ali ne mora). U našem slučaju, ovi konveksni skupovi odgovaraju tzv. **konveksnim ljsuskama** (engl. *convex hull*) primjera iz dviju klasa. Konveksna ljska je najmanji konveksni skup koji sadrži sve primjere dotične klase. U dvodimenzionalnom prostoru to će biti poligon, a općenito n -dimenzionalni **politop**.

Sad se možemo pitati: koja je veza maksimalne marginе i konveksnih ljsusaka? Odgovor je da, ako maksimiziramo marginu, onda će hiperravnina zapravo biti **simetrala spojnica** dviju konveksnih ljsusaka:



Na lijevoj slici prikazane su dvije konveksne ljske i spojnica između njih. Spojnica povezuje dvije najbliže točke tih dviju konveksnih ljsaka. Okomica spajnice je pravac koji razdvaja primjere iz dviju klasa. Takav pravac ujedno maksimizira marginu (udaljenost između pravca i konveksnih ljsaka s obje strane), tj. primjer iz prve klase koji je najbliži pravcu i primjer iz druge klase koji je najbliži pravcu su jednako udaljeni od toga pravca. Na desnoj slici prikazana je slična situacija, ali je sada jedan brid konveksne ljske paralelan s pravcem, te su dva primjera iz jedne klase (crvene) i jedan primjer iz druge klase (plave) jednako udaljeni od pravca.

Dobro, sada kada znamo kako želimo postaviti hiperravninu (tako da maksimizira marginu), možemo se zapitati kako ćemo to postići. Kako možemo reći algoritmu strojnog učenja koju hiperravninu da *preferira*? Odgovor je: trebamo definirati **pristranosti preferencijom**. Kako definiramo pristranost preferencije kod algoritma strojnog učenja? Tako da tu pristranost ugradimo u empirijsku pogrešku i optimizacijski postupak. Pa, hajdemo onda definirati te komponente algoritma, tako da dobijemo upravo hipotezu s maksimalnom marginom.

Prisjetimo se najprije kako smo to radili do sada, kod poopćenih linearnih modela. Recept je bio sljedeći. Najprije smo definirali vjerojatnost oznaka skupa označenih primjera. To znači da smo pretpostavili da izlaz modela odgovara nekoj teorijskoj distribuciji (Gaussovoj, Bernoullijskoj, multinoullijskoj), koja je upravljana parametrima \mathbf{w} . Zatim smo napisali izraz za logaritam vjerojatnosti oznaka pod takvom distribucijom i tretirali ga kao funkciju parametara \mathbf{w} . Nakon toga izveli smo **empirijsku pogrešku** kao negativnu vrijednost tog logaritma. Konačno, nakon toga razvili smo **optimizacijski postupak**, već ovisno o tome je li pogreška imala rješenje u zatvorenoj formi ili nije.

Kod SVM-a, međutim, ići ćemo posve drugim putem. Krenut ćemo odmah od onoga što želimo dobiti – **maksimalne margine** – i direktno nju definirati kao optimizacijski problem. Onda će iz toga, praktički kao nusprodukt, ispasti funkcija pogreške i funkcija gubitka.

1.1 Formulacija optimizacijskog problema

Formalizirajmo sada problem maksimalne margine. U nastavku će nam biti lakše težinu w_0 tretirati odvojeno od drugih težina, kao što smo radili kada smo pričali o geometriji linearног modela. Model SVM-a je najjednostavniji mogući:

$$h(\mathbf{x}; \mathbf{w}, w_0) = \mathbf{w}^T \mathbf{x} + w_0$$

Granica između klase je hiperravnina $h(\mathbf{x}) = 0$. Kao i uvijek, to je $(n - 1)$ -dimenzijska hiperravnina ugrađena u n -dimenzijski prostor. Oznake primjera modela neka su $y \in \{-1, +1\}$; tako će biti lakše nego da radimo sa 0 i 1.

Predikcija modela ovisi o tome je li primjer na jednoj ili drugoj strani hiperravnine, što možemo detektirati na temelju predznaka. Dakle, predikcija oznake je:

$$y = \text{sgn}(h(\mathbf{x}))$$

Uz našu pretpostavku da su primjeri linearno odvojivi, postoje težine \mathbf{w} i w_0 takve da:

$$\forall (\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D}. y^{(i)}h(\mathbf{x}^{(i)}) \geq 0$$

Ovo jednostavno slijedi iz pretpostavke da su primjeri linearno odvojivi. Naime, ako su primjeri linearno odvojivi, onda ih hipoteza $h(\mathbf{x})$ sve ispravno klasificira, a to znači da je $h(\mathbf{x}^{(i)}) = y^{(i)}$ za svaki označeni primjer $(\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D}$, a to je isto kao da smo napisali $y^{(i)}h(\mathbf{x}^{(i)}) \geq 0$ (ili je primjer pozitivan pa je i izlaz modela pozitivan ili nula, pa je umnožak pozitivan ili nula, ili je primjer negativan pa je izlaz modela negativan, pa je umnožak negativan; sjetite se da smo upravo umnožak $y^{(i)}h(\mathbf{x}^{(i)})$ koristili kod usporedbe funkcije gubitaka).

Koliko hipoteza postoji za koje ovo vrijedi? (Pretpostavite $\mathcal{X} \in \mathbb{R}^n$.) Odgovor je: ima ih beskonačno mnogo, odnosno postoji beskonačno mnogo (w_0, \mathbf{w}) za koje ovo vrijedi.

Sad uvodimo induktivnu pristranost preferencijom: nas zanima ona hipoteza koja daje rješenje **maksimalne margine**. Prisjetimo se: margina je udaljenost hiperravnine do najbližeg primjera. Dakle, želimo da hiperravnina prolazi tako da je najudaljenija od najbližih primjera dviju klasa. Zapravo, kad se prisjetimo slike s konveksnim ljkuskama, to znači da je hiperravnina upravo jednako udaljena od najbližih primjera iz obaju klasa. Znamo da je **predznačena udaljenost** primjera od hiperravnine jednaka:

$$d = \frac{h(\mathbf{x})}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T \mathbf{x}^{(i)} + w_0}{\|\mathbf{w}\|}$$

Nas zanimaju samo hiperravnine koje ispravno sve klasificiraju primjere. To znači da vrijedi $y^{(i)}h(\mathbf{x}^{(i)}) \geq 0$, bio primjer pozitivan ili negativan. Dalje, to znači da je

$$\frac{y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0)}{\|\mathbf{w}\|}$$

nepredznačena udaljenost primjera $\mathbf{x}^{(i)}$ od hiperravnine, bio on pozitivan ili negativan (dakle, uвijek pozitivan broj, neovisno o oznaci primjera). Nadalje, po definiciji, margina je udaljenost hiperravnine do **najbližeg primjera**:

$$\min_i \left\{ \frac{y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0)}{\|\mathbf{w}\|} \right\} = \frac{1}{\|\mathbf{w}\|} \min_i \{y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0)\}$$

gdje smo na desnoj strani izlučili $\|\mathbf{w}\|$ izvan funkcije min, budući da je norma vektora težina za sve primjere ista. Provedite ovdje nekoliko minuta dok se niste uvjerili da razumijete izraz koji smo upravo napisali. Funkcija min iterira po svim primjerima iz \mathcal{D} i računa udaljenost primjera od hiperravnine, te nalazi najmanju takvu udaljenost, tj. udaljenost do najbližeg primjera, bilo pozitivnog ili negativnog. I sada, budući da želimo **maksimalnu marginu**, ovu udaljenost koju smo upravo napisali želimo maksimizirati, tj. tražimo takvu hiperravninu (takve parametre \mathbf{w} i w_0) koji će maksimizirati tu udaljenost:

$$\operatorname{argmax}_{\mathbf{w}, w_0} \left\{ \frac{1}{\|\mathbf{w}\|} \min_i \{y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0)\} \right\}$$

Opet, provedite ovdje nekoliko minuta dok se niste uvjerili da razumijete što smo ovdje napisali. Konceptualno, funkcija argmax iterira po svim mogućim hiperravninama, za svaku hiperravninu izračunavamo pomoću funkcije min udaljenost do najbližeg primjera, te odabiremo onu hiperravninu, tj. one parametre \mathbf{w} i w_0 , za koju je ta udaljenost najveća. Ako je \mathcal{D} linearno odvojiv, a pretpostavili smo da jest, onda postoji samo jedna takva hiperravnina. Također, za tu hiperravninu postojat će barem dva primjera koji će od nje biti jednakо udaljeni, i to jedan pozitivan s jedne strane hiperravnine te jedan negativan s druge strane hiperravnine. Naime, kada to ne bi bilo tako – kada bi jedan primjer iz jedne klase bio bliži hiperravnini a drugi iz druge klase malo dalji – onda ta hiperravnina ne bi bila rješenje maksimalne margine, jer bismo je uвijek mogli malo odmaknuti od bližeg primjera i približiti onom daljem primjeru i time maksimizirati marginu. Dakle, hiperravnina koja maksimizira marginu sigurno je pozicionirana tako da su njoj najbliži primjeri iz pozitivne i negativne klase od nje jednakо udaljeni. Primijetite da sve ovo slijedi iz gornjeg izraza, tj. ne moramo više uvoditi nikakve dodatne uvjete.

Ovime smo definirali kakvu hiperravninu želimo, odnosno definirali smo optimizacijski problem. Nažalost, ovako definiran optimizacijski problem ne možemo izravno riješiti: poteškoća je u tome što imamo min-izraz unutar argmax-izraza. Umjesto toga, problem trebamo nekako preformulirati, tako da postane jednostavniji. Pokazuje se da to možemo napraviti, i to tako da se riješimo min funkcije. Ideja je da pretpostavimo da je za primjer koji je najbliži margini izlaz modela $\mathbf{w}^T \mathbf{x} + w_0$ jednak nekoj konstanti, pa onda uopće ne trebamo tražiti najbliži primjer. Kako to možemo napraviti? Pa, trik je u tome da se sjetimo da vektor težina (w_0, \mathbf{w}) možemo

proizvoljno skalirati, a da to neće utjecati na orijentaciju hiperravnine niti na udaljenosti između primjera od hiperravnine, ali će međutim to utjecati na izlaz hipoteze. Prepostavimo onda da je vektor težina (w_0, \mathbf{w}) skaliran upravo tako da hipoteza za najbliži pozitivan primjer daje izlaz $+1$ te, posljedično, za najbliži negativan primjer daje izlaz -1 . Drugim riječima, prepostavimo da za primjer koji je najbliži hiperravnini vrijedi:

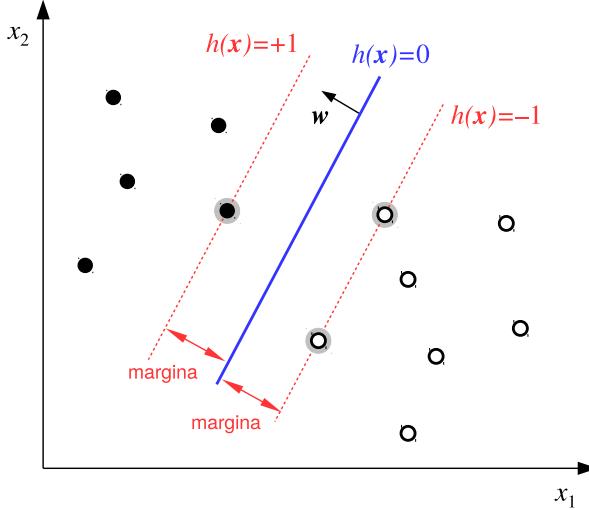
$$y^{(i)}(\mathbf{w}^T \mathbf{x} + w_0) = 1$$

Ovo može vrijediti za više od jednog primjera. Zapravo, primjetite da će ovo sigurno vrijediti za barem dva primjera, po jedan sa svake strane hiperravnine, jer inače hiperravnina ne bi bila na poziciji maksimalne margini.

Gornji uvjet vrijedi, dakle, za primjere iz \mathcal{D} koji su najbliži margini, a takvih je barem dva (po jedan sa svake strane), a može ih biti i više. Što je sa svim drugim primjerima u \mathcal{D} ? Svi drugi primjeri bit će još udaljeniji od margini, dakle za njih će izlaz modela biti $h(\mathbf{x}) > 1$ ako su pozitivni odnosno $h(\mathbf{x}) < -1$ ako su negativni. Dakle, možemo reći da za sve primjere u \mathcal{D} (i one najbliže hiperravninini i one koji to nisu), od $i = 1$ do $i = N$, vrijedi sljedeće:

$$y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \geq 1, \quad i = 1, \dots, N$$

Prikažimo to grafički:



Na slici je prikazan dvodimenzionalni ulazni prostor s primjerima klase $y = 1$ (puni kružići) i klase $y = 0$ (prazni kružići). Granica je pravac za koji $h(\mathbf{x}) = 0$. Pozitivni primjeri (oni za koje $y = 1$) su na pozitivnoj strani pravca, tj. strani u smjeru normale \mathbf{w} . Tri su primjera najbliža pravcu $h(\mathbf{x}) = 0$, i to jedan primjer iz klase $y = 1$ i dva primjera iz klase $y = 0$ (ta tri primjera označena su sivim rubom). Izlaz hipoteze za te primjere je $h(\mathbf{x}) = +1$ (za primjer iz klase $y = 1$) odnosno $h(\mathbf{x}) = -1$ (za dva primjera iz klase $y = 0$). Da je to tako slijedi iz našeg uvjeta da za primjere koji su najbliži pravcu (odnosno općenito hiperravnini) mora vrijediti $yh(\mathbf{x}) = 1$. Na slici vidimo i marginu, a to je udaljenost od pravca do najbližeg primjera s jedne i druge strane. Za tri primjera za koja vrijedi $yh(\mathbf{x}) = 1$ kažemo da se nalaze "na margini".

Nakon ovog zgodnog trika – da za primjere koji su najbliži hiperravnini prepostavimo $yh(\mathbf{x}) = 1$ – optimizacijski se problem iz:

$$\underset{\mathbf{w}, w_0}{\operatorname{argmax}} \left\{ \frac{1}{\|\mathbf{w}\|} \underbrace{\min_i \{y^{(i)}(\mathbf{w}^T \mathbf{x} + w_0)\}}_{=1} \right\}$$

svodi na:

$$\underset{\mathbf{w}, w_0}{\operatorname{argmax}} \frac{1}{\|\mathbf{w}\|}$$

Međutim, ovo vrijedi samo ako vrijedi naša pretpostavka, stoga i nju moramo nekako ugraditi u optimizacijski postupak. To radimo tako da u optimizacijski postupak dodamo sljedeća **ograničenja**:

$$y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \geq 1, \quad i = 1, \dots, N$$

Došli smo dakle, do toga da želimo maksimizirati $\frac{1}{\|\mathbf{w}\|}$ uz gornja ograničenja. Sada možemo primijetiti da je maksimizator od $\frac{1}{\|\mathbf{w}\|}$ ekvivalentan minimizatoru od $\|\mathbf{w}\|$, a taj je pak ekvivalentan minimizatoru od $\|\mathbf{w}\|^2$ (jer je L_2 -norma konveksna i nenegativna funkcija). Još ćemo sve to pomnožiti s $\frac{1}{2}$ radi kasnije matematičke jednostavnosti. Konačna formulacija optimizacijskog problema maksimalne margine onda je sljedeća:

$$\underset{\mathbf{w}, w_0}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{w}\|^2$$

uz ograničenja:

$$y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \geq 1, \quad i = 1, \dots, N$$

Pogledajmo što smo zapravo napravili. Krenuli smo od problema maksimalne margine, i napisali ga kao maksimizaciju udaljenosti između najbližih primjera. To nam je dalo izraz argmax-min, koji nije prikladan za optimizaciju. Onda smo izveli lukav trik: pretpostavili smo da za primjere koji su najbliži hiperravnini hipoteza daje ± 1 . To nam je omogućilo da se riješimo izraza min. Konačno, uz malo algebре, došli smo do toga da želimo minimizirati $\frac{1}{2} \|\mathbf{w}\|^2$ uz ograničenja $y^{(i)} h(\mathbf{x}^{(i)}) \geq 1$.

Naš optimizacijski problem sveo se na ciljnju funkciju koju želimo minimizirati i ograničenja koja pritom moramo poštovati. Budući da je ciljna funkcija **konveksna**, ovo je tipičan problem koji se u teoriji optimizacije naziva **konveksna optimizacija uz ograničenja**, odnosno preciznije problem **kvadratnog programiranja**.

Očito, da bismo mogli dalje, trebamo pogledati o čemu se tu radi.

2 Optimizacija uz ograničenja

Općenito, **optimizacijski problem uz ograničenja** (engl. *constrained optimization problem*) definiran je na sljedeći način:

$$\begin{aligned} & \text{minimizirati} && f(\mathbf{x}) \\ & \text{uz ograničenja} && g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m \\ & && h_i(\mathbf{x}) = 0, \quad i = 1, \dots, p \end{aligned}$$

Pri čemu je funkcija $f : \mathbb{R}^n \rightarrow \mathbb{R}$ **ciljna funkcija** (engl. *objective function*) koju minimiziramo, $h_i : \mathbb{R}^n \rightarrow \mathbb{R}$ su **ograničenja jednakosti** (engl. *equality constraints*), a $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ su **ograničenja nejednakosti** (engl. *inequality constraints*). Primijetite da možemo imati više ograničenja jednakosti i više ograničenja nejednakosti (a možemo imati i samo jednu od te dvije vrste ograničenja).

Ovakav oblik optimizacijskog problema (minimizacija ciljne funkcije, ograničenja definirana tako da su manja ili jednaka nuli) naziva se **standardni oblik**. Standardni oblik možda se čini malo ograničavajućim, ali to zapravo nije. Naime, ako umjesto minimizacije želimo maksimizaciju funkcije $f(\mathbf{x})$, onda možemo jednostavno minimizirati funkciju $-f(\mathbf{x})$. Nadalje, sva ograničenja (ne)jednakosti daju se uvijek svesti na standardni oblik. Konkretno, ograničenje jednakosti $h(\mathbf{x}) = c$ možemo napisati kao $h(\mathbf{x}) - c = 0$, dok ograničenja nejednakosti $g(\mathbf{x}) \leq c$ ili $g(\mathbf{x}) \geq c$ možemo napisati kao $g(\mathbf{x}) - c \leq 0$ odnosno $c - g(\mathbf{x}) \leq 0$.

Kod optimizacije s ograničnjima tražimo minimum koji zadovoljava sva ograničenja. Točke koje zadovoljavaju ograničenja nazivamo **ostvarivim točkama** (engl. *feasible points*) ili **ostvarivim područjem** (engl. *feasibility region*).

Poseban slučaj gornjeg optimizacijskog problema jest onaj kod kojega je funkcija $f(\mathbf{x})$ konveksna. Tada govorimo o **konveksnom optimizacijskom problemu** (engl. *convex optimization problem*). Nadalje, poseban slučaj konveksnog optimizacijskog problema je **kvadratni program** (engl. *quadratic program, QP*), kod kojega je ciljna funkcija kvadratna (pa time i konveksna), a ograničenja su afne funkcije. Primijetite da je upravo takav optimizacijski problem maksimalne margine. Naime, ciljna funkcija je kvadratna:

$$\frac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

dok su ograničenja nejednakosti linearna:

$$y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) \geq 1, \quad i = 1, \dots, N$$

Postoji niz metoda za rješavanje kvadratnog programa, npr. **metode kazne** (engl. *penalty methods*), **metode unutarnje točke** (engl. *interior point methods / barrier methods*), **koordinatni spust**, metoda **konjugiranog gradijenta** i **Lagrangeova dualnost**. Mi ćemo koristiti zadnje navedenu metodu, koja je, u kombinaciji s algoritmom **slijedne minimalne optimizacije** (engl. *sequential minimal optimization, SMO*), bila prva korištena za rješavanje problema maksimalne margine. Glavna ideja te metode jest da ćemo preći u tzv. **dualnu formulaciju optimizacijskog problema**. Uskoro ćemo vidjeti što to zapravo znači. Vidjet ćemo i koje su prednosti tog pristupa.

Pogledajmo, dakle, kako pristupiti kvadratnom programiranju preko Lagrangeove dualnosti.

3 Lagrangeova dualnost

Lagrangeova dualnost zasniva se na metodi **Lagrangeovih multiplikatora**. Ideja metode Lagrangeovih multiplikatora jest da **preformuliramo** optimizacijski problem s ograničenjima (ne nužno konveksan!) tako da ta ograničenja eksplicitno **ugradimo** u ciljnu funkciju. Pogledajmo kako.

3.1 Lagrangeova funkcija

Početni problem je:

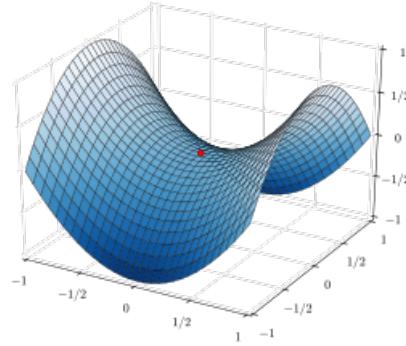
$$\begin{aligned} & \text{minimizirati} && f(\mathbf{x}) \\ & \text{uz ograničenja} && g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m \\ & && h_i(\mathbf{x}) = 0, \quad i = 1, \dots, p \end{aligned}$$

Ciljnu funkciju i ograničenja kombiniramo u novu funkciju:

$$L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \sum_{i=1}^m \alpha_i g_i(\mathbf{x}) + \sum_{i=1}^p \beta_i h_i(\mathbf{x})$$

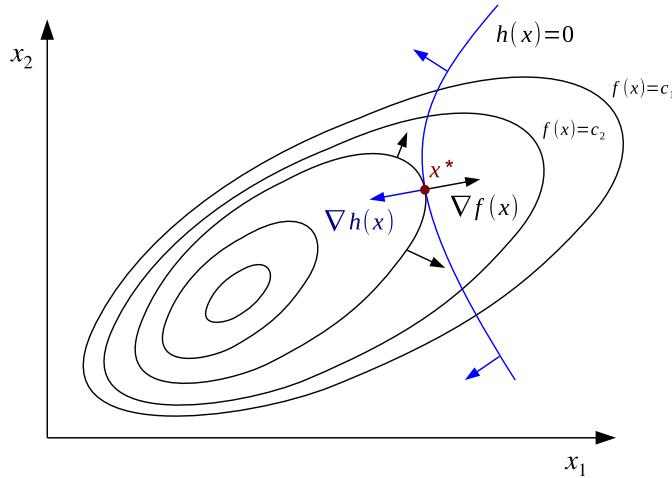
gdje $\alpha_i \geq 0$. Ovu funkciju nazivamo **Lagrangeova funkcija**. Vrijednosti α_i i β_i su **Lagrangeovi multiplikatori** (množitelji) za ograničenja nejednakosti odnosno ograničenja jednakosti.

Rješenje originalnog optimizacijskog problema s ograničenjem je točka u kojoj je gradijent Lagrangeove funkcije jednak nuli, $\nabla L = 0$, tj. **stacionarna točka** Lagrangeove funkcije. Pokazuje se da je ta stacionarna točka zapravo **sedlo (saddle point)** Lagrangeove funkcije i ta je točka minimum funkcije po \mathbf{x} i maksimum funkcije po $\boldsymbol{\alpha}$ i $\boldsymbol{\beta}$. Ovako to izgleda:



Ovo je vrlo zanimljivo, i ovo je sedlo vrlo lijepo, no otkud sve to? Hajdemo ovo pogledati malo detaljnije. Lagrangeova funkcija kodira dvije vrste ograničenja: **ograničenja jednakosti i ograničenja nejednakosti**. Pogledajmo zasebno ta dva slučaja. Fokusirat ćemo se na slučaj kada je ciljna funkcija konveksna (premda Lagrangeova metoda nije ograničena samo na konveksne funkcije; jedini zahtjev jest da su ciljna funkcija i funkcije ograničenja derivabilne). Bez smanjenja općenitosti, ograničit ćemo se na funkciju dvije varijable, kako bismo to mogli skicirati.

3.2 Ograničenja jednakosti



Na slici su prikazane izokonture ciljne funkcije $f(\mathbf{x})$. To je funkcija čiji minimum tražimo. Minimum se nalazi negdje u sredini najmanje konture. Funkcija $h(\mathbf{x}) = 0$ (plava krivulja) je ograničenje koje naše rješenje mora zadovoljiti. To znači da u obzir za rješenje dolaze samo one točke koje se nalaze na toj krivulji. Na našoj slici $h(\mathbf{x}) = 0$ je krivulja, no u općenitom slučaju, $h(\mathbf{x}) = 0$ definira n -dimenzijsku površinu ugrađenu u prostor $\mathbb{R}^n \times \mathbb{R}$. U svakoj točki na površini $h(\mathbf{x}) = 0$, gradijent $\nabla h(\mathbf{x})$ bit će okomit na tu površinu (to mora biti po definiciji gradijenta). Neka je točka \mathbf{x}^* točka na površini ograničenja $h(\mathbf{x}) = 0$ koja minimizira $f(\mathbf{x})$ (označena crveno). To je točka u kojoj smo se uspjeli najviše približiti globalnome minimumu funkcije $f(\mathbf{x})$, ali smo ipak ostali na površini $h(\mathbf{x}) = 0$ koja definira ostvarive točke. (Kao muha koja kroz staklo želi doći što blizu pekmezu; doći će do točke na staklu gdje je najbliža pekmezu, ali je i dalje na staklu).

Sad primijetite da vektor $\nabla f(\mathbf{x}^*)$ također mora biti okomit na površinu $h(\mathbf{x}) = 0$. U protivnom bismo se naime uvijek mogli pomaknuti po površini ograničenja tako da se vrijednost $f(\mathbf{x})$ smanji. Tek ako je gradijent okomit, znači da smo na minimalnoj mogućoj točki.

Temeljem ovih opažanja, zaključujemo da u točki \mathbf{x}^* vektori gradijentna ∇f i ∇h moraju biti **kolinearni (paralelni ili antiparalelni)**. To onda znači da u točki \mathbf{x}^* mora postojati konstanta β

za koju vrijedi:

$$\nabla f(\mathbf{x}^*) + \beta \nabla h(\mathbf{x}^*) = 0$$

tj. točka u kojoj linearna kombinacija vektora ∇f i ∇h iščezava. Točka \mathbf{x}^* za koju ovo vrijedi upravo je rješenje našeg početnog optimizacijskog problema s ograničenjem!

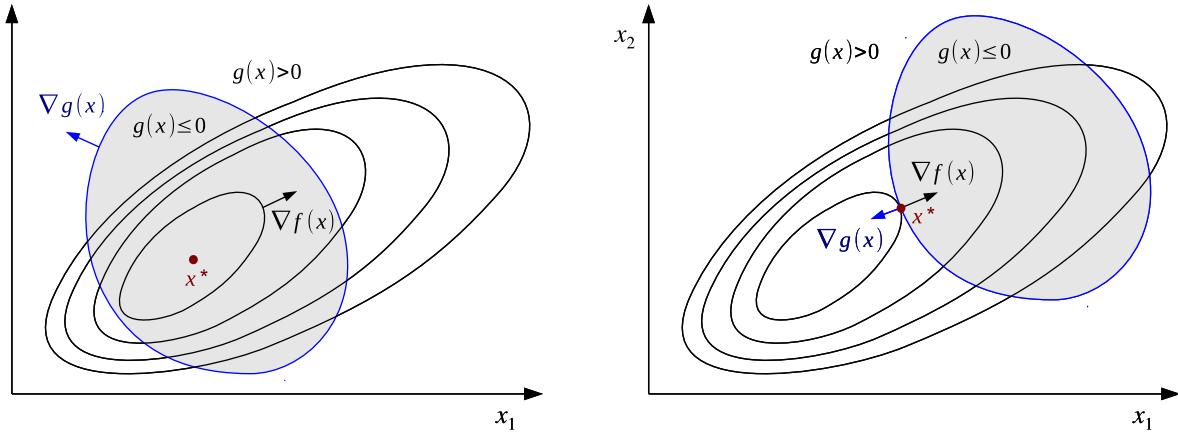
Sada pretpostavimo da ova jednadžba odgovara stacionarnoj točki neke funkcije. Ako je tako, onda bi ta funkcija bila ova:

$$L(\mathbf{x}, \beta) \equiv f(\mathbf{x}) + \beta h(\mathbf{x})$$

i to je upravo naša Lagrangeova funkcija! Pa zaključujemo: stacionarna točka Lagrangeove funkcije je rješenje našeg problema s ograničenjem.

3.3 Ograničenja nejednakosti

Pogledajmo sada slučaj s **ograničenjima nejednakosti**, jer to je zapravo ono što će nam trebati za naš problem maksimalne margine. Tu su moguća dva slučaja: minimum funkcije $f(\mathbf{x})$ nalazi se unutar ostvarivog područja ili se nalazi izvan.



Na lijevoj slici prikazana je situacija kada je minimum *unutar* ostvarenog područja. Tada kažemo da ograničenje **nije aktivno**, te funkcija $g(\mathbf{x})$ ne igra nikakvu ulogu.

Na desnoj slici prikazana je situacija kada je minimum *izvan* ostvarivog područja. Tada kažemo da je ograničenje **aktivno** i točka minimuma \mathbf{x}^* nalazi se na površini $g(\mathbf{x}) = 0$, što je bliže moguće neograničenom minimumu. To nadalje znači da sve točke \mathbf{x} u ostvarivom području imaju vrijednost $f(\mathbf{x})$ veću od minimuma, pa gradijent $\nabla f(\mathbf{x})$ pokazuje prema ostvarivom području, dok $\nabla g(\mathbf{x})$ pokazuje od njega. Posljedično, $\nabla f(\mathbf{x})$ i $\nabla g(\mathbf{x})$ su antiparalelni vektori te vrijedi:

$$\nabla f(\mathbf{x}^*) = -\alpha \nabla g(\mathbf{x}^*)$$

za neku konstantu $\alpha > 0$.

Prema tome, uvezši u obzir oba ova slučaja, minimizaciji $f(\mathbf{x})$ uz uvjet $g(\mathbf{x})$ odgovara nalaženje **stacionarne točke** sljedeće Lagrangeove funkcije:

$$L(\mathbf{x}, \alpha) \equiv f(\mathbf{x}) + \alpha g(\mathbf{x})$$

uz $\alpha \geq 0$. Ako $\alpha = 0$, onda ograničenje $g(\mathbf{x})$ nije aktivno.

Primjetite da za točku ostvarivog minimuma \mathbf{x}^* mora vrijediti ili $\alpha = 0$ (neaktivno ograničenje) ili $g(\mathbf{x}) = 0$ (aktivno i ispoštovano ograničenje). To sažeto možemo napisati kao:

$$\alpha g(\mathbf{x}) = 0$$

Ovaj se uvjet naziva **komplementarna labavost** (engl. *complementary slackness*).

Početna ograničenja jednakosti i nejednakosti, zajedno s ova dva uvjeta koja smo upravo izveli ($\alpha \geq 0$ i $\alpha g(\mathbf{x}) = 0$), čine tzv. **Karush-Kuhn-Tuckerove (KKT) uvjeti**. To su uvjeti koji nužno vrijede u točki rješenja.

Tako smo, dakle, dobili Lagrangeovu funkciju i pripadne uvjete KKT. Evo sada sve na jednom mjestu:

► Lagrangeova funkcija i uvjeti KKT

$$\begin{aligned} & \text{minimizirati} && f(\mathbf{x}) \\ & \text{uz ograničenja} && g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m \\ & && h_i(\mathbf{x}) = 0, \quad i = 1, \dots, p \end{aligned}$$

Lagrangeova funkcija:

$$L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \sum_{i=1}^m \alpha_i g_i(\mathbf{x}) + \sum_{i=1}^p \beta_i h_i(\mathbf{x})$$

U stacionarnoj točki vrijede uvjeti KKT:

$$\begin{aligned} g_i(\mathbf{x}) &\leq 0, \quad i = 1, \dots, m \\ h_i(\mathbf{x}) &= 0, \quad i = 1, \dots, p \\ \alpha_i &\geq 0, \quad i = 1, \dots, m \\ \alpha_i g_i(\mathbf{x}) &= 0, \quad i = 1, \dots, m \end{aligned}$$

Primijetite da Lagrangeove funkcije za ograničenje jednakosti i nejednakosti možemo kombinirati u jednu funkciju s oba ograničenja. Također, ako imamo više ograničenja, samo ih sve pridodamo u Lagrangeovu funkciju.

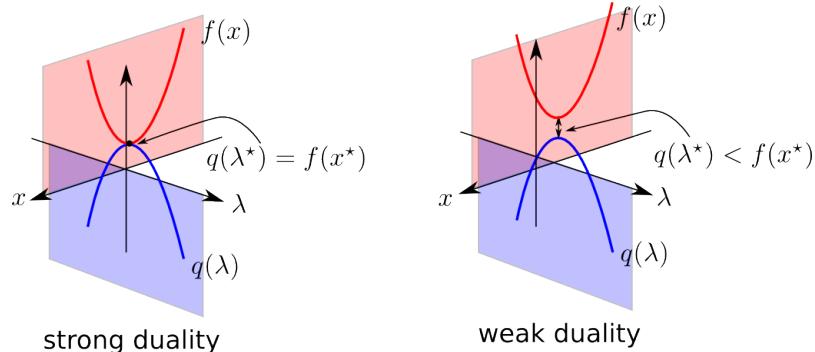
Sada smo se (nadam se) uvjerili da Lagrangeova funkcija i uvjeti KKT imaju smisla. No, kako nalazimo stacionarnu točku Lagrangeove funkcije? To nas dovodi do tzv. **načela dualnosti**.

3.4 Načelo dualnosti

U teoriji optimizacije postoji tzv. **načelo dualnosti** (engl. *duality principle*), koje nam kaže da optimizacijski problem možemo sagledavati na dva načina:

- **Primarni problem** (engl. *primal problem*): minimizacija funkcije $f(\mathbf{x})$
- **Dualni problem** (engl. *dual problem*): nalaženje **donje ograde** primarnog problema

Odnos između primarnog i dualnog optimizacijskog problema možemo prikazati ovako:



Slike prikazuju primarni problem (crvena krivulja) i dualni problem (plava krivulja). Primarni problem i dualni problem imaju svaki svoje varijable po kojima optimiziramo: primarni problem optimiziramo po primarnim varijablama (na slici je to varijabla x), a dualni problem po dualnim varijablama (na slici je to varijabla λ). Primarne varijable i dualne varijable su različite varijable, pa je na slici prikazano tako da su dimenzije primarnog problema i dualnog problema različite (primarni problem: crvena ravnina, dualni problem: plava ravnina). Plava krivulja odgovara dualnom problemu, i ona je donja ograda primarnog problema (dakle, minimum funkcije $f(\mathbf{x})$ veći je ili jednak vrijednosti plave krivulje). Općenito, rješenja primarnog i dualnog problema se ne moraju poklapati već može postojati tzv. **procjep dualnosti** (engl. *duality gap*). To je prikazano na desnoj slici. Uz određene uvjete, kod konveksne optimizacije dualni procjep jednak je nuli. Tada govorimo o **jakoj dualnosti** (engl. *strong duality*) (lijeva slika). Rješenje se onda nalazi u točki sedla: minimum po primarnim varijablama (crvena krivulja) a maksimum po dualnim varijablama (plava krivulja).

U našem slučaju vrijedi jaka dualnost, budući da je problem maksimalne margine konveksan optimizacijski problem (dapače, to je kvadratni program). To znači da umjesto primarnog problema možemo rješavati dualni problem, i da ćemo dobiti identično rješenje. Ovdje se odmah nameće pitanje zašto bismo to uopće htjeli? Zašto rješavati dualni problem umjesto primarnog, kada ćemo ionako dobiti isto rješenje? Odgovor na to pitanje ostavit ćemo za kasnije. Ovdje ćemo samo reći da rješavanje problema maksimalne margine u dualu ima niz važnih praktičnih prednosti koje su vrlo primamljive.

Pogledajmo kako načelo dualnosti izgleda kod Lagrangeove funkcije – to je **Lagrangeova dualnost**. Lagrangeova funkcija $L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ je funkcija **primarnih varijabli \mathbf{x}** i **dualnih varijabli $\boldsymbol{\alpha}$ i $\boldsymbol{\beta}$** . Već znamo da je rješenje optimizacijskog problema stacionarna točka Lagrangeove funkcije, tj. točka za koju vrijedi:

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0$$

Međutim, rješenje ove jednadžbe ne mora dovesti do uklanjanja dualnih varijabli. Općenito, dobit ćemo rješenje koje, za neke $\boldsymbol{\alpha}$ i $\boldsymbol{\beta}$, minimizira Lagrangeovu funkciju L po primarnim varijablama \mathbf{x} . Drugim riječima, dobit ćemo funkciju:

$$\tilde{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

Ova se funkcija naziva **dualna Lagrangeova funkcija** (primijetite tildu!). Provedite ovdje neko vrijeme da shvatite što ova funkcija zapravo radi. Za fiksirani $\boldsymbol{\alpha}$ i $\boldsymbol{\beta}$ (dualne varijable), funkcija vraća minimum Lagrangeove funkcije po primarnoj varijabli \mathbf{x} . Zbog načela dualnosti, ova funkcija je **donja ograda** primarnog problema: to znači da je minimum od $L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ veći ili jednak vrijednostima $\tilde{L}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ za svaki $\boldsymbol{\alpha}$ i $\boldsymbol{\beta}$. Nadalje, zato što vrijedi jaka dualnost, minimum primarnog problema možemo pronaći tako da *maksimiziramo* dualni problem, je će nas to dovesti u točku sedla: minimum po primarnim parametrima, a maksimum po dualnim parametrima (v. prethodnu sliku). Drugim riječima, trebamo riješiti sljedeći konveksni optimizacijski problem:

$$\begin{aligned} & \text{maksimizirati} && \tilde{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}) \\ & \text{uz ograničenja} && \alpha_i \geq 0, \quad i = 1, \dots, m \end{aligned}$$

Zaključujemo: **minimizacija ciljne funkcije** istovjetna je **maksimizaciji dualne funkcije** (ako vrijedi jaka dualnost, što kod nas jest slučaj!).

Sažmimo sve ovo na jednom mestu. Želimo riješiti sljedeći optimizacijski problem:

► Konveksna optimizacija u dualu

$$\begin{aligned}
& \text{minimizirati} && f(\mathbf{x}) \\
& \text{uz ograničenja} && g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m \\
& && h_i(\mathbf{x}) = 0, \quad i = 1, \dots, p
\end{aligned}$$

Pripadna Lagrangeova funkcija je:

$$L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \sum_{i=1}^m \alpha_i g_i(\mathbf{x}) + \sum_{i=1}^p \beta_i h_i(\mathbf{x})$$

Dualna Lagrangeova funkcija je:

$$\tilde{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

Dualni problem je:

$$\begin{aligned}
& \text{maksimizirati} && \tilde{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}) \\
& \text{uz ograničenja} && \alpha_i \geq 0, \quad i = 1, \dots, m
\end{aligned}$$

Ako je problem konveksan, vrijedi jaka dualnost, pa je rješenje dualnog problema ujedno i rješenje primarnog problema.

A sada kada sve ovo znamo, vratimo se našem poslu – optimizaciji maksimalne margine...

4 Optimizacija maksimalne margine

Dakle, naš kvadratni program maksimalne margine riješit ćemo tako da ćemo ograničenja uvesti u Lagrangeovu funkciju, koja je funkcija originalnih (primarnih) varijabli i novih (dualnih) varijabli. Zatim ćemo iskazati minimum Lagrangeove funkcije po primarnim varijablama, što će nam dati dualnu Lagrangeovu funkciju. Naposlijetku ćemo maksimizirati tu funkciju, jer će nam njezina maksimizacija dati minimum primarne funkcije, budući da se radi o konveksnom problemu. Brilijantno!

Pa, krenimo. Prisjetimo se, optimizacijski problem maksimalne margine bio je:

$$\operatorname{argmin}_{\mathbf{w}, w_0} \frac{1}{2} \|\mathbf{w}\|^2$$

uz uvjete:

$$y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \geq 1, \quad i = 1, \dots, N$$

Primijetite da imamo onoliko ograničenja koliko imamo primjera. Napišimo Lagrangeovu funkciju:

$$L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i \left\{ y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1 \right\}$$

gdje je $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_N)$, $\alpha_i \geq 0$, vektor Lagrangeovih multiplikatora, po jedan za svaki primjer. Lagrangeova funkcija L je funkcija primarnih varijabli \mathbf{w} i w_0 te dualnih varijabli $\boldsymbol{\alpha}$.

Sada prelazimo na dualnu formulaciju: po definiciji, dualna Lagrangeova funkcija je ona koja minimizira Lagrangeovu funkciju po primarnim varijablama. Dakle:

$$\tilde{L}(\boldsymbol{\alpha}) = \min_{\mathbf{w}, w_0} L(\mathbf{w}, w_0, \boldsymbol{\alpha})$$

Dobra stvar je da u ovom slučaju minimizacija Lagrangeove funkcije po primarnim varijablama ima rješenje u zatvorenoj formi. Minimizator (\mathbf{w}^*, w_0^*) dobivamo deriviranje po \mathbf{w} odnosno w_0 i izjednačavanje s nulom:

$$\begin{aligned}\frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\alpha})}{\partial \mathbf{w}} = 0 &\Rightarrow \mathbf{w} = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)} \\ \frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\alpha})}{\partial w_0} = 0 &\Rightarrow \sum_{i=1}^N \alpha_i y^{(i)} = 0\end{aligned}$$

Posebno upamtite ovu prvu jednakost za \mathbf{w} , koji nam je bitna je povezuje primarne varijable \mathbf{w} s dualnim varijablama $\boldsymbol{\alpha}$.

Ove dvije jednakosti sada uvrštavamo nazad u Lagrangeovu funkciju, kako bismo dobili njezin minimum, odnosno **dualnu Lagrangeovu funkciju**:

$$\begin{aligned}\tilde{L}(\boldsymbol{\alpha}) &= \min_{\mathbf{w}, w_0} \left(\frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i \{y^{(i)} (\mathbf{w}^\top \mathbf{x}^{(i)} + w_0) - 1\} \right) \\ &= \min_{\mathbf{w}, w_0} \left(\frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} - w_0 \underbrace{\sum_{i=1}^N \alpha_i y^{(i)}}_{=0} + \sum_{i=1}^N \alpha_i \right) \\ &= \frac{1}{2} \sum_{i=1}^N \alpha_i y^{(i)} (\mathbf{x}^{(i)})^\top \sum_{j=1}^N \alpha_j y^{(j)} \mathbf{x}^{(j)} - \sum_{i=1}^N \alpha_i y^{(i)} (\mathbf{x}^{(i)})^\top \sum_{j=1}^N \alpha_j y^{(j)} \mathbf{x}^{(j)} + \sum_{i=1}^N \alpha_i \\ &= \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} (\mathbf{x}^{(i)})^\top \mathbf{x}^{(j)}\end{aligned}$$

Ovaj izvod ne izgleda privlačno, ali je barem rezultat relativno pristojan. Uvjerite se da možete napraviti ovaj izvod.

Pogledajmo sada u kompletu kako izgleda **dualni optimizacijski problem SVM-a**:

► Dualni optimizacijski problem SVM-a

Maksimizirati:

$$\sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} (\mathbf{x}^{(i)})^\top \mathbf{x}^{(j)}$$

uz ograničenja

$$\begin{aligned}\alpha_i &\geq 0, \quad i = 1, \dots, N \\ \sum_{i=1}^N \alpha_i y^{(i)} &= 0\end{aligned}$$

U točki rješenja vrijede uvjeti KKT:

$$\begin{aligned}y^{(i)} (\mathbf{w}^\top \mathbf{x}^{(i)} + w_0) &\geq 1, \quad i = 1, \dots, N \\ \alpha_i &\geq 0, \quad i = 1, \dots, N \\ \alpha_i (y^{(i)} h(\mathbf{x}^{(i)}) - 1) &= 0, \quad i = 1, \dots, N\end{aligned}$$

Izvedenu jednakost $\sum_{i=1}^N \alpha_i y^{(i)} = 0$ morali smo dodati kao ograničenje jer se varijable α_i nismo riješili u dualnoj Lagrangeovoj funkciji (za razliku od primarne varijable \mathbf{w} , koje smo se riješili, pa prvu izvedenu jednakost više ne trebamo kao ograničenje.)

Ovo je i dalje problem **kvadratnog programiranja**, s time da, za razliku od primarnog problema, pored ograničenja nejednakosti, imamo i ograničenja jednakosti. Tako gledano, ispada da prijelazom iz primarne u dualnu formulaciju nismo baš puno dobili, naprotiv, imamo ograničenja jednakosti koja prije nismo imali. Međutim, kako smo već spomenuli, dualna formulacija ima niz prednosti. Jedna od njih je da se na ovaj kvadratni problem sad može primijeniti algoritam **slijedne minimalne optimizacije (SMO)**. Taj algoritam upravo iskorištava ograničenja jednakosti kako bi problem razbio na potprobleme, koje onda rješava analitički, te je zbog toga učinkovitiji od algoritama za općenite kvadratne programe. Nećemo ići u detalje; trebate samo znati da taj algoritam postoji i da se primjenjuje u dualnoj formulaciji.

Međutim, važan detalj koji ovdje trebamo uočiti jest da je, prelaskom iz primarnog u dualni problem, došlo do promjene u varijablama: primarni problem imao je $n+1$ primarnih varijabli, dok dualni ima N dualnih varijabli. Da li se ovo računalno isplati? Da, ako $N \ll n$, tj. **ako je broj primjera mnogo manji od broja značajki**. Tipične domene gdje je to redovito slučaj su bioinformatika, analiza teksta i analiza slike. Općenito, složenost kvadratnog programiranja od N varijabli je $\mathcal{O}(N^3)$. Međutim, SMO je prilagođen upravo ovom problemu i ima složenost $\mathcal{O}(N^2)$.

5 Dualni model SVM-a

Vratimo se sada na početak današnjeg prevanja. Na početku smo definirali **model SVM-a**, i to je bio linearan model:

$$h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

Međutim, to je bila primarna formulacija modela. U dualnoj formulaciji više nemamo primarne parametre, tj. nemamo više težine. Umjesto toga imamo dualne parametre α . Prijelaz s primarnog na dualni model nam omogućava jednadžba koju smo već bili izveli:

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)}$$

Uvrštavanjem ovoga u izraz za primarni model, dobivamo:

$$h(\mathbf{x}) = \underbrace{\mathbf{w}^T \mathbf{x} + w_0}_{\text{Primarno}} = \underbrace{\sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^T \mathbf{x}^{(i)}}_{\text{Dualno}} + w_0$$

Kako funkcioniра predikcija u dualnom modelu? Da bismo klasificirali primjer \mathbf{x} , računamo **skalarni produkt** između \mathbf{x} i svih primjera $\mathbf{x}^{(i)}$ iz skupa \mathcal{D} , pomnožen s težinom α_i i predznakom $y^{(i)}$. Računanje skalarnog produkta $\mathbf{x}^T \mathbf{x}^{(i)}$ je zapravo računanje **sličnosti** između vektora \mathbf{x} i $\mathbf{x}^{(i)}$, budući da:

$$\mathbf{x}^T \mathbf{y} = \sum_{i=1}^n x_i y_i$$

Ovaj umnožak će biti to veći što se vektori podudaraju u više komponenata, a to znači da su vektori to sličniji.

Dakle, umjesto da pohranjujemo težine \mathbf{w} , kod dualnog modela trebamo pohraniti primjere i njihove oznake. Umjesto:

$$h(\mathbf{x}; w_0, \mathbf{w})$$

19

imamo:

$$h(\mathbf{x}; \boldsymbol{\alpha}, \mathcal{D})$$

Pritom težinu w_0 ne trebamo, jer se ona može izračunati naknadno iz parametara α i primjera za učenje \mathcal{D} .

Objasnimo sada napokon zašto se ovaj algoritam zove **stroj potpornih vektora**. Iz uvjeta komplementarne labavosti:

$$\alpha_i(y^{(i)}h(\mathbf{x}^{(i)}) - 1) = 0$$

slijedi da za svaki primjer $\mathbf{x}^{(i)}$ iz \mathcal{D} vrijedi $\alpha_i = 0$ ili $y^{(i)}h(\mathbf{x}^{(i)}) = 1$. U prvom slučaju, ako $\alpha_i = 0$, onda taj vektor ne figurira u izračunu funkcije h . U drugom slučaju, ako $y^{(i)}h(\mathbf{x}^{(i)}) = 1$, onda to znači da primjer leži točno na margini (prisjetite se, $y^{(i)}h(\mathbf{x}^{(i)}) = 1$ je upravo bio uvjet koji smo definirali za primjere koji su najbliži hiperravnini, i to su primjeri koji leže na margini). Prema tome, iz ova dva slučaja slijedi da se u dualnoj formulaciji kao pribrojnici pojavljuju samo vektori koji leže točno na margini. Te vektore nazivamo **potporni vektori** (engl. *support vectors*). Svi ostali vektori za koje $\alpha_i = 0$ uopće ne utječu na izlaz modela i možemo ih zanemariti kada radimo predikciju.

Alternativni pogled na ovo jest da je hiperravnina (u primarnom problemu) definirana linearnom kombinacijom potpornih vektora (u dualu). Ovo je vrlo bitno. To znači da će SVM model zapravo biti vrlo učinkovit kod predikcije. U mnogo slučajeva, umjesto da pohranjujemo vrlo mnogo težina \mathbf{w} , bit će dovoljno pohraniti manji broj potpornih vektora i pripadnih parametara $\boldsymbol{\alpha}$. Efektivno ćemo dobiti **rjetki model**.

Sažetak

- Stroj potpornih vektora je **linearan model** s **maksimalnom marginom** između primjera dviju klasa, što daje **dobru generalizaciju**
- Problem se svodi na **konveksnu optimizaciju s ograničenjima (kvadratno programiranje)**
- Primjenom **Lagrangeove dualnosti** problem se iz **primarne formulacije** može prebaciti u **dualnu formulaciju**
- Dualna optimizacijski problem učinkovito je rješiv algoritmom **SMO**
- Predikcija se radi na temelju **usporedbe** ulaznog primjera i odabranih označenih primjera, tzv. **potpornih vektora**

Bilješke

1 Pokušajmo odmah razriješiti misterij ovog čudnog naziva: zašto **stroj potpornih vektora**? Prvo, zašto "stroj". To nije skroz jasno. SVM tu nije usamljen, ima još strojeva u strojnom učenju, npr. Boltzmannov stroj (nećemo raditi), jezgreni stroj (radit ćemo), Helmholtzov stroj (nećemo raditi). Ovdje su dva objašnjenja: <https://stats.stackexchange.com/q/261041/93766>. Prema prvom, "stroj" dolazi jednostavno od "strojnog učenja", a prema drugom od povijesne povezanosti algoritama i strojeva, jer su u ranim danima računarske znanosti mnogi algoritmi doista bili fizički implementirani kao specijalizirani strojevi (slično bi se moglo reći i za "Turingov stroj"). Ovo se pitanje nedavno raspravljalo i na Twitteru: https://twitter.com/sam_power_825/status/1314921722630504448 (hvala Domagoju na informaciji). Što se drugog dijela naziva tiče, "potporni vektori", to je lako objasniti jednom kada se shvati kako algoritam radi, i to ćemo napraviti danas. No, objasnimo ovdje za nestrepljive barem ukratko ideju. SVM je linearan klasifikacijski model koji dakle primjere dviju klasa razdvaja hiperravninom. Tu hiperravninu možemo definirati vektorom težina \mathbf{w} , kao što to radi npr. logistička regresija, ali se, alternativno, hiperravnina može definirati kao linearna kombinacija primjera \mathbf{x} iz skupa označenih primjera \mathcal{D} . Pritom nam ne trebaju svi primjeri iz \mathcal{D} , nego

samo neki. Ti neki primjeri koji nam trebaju zapravo su potpornji za tu hiperravninu, pa ih zato nazivamo **potporni vektori**.

- 2** SVM su osmisili ruski matematičari Vladimir Vapnik i Alexey Chervonenkis, i to još 1963. godine na "Institutu za upravljačku znanost" (Institut Problem Upravleniya) u Moskvi. Vapnik je potom u devedestima emigrirao u SAD i radio za AT&T Bell Labs u New Jerseyu, gdje je, u suradnji s kolegom Isabelle Guyon i kolegom Bernardom Boserom (u međuvremenu u braku), 1992. godine nadogradio ideju SVM-a s tzv. **jezgrenim funkcijama** (Boser et al., 1992). Jezgrena funkcija, o kojima ćemo mi pričati kroz dva tjedna, implicitan je način definiranja funkcije preslikavanja u prostor značajki više dimenzije, što je omogućilo primjenu SVM-a na nelinearne probleme. Daljnje proširenje bila je formulacija SVM-a s **mekom marginom** (Cortes and Vapnik, 1995), koju je Vapnik objavio zajedno s danskom znanstvenicom Corinnom Cortes 1995. godine, a koja je u to doba također radila u AT&T Bell Labs. SVM s mekom marginom omogućio je primjenu na (ne)linearno neodvojive probleme (probleme koji su linearne neodvojivi i probleme koji ostaju neodvojivi nakon preslikavanja u prostor značajki), i to je danas standardna formulacija SVM-a, koju ćemo i mi raditi. Inače, Vapnik od 2014. godine radi u Facebookovom istraživačkom centru FAIR, zajedno sa jakim imenima dubokog učenja, dok je Chervonenkis nažalost prije nekoliko godina tragično skončao u močvari u okolini Moskve.
- 3** Dokaz da maksimalna margina smanjuje pogrešku generalizacije temelji se na dokazivanju gornje ograde na složenost modela karakterizane pomoću tzv. **Vapnik-Chervonenkisove dimenzije (VC-dimenzije)**. VC-dimenziju osmislili su 1974. godine upravo Vladimir Vapnik i Alexey Chervonenkis, tvorci izvirne ideje SVM-a. Pojednostavljeno, VC-dimenzijska je mjera složenosti modela definirana kao najveći broj primjera koje binarni klasifikator može točno razdvojiti, i to neovisno o označama tih primjera (tj. za sva moguća označavanja). Npr., VC-dimenzijska pravca u dvodimenzijskome ulaznom prostoru je 3, jer pravac može razdvojiti najviše tri primjera u dvodimenzijskom prostoru, ako u obzir uzmemos sve moguće označke tih primjera (kojih ukupno ima $2^3 = 8$). Ako dodamo četvrti primjer, onda imamo ukupno 16 mogućih označavanja, od kojih međutim 2 ne možemo razdvojiti pravcem (notorni "XOR" problem), što znači da VC-dimenzijska pravca nije veća od 3. Što je VC-dimenzijska veća, to je model veće složenosti. Više o VC-dimenzijskoj možete pročitati u poglavlju 2.2 u skripti, a mnogo detaljnije u izvrsnoj knjizi (Shalev-Shwartz and Ben-David, 2014) (poglavlje 6). Ideja dokaza da maksimalna margina SVM-a daje najbolju generalizaciju temelji se na tome da se pokaže da maksimalna margina ograničava VC-dimenzijsku SVM-a, što znači da ograničava njegovu složenost, a to znači da sprječava prenaučenost i time poboljšava generalizaciju. Detalji ovoga vrlo su lijepo i didaktično pojašnjeni u (Mount, 2015) (dostupno ovdje: https://winvector.github.io/margin_margin.pdf). U (Shalev-Shwartz and Ben-David, 2014) (dio 26.3) možete naći dokaz koji se temelji na alternativnoj karakterizaciji složenosti modela pomoću **Rademacherove složenosti**.

- 4** **Teorem o odvajanju hiperravninom** (engl. *hyperplane separation theorem*) glasi ovako: Neka su A i B dva disjunktna neprazna konveksna podskupa u \mathbb{R}^n . Onda postoji ne-nul vektor \mathbf{w} i realan broj c takav da:

$$\forall \mathbf{x}^A \in A, \forall \mathbf{x}^B \in B. (\mathbf{w}^T \mathbf{x}^A \geq c) \wedge (\mathbf{w}^T \mathbf{x}^B \leq c)$$

Pritom je $\mathbf{w}^T \mathbf{x} = c$, gdje je \mathbf{w} vektor normale, odvajajuća hiperravnina. U strojnom učenju, mi to uvijek formuliramo tako da je $c = 0$, tj. $\mathbf{w}^T \mathbf{x} = 0$. Dokaz i više detalja možete pronaći na https://en.wikipedia.org/wiki/Hyperplane_separation_theorem odnosno u poglavlju 2.5 u (Boyd et al., 2004)

Inače, autor teorema o odvajanju hiperravninom je poljsko-njemački matematičar Hermann Minkowski. Minkowski je inače najpoznatiji po svojoj geometrijskoj interpretaciji teorije relativnosti Alberta Einsteina (kojemu je bio profesor matematike dok je ovaj studirao u Zürichu, na današnjem ETH Zürich) iz 1907. godine u smislu četverodimenzijskog prostora u kojem se prostor i vrijeme isprepliću, tzv. prostorvrijeme ili prostorno-vremenski kontinuum (*njam. Minkowski-Raum*), a koju je Einstein potom ugradio u svoju opću teoriju relativnosti. U računarstvu, pak, Minkowski nam je više poznat po **Minkowskijevoj udaljenosti**, koja je generalizacija euklidiske udaljenosti i Manhattan-udaljenosti:

$$D(\mathbf{x}, \mathbf{y}) = \left(\sum_i |x_i - y_i|^p \right)^{\frac{1}{p}}$$

gdje za $p = 2$ i $p = 1$ dobivamo euklidsku udaljenost odnosno Manhattan-udaljenost. To je očigledno povezano s p -normom, o kojoj smo pričali u kontekstu regularizacije, i to u smislu da je Minkowskijeva

udaljenost između dvije točke jednaka p -normi razlike radijvektora tih točaka:

$$D(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p$$

Drugim riječima, regularizaciju p -normom vektora težina \mathbf{w} mogli smo definirati i kao Mikowskijevu udaljenost vektora težina \mathbf{w} od nul-vektora.

- 5** Definicije **konveksnog skupa** prisjetili smo se na pretprošlom predavanju u kontekstu definicije konveksne funkcije (domena konveksne funkcije mora biti konveksni skup). Prisjetimo se ovdje opet definicije konveksnog skupa. Skup A je konveksan ako i samo ako

$$\forall \mathbf{x}_1, \dots, \mathbf{x}_n \in A, \forall \alpha_1, \dots, \alpha_n. \sum_i \alpha_i = 1 \quad \text{vrijedi} \quad \sum_{i=1}^n \alpha_i \mathbf{x}_i \in A$$

tj. svaka linearna kombinacija točaka iz A je i sama unutar A .

- 6** Formalno (Boyd et al. (2004), poglavlj 2.1.4), **konveksna ljska** skupa C , označena s $\text{conv } C$, je skup svih konveksnih kombinacija točaka iz C :

$$\text{conv } C = \left\{ \alpha_1 x_1 + \dots + \alpha_k x_k \mid x_i \in C, \alpha_i \geq 0, i = 1, \dots, k, \sum_i \alpha_i = 1 \right\}$$

- 7** Ili n -dimenzijska ravnina ugrađena u $(n+1)$ -dimenzijski prostor, ako uvedemo "dummy" značajku $x_0 = 1$ i težinu w_0 uključimo u vektor \mathbf{w} , tj. ako model definiramo kao homogenu funkciju umjesto kao afinu funkciju. No ova razlika je nebitna; nastavljamo s izdvojenom težinom w_0 jer će nam tako biti lakše.

- 8** Funkcija predznaka ovdje je (što je uobičajeno u strojnom učenju) definirana kao:

$$\begin{aligned} \text{sgn} : \mathbb{R} &\rightarrow \{-1, +1\} \\ \text{sgn}(x) &= \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases} \end{aligned}$$

a ne sa trećom vrijednošću, $\text{sgn}(0) = 0$, kako je ta funkcija uobičajeno definirana u matematici.

- 9** Već smo bili spomenuli, kada smo pričali o linearnim diskriminativnim modelima, da množenje težina (w_0, \mathbf{w}) s faktorom $\alpha > 0$ nema utjecaja na položaj i orientaciju hiperravnine, niti na udaljenost primjera od hiperravnine. Udaljenosti su nepromijenjene budući da:

$$d = \frac{h(\mathbf{x}; \alpha \mathbf{w})}{\|\alpha \mathbf{w}\|} = \frac{\alpha \mathbf{w}^T \mathbf{x} + \alpha w_0}{\|\alpha \mathbf{w}\|} = \frac{\alpha (\mathbf{w}^T \mathbf{x} + w_0)}{\alpha \|\mathbf{w}\|} = \frac{\mathbf{w}^T \mathbf{x} + w_0}{\|\mathbf{w}\|} = \frac{h(\mathbf{x}; \mathbf{w})}{\|\mathbf{w}\|}$$

- 10** Također primijetite da smo na ovaj način zapravo osigurali korespondenciju 1:1 između parametara hiperravnine (w_0, \mathbf{w}) i hipoteze h . Općenito vrijedi da jednu te istu funkciju h možemo definirati s beskonačno mnogo različitih težina (w_0, \mathbf{w}) , koje se razlikuju samo za konstantan faktor. Uvedenim ograničenjem, međutim, eliminirali smo sve vektore težina osim jednog (onog koji zadovoljava gornje ograničenje).

- 11** S obzirom da smo objasnili svaki korak u ovoj transformaciji, trebalo bi biti jasno da rješavamo isti optimizacijski problem od kojega smo krenuli, a to je maksimizacija margine uz pretpostavku linearne odvojivih klasa. Međutim, formulacija do koje smo došli opire se intuiciji. Naime, kako to da minimizacija druge norme vektora težina \mathbf{w} (sans w_0), uz navedena ograničenja, daje rješenje maksimalne margine? Pokušajmo to objasniti. Prvo opažanje je da, što je norma težina \mathbf{w} manja (tj. što je vektor normale kraći), to su primjeri za koje $yh(\mathbf{x}) = 1$ dalje od hiperravnine. Zašto je to tako? Zato što je izlaz hipoteze h jednak $\mathbf{w}^T \mathbf{x}$, pa će za neki fiksni \mathbf{x} izlaz biti to manji što je \mathbf{w} manji, a to znači da se primjer za koji $yh(\mathbf{x}) = 1$ onda nalazi dalje od hiperravnine. To zapravo znači da minimizacijom od \mathbf{w} mi efektivo širimo marginu. Naravno, tu su i ograničenja, i ona zahtijevaju da za sve primjere vrijedi $yh(\mathbf{x}) \geq 1$, što znači da se primjeri ne mogu nalaziti "unutar margine". Imamo, dakle, dva suprotstavljenja zahtjeva: s jedne strane, minimiziramo $\|\mathbf{w}\|$, čime širimo marginu,

s druge strane, imamo ograničenja koja sprječavaju da margina bude tako široka da u nju uđu neki primjeri. Posljedica ovoga je da dobivamo što širu marginu, ali točno takvu da su primjeri koji su najbliži margini oni za koje $yh(\mathbf{x}) = 1$, i svi primjeri su ispravno klasificirani. Ova opozicija između što šire marge i poštovanja ograničenja analogna je argmax-min kriteriju od kojega smo početno krenuli.

- 12** Nemojte da vas zbuni naziv “programiranje”. Ne misli se o programiranju u smislu kodiranja, u C-u ili nedajbože Javi. Radi se o **matematičkom programiranju**, što je stari naziv za **(matematičku optimizaciju)**.

- 13** Formalno, **konveksni optimizacijski problem** (engl. *convex optimization problem*) definiran je kao:

$$\begin{aligned} & \text{minimizirati} && f(\mathbf{x}) \\ & \text{uz ograničenja} && g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m \\ & && \mathbf{a}_i^T \mathbf{x} - b_i = 0, \quad i = 1, \dots, p \end{aligned}$$

gdje su ciljna funkcija f i ograničenja nejednakosti g_i **konveksne funkcije**, dok su ograničenja jednakosti h_i **afine funkcije**. U ovom slučaju, ograničenje na minimizaciju od $f(\mathbf{x})$ doista predstavlja ograničenje, jer maksimizacija od $-f(\mathbf{x})$ više nije konveksan problem budući da funkcija $-f(\mathbf{x})$ nije konveksna nego konkavna. Međutim, u praksi nas maksimizacija konveksnih funkcija i minimizacija konkavnih funkcija ne zanimaju. Zanimaju nas minimizacija konveksnih funkcija i maksimizacija konkavnih funkcija, a to je obuhvaćeno gornjom definicijom konveksnoga optimizacijskog problema. Detaljnije u (Boyd et al., 2004) (poglavlje 4).

- 14** Formalno, **kvadratni program** (engl. *quadratic program, QP*) definiran je kao:

$$\begin{aligned} & \text{minimizirati} && \frac{1}{2} \mathbf{x}^T \mathbf{P} \mathbf{x} + \mathbf{q}^T \mathbf{x} + r \\ & \text{uz ograničenja} && \mathbf{G} \mathbf{x} \leq \mathbf{h} \\ & && \mathbf{A} \mathbf{x} = \mathbf{b} \end{aligned}$$

gdje je \mathbf{P} simetrična i realna matrica dimenzija $n \times n$, \mathbf{q} je n -dimenzijski vektor realnih brojeva, \mathbf{G} je realna matrica dimenzija $n \times m$, a \mathbf{A} je realna matrica dimenzija $p \times n$. Notacija $\mathbf{G} \mathbf{x} \leq \mathbf{h}$ označava da su sve komponente vektora $\mathbf{G} \mathbf{x}$ manje od ili jednake njima odgovarajućim komponentama vektora \mathbf{h} . Primjetite da to zapravo znači da su ograničenja nejednakosti (isto kao i ograničenja jednakosti) linearna (odnosno affine funkcije). Detaljnije o kvadratnom programiranju možete pročitati u (Boyd et al., 2004) (poglavlje 4).

Lako se pogubiti među svim ovim varijantama optimizacijskih problema; u snalaženju može pomoći ova taksonomija optimizacijskih problema: <https://neos-guide.org/content/optimization-taxonomy>. U toj taksonomiji, kvadratno programiranje nalazimo kao deterministički, kontinuirani, ograničeni optimizacijski problem s linearnim ograničenjima. Druga vrsta konveksnog optimizacijskog problema, koji je jednostavniji od kvadratnog programa, a koji se u računarstvu također često koristi, je **linearni program**, kod kojega je ciljna funkcija linearna (ograničenja su istog oblika kao i za kvadratni program).

- 15** Metodu **Lagrangeovih mnoštavki** osmislio je talijanski matematičar i astronom Joseph-Louis Lagrange u 18. stoljeću. Lagrange je jedan od najvećih matematičara, poznat po nizu važnih doprinosa matematici (u diferencijalnom računu, varijacijskom računu i teoriji brojeva), fizici (teorijska mehanika) i astronomiji (problem triju tijela). Lijep opis Lagrangeova života i stvaralaštva možete pročitati ovdje: <https://www.famousscientists.org/joseph-louis-lagrange/>. Langrangea smo već bili spomenuli u kontekstu regresije i Lagrangeovog interpolacijskog teorema.

- 16** **Karush-Kuhn-Tuckerovi (KKT) uvjeti** (engl. *Karush-Kuhn-Tucker (KKT) conditions*) su nužni (ali ne i dovoljni) uvjeti za optimalnost rješenja nelinearnog optimizacijskog problema, pod uvjetom da optimizacijski problem zadovoljava tzv. **uvjeti regularnosti** (engl. *regularity conditions*) (koji ovise o

konkretnom optimizacijskom problemu). Uvjeti KKT su:

$$\begin{aligned} g_i(\mathbf{x}) &\leq 0, \quad i = 1, \dots, m \\ h_i(\mathbf{x}) &= 0, \quad i = 1, \dots, p \\ \alpha_i &\geq 0, \quad i = 1, \dots, m \\ \alpha_i g_i(\mathbf{x}) &= 0, \quad i = 1, \dots, m \end{aligned}$$

Prva dva uvjeta su uvjeti **primarne ostvarivosti** (engl. *primal feasibility*), treći uvjet je uvjet **dualne ostvarivosti** (engl. *dual feasibility*), a četvrti uvjet je već spomenuti uvjet **komplementarne lababosti** (engl. *complementarity slackness*).

Uvjeti KKT su nužni i *dovoljni* uvjeti za optimalnost rješenja ako je optimizacijski problem konveksan (tj. ciljna funkcija f i funkcije ograničenja nejednakosti g_i su konveksne, a funkcije ograničenja jednakosti h_i su afine) te su funkcije g_i derivabilne. KKT uvjeti zapravo proširuju metodu Lagrangeovih multiplikatora, koja dopušta samo ograničenja jednakosti. Budući da mi ovdje koristimo i uvjete nejednakosti, točnije bi bilo reći da koristimo optimizacijsku metodu KKT, a ne metodu Lagrangeovih multiplikatora.

Uvjete KKT predložili su matematičari Harold Kuhn i Albert Tucker 1951. godine, ali ih je William Karush objavio još kao student desetak godina ranije u svojem diplomskom radu (za što je nagrađen s prvim "K" u kratici KKT).

- 17 Kod konveksnih optimizacijskih problema, uvjet regularnosti (koji je preduvjet za uvjete KKT) jest da postoji točka \mathbf{x} takva da $h(\mathbf{x}) = 0$ i $g_i(\mathbf{x}) < 0$, gdje je h ograničenje jednakosti, a g_i ograničenje nejednakosti. To je tzv. **Slaterov uvjet regularnosti**. Slaterovi uvjeti ujedno su dovoljni uvjeti za jaku dualnost konveksnog optimizacijskog problema.
- 18 Za nestrpljive: rješavanje problema maksimalne margine SVM-a u dualnoj formulaciji ima tri važne prednosti nad rješavanjem tog problema u primarnoj formulaciji. Prvo, u dualnoj formulaciji može se primijeniti algoritam **slijedne minimalne optimizacije** (engl. *sequential minimal optimization, SMO*), koji je učinkovit algoritam specijaliziran za kvadratno programiranje baš za SVM. (Mi, nažalost, nećemo imati vremena pričati o tom algoritmu.) Drugo, u dualnoj formulaciji model ćemo moći definirati preko tzv. **potpornih vektora**, što ima svoje prednosti u pogledu računalne učinkovitosti. Treće, u dualnoj formulaciji moći ćemo koristiti tzv. **jezgreni trik**, i time zaobići potrebu da ručno i eksplicitno definiramo funkciju preslikavanja. Mogućnost korištenja jezgrenog trika zapravo je razlog zašto je SVM tako moćan i popularan algoritam.
- 19 Algoritam **slijedne minimalne optimizacije** (engl. *sequential minimal optimization, SMO*) osmislio je američki računalni znanstvenik John Platt 1998. godine (Platt, 1998). Platt, koji trenutačno radi u Googleu, osim po algoritmu SMO i tome da je otkrio dva asteroida promjera oko 25 km, poznat je i po metodi **Plattovog skaliranja**, kojom omogućava da se izlaz SVM-a vrlo jednostavno transformira u probabilistički izlaz u intervalu $(0, 1)$. Više o tome idući put.
- 20 Težinu w_0 možemo izračunati na temelju činjenice da za potporne vektore vrijedi $y^{(i)}h(\mathbf{x}^{(i)}) = 1$. Neka je S skup indekasa potpornih vektora $\mathbf{x}^{(i)}$. Ako u $y^{(i)}h(\mathbf{x}^{(i)}) = 1$ uvrstimo

$$h(\mathbf{x}) = \sum_{i=1}^N \alpha_i y^{(i)} (\mathbf{x})^T \mathbf{x}^{(i)} + w_0$$

onda dobivamo:

$$y^{(i)}h(\mathbf{x}^{(i)}) = y^{(i)} \left(\sum_{j \in S} \alpha_j y^{(j)} (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)} + w_0 \right) = 1$$

Iz toga dobivamo:

$$w_0 = y^{(i)} - \sum_{j \in S} \alpha_j y^{(j)} (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)}$$

pri čemu smo iskoristili $1/y^{(i)} = y^{(i)}$ jer $y^{(i)} \in \{-1, +1\}$. Ovu jednadžbu možemo izračunati na temelju jednog, proizvoljno odabranog označenog primjera $(\mathbf{x}^{(i)}, y^{(i)})$. Međutim, zbog numeričkih odstupanja nećemo za svaki odabir dobiti isto rješenje, pa je bolje izračunati prosjek nad svim potpornim vektorima:

$$w_0 = \frac{1}{|S|} \sum_{i \in S} \left(y^{(i)} - \sum_{j \in S} \alpha_j y^{(j)} (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)} \right)$$

Literatura

- B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.
- S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- J. Mount. How sure are you that large margin implies low vc dimension? *Win-Vector Blog*, 2015.
- J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.
- S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

9. Stroj potpornih vektora II

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v2.7

Zadnji put bavili smo se **strojem potpornih vektora** te smo definirali njegov model i optimizacijski postupak. Cilj optimizacije bio je nalaženje hiperravnine koja **maksimizira marginu** između primjera dviju klase, jer maksimalna margina daje najbolju generalizaciju. Taj se optimizacijski problem sveo na **kvadratno programiranje**, koje smo riješili metodom Lagrangeovih multiplikatora. Također smo pogledali kako se iz primarne formulacije problema možemo prebaciti u **dualnu**. U konačnici smo tako dobili dualni model SVM-a, gdje se predikcija za novi primjer ne radi na temelju težina značajki, nego na temelju sličnosti primjera s nekim prototipnim primjerima, koje smo nazvali **potporna vektori**.

Danas nastavljamo tamo gdje smo stali zadnji put. Glavne dvije teme kojima ćemo se baviti jesu **meka margin** i **gubitak zglobnice**. Prva se tema tiče relaksacije pretpostavke o linearnoj odvojivosti primjera, koje smo se držali prošli put, ali koja je vrlo nerealna. Ova izmjena iziskivat će da malo promijenimo optimizacijski problem, ali je dobra vijest da je ta promjena trivijalna i da ne moramo angažirati nikakvu novu matematičku artiljeriju. Druga se tema tiče alternativnog pogleda na SVM, gdje – umjesto da problem maksimalne margine formaliziramo kao optimizaciju uz ograničenja – optimizaciju provodimo na način koji je sličniji onome što smo radili do sada: gradijentnim spustom po funkciji pogreške definiranoj kao očekivanje funkcije gubitka. Na koncu ćemo pogledati još neke važne detalje u vezi s primjenom SVM-a u praksi.

1 Podsjetnik

Prisjetimo se najprije što smo sve naučili zadnji put. Krenuli smo od pretpostavke da su primjeri iz skupa za učenje linearno odvojivi. Zatim smo definirali problem **maksimalne margine**, a onda ga preformulirali kao problem **optimizacije uz ograničenja**. Ograničenja smo definirali tako da za primjere koji su najbliži hiperravnini (primjeri na margini) vrijedi $y\hat{h}(\mathbf{x}) = 1$:

► Maksimalna margin

Početni problem:

$$\underset{\mathbf{w}, w_0}{\operatorname{argmax}} \left\{ \frac{1}{\|\mathbf{w}\|} \min_i \{y^{(i)}(\mathbf{w}^T \mathbf{x} + w_0)\} \right\}$$

Reformulirani problem:

$$\underset{\mathbf{w}, w_0}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{w}\|^2$$

uz ograničenja:

$$y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \geq 1, \quad i = 1, \dots, N$$

Zaključili smo da je ovo **kvadratni program**, budući da je ciljna funkcija konveksna a ograničenja nejednakosti su afne funkcije. Taj se problem može riješiti metodom **Lagrangeovih multiplikatora**, kod koje ograničenja ugrađujemo u ciljnu funkciju, tzv. **Lagrangeovu funkciju**, koja sadrži dodatne (dualne) varijable:

► Maksimalna margina – primarni problem

$$\operatorname{argmin}_{\mathbf{w}, w_0} \frac{1}{2} \|\mathbf{w}\|^2$$

uz ograničenja:

$$y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \geq 1, \quad i = 1, \dots, N$$

Lagrangeova funkcija:

$$L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i \left\{ y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1 \right\}$$

gdje $\alpha_i \geq 0$. Minimizator (\mathbf{w}^*, w_0^*) je stacionarna točka (sedlo) od L . U toj točki vrijede uvjeti KKT:

$$\begin{aligned} y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) &\geq 1, & i = 1, \dots, N \\ \alpha_i &\geq 0, & i = 1, \dots, N \\ \alpha_i (y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1) &= 0, & i = 1, \dots, N \end{aligned}$$

Ovo je **primarna** formulacija problema, čije je rješenje minimum po primarnim varijablama (\mathbf{w}, w_0) te maksimum po dualnim varijablama $\boldsymbol{\alpha}$, tj. sedlo Lagrangeove funkcije. Međutim, kod SVM-a iz više nam je razloga bolje preći u **dualnu** formulaciju problema. To radimo tako da najprije definiramo **dualnu Langrangeovu funkciju**:

► Maksimalna margina – prijelaz u dual

$$L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i \left\{ y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1 \right\}$$

$$\begin{aligned} \frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\alpha})}{\partial \mathbf{w}} &= 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)} \\ \frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\alpha})}{\partial w_0} &= 0 \quad \Rightarrow \quad \sum_{i=1}^N \alpha_i y^{(i)} = 0 \end{aligned}$$

Dualna Lagrangeova funkcija:

$$\tilde{L}(\boldsymbol{\alpha}) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)}$$

Dualna Lagrangeova funkcija je funkcija isključivo dualnih varijabli $\boldsymbol{\alpha}$ i ona nam za zadane vrijednosti dualne varijable daje minimum Lagrangeove funkcije po primarnim varijablama (\mathbf{w}, w_0) . S obzirom da je dualna Lagrangeova funkcija donja ograda primarnog (minimizacijskog) problema, te da vrijedi jaka dualnost (procijep dualnosti jednak je nuli), to je rješenje primarnog problema istovjetno maksimumu dualne Lagrangeove funkcije po dualnim varijablama

α , uz određena ograničenja na te dualne varijable. Konkretno, dualna formulacija optimizacijskog problema maksimalne marge je:

► **Maksimalna marga – dualni problem**

$$\underset{\alpha}{\operatorname{argmax}} \left(\sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)} \right)$$

uz ograničenja:

$$\alpha_i \geq 0, \quad i = 1, \dots, N$$

$$\sum_{i=1}^N \alpha_i y^{(i)} = 0$$

U točki rješenja vrijede uvjeti KKT:

$$\begin{aligned} y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) &\geq 1, & i = 1, \dots, N \\ \alpha_i &\geq 0, & i = 1, \dots, N \\ \alpha_i (y^{(i)} h(\mathbf{x}^{(i)}) - 1) &= 0, & i = 1, \dots, N \end{aligned}$$

Ovaj optimizacijski problem opet je problem kvadratnog programiranja, ovaj put uz ograničenja nejednakosti i ograničenja jednakosti. Prednost ovakve formulacije (između ostalog) jest da se može učinkovito riješiti algoritmom **slijedne minimalne optimizacije (SMO)**, o kojem, međutim, nismo pričali (niti nećemo). Konačno, utvrdili smo da između modela primarne i dualne formulacije SVM-a možemo uspostaviti sljedeću vezu:

► **Primarni i dualni model**

$$\begin{aligned} \mathbf{w} &= \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)} \\ h(\mathbf{x}) &= \underbrace{\mathbf{w}^T \mathbf{x} + w_0}_{\text{Primarno}} = \underbrace{\sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^T \mathbf{x}^{(i)}}_{\text{Dualno}} + w_0 \end{aligned}$$

Veza se temelji na činjenici da vektor težina \mathbf{w} možemo napisati kao linearu kombinaciju vektora \mathbf{x} , pomnoženih s njihovom oznakom $y^{(i)}$. U linearnej kombinaciji sudjeluju samo oni vektori za koje $\alpha_i > 0$, i te vektore nazivamo **potporni vektori**.

2 Meka marga

Rješenje za vektor α bit će takvo rješenje koje maksimizira marginu uz pretpostavku **linearno odvojivih** primjera. Prisjetimo, tu smo pretpostavku kodirali ovim ograničenjima:

$$y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \geq 1, \quad i = 1, \dots, N$$

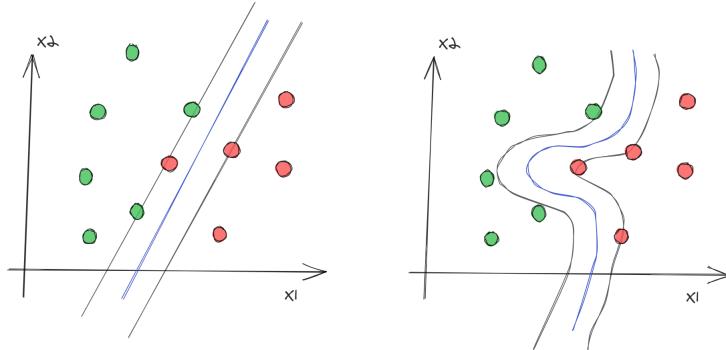
No što ako primjeri nisu linearno odvojivi? Tada jednostavno ne postoji rješenje koje bi zadovoljilo gornji uvjet, pa je ostvarivo područje prazan skup! Drugim riječima, ne postoji hiperravnina koja točno odvaja primjere.

To nije dobro. Ne želimo situaciju u kojoj nemamo nikakvo rješenje. Npr., sjetite se algoritma logističke regresije: ako primjeri nisu linearno odvojivi, ipak ćemo dobiti neko rješenje, i to ono koje minimizira empirijsku pogrešku na skupu za učenje. (Kod perceptronu nećemo dobiti nikakvo rješenje ako primjeri nisu linearno odvojivi, tj. algoritam perceptronu u tom slučaju ne konvergira, ali to smo već ustanovili da je nedostatak koji ne možemo prihvati.) Kako bismo to riješili? Ako primjeri nisu linearno odvojivi, želimo da nam algoritam SVM-a, kao i algoritam logističke regresije, da onu hipotezu koja je najbolja moguća, kad već ne postoji savršena hipoteza.

Vjerojatno prva stvar koja nam pada na pamet jest da preslikamo primjere pomoću funkcije ϕ u prostor značajke više dimenzije, kao što smo radili do sada. To je vrlo dobra ideja, i to ćemo svakako raditi, jer ćemo na taj način, kao i do sada, iz linearog modela efektivno dobiti nelinearan model. Međutim, to nije principijelno rješenje, i to iz dva razloga.

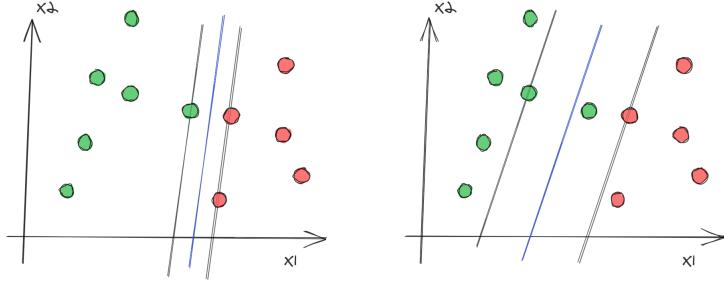
Prvi problem je da se vrlo lako može dogoditi da ne uspijemo primjere preslikati u prostor gdje bi oni bili linearno odvojivi. Općenito nemamo garanciju da ćemo primjere uspjeti preslikati u prostor značajki u kojem su oni linearno odvojivi, odnosno općenito ne znamo koju funkciju ϕ trebamo odabrati, a da bi preslikavanje rezultiralo linearno odvojivim problemom.

Drugi problem je: želimo li stvarno bilo koji ulazni problem preslikati u prostor gdje je on linearno odvojiv? Što ako u ulaznom prostoru postoji **šum**, i dvije klase zbog šuma nisu linearno odvojive? Ako ih preslikamo u toliko visokodimenzionalni prostor da one postaju odvojive, sigurno ćemo dobiti **prenaučen** model! Pogledajmo primjer takve situacije:



Ako je najlijeviji crveni primjer na lijevoj slici šum (bilo pogrešno označen ili pogrešno smješten), onda ne bismo željeli da taj primjer utječe na granicu i marginu, i idealno bi bilo da granica i margina izgledaju ovako kako je prikazano na lijevoj slici. Ako, međutim, primjere preslikamo u prostor značajki dovoljno visoke dimenzije i tamo ih odvojimo linearnom granicom, onda će u ulaznom prostoru granica biti visoko nelinarna i previše prilagođena šumu, kao što prikazuje desna slika.

No, čak i ako primjeri jesu linearno odvojivi u ulaznom prostoru, trenutačni postupak insistira na nalaženju maksimalne margine, čak i onda kada, opet zbog šuma, to možda nije najpametnije:



Na lijevoj slici imamo zeleni primjer koji je šum (bilo pogrešno označen ili pogrešno smješten). Primjeri su, međutim, linearno odvojivi, i granica maksimalne margine je poprilično uska. Na desnoj je slici prikazana margina koja međutim dopušta da dotični zeleni primjer bude unutar margine (ovdje čak i na suprotnoj strani granice!). No, takva margina je znatno šira, što znači da će model bolje generalizirati.

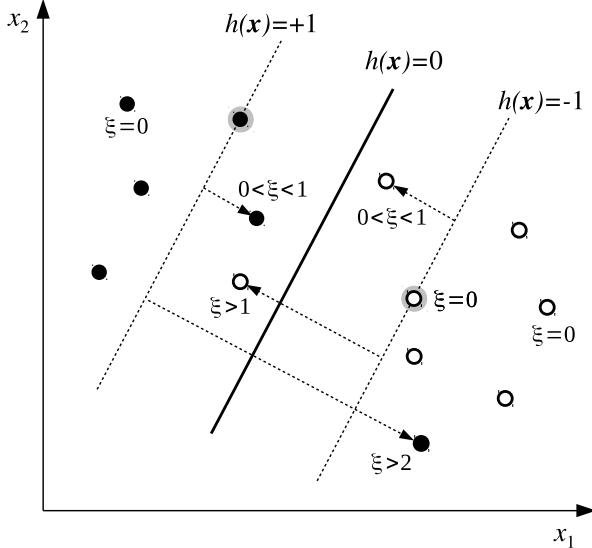
Dakle naš problem je što trenutna formulacija optimizacijskog postupka inzistira na linearnej odvojivosti primjera. Ako primjeri nisu linearno odvojivi, onda nema rješenja. Ili imamo rješenje maksimalne margine, ili nemamo nikakvo rješenje! Očito je da nam treba malo više fleksibilnosti. Sjetite se kako smo to radili do sada: željeli smo minimizirati empirijsku pogrešku, ali smo dopuštali da se primjeri pogrešno klasificiraju i to je uzrokovalo gubitke. Ideja je da istu stvar napravimo i ovdje: dopustit ćemo da primjeri ulijeću u marginu, pa čak i da budu s krive strane granice, ali ćemo onda malo kažnjavati hipotezu ako se to dogodi. Takva formulacija problema naziva se **meka margina** (engl. *soft margin*). Formulaciju koju smo imali prije, gdje nismo dopuštali nikakve pogreške, onda nazivamo **tvrda margina** (engl. *hard margin*). 1

2.1 Formulacija meke margine

Meku marginu ćemo vrlo jednostavno ostvariti tako da malo promijenimo optimizacijska ograničenja, na sljedeći način:

$$y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \geq 1 - \xi_i, \quad i = 1, \dots, N$$

Ovdje smo uveli tzv. **rezervnu varijablu** (engl. *slack variable*) ξ_i , po jednu za svaki primjer $\mathbf{x}^{(i)}$. Vrijedi $\xi_i \geq 0$. Ta nam varijabla govori koliko je primjer ušao u marginu ili čak prešao granicu. Naime, ako je $\xi_i = 0$, onda znači da je primjer na ispravnoj strani granice i izvan margine, i imamo istu situaciju (odnosno ograničenja) kao što smo imali ranije s tvrdom marginom. Međutim, ako je $\xi_i > 0$, onda znači da više ne vrijedi $y^{(i)}h(\mathbf{x}^{(i)}) \geq 1$, nego je sada $y^{(i)}h(\mathbf{x}) < 1$, a to znači da je primjer $\mathbf{x}^{(i)}$ ušao u marginu ili je čak prešao granicu. Konkretno, ako je $\xi_i = 1$, onda to znači da je $y^{(i)}h(\mathbf{x}^{(i)}) = 0$, tj. primjer je točno na granici. Nadalje, ako je $\xi_i > 1$, onda to znači da je $y^{(i)}h(\mathbf{x}^{(i)}) < 0$, što znači da je primjer na drugoj strani hiperravnine i da je pogrešno klasificiran. Navedene slučajeve ilustrira sljedeća skica:



Slika prikazuje dvodimenzionalni ulazni prostor s primjerima iz dviju klasa (crni i bijeli kružići). Granica između klasa je hiperravnina (ovdje pravac) za koji $h(\mathbf{x}) = 0$. Primjeri za koje $h(\mathbf{x}) = 1$ i $h(\mathbf{x}) = -1$ su potporni vektori i oni su pozicionirani točno na margini. Na ovoj slici imamo dva potpora vektora, po jedan iz svake klase. Za potporne vektore vrijedi $\xi_i = 0$ i $y h(\mathbf{x}) = 1$. Za primjere koji su izvan margine, a točno klasificirani, vrijedi $\xi_i = 0$ i $y h(\mathbf{x}) > 1$. Za primjere koji su ušli u marginu, ali su još uvijek na ispravnoj strani granice, vrijedi $0 < \xi_i < 1$, tj. ti primjeri će biti kažnjeni sa ξ_i koji je manji od jedinice. Na slici imamo dva takva primjera, po jedan iz svake klase. Primjeri koji pređu na krivu stranu hiperravnine imaju $\xi_i > 1$, tj. ti su primjeri kažnjeni još više. Na slici imamo jedan takav primjer koji je prešao u poluprostor pozitivne klase. Konačno, primjer koji pređe ne samo na pogrešnu stranu hiperravnine nego još izade i iz margine s pogrešne strane imat će $\xi_i > 2$. Na slici je to jedan primjer koji je prešao u poluprostor negativne klase. Ukupno, za četiri primjera vrijedi $\xi_i > 0$, odnosno četiri primjera ne poštuju ograničenja tvrde margine.

Pa, sažmimo: Ako je primjer $\mathbf{x}^{(i)}$ ispravno klasificiran, onda $\xi_i = 0$. Međutim, ako primjer $\mathbf{x}^{(i)}$ nije ispravno klasificiran, onda $\xi_i > 0$ i ξ_i je jednak odstupanju izlaza modela od točne klasifikacije, tj. $\xi_i = |y^{(i)} - h(\mathbf{x}^{(i)})|$. Primjeri koji su na pogrešnoj strani imat će $\xi_i > 1$. Primjeri koji su na ispravnoj strani granice (ili leže upravo na njoj), ali su ušli unutar margine, imat će $0 < \xi_i \leq 1$.

Ovime smo dobili željenu fleksibilnost. Ovako modificirana ograničenja dopuštaju da pojedini primjeri uđu unutar margine ili čak s druge strane granice. No, ne želimo da to bude masovna pojava. Želimo to dopustiti, ali ne previše. Ovdje, dakle, imamo dva kriterija: želimo maksimalnu marginu koja dopušta pogrešne klasifikacije i ulete u marginu, ali istovremeno želimo da je takvih situacija što manje. Ova dva kriterija međusobno su oprečna: možda možemo dobiti savršenu klasifikaciju, ali onda bi margin mogla biti vrlo uska. Obrnuto, možemo dobiti široku marginu, ako dopustimo ulaz mnogih primjera u marginu. Postavlja se pitanje kako ostvariti ovakav kompromis pri optimizaciji?

Prisjetimo se: naša ciljna funkcija koju smo minimizirali bila je:

$$\frac{1}{2} \|\mathbf{w}\|^2$$

i to daje maksimalnu, ali tvrdnu marginu. Kako ostvariti kompromis? Na isti način kao što smo to napravili kod regularizacije: linearom kombinacijom dvaju uvjeta. Ako dozvoljavamo pogreške, a pogreške modeliramo sa ξ_i , onda ćemo ciljnu funkciju redefinirati ovako:

$$\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i$$

Hiperparametar $C > 0$ određuje **kompromis** između širine margine i tvrdoće margine. Veći C dovodi do većeg kažnjavanja pogrešne klasifikacije, što će za posljedicu imati "čišću" odnosno tvrdju marginu. Manji C znači da nas je manje briga za ulete u marginu i za pogrešne klasifikacije, što će nam dati poroznu, ali široku marginu. Kako to utječe na **složenost modela**? Veći C daje tvrde margine, dakle složenije modele. Manji C daje mekše margine, dakle jednostavnije modele.

Napišimo sad u kompletu naš optimizacijski problem:

$$\underset{\mathbf{w}, w_0, \xi}{\operatorname{argmin}} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \right\}$$

uz ograničenja:

$$\begin{aligned} y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) &\geq 1 - \xi_i, & i = 1, \dots, N \\ \xi_i &\geq 0, & i = 1, \dots, N \end{aligned}$$

Primijetite da su varijable ξ_i također varijable po kojima optimiramo, i želimo da su one što manje, kako bi drugi pribrojnik u ciljnoj funkciji bio što manji. Međutim, to nisu parametri modela (za razliku od dualne varijable α , koja jest parametar modela u dualnoj formulaciji).

Ovimo smo definirali optimizacijski problem. Iduće pitanje na koje moramo odgovoriti jest: kako ovo optimirati?

2.2 Dualno kvadratno programiranje

Kao i kod tvrde margine, optimizacijski problem koji smo upravo definirali je **kvadratni program**. Kao i kod tvrde margine, prebacit ćemo se u **dualni problem**, zbog prednosti koje smo bili natuknuli prošli put. No, krenimo redom. Najprije definiramo **Lagrangeovu funkciju**:

$$L(\mathbf{w}, w_0, \xi, \alpha, \beta) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \alpha_i (y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1 + \xi_i) - \sum_{i=1}^N \beta_i \xi_i$$

Ova Lagrangeova funkcija ima primarne varijable \mathbf{w} , w_0 i ξ te dualne varijable α i β . Rješenje je stacionarna točka (sedlo) ove funkcije, koje je minimum po primarnim varijablama i maksimum po dualnim varijablama. U točki rješenja vrijedit će sljedećih šest uvjeta KKT:

$$\begin{aligned} y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) &\geq 1 - \xi_i & i = 1, \dots, N \\ \xi_i &\geq 0 & i = 1, \dots, N \\ \alpha_i &\geq 0 & i = 1, \dots, N \\ \beta_i &\geq 0 & i = 1, \dots, N \\ \alpha_i (y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1 + \xi_i) &= 0 & i = 1, \dots, N \\ \beta_i \xi_i &= 0 & i = 1, \dots, N \end{aligned}$$

Sada želimo preći u dualni problem. Za to moramo definirati **dualnu Lagrangeovu funkciju**, koja je funkcija dualnih varijabli i koja minimizira Lagrangeovu funkciju po primarnim varijablama. Dakle, prvo trebamo izračunati minimum Lagrangeove funkcije po primarnim varijablama. U tu svrhu deriviramo Lagrangeovu funkciju po \mathbf{w} , w_0 i ξ_i i izjednačavamo s nulom, što nam daje:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} = 0 &\Rightarrow \mathbf{w} = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)} \\ \frac{\partial L}{\partial w_0} = 0 &\Rightarrow \sum_{i=1}^N \alpha_i y^{(i)} = 0 \\ \frac{\partial L}{\partial \xi_i} = 0 &\Rightarrow \alpha_i = C - \beta_i \end{aligned}$$

Ove jednakosti sada uvrštavamo nazad u Lagrangeovu funkciju L , kako bismo dobili dualnu Lagrangeovu funkciju:

$$\tilde{L}(\boldsymbol{\alpha}) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)}$$

Primijetite da je ovo posve ista dualna Lagrangeova funkcija kao i ona za tvrdu marginu! Međutim, ono što je ovdje drugačije su ograničenja. Naime, imamo multiplikatore β_i , koje nismo imali kod tvrde margine. Budući da $\beta_i \geq 0$, a da smo gore izveli jednakost $\alpha_i = C - \beta_i$, to znači da:

$$\begin{aligned}\alpha_i &= C - \beta_i \\ \beta_i &= C - \alpha_i \geq 0 \\ \alpha_i &\leq C\end{aligned}$$

Također, otprije znamo da vrijedi $\alpha_i \geq 0$, pa zaključujemo da mora vrijediti ograničenje $0 \leq \alpha_i \leq C$. Dakle, naš dualni problem je:

$$\underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \left(\sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)} \right)$$

uz ograničenja:

$$\begin{aligned}0 &\leq \alpha_i \leq C, \quad i = 1, \dots, N \\ \sum_{i=1}^N \alpha_i y^{(i)} &= 0, \quad i = 1, \dots, N\end{aligned}$$

(Ograničenje jednakosti moramo uvrstiti jer nismo eliminirali varijablu α_i .) Primijetite da je jedina razlika u odnosu na tvrdu marginu ta što, pored donje ograde, postoji i gornja ograda na α , tj. ograničenje $\alpha_i \leq C$. I ponovo se ovdje radi o kvadratnom programiranju, koje se može učinkovito (u $\mathcal{O}(N^2)$) riješiti algoritmom **slijedne minimalne optimizacije (SMO)**.

2.3 Predikcija

Kao što smo ustanovili zadnji put, u ovoj dualnoj formulaciji, jednom kada je model naučen i kada imamo izračunat vektor $\boldsymbol{\alpha}$, za klasifikaciju novog primjera koristimo izraz:

$$h(\mathbf{x}; \boldsymbol{\alpha}, \mathcal{D}) = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^T \mathbf{x}^{(i)} + w_0$$

Prisjetimo se, predikcija funkcioniра tako da primjer \mathbf{x} uspoređujemo po sličnosti s prototipnim primjerima iz \mathcal{D} . Ta se uporedba čini pomoću skalarnog produkta $\mathbf{x}^T \mathbf{x}^{(i)}$. Rezultat toga jest da će prototipni primjeri koji su najsličniji ulaznom primjeru imati najviše utjecaja na njegovu klasifikaciju. Prototipni primjeri su oni za koje je $0 < \alpha_i < C$, i njih zovemo **potporni vektori**. Iz uvjeta KKT slijedi, pomalo protivno intuiciji, da potporni vektori za koje $\alpha_i < C$ leži na margini, dok potporni vektori za koje $\alpha_i = C$ mogu ležati i unutar margini.

Primijetite da se, kao i kod tvrde margine, u dualnom modelu SVM-a više ne pojavljaju težine \mathbf{w} . Parametar modela više nije n -dimenzijski vektor težina, nego su parametri N -dimenzijski vektor $\boldsymbol{\alpha}$ i pripadni potporni vektori, odnosno označeni primjeri $(\mathbf{x}_i, y^{(i)})$ za koje $\alpha_i > 0$. Pritom, kao i kod tvrde margine, parametar w_0 možemo izračunati pomoću potpornih vektora, dakle w_0 nam ne treba kao zaseban parametar.

3 Gubitak zglobnice

Naučili smo već mnogo toga o SVM-u. Pristup koji smo do sada razmatrali, a koji se svodi na formalizaciju problema maksimalne margine kao problema optimizacije uz ograničenja, standardni je pristup i on odgovara povijesnom razvoju SVM-a. Primijetite, međutim, da u tom pristupu uopće nismo pričali o funkciji gubitka niti o pogrešci, kao što smo to radili kod pretvodnih algoritama, npr. linearne ili logističke regresije, već smo izravno definirali optimizacijski problem kao kvadratni program. Međutim, sada kada smo objasnili osnovne principe SVM-a, možemo razmotriti i alternativnu formulaciju optimizacijskog problema SVM u smislu gubitka i funkcije pogreške. To će nam omogućiti dvije stvari. Prvo, da SVM ne rješavamo kvadratnim programiranjem, nego npr. nekom varijantom **gradijentnog spusta**. I drugo, da SVM izravno usporedimo s drugim linearnim modelima koje smo već upoznali.

Krenimo od toga da se prisjetimo kako izgleda ciljna funkcija **primarnog** optimizacijskog problema za slučaj meke margine:

$$\underset{\mathbf{w}, w_0, \xi}{\operatorname{argmin}} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \right\}$$

Zastanite ovdje na trenutak i pogledajte još jednom ciljnu funkciju koju smo upravo napisali. Izgleda li vam ova funkcija poznato? Sjetite se, npr., logističke regresije. Pa, ovo izgleda baš kao **L2-regularizirana empirijska pogreška**. Naime, nju smo definirali ovako:

$$E_R(\mathbf{w} | \mathcal{D}) = \sum_{i=1}^N L(y^{(i)}, h(\mathbf{x}^{(i)}; \mathbf{w})) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

Ako ovdje stavimo $\lambda = 1/C$, dobit ćemo (do na faktor C) identičan izraz gornjoj ciljnoj funkciji!

Prema tome, možemo zaključiti da je ciljna funkcija primarnog optimizacijskog problema zapravo L2-regularizirana funkcija empirijske pogreške! Pritom izraz $\frac{1}{2} \|\mathbf{w}\|^2$, kojim osiguravamo da margina bude što šira, zapravo odgovara **regularizacijskom izrazu**, dok izraz $C \sum \xi_i$, kojime kažnjavamo primjere koji ulaze u marginu, odgovara **funkciji gubitka**. Ova prva korespondencija vjerojatno više iznenađuje nego ova druga. Međutim, da tendencija za širom marginom odgovara regularizaciji već smo bili napomenuli, budući da je cijela ideja maksimalne margine upravo ostvariti veću generalizaciju modela, a to znači manju sklonost prenaučenosti, što znači manju složenost, i to upravo ostvarujemo regularizacijom. Dakle, kada malo razmislimo, obje korespondencije imaju smisla. Međutim, ono što je zanimljivo je razlika u načinu kako smo došli do izraza za ciljnu funkciju SVM-a i za L2-regulariziranu pogrešku linearnog modela. Naime, izraz koji odgovara regularizacijskom faktoru imali smo u priči od SVM-a otpočetka, jer je taj izraz bio sastavni dio kriterija maksimalne margine, dok smo izraz s rezervnim varijablama dodali naknadno, kako bismo ostvarili meku marginu. Kod poopćenih linearnih modela išli smo obrnuto: prvo smo definirali empirijsku pogrešku kao očekivanje funkcije gubitka, a tek potom dodali regularizacijski izraz.

Vratimo se ciljnoj funkciji SVM-a i pokušajmo je napisati kao L2-regulariziranu pogrešku. Vidimo da regularizacijski faktor već imamo. Pogledajmo možemo li nekako napisati ovaj ξ_i u obliku funkcije gubitka. Prisjetimo se da za primjere koji su na ispravnoj strani granice i izvan margine vrijedi $y^{(i)}h(\mathbf{x}^{(i)}) \geq 1$ te da ti primjeri ne nanose gubitak. Primjere koji su unutar margine ili na pogrešnoj strani granice nanose gubitak $\xi_i = |y^{(i)} - h(\mathbf{x}^{(i)})|$, što je jednako $1 - y^{(i)}h(\mathbf{x}^{(i)})$. To onda znači da funkciju gubitka možemo definirati kao:

$$L(y, h(\mathbf{x})) = \max(0, 1 - yh(\mathbf{x})).$$

Ako je $yh(\mathbf{x}) \geq 1$, onda će $1 - yh(\mathbf{x}) \leq 0$, pa je gubitak 0. Funkcija max pritom osigurava da gubitak ne postane negativan. Ako je $yh(\mathbf{x}) < 1$, onda onda je gubitak veći od nule.

Kao i inače, funkcija gubitka L je funkcija dvije varijable. Kao što smo to radili do sada, radi usporedbe s drugim funkcijama gubitka, preoblikovat ćemo ju u funkciju jedne varijable, $y\mathbf{w}^T \mathbf{x}$. Ovdje je to jednostavno:

$$L(y\mathbf{w}^T \mathbf{x}) = \max(0, 1 - y\mathbf{w}^T \mathbf{x}).$$

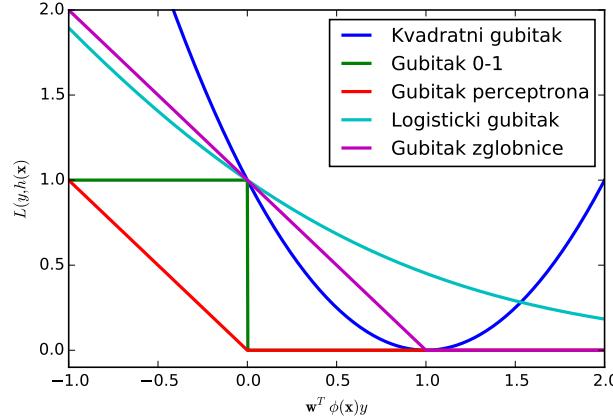
Ovu funkciju gubitka nazivamo **gubitak zglobnice** (engl. *hinge loss*), budući da izgledom podjeća na zglobnicu. Konačno, uvrštavanjem gubitka zglobnice u ciljnu funkciju dobivamo **funkciju pogreške SVM-a**:

$$E(\mathbf{w}|\mathcal{D}) = \sum_{i=1}^N \max(0, 1 - y^{(i)}\mathbf{w}^T \mathbf{x}) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

gdje $\lambda = 1/C$. Provjerimo još samo vezu između hiperparametara λ i C : veći C daje složeniji model, budući da više kažnjavamo netočne klasifikacije ili ulaske u marginu. Veći C odgovara manjem λ . Manji λ daje manje regularizirani model, tj. složeniji model. Dakle, ovo je u redu. Možemo zaključiti: funkcija pogreške SVM-a je L2-regularizirano očekivanje gubitka zglobnice. (Pritom smo izostavili faktor $\frac{1}{N}$.)

Pogrešku SVM-a možemo minimizirati npr. **stohastičkim gradijentnim spustom**, i to primjenom **podgradijenta**, budući da gubitak zglobnice nije diferencijabilan u svim točkama. Međutim, nećemo dalje o tome.

Umjesto toga, usredotočimo se radije na funkciju gubitka zglobnice i usporedimo je s drugim funkcijama gubitka koje smo do sada susreli. Grafikon svih tih funkcija izgleda ovako:



Gubitak zglobnice prikazan je ljubičastom bojom. Vidimo da je gubitak zglobnice, isto kao i funkcija logističkog gubitka, aproksimacija funkcije gubitka 0-1, koju bismo idealno željeli minimizirati, ali koja nije konveksna i uglavnom je konstantna, pa stoga nije pogodna za optimizaciju. Zapravo, sve funkcije gubitka koje smo dosada promatrali (osim perceptronu, koji je ionako crna ovca u ovoj našoj priči) su tzv. **nadomjesne (surogatne) funkcije gubitka** za gubitak 0-1. Točnije, to su **konveksne gornje međe funkcije gubitka 0-1**. Usporedbimo gubitak zglobnice s gubitkom perceptronu i logističkim gubitkom. Gubitak zglobnice i gubitak perceptronu zapravo su istog oblika, jedina je razlika da je gubitak zglobnice pomaknut udesno za 1. Međutim, efekt tog pomaka je značajan: za razliku od perceptronu, SVM će kažnjavati i one primjere za koje $0 < \mathbf{w}^T \mathbf{x}y < 1$, a to su upravo primjeri koji ulaze u marginu. Od logističkog se gubitka funkcija zglobnice razlikuje po tome što uopće ne kažnjava ispravno klasificirane primjere izvan margine, što rezultira time da su **rješenja rjeđa** (više je težina pritegnuto na nulu nego što je to slučaj kod logističke regresije). Zašto? Zato što primjeri koji su na ispravnoj strani margine ne nanose nikakav gubitak, pa nije potrebno regrutirati težine, makar u malom iznosu, da bi se gubitak dodatno smanjio.

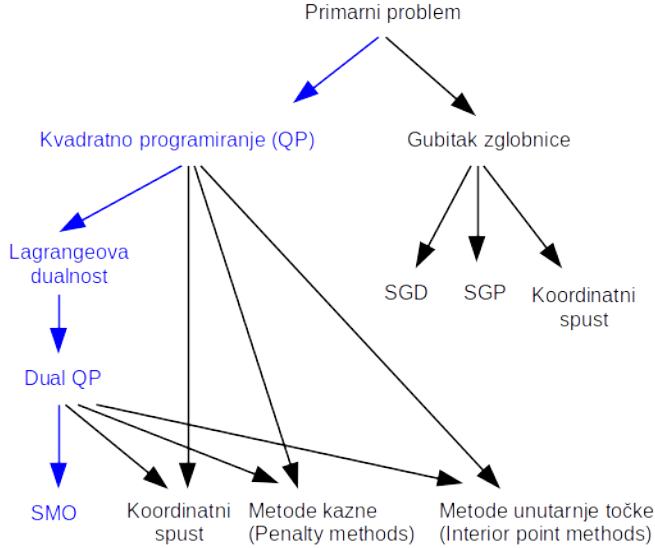
5

6

7

4 Alternativni algoritmi SVM-a

Optimizacija pogreške SVM-a gradijentnim spustom predstavlja alternativu optimizaciji u dualu pomoću algoritma SMO. Međutim, ova dva postupka ne iscrpljuju sve mogućnosti optimizacije SVM-a. Te mogućnosti grafički možemo prikazati ovako:



Plavom je bojom označen "klasičan" put, kojim smo i mi isli. On uključuje prelazak iz primarnog problema u dualni, primjenu Lagrangeove dualnosti i rješavanje dualnog kvadratnog programa algoritmom SMO. Umjesto algoritma SMO, mogu se koristiti druge optimizacijske metode, poput koordinatnog spusta, metode kazne ili metode unutarnje točke. Alternativno, umjesto prelaska u dualnu formulaciju, istim se metodama može optimirati izravno primarna formulacija problema maksimalne marge. Konačno, umjesto kvadratnog programiranja, može se minimizirati pogreška SVM-a definirana preko gubitka zglobnice, i to opet različitim postupcima, uključivo stohastičkim gradijentnim spustom (SGD), stohastičkom projekcijom (pod)gradijenta (engl. *stochastic (sub)gradient projection*) i koordinatnim spustom.

Sve ove mogućnosti rezultirale su time da postoji niz optimizacijskih algoritama (rješavača) za SVM. Ovdje je dan pregled nekih od njih:

Algorithm	Citation	SVM type	Optimization type	Style	Runtime
SMO	[Platt, 1999]	Kernel	Dual QP	Batch	$\Omega(n^2d)$
SVM ^{light}	[Joachims, 1999]	Kernel	Dual QP	Batch	$\Omega(n^2d)$
Core Vector Machine	[Tsang et al., 2005, 2007]	SL Kernel	Dual geometry	Batch	$O(s/\rho^4)$
SVM ^{perf}	[Joachims, 2006]	Linear	Dual QP	Batch	$O(ns/\lambda\rho^2)$
NORMA	[Kivinen et al., 2004]	Kernel	Primal SGD	Online(-style)	$\tilde{O}(s/\rho^2)$
SVM-SGD	[Bottou, 2007]	Linear	Primal SGD	Online-style	Unknown
Pegasos	[Shalev-Shwartz et al., 2007]	Kernel	Primal SGD/SGP	Online-style	$\tilde{O}(s/\lambda\rho)$
LibLinear	[Hsieh et al., 2008]	Linear	Dual coordinate descent	Batch	$O(nd \cdot \log(1/\rho))$
SGD-QN	[Bordes and Bottou, 2008]	Linear	Primal 2SGD	Online-style	Unknown
FOLOS	[Duchi and Singer, 2008]	Linear	Primal SGP	Online-style	$\tilde{O}(s/\lambda\rho)$
BMRM	[Smola et al., 2007]	Linear	Dual QP	Batch	$O(d/\lambda\rho)$
OCAS	[Franc and Sonnenburg, 2008]	Linear	Primal QP	Batch	$O(nd)$

Table 1: A comparison of various SVM solvers discussed in this document. "QP" refers to a quadratic programming technique, "SGD" to stochastic (sub)gradient descent, and "SGP" to stochastic (sub)gradient projection. "SL" means the method only works with square-loss. The runtime is for a problem with n training examples and d features, with an average of s non-zero features per example. λ is the SVM regularization parameter, and ρ the optimization tolerance. "Unknown" means there is no known formal bound on the runtime.

Prvionavedeni algoritam je Plattov SMO, koji smo već više puta spomenuli. Taj algoritam može raditi s **jezgrama** (engl. *kernels*) (što znači da granica između klasa može biti nelinearna; više o tome idući put), optimizacija se provodi kvadratnim programiranjem u dualu nad svim primjerima (batch), a složenost treniranja modela je kvadratna u broju primjera. Drugi algoritmi

nude neke druge prednosti, pa su prikladniji u drugim situacijama. Npr., poznati algoritam *LibLinear* prikladan je za slučajeve kada je granica između klasa linearna, jer ima linearu složenost u broju primjera i broju značajki. Već smo naglasili da je jedna od prednosti dualnih pristupa mogućnost korištenja jezgri, međutim jezgre se zapravo mogu koristiti i u primarnoj formulaciji (npr., algoritam Pegasos).

5 Napomene

SVM dolazi s nizom napomena. Kao i uvijek, vrag je u detaljima. Pogledajmo o čemu se radi.

5.1 SVM za regresiju

Mi smo se bavili SVM-om za klasifikaciju. Međutim, postoji i **stroj potpornih vektora za regresiju** (engl. *support vector regression, SVR*). Taj algoritam koristi istu ideju o margini kao i SVM, samo što ovog puta baš želimo da primjeri budu unutar margine, koja je oko krivulje koju aproksimiramo. Nećemo ići u detalje ovoga.

10

5.2 Hiperparametar C

SVM je vrlo osjetljiv na hiperparametar C , koji definira složenost modela. Treba dobro odabratи vrijednost tog hiperparametra, odnosno treba napraviti **odabir modela**. Kako? Primijetite da, ako optimirate taj hiperparametar na skupu za učenje, dobit ćeete prenaučen model (veliki C). Zašto? Zato što, što veći C , to složeniji model, pa to manja empirijska pogreška na skupu za učenje. Zbog toga parametar C , isto kao i parametar λ kod regularizirane logističke regresije, trebamo optimirati na izdvojenom skupu označenih primjera, tj. trebamo raditi **unakrsnu provjeru**.

5.3 Skaliranje

Sjetimo se kako dualni SVM radi predikciju:

$$h(\mathbf{x}) = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^T \mathbf{x}^{(i)} + w_0$$

Zamislite što se događa ako radimo klasifikaciju potencijalnih korisnika kredita banke, te imamo značajke, npr. dob osobe i prihod. Što će u skalarnom produktu dominirati? Dominirat će prihod, jer ima veću skalu. To nije dobro. Da bismo to izbjegli, trebamo skalirati značajke. Kako? Tipično radimo **max-min normalizaciju** ili **standardizaciju**. No, budući da ćeete to raditi u labosima, ovdje vam ne želim pokvariti zabavu.

5.4 Višeklasje

Kod SVM-a u principu koristimo shemu OVO. Shema OVR se nepreporuča zbog neuravnoteženosti klasa.

5.5 Probabilistički izlaz

SVM nema probabilistički izlaz. Izlaz iz SVM-a je proporcionalan udaljenosti primjera od hiper-ravnine, i ta vrijednost nije ograničena na interval $(0, 1)$ i ne odgovara vjerojatnosti pripadanja primjera nekoj klasi. Naime, za razliku od poopćenih linearnih modela, SVM nije probabilistički model i nije izведен na temelju pretpostavke da se označe pokoravaju nekoj teorijskoj distribuciji. Međutim, u praksi često ipak želimo na neki način izračunati vjerojatnost klasifikacije primjera u neku klasu, pa makar i bez teorijske podloge. Rješenje u takvim slučajevima

je **Plattovo skaliranje (Plattova kalibracija)**. Ideja je da se jednostavno na izlaz SVM-a naliđe sigmoidna funkcija (odnosno, da treniramo logističku regresiju s izlazom SVM-a $h(\mathbf{x})$ kao jednom jedinom značajkom):

$$P(y = 1|\mathbf{x}) = \sigma(ah(\mathbf{x}) + b)$$

pri čemu se parametri a i b optimiraju na izdvojenom skupu označenih primjera.

Ovo je praktično, ali problematično. Problem je što nemamo nikakvu garanciju da će ovo biti dobro **kalibrirano**, budući da iza ovoga ne stoji nikakav vjerojatnosni model. Pogotovo je to problem ako imamo više klase, jer neće biti nikakve povezanosti između njih, kao što je to bilo kod multinomijalne logističke regresije, koja koristi funkciju softmax. Stoga je preporuka za Plattovu kalibraciju: koristite, ako baš morate. Ako želite dobiti vjerojatnosti kojima vjerujete, koristite neki probabilistički model (npr., logističku regresiju).

5.6 Nelinearnost

Ovo pitanje smo predugo odgađali, pa je vrijeme da odgovorimo i na njega: kako dobiti nelinearnu granicu? Pa, kao i uviјek: preslikavanjem u prostor značajki, pomoću funkcije ϕ . Prostor u koji preslikavamo tipično je više dimenzije od ulaznog prostora, čime povećavamo vjerojatnost da će primjeri u tom prostoru značajki biti linearno odvojivi. Ako, dakle, koristimo preslikavanje u prostor značajki, model SVM-a je:

$$h(\mathbf{x}) = \sum_{i=1}^N \alpha_i y^{(i)} \phi(\mathbf{x})^T \phi(\mathbf{x}^{(i)}) + w_0$$

Mođutim, ovo je tek početak za nešto mnogo moćnije: **jezgrene funkcije**. Jezgrene funkcije sastavni su i ključni dio SVM-a, i o njima ćemo pričati idući put.

Sažetak

- **Meka marga** omogućava kompromis između pogrešne klasifikacije i maksimalne marge, sprječavajući **prenaučenost**
- Problem se opet svodi na **kvadratno programiranje**
- Empirijska pogreška SVM-a je **L2-regularizirano** očekivanje **gubitka zglobnice**
- Postoje razni algoritmi SVM-a, uključivo **SGD** u primarnoj formulaciji
- SVM je osjetljiv na **hiperparametar C** i **skaliranje** značajki
- SVM **nema probabilistički izlaz**, ali može se dobiti, ako baš treba
- Nelinearnost dobivamo preslikavanjem odnosno **jezgrenim funkcijama** (više drugi put)

Bilješke

1 Formulacija SVM-a s mekom marginom je standardna formulacija SVM-a. Predložili su je 1995. godine Corinna Cortes i Vladimir Vapnik ([Cortes and Vapnik, 1995](#)), kao nadogradnju na raniji model SVM-a s jezgrenim funkcijama ([Boser et al., 1992](#)).

2 Ograničenje na dualnu varijablu α je dakle $0 \leq \alpha_i \leq C$. Ograničenja ovog oblika, gdje postoji gornja i donja ograda, nazivamo **ograda varijable** (engl. *variable bounds*) ili **ograničenjima kutije** (engl. *box constraints*). Striktno gledano, optimizacijski problem s ogradama nije u standardnoj formi, no lako ga se može pretvoriti u standardnu formu tako da se ograde zamijene dvama odgovarajućim ograničenjima nejednakosti. U našem slučaju, ograde varijable $0 \leq \alpha_i \leq C$ zamijenili bismo ograničenjima $-\alpha_i \leq 0$ i $\alpha_i - C \leq 0$. Detaljnije u ([Boyd et al., 2004](#)) (poglavlje 4).

3 Prisjetimo se, kod tvrde marge **potporni vektori** su oni vektori $\mathbf{x}^{(i)}$ iz skupa označenih primjera \mathcal{D} za koje vrijedi $\alpha_i > 0$. Kod meke marge α_i je omeđen odozdo i odozgo, $0 \leq \alpha_i \leq C$, a potporni

su vektori oni vektori $\mathbf{x}^{(i)}$ iz \mathcal{D} za koje vrijedi $0 < \alpha \leq C$. Pokažimo da su potporni vektori za koje vrijedi $\alpha_i < C$ na margini, dok vektori za koje $\alpha_i = C$ mogu ležati i unutar margine. Prisjetimo se da vrijede $\alpha_i = C - \beta_i$ i uvjet komplementarne labavosti $\beta_i \xi_i = 0$. Ako $\alpha_i < C$, onda iz $\alpha_i = C - \beta_i$ slijedi $\beta_i > 0$, te iz toga i $\beta_i \xi_i = 0$ slijedi $\xi_i = 0$, što znači da je primjer na margini. Ako $\alpha_i = C$, onda iz $\alpha_i = C - \beta_i$ slijedi $\beta_i = 0$, te iz toga i $\beta_i \xi_i = 0$ slijedi $\xi_i = 0$ ili $\xi_i > 0$, odnosno primjer je na margini ili unutar nje.

- 4 Kako na temelju dualnih parametara i potpornih vektora izračunati težinu w_0 objasnili smo u temi 8 (bilješka 21), gdje smo pričali o tvrdoj margini. Međutim, kod meke margine treba paziti da se izračun težine w_0 radi isključivo na temelju potpornih vektora koji su na margini (dakle onih za koje $\alpha_i < C$), a ne i na temelju onih koji su unutar margine (oni za koje $\alpha_i = C$). Naime, izračun se temelji na jednakosti $y^{(i)} h(\mathbf{x}^{(i)}) = 1$, koja vrijedi samo za potporne vektore $\mathbf{x}^{(i)}$ koji su točno na margini.
- 5 **Zglobnica** (baglama, pant, šarka) je željezni okov pomoću kojega su za okvir pričvršćena vrata, prozorska krila, poklopac na kutiji i sl. http://hjp.znanje.hr/index.php?show=search_by_id&id=d1hnURI%3D
- 6 Ova sličnost između **gubitka perceptron** i **gubitka zglobnice** upućuje na to da su i algoritam perceptron i algoritma SVM-a donekle slični. To je točno, međutim, očigledno postoje i različitosti. Usporedimo ih po trima komponentama. Model perceptron i model SVM-a su zapravo identični: oba modela su skalarni produkt vektora težina i vektora značajki. Perceptron ima aktivacijsku funkciju praga, međutim ona zapravo ne figurira kod izračuna gubitka, a ista bi se funkcija praga mogla dodati i modelu SVM-a za potrebe krajnje klasifikacijske predikcije. Funkcije gubitka razlikuju se za pomak od 1 duž x -osi, zbog čega algoritam SVM-a kažnjava i one primjere koji su ispravno klasificirani, ali su unutar margini. Osim po funkciji gubitka, funkcije pogreške razlikuju se još i po tome što pogreška SVM-a uključuje i regularizacijski faktor, kojim se promovira širina margini, dok perceptron toga nema. Konačno, optimizacijski se postupci razlikuju zato jer perceptron uči na pojedinačnim primjerima (online), dok SVM može učiti na svim primjerima (kod optimizacije kvadratnim programiranjem) ili, ako optimiramo gradijentnim spustom, na minigrupama primjera ili na pojedinačnim primjerima (stohastički gradijentni spust). Učenje SVM-a stohastičkim gradijentnim spustom najviše nalikuje učenju perceptron, premda postoji i razlika u pogledu kriterija zaustavljanja (učenje perceptron zaustavlja se tek kada su svi primjeri zaredom točno klasificirani, dok se stohastički gradijenti spust zaustavlja kada je gradijent jednak nuli ili dovoljno blizu tome). Detaljniju usporedbu algoritama perceptron i SVM-a možete pronaći u ([Collobert and Bengio, 2004](#)).
- 7 Vrlo je zanimljivo svojstvo SVM-a da rezultira rijetkim modelima, premda zapravo koristi L2-regularizaciju a ne L1-regularizaciju. Rijetkost je ovdje posljedica toga što funkcija gubitka ne kažnjava ispravno klasificirane primjere izvan margini, a ne toga što provodimo L1-regularizaciju. Ovo je netipično u usporedbi s poopćenim linearnim modelima, i mnogim drugim probabilističkim modelima strojnog učenja. Zbog toga [Murphy \(2012\)](#) konstatira da je SVM iz probabilističke perspektive “vrlo neprirođan”, budući da (1) rijetkost ostvaruje pomoću funkcije gubitka umjesto pomoću regularizacijskog izraza (odnosno apiorne gustoće vjerojatnosti nad težinama), (2) uvodi jezgre pomoću jezgrenog trika, umjesto da su one eksplicitno ugrađene u model i (3) ne rezultira vjerojatnosnim izlazom, što dovodi do toga da vjerojatnosti nisu dobro kalibrirane, pogotovo kod višeklasne klasifikacije ([Murphy, 2012](#)) (poglavlje 14.5).
- 8 Preuzeto iz ([Menon, 2009](#)).
- 9 Algoritam SMO implementiran je u enormno popularnoj bilblioteci **LibSVM** ([Chang and Lin, 2011](#)), koja se pak koristi u mnogim drugim alatima, uključivo alatu **sklearn**, koji koristimo za labose.
- 10 Zainteresirane upućujem na ([Drucker et al., 1997](#)) i ([Smola and Schölkopf, 2004](#)).
- 11 Plattovo skaliranje osmislio je John Platt 1999. godine. Detalje možete naći u ([Platt et al., 1999](#)). Platt je godinu dana prije toga osmislio algoritam **slijedne minimalne optimizacije (SMO)**.
- 12 Intuitivno, probabilistički model je dobro **kalibriran** ako vjerojatnosti koje daje doista odgovaraju udjelima u skupu primjera. Npr., ako za neki podskup primjera vrijedi $h(\mathbf{x}) \sim 0.7$, onda za dobro

kalibrirani model očekujemo da će u tom podskupu primjera doista njih $\sim 70\%$ biti pozitivni.

Literatura

- B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.
- S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- C.-C. Chang and C.-J. Lin. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.
- R. Collobert and S. Bengio. Links between perceptrons, mlps and svms. In *Proceedings of the twenty-first international conference on Machine learning*, page 23, 2004.
- C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- H. Drucker, C. J. Burges, L. Kaufman, A. J. Smola, and V. Vapnik. Support vector regression machines. In *Advances in neural information processing systems*, pages 155–161, 1997.
- A. K. Menon. Large-scale support vector machines: algorithms and theory. *Research Exam, University of California, San Diego*, 117, 2009.
- K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.
- J. Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.
- A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

10. Jezgrene metode

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v1.9

Prošla dva puta bavili smo se strojem potpornih vektora. Objasnili smo model tog algoritma i optimizacijski postupak koji se temelji na ideji maksimalne margine. Na zadnjem predavanju to smo malo proširili i govorili smo o mekoj margini, koja dopušta malo pogrešnu klasifikaciju i efektivno time sprječava prenaučenost.

Danas ćemo pogledati nešto što je povezano s SVM-om, ali je i šire od toga: **jezgrene metode** (engl. *kernel methods*). Jezgrene metode općenit su i vrlo moćan radni okvir za modeliranje nelinearnosti. Vidjet ćemo da jezgrene metode dolaze u dva osnovna okusa: **jezgreni strojevi** i **jezgreni trik**.

1

1 Jezgrene funkcije

Do sada smo uvijek prepostavljali da je primjer **x vektor** značajki, odnosno da primjeri žive u nekom n -dimenzijskome vektorskem prostoru, $\mathcal{X} = \mathbb{R}^n$. No, ponekad nije jasno kako primjer možemo prikazati kao vektor značajki. Npr., recimo da želimo napraviti klasifikator koji će klasificirati riječi (npr., za neku riječ hrvatskoga jezika želimo odrediti je li ta riječ latinizam). U tom slučaju naši primjeri **x** su riječi, tj. kao nizova znakova (stringovi). Nije očito kako bismo niz znakova prikazali kao vektor u nekom n -dimenzijskom prostoru. Ili, recimo da radimo klasifikaciju DNK nizova. Kako biste u tom slučaju prikazali značajke? Što ako radimo klasifikaciju grafova, np. klasificiramo strukture molekula koje su prikazane kao graf? Nije naš jasno kako bismo grafove prikazali kao vektor značajki. U takvim je slučajevima možda lakše računati **sličnost** između primjera, nego ih vektorizirati. Npr., lakše mi je izračunati sličnost između dviju riječi, dva DNK niza, ili čak dva grafa, nego svaki od njih prebaciti u prostor značajki. U to slučaju, umjesto vektorskog prostora $\mathcal{X} = \mathbb{R}^n$, imati ćemo neki **apstraktan prostor** \mathcal{X} . Taj prostor nije nužno vektorski, tj. možda ga ne znamo ga definirati pomoću značajki, ali znamo kako izračunati sličnost između primjera $\mathbf{x} \in \mathcal{X}$.

Općenito, sličnost dvaju primjera računamo pomoću jezgrene funkcije. **Jezgrena funkcija** (engl. *kernel function*) (ili samo **kernel** – koristit ćemo i taj naziv) je realna funkcija koja kao argumenate uzima dva primjera iz \mathcal{X} i daje nam sličnost između tih primjera:

$$\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$$

Tipično vrijedi:

$$\kappa(\mathbf{x}, \mathbf{x}') \geq 0$$

i

$$\kappa(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x}', \mathbf{x})$$

tj. jezgrena funkcija je nenegativna i simetrična. Ako je domena jezgrene funkcije dodatno ograničena na $[0, 1]$, dakle $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow [0, 1]$ i ako $\kappa(\mathbf{x}, \mathbf{x}) = 1$, onda govorimo o **mjeri sličnosti** (engl. *similarity measure*).

3

Kao što smo već napomenuli, jezgrene funkcije imaju itekako smisla kada su podatci s kojima radimo na neki način strukturirani. Ako se radi o nizovima znakova, npr., riječima ili nizovima

DNK, onda koristimo **jezgre nizova znakova** ili (engl. *string kernele*), kojima uspoređujemo sličnost dvaju znakovnih nizova. Ako je struktura podataka složenija od nizova, onda idemo na **jezgre stabala** (engl. *tree kernels*) ili **jezgre grafova** (engl. *graph kernels*). Prve se često koriste u području obrade prirodnog jezika, gdje se uspoređuju sintaktički grafovi rečenica. Jezgre grafova još su općenitije: sličnost između grafova tipično računaju na temelju broja zajedničkih podgrafova ili broja zajedničkih šetnji u oba grafa, a mogu se definirati i za težinske ili usmjerene grafove.

4

► PRIMJER

Želimo raditi klasifikaciju riječi (koje se sastoje od slova) ili klasifikaciju nizova DNK (koja se sastoji od nukleotida A, T, G, C). Nije baš jasno kako bismo takve nizove najbolje prialzali kao vektore. Doduše, možemo smisliti razne načine, ali lakše je definirati sličnost dvaju znakovnih nizova, odnosno definirati **jezgru znakovnih nizova**. Nju možemo definirati na različite načine, npr., možemo brojati koliko nizovi imaju podudarnih podnizova, ili možemo koristiti **Levenshteinovu udaljenost** (broj operacija umetanja i brisanja potrebnih da bi se jedan niz sveo na drugi). Npr., *zagreb* i *zagrebački* imaju zajedničke podnizove *zagreb*, *zagre*, *zagr*, *zag*, *za*, *z*, *agreb*, *agre*, itd. Takvu jezgru možemo definirati ovako:

$$\kappa(\mathbf{x}, \mathbf{x}') = \sum_{s \in \mathcal{A}^*} c(s, x)c(s, x')$$

gdje je \mathcal{A}^* skup svih nizova znakova abecede \mathcal{A} (* je Kleeneov operator), a c brojač koliko puta se podniz s pojavio u nizu x .

Dakle, jedna jasna motivacija za korištenje jezgrenih funkcija je situacija kada primjeri nisu vektori i imaju neku internu strukturu. Međutim, jezgrene funkcije možemo koristiti i onda kada primjeri jesu vektori (bilo da su vektori u ulaznom prostoru ili da su vektori nakon preslikavanja u prostor značajki). Dapače, to se vrlo često radi, budući da jezgrene funkcije imaju i druge prednosti, kao što ćemo vidjeti.

Pogledajmo sada nekoliko primjera jezgrenih funkcija koji rade s primjerima koji se mogu prikazati kao vektori. Najjednostavnija je **linearna jezgra**, koja je jednostavno skalarni umnožak dvaju vektora primjera:

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

Ova jezgra je simetrična, ali nije nenegativna niti je odozgo ograničena na 1. (Zapravo, mi smo na ovaj način već računali sličnost primjera kod SVM-a. To će reći da SVM zapravo koristi jezgru. Tako je, i na to ćemo se vratiti.)

Jedna moćna alternativa linearnoj jezgri je **Gaussova jezgra**:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

Ovo je isto kao gustoća vjerojatnosti normalne (Gaussove) distribucije centrirane u \mathbf{x} , samo što nema normalizacijskog izraza $1/\sqrt{2\pi\sigma^2}$ (kojim se osigurava da integral gustoće vjerojatnosti bude jednak jedinice). Prisjetimo se, $\|\mathbf{x} - \mathbf{x}'\|^2$ je kvadrat euklidske udaljenosti između \mathbf{x} i \mathbf{x}' :

$$\|\mathbf{x} - \mathbf{x}'\|^2 = (\mathbf{x} - \mathbf{x}')^T (\mathbf{x} - \mathbf{x}') = \sum_{j=1}^n (x_j - x'_j)^2$$

Ovo se lijepo može interpretirati upravo kao **sličnost** između \mathbf{x} i \mathbf{x}' jer će imati vrijednost 1 za $\mathbf{x} = \mathbf{x}'$, a vrijednost nula kako udaljenost između ta dva vektora ide prema beskonačno. Parametar σ^2 je **varijanca** normalne distribucije: što je varijanca veća, to je Gaussovo “zvono” šire. U ovom kontekstu σ^2 se također zove **širina pojasa** (engl. *bandwidth*): što je σ^2 veća, to će biti veća sličnost između primjera koji su udaljeniji. S druge strane, to što je σ^2 manja, to smo striktniji u pogledu što je slično a što nije. Nekad koristimo $\gamma = 1/2\sigma^2$, pa pišemo:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2\right)$$

gdje parametar $\gamma = 1/2\sigma^2$ nazivamo **preciznost** (inverz varijance). Zapravo, $1/\sigma^2$ zovemo preciznost, ali nema veza, razlika je u konstantnom faktoru. Koji je efekt parametra γ odnosno σ^2 na izračun sličnosti između primjera? Što je preciznost veća (odnosno σ^2 manja), to je distribucija uža i to bliže moraju biti primjeri da bismo ih smatrali sličnima. Suprotno, što je preciznost manja (odnosno σ^2 veća), to je distribucija šira i to su nam primjeri općenito međusobno sličniji.

Jezebre ovog tipa, koje su neka funkcija udaljenosti između primjera, $\|\mathbf{x} - \mathbf{x}'\|$, zovemo **radijalne bazne funkcije** (engl. *radial basis functions, RBF*). Dakle, jezgra RBF općenito je definirana kao:

$$\kappa(\mathbf{x}, \mathbf{x}') = f(\|\mathbf{x} - \mathbf{x}'\|)$$

gdje je f neka funkcija, npr., eksponencijalna funkcija kao Kod Gaussove jezgre. Gaussovou jezgri slična je **eksponencijalna jezgra**, koja također koristi eksponencijalnu funkciju, ali ne kvadrira normu, tj.:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|)$$

Još jedna često korištena jezgra RBF je **inverzna kvadratna jezgra**:

$$\kappa(\mathbf{x}, \mathbf{x}') = \frac{1}{1 + \|\mathbf{x} - \mathbf{x}'\|^2}$$

Osim što jezgra RBF ne mora nužno biti Gaussova jezgra, ona ne mora koristiti ni euklidsku udaljenost. Problem s euklidskom udaljenošću je što sve značajke (odnosno sve dimenzije) tretira jednakim važnim. Sjetimo se primjera s godinama i prihodima: ne želimo da prihodi dominiraju u izračunu udaljenosti samo zato što imaju veću mjeru skalu. Jedan način da to riješimo, koji smo spomenuli zadnji put, jest da normaliziramo značajke, pa tek onda računamo euklidsku udaljenost. Međutim, odnose između značajki možemo još bolje modelirati ako koristimo poopćenje euklidske udaljenosti, tzv. **Mahalanobisovu udaljenost**:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{x}')\right)$$

gdje je **$\boldsymbol{\Sigma}$ kovarijacijska matrica**, koja kodira kovarijaciju između značajki (dimenzija). Kovarijanca nam kazuje koliko varijable zajedno variraju, odnosno koliko zajednički rastu ili padaju. Npr., može se pokazati da prihod pozitivno korelira s dobi više nego neki drugi par značajki. Onda želimo da, ako dvije osobe imaju slične prihode i sličnu dob, da to u izračunu sličnosti ima manju težinu nego ako imaju nešto drugo slično. To upravo možemo postići Mahalanobisovom udaljenošću. Ako kovarijacijsku matricu postavimo na jediničnu matricu, $\boldsymbol{\Sigma} = \mathbf{I}$, što znači da nema kovarijacija između značajki i da je varijanca svake značajke jednaka, onda Mahalanobisova udaljenost degenerira na euklidsku udaljenost.

Izvršno, sada znamo što su to jezgrene funkcije! Iduće pitanje je kako ih iskoristiti za klasifikaciju i regresiju. Tu imamo dvije mogućnosti: prva mogućnost su **jezgreni strojevi**, a druga mogućnost su **kernelizacija**, odnosno primjena tzv. **jezgrenog trika**. Pogledat ćemo obje mogućnosti. Krenimo najprije s jezgrenim strojevima.

2 Jezgreni strojevi

Jezgreni strojevi imaju mističan naziv, ali zapravo nisu ništa drugo nego jedna vrsta poopćenih linearnih modela. Pa, prisjetimo se najprije što je to poopćeni linearni model, pa ćemo polako doći do jezgrenih strojeva. **Poopćeni linearni model** definirali smo ovako:

$$h(\mathbf{x}) = f(\mathbf{w}^\top \phi(\mathbf{x}))$$

gdje je ϕ funkcija koja je preslikavala iz ulaznog prostora u prostor značajki:

$$\phi(\mathbf{x}) = (1, \phi_1(\mathbf{x}), \dots, \phi_m(\mathbf{x}))$$

gdje je

$$\{\phi_1, \phi_2, \dots, \phi_m\}$$

skup od m **baznih funkcija** (nelinearnih funkcija ulaznih varijabli), svaka od kojih uzima cijeli primjer i vraća jedan broj, $\phi_j : \mathbb{R}^n \rightarrow \mathbb{R}$.

Problem s baznim funkcijama očito je taj da ih moramo nekako definirati, a da ne znamo unaprijed koje su dobre funkcije za naš problem, tj. koje nam bazne funkcije daju linearu odvojivost u prostoru značajki. Taj smo problem već bili dotakli kada smo pričali o logističkoj regresiji. Tada smo opisali ideju **adaptivnih baznih funkcija**, tj. ideju da bazne funkcije automatski prilagodimo našim podatcima, umjesto da ih unaprijed ručno definiramo. Također smo spomenuli da to možemo napraviti na dva načina. Jedan način je da bazne funkcije parametriziramo te da te parametre učimo iz podataka, što nas je spektakularno dovelo do neuronskih mreža. Drugi je način da u prostoru primjera definiramo neke referentne primjere te da onda bazne funkcije definiramo tako da mjere sličnost između ulaznog primjera i dotičnih referentnih primjera. Drugim riječima, ideja je da za bazne funkcije koristimo **jezgrene funkcije**.

Danas se bavimo ovim drugim načinom. Dakle, jezgrene funkcije ϕ_j definirat ćemo tako da uspoređuju ulazni primjer s nekim unaprijed odabranim referentnim primjerima u ulaznom prostoru. Označimo te odabrane primjere sa $\mu^j \in \mathcal{X}$, i neka ih ima ukupno m . Dakle, svaka bazna funkcija ϕ_j bit će definirana kao jezgrena funkcija koja mjeri sličnost u odnosu na primjer μ^j , tj. $\phi_j(\mathbf{x}) = \kappa(\mathbf{x}, \mu_j)$.

S tako definiranim baznim funkcijama, imamo ovakvu funkciju preslikavanja:

$$\phi(\mathbf{x}) = (1, \kappa(\mathbf{x}, \mu^1), \kappa(\mathbf{x}, \mu^2), \dots, \kappa(\mathbf{x}, \mu^m))$$

I to je to! Poopćeni linearni model gdje vektor značajki ima upravo ovakav oblik nazivamo **jezgreni stroj** (engl. *kernel machine*).

Sad možemo jednostavno raditi *business as usual*, koristeći tako definirano preslikavanje, npr., u logističkoj regresiji:

$$h(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T \phi(\mathbf{x}))$$

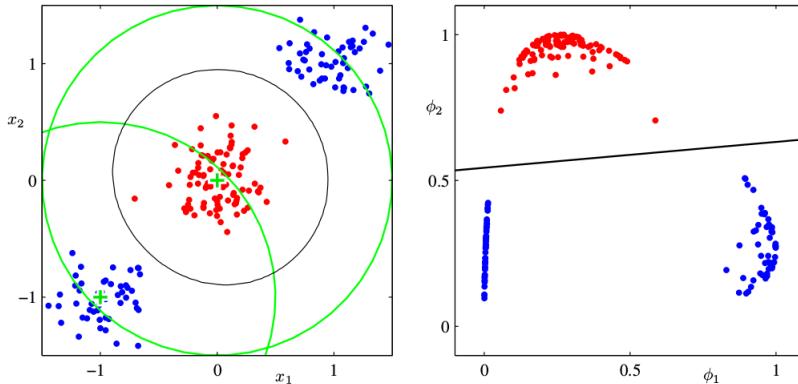
ili linearnoj regresiji:

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x})$$

ili SVM-u, koji ima isti model kao i linearna regresija. Pogledajmo nekoliko primjera.

► PRIMJER

6



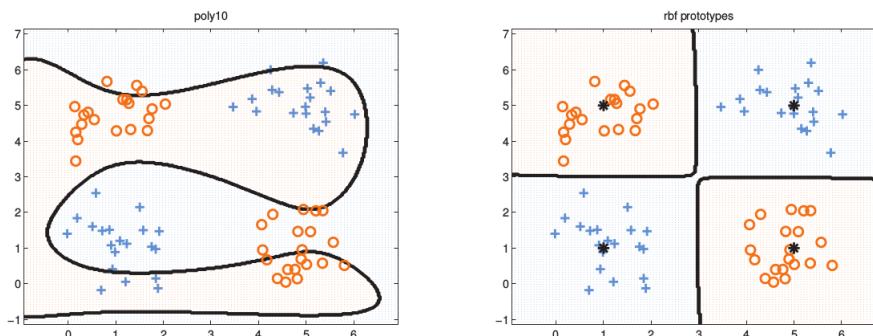
Na lijevoj slici prikazan je dvodimenzionalni ulazni prostor ($n = 2$), u kojem imamo primjere iz dviju klase (crvena i plava). Klase očito nisu linearno odvojive. Kako bismo ostvarili linearnu odvojivost, primjere ćemo preslikati u prostor značajki funkcijom

$$\phi(\mathbf{x}) = (1, \phi_1(\mathbf{x}), \phi_2(\mathbf{x})) = (1, \kappa(\mathbf{x}, \mu^1), \kappa(\mathbf{x}, \mu^2))$$

gdje ćemo za bazne funkcije koristiti jezgre RBF. Preslikavanje ϕ_1 definirano je jezgrenom funkcijom sa središtem u $(-1, 1)$, a ϕ_2 je preslikavanje definirano jezgrenom funkcijom sa središtem u $(0, 0)$. Zeleni križići prikazuju središta tih dviju jezgrenih funkcija, a zelene kružnice prikazuju izokonture. Budući da imamo samo dvije jezgrevne funkcije, preslikavamo u prostor značajki dimenzija $m = 2$ (ako zanemarimo težinu w_0). U prostoru značajki sada su na klase linearne odvojive (uzmite si vremena da shvatite zašto je to tako). Linearna granica (crni pravac) u prostoru značajki odgovara nelinearnoj granici (crnoj kružnici) u ulaznom prostoru. Primijetite da u ovom slučaju nismo preslikavali u prostor više dimenzije (imamo $n = m$), međutim preslikavanje je nelinearno, pa smo u prostoru značajki uspjeli ostvarili linearno odvajanje klasa.

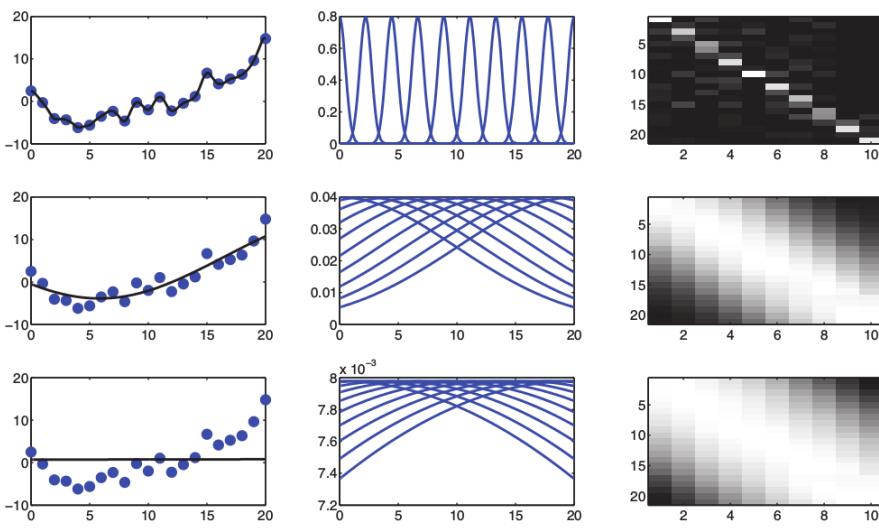
► PRIMJER

Ovaj je primjer sličan prethodom. Ponovno se radi o binarnom klasifikacijskom problemu u dvo-dimensijskome ulaznom prostoru, koji nije linearne odvojiv (XOR-problem). Ako preslikavanje u prostor značajki definiramo kao jezgrevni stroj sa četiri jezgre RBF, i središta tih jezgara postavimo u sredine svake od četiri klase u ulaznom prostoru, onda dobivamo nelinearnu granicu između klasa kakva je prikazana na desnoj slici. Na lijevoj je slici prikazana granica koju bismo dobili preslikavanjem polinomom 10. stupnja, koji za razliku od jezgre RBF, na ovom problemu ne ostvaruje savršeno točnu klasifikaciju.



► PRIMJER

Pogledajmo i jedan primjer s (jednostavnom) regresijom:



a Lijeva slika prikazuje ulazni prostor, u kojem imamo 21 primjer. Ovdje se radi o jednostavnoj regresiji i naš ulazni prostor je jednodimenzionalni. Za preslikavanje koristimo 10 Gaussovih jezgrenih funkcija. Srednja slika prikazuje te Gaussove jezgrene funkcije, koje su ulaznom prostoru ekvidistantno postavljene. Desna slika prikazuje matricu dizajna: retci su primjeri, a stupci su značajke. U ovom slučaju značajke su vrijednosti Gaussove jezgre, njih ukupno 10. Svjetlije ćelije odgovaraju većim vrijednostima jezgrevne funkcije (tj. većoj sličnosti između primjera i referentnog primjera), a tamnije ćelije odgovaraju manjim vrijednostima jezgrevne funkcije (tj. manjoj sličnosti između primjera i referentnog primjera). Drugim riječima, na toj slici vidimo koliko je svaki od 21 primjera blizu središta svake od 10 Gaussovih distribucija. Vidimo prenaučen (prvi red), dobar (srednji red) i podnaučen model (donji red).

Ovo sve je vrlo lijepo, no postoji jedan problem koji smo prešutjeli: kako definirati referentne primjere μ_j ? Mogli bismo ih rasporediti uniformno (ekvidistantno) po prostoru, kao što je to bio slučaj u posljednja dva gornja primjera. To je izvedivo ako je prostor niske dimenzije, ali ako imamo visokodimenzionalni prostor, to postaje problematično. Trebalo bi nam vrlo mnogo (eksponencijalno mnogo u broju dimenzija) takvih primjera, a da uniformno pokrijemo visokodimenzionalni prostor kroz sve dimenzije. Što bi bila alternativa uniformnom raspoređivanju referentnih primjera? Nekako bismo htjeli da ti primjeri u prostoru budu postavljeni tamo gdje ima pravih primjera, a ne tamo gdje ih nema. Kako to ostvariti? Pa, upravo tako da za referentne primjere uzmemos primjere iz skupa za učenje. Onda nam se očito ne može dogoditi da tako definirani referentni primjeri budu u nekim dijelovima ulaznog prostora gdje nema pravih primjera. Ako, dakle, za referentne primjere odaberemo primjere iz skupa za učenje, onda za funkciju preslikavanja u prostor značajki imamo:

$$\phi(\mathbf{x}) = (1, \kappa(\mathbf{x}, \mathbf{x}^{(1)}), \kappa(\mathbf{x}, \mathbf{x}^{(2)}), \dots, \kappa(\mathbf{x}, \mathbf{x}^{(N)}))$$

Sada je dimenzija prostora značajki jednaka prostoru primjera, $m = N$, dakle imat ćemo onoliko parametara koliko imamo primjera u skupu za učenje.

Međutim, to bi opet moglo biti problematično, jer primjera može biti previše. Stoga se postavlja pitanje: kako možemo smanjiti broj referentnih primjera? Primijetite da je u ovom slučaju to je isto kao da smo pitali: kako možemo smanjiti broj značajki? Kako smo to radili do sada? Regularizacijom! Međutim, to onda mora biti **L_1 -regularizacija**, kako bismo težine pritegnuli skroz na nulu. Dakle, ako primijenimo L_1 -regularizaciju, pritegnut ćemo neke težine (u prostoru značajki!) na nulu, što efektivno znači da smo neke od jezgri izbacili, a to je isto kao

da smo odabrali neki podskup primjera iz skupa za učenje da nam služe kao referentni primjeri. Takve primjere onda možemo smatrati **prototipnim primjerima**. Nadalje, takve jezgreni strojevi, koji kao referentne primjere za izračun jezgri koriste manji broj primjera iz skupa za učenje, nazivamo **rijetki jezgreni strojevi** (engl. *sparse kernel machines*) ili **rijetki vektorski strojevi** (engl. *sparse vector machines*). Konkretno, kada koristimo L_1 -regularizaciju, to zovemo **L1VM**. Ako koristimo L_2 -regularizaciju, onda je to **L2VM**, no primijetite da nam to neće dati rijetki model, iz razloga koji nam je već poznat (L_2 -regularizacija ne potiskuje težine do nule).

Sve ovo vas možda malo podsjeća na SVM. Sjetite se da smo tamo upravo imali podskup primjera iz skupa za učenje, koje smo zvali **potporni vektori**, s kojima smo uspoređivali druge primjere. Međutim, kod SVM-a nismo baš koristili te primjere da bismo preslikali u prostor značajki na ovakav način, nego je cijela ta priča ispala nekako “automatski” kod prijelaza iz primarnog u dualni optimizacijski problem.

Sada je zgodan trenutak da se malo vratimo na SVM i pogledamo to u svjetlu ove današnje priče. Vrijeme je da pogledamo drugi način kako možemo iskoristiti jezrene funkcije. Vrijeme je za – **jezgreni trik!**

3 Jezgreni trik

3.1 Nelinearan SVM

Vratimo se na pitanje s kojim smo bili završili prošlotjedno predavanja: kako od SVM možemo dobiti **nelinearan model**, tj. model s nelinearnom granicom između klasa u ulaznom prostoru? Znamo, jer smo to već mnogo puta rekli, da to možemo lako ostvariti tako da primjere preslikamo u prostor više dimenzije pomoću funkcije ϕ . Nadam se da će, nakon preslikavanja, ti primjeri u prostoru značajki biti linearne odvojivi, ili barem blizu toga da budu linearne odvojivi – ne moraju biti savršeno linearne odvojivi jer imamo **meku marginu**. Za funkciju preslikavanja možemo koristiti uobičajene funkcije koje smo koristili do sada (npr., polinomijalno preslikavanje) ili možemo koristiti raznorazne jezrene funkcije, kao što bismo to radili kod jezgrenih strojeva.

Međutim, kod SVM-a možemo napraviti nešto puno bolje. Nema potrebe da prvo preslikavamo primjere u prostor značajki, pa onda primjenjujemo SVM-a. Umjesto toga, možemo direktno iskoristiti jezgenu funkciju unutar samo algoritma (tj. unutar modela i optimizacijskog postupka). Dovoljno je primijetiti da SVM zapravo već koristi jezgru! Naime, prisjetimo se kako izgleda model SVM u dualnoj formulaciji:

$$h(\mathbf{x}) = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^T \mathbf{x}^{(i)} + w_0$$

Ovaj skalarni umnožak $\mathbf{x}^T \mathbf{x}^{(i)}$ nije ništa drugo nego **linearna jezgra**, koji računa sličnost primjera s potpornim vektorima! Ista se stvar javlja i kod treniranja SVM-a. Prisjetimo se, u dualnoj formulaciji, optimizacijski problem je ovaj:

$$\underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \left(\sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)} \right)$$

uz određena ograničenja, koja ovdje više nećemo ponavljati. Vidimo da se i ovdje pojavljuje skalarni umnožak.

Štoviše, jedina mjesto gdje se u algoritmu SVM-a pojavljuju primjeri \mathbf{x} zapravo ova dva mesta gdje se oni pojavljuju u obliku skalarnog umnoška. Drugim riječima, vektori \mathbf{x} nikada se ne pojavljuju sami, nego uvijek u skalarnom umnošku. To nam otvara jednu fantastičnu mogućnost. Zašto ne bismo skalarni umnožak na ta dva mesta jednostavno zamijenili jezgrenom

funkcijom? Dakle, model bi bio:

$$h(\mathbf{x}) = \sum_{i=1}^N \alpha_i y^{(i)} \kappa(\mathbf{x}, \mathbf{x}^{(i)}) + w_0$$

a ciljna funkcija kod optimizacije bila bi:

$$\sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

pri čemu $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$, tj. koristimo **linearnu jezgru**. Što smo time dobili? Pa, iskreno, još ništa. Samo smo napravili zamjenu jednog izraza (skalarnog umnoška) ekvivalentnim izrazom (linearnom jezgrom). Međutim, zašto stati na linearnej jezgri? To nam neće dati nelinarnost. Nelinearnost bismo dobili da smo primjere preslikali pomoću funkcije ϕ . U tom slučaju, umjesto skalarnog umnoška $\mathbf{x}^T \mathbf{x}^{(i)}$, u izrazu za model i u izrazu za ciljnu funkciju imali bismo $\phi(\mathbf{x})^T \phi(\mathbf{x}^{(i)})$. Pa, dakle, ako želimo biti općenitiji, jezgru ćemo definirati da uključuje to preslikavanje, tj. ovako:

$$\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

Vidimo da ovako definirana jezgrena funkcija mjeri **sličnost** dvaju vektora nakon njihovog preslikavanja u **prostor značajki**, gdje je ta sličnost definirana konkretno pomoću **skalarnog umnoška**. Ako preslikavanja nema, tj. $\phi(\mathbf{x}) = (\mathbf{x})$, onda smo nazad na linearnej jezgri.

I to je zapravo taj famozni **jezgreni trik** (engl. *kernel trick*): jednostavno ćemo zamijeniti svako pojavljivanje ovog skalarnog umnoška u prostoru značajki jezgrenom funkcijom! No, zašto je to trik? Tu dolazimo do fantastičnog dijela. To je trik zato što mi nećemo stvarno izračunati tu jezgrenu funkciju kao skalarni umnožak u prostoru značajki. Naime, mi uopće nećemo preslikavati primjere u prostor značajki pomoću funkcije ϕ , nego ćemo direktno izračunati sličnost dvaju primjera pomoću jezgrene funkcije, i onda taj broj koji dobijemo umetnuti na mjesto skalarnog umnoška. Ono što je tu fantastično jest što možemo upotrijebiti (skoro pa) bilo koju jezgrenu funkciju, uključivo i one koje uopće nisu definirane kao skalarni umnožak vektora. Drugim riječima, tretirat ćemo jezgrenu funkciju kao "crnu kutiju": unutra ulaze dva primjera (bilo vektorska ili ne), van izlazi neki broj, a taj broj odgovara njihovoj sličnosti u prostoru značajki. Pritom nas uopće ne zanima da primjere doista preslikamo u prostor značajki, nego samo koliko bi iznosio njihov skalarni umnožak kada bismo ih doista preslikali. Poanta ovoga je da nema eksplicitnog preslikavanja, nego je preslikavanje **implicitno**!

Ovakav se pristup općenito naziva **inverzno oblikovanje**: umjesto da izravno preslikavamo primjere, mi izabiremo neku jezgrenu funkciju κ pa treniramo model s tom funkcijom, i istu tu funkciju koristimo kod predikcije. Inverzno oblikovanje ima tri vrlo važne prednosti nad preslikanjem primjera u prostor značajki:

1. **Smanjenje računalne složenosti:** izračun jezgredne funkcije često je jednostavniji nego izračun dvaju preslikavanja pa zatim izračun skalarnog umnoška;
2. **Jednostavnost:** kao što smo već bili napomenuli, često je lakše definirati jezgrednu funkciju κ nego preslikavanje ϕ , a pored toga postoji niz standardnih jezgrednih funkcija. Ova prednost osobito dolazi do izražaja ako primjeri imaju nekakvu strukturu (znakovni nizovi, stabla, grafovi i sl.). Tada je puno lakše definirati sličnost nego značajke, a kamo li preslikavanje;
3. **Veća složenost modela:** Prostor značajki koji odgovara jezgrednoj funkciji može biti vrlo visokodimenzijski, potencijalno beskonačno dimenijski! To ne možemo dobiti nikakvim preslikavanjem.

Sve ovo zvuči super! Jezgreni trik je super stvar, to je to! Međutim, ima jedna mala začkuljica. Pogledajmo još jednom kako smo definirali jezgenu funkciju:

$$\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

Imajući na umu ovu definiciju, postavlja se pitanje: mogu li ovdje koristiti *baš svaku* jezgenu funkciju? Odgovor je: ne mogu. Jezgrena funkcija mora odgovarati **skalarnom umnošku** primjera u prostoru značajki. Inače jezgreni trik ne funkcioniра. Drugim riječima, skalarni umnožak $\phi(\mathbf{x})^T \phi(\mathbf{x}')$ smijem zamijeniti bilo kakvom jezgrenom funkcijom $\kappa(\mathbf{x}, \mathbf{x}')$, pod uvjetom da ta jezgrena funkcija odgovara onome što bih dobio kada bih doista preslikao vektore u prostor značajki i tamo ih skalarno pomnožio. S obzirom da jezgenu funkciju možemo definirati kako god to želimo (sve dok ona zadovoljava osnovne uvjete mjere sličnosti: nenegativnost i simetričnost), očito je da neće svaka jezgrena funkcija zadovoljavati ovaj naš uvjet. Npr., jezgrena funkcija $\kappa(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$ nije valjana jezgra jer ne postoji preslikavanje ϕ takvo bi $\phi(\mathbf{x})^T \phi(\mathbf{x}')$ bilo jednako $\|\mathbf{x} - \mathbf{x}'\|$.

Srećom, postoji puno vrlo dobrih jezgri koje zadovoljavaju ovaj uvjet. Takve jezgre nazivamo **Mercerove jezgre** ili **pozitivno definitne jezgre** ili **valjane jezgre** (engl. *valid kernels*). Usredotočimo se u nastavku na takve jezgre.

3.2 Mercerove jezgre

U nastavku će nam biti korisno da jezgenu funkciju prikažemo matrično. Primijetite, naime, da kod treniranja modela računamo vrijednost jezgrene funkcije između svih parova primjera iz skupa primjera \mathcal{D} . Te vrijednosti možemo izračunati unaprijed i pohraniti u simetičnu matricu dimenzija $N \times N$. Tu ćemo matricu nazvati **jezgrena matrica** (engl. *kernel matrix*) i označiti sa \mathbf{K} . Dakle, jezgrena matrica izgleda ovako:

$$\mathbf{K} = \begin{pmatrix} \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) & \dots & \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(N)}) \\ \kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(1)}) & \kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(2)}) & \dots & \kappa(\mathbf{x}^{(2)}, \mathbf{x}^{(N)}) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}^{(N)}, \mathbf{x}^{(1)}) & \kappa(\mathbf{x}^{(N)}, \mathbf{x}^{(2)}) & \dots & \kappa(\mathbf{x}^{(N)}, \mathbf{x}^{(N)}) \end{pmatrix}$$

Sada se pitamo: ako je matrica \mathbf{K} doista rezultat skalarnih umnožaka u nekom vektorskому prostoru značajki, kakva ona mora biti, a da možemo reći da taj prostor stvarno postoji? Drugim riječima: kakva mora biti jezgrena matrica ako je jezgra Mercerova? Može se dokazati da je svaka matrica skalarnih umnožaka pozitivno definitna, i obrnuto, svaka pozitivno definitna matrica je matrica skalarnih umnožaka za neki skup vektora u nekom vektorskemu prostoru (prisjetimo se, matrica \mathbf{A} je pozitivno semidefinitna ako vrijedi $\mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0$). Takvu matricu zovemo **Gramova matrica** (taj smo naziv već susreli kada smo pričali u regresiji). Rekli smo već da je Mercerova jezgra ona jezgra koja odgovara skalarnome umnošku vektora u prostoru značajki. Prema tome, ako je jezgra κ Mercerova, onda je matrica \mathbf{K} nužno pozitivno semidefinitna, tj. jezgrena matrica je Gramova matrica, definirana na sljedeći način:

$$\mathbf{K} = \Phi \Phi^T$$

gdje je Φ je matrica dizajna

Sada kada znamo da Mercerova jezgra daje vrijednost skalarnog umnoška dvaju vektora u prostoru značajki, možemo se zapitati: kakav je taj prostor? Savim sigurno, to je **vektorski prostor**, koji očito mora imati definiranu **operaciju skalarnog umnoška**. Međutim, taj prostor može imati vrlo mnogo dimenzija, uključivo beskonačno mnogo dimenzija. Takav vektorski prostor, koji ima definiran **skalarni umnožak**, ali koji je proizvoljne dimenzije, uključivo beskonačnog broja dimenzija, naziva se **Hilbertov prostor** \mathcal{H} . To uključuje i konačno-dimenzijski višedimenzijski prostor \mathbb{R}^n , s kakvim smo do sada radili. Dakle, da bi jezgreni trik funkcionirao, jezgrena funkcija mora biti Mercerova jezgra, odnosno matrica \mathbf{K} mora biti pozitivno semidefinitna, i onda će ona odgovarati skalarnom umnošku u nekom Hilbertovom prostoru.

3.3 Izgradnja jezgri

Toliko o teoriji. Okrenimo se sada praktičnim pitanjima. Osnovno pitanje očito glasi: kako izgraditi Mercerove jezgre? Možemo li izmisliti neku jezgrenu funkciju i onda dokazati da je Mercerova? Mogli bismo, ali to je općenito dosta zahtjevno i traži dosta znanja funkcijске analize. Alternativa je da izračunamo jezgrenu matricu \mathbf{K} na našem skupu primjera \mathcal{D} te da provjerimo je li to Gramova matrica, tj. je li ta matrica pozitivno semidefinitna. Očito, to nije formalan dokaz da je jezgra Mercerova, ali u praksi može biti dovoljno dobro, pogotovo ako je skup \mathcal{D} razmjerno velik.

Međutim, postoji niz **standardnih jezgrenih funkcija** za koje se zna da su Mercerove jezgre. Na primjer:

- **Linearna jezgra:** $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$, koja efektivno daje linearan model
- **Polinomijalna jezgra:** $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^p$, koja uključuje linearne članove, kombinacije ulaznih varijabli (interakcije) i potencije do uključivo stupnja p
- **Gaussova RBF jezgra**, koju smo već vidjeli,
- **Jezgra znakovnih nizova**, o kojoj smo također već pričali.
- ... i mnoge druge.

Za linearnu jezgru nam je jasno da je Mercerova, jer očito odgovara skalarnom umnošku, i to izravno u ulaznom prostoru primjera. Što je s polinomijalnom jezgrom? Može se dokazati da je i ona Mercerova. No, umjesto toga, da dobijemo osjećaj da je to tako, pogledajmo jedan primjer.

► PRIMJER

Neka je jezgrena funkcija $\kappa(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2$. Provjerimo odgovara li ova jezgrena funkcija skalarnom umnošku u nekom prostoru značajki. To ćemo napraviti tako da ćemo raspisati izraz $(\mathbf{x}^T \mathbf{z})^2$, i zatim provjeriti odgovara li on doista skalarnom umnošku neka dva vektora. Jednostavnosti radi, a ne smanjujući općenitost, ograničimo se na dvodimenzionalni prostor, $n = 2$. Prvi vektor neka je $\mathbf{x} = (x_1, x_2)$, a drugi $\mathbf{z} = (z_1, z_2)$. Onda imamo:

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^T \mathbf{z})^2 = ((x_1, x_2)^T (z_1, z_2))^2 = (x_1 z_1 + x_2 z_2)^2 \\ &= (x_1 z_1)^2 + 2(x_1 z_1)(x_2 z_2) + (x_2 z_2)^2 = x_1^2 z_1^2 + \sqrt{2} x_1 x_2 \sqrt{2} z_1 z_2 + x_2^2 z_2^2 \\ &= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)^T (z_1^2, \sqrt{2} z_1 z_2, z_2^2) = \phi(\mathbf{x})^T \phi(\mathbf{z})\end{aligned}$$

Vidimo da jezgrena funkcija $\kappa(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2$ doista odgovara skalarnom umnošku u prostoru značajki, i to s preslikavanjem definiranim kao $\phi(\mathbf{x}) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)$.

Primijetimo također da je u ovom slučaju **računalno jednostavnije** izravno izračunati jezgenu funkciju (ukupno 4 računske operacije) nego skalarni umnožak u prostoru značajki (2×4 operacija za preslikavanje i dodatnih 5 operacija za skalarni umnožak = 13 operacija).

3.4 Gaussova jezgra

Vrlo lijepo, linearna i polinomijalna jezgra su Mercerove. A što je sa Gaussovom jezgrom? Prisjetimo se, Gaussova jezgra definirana je ovako:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right)$$

Može se dokazati da je Gaussova jezgra također Mercerova, no to ćemo ovdje preskočiti. Ono što je još zanimljivije jest pitanje kako izgleda prostor značajki u koji ona preslikava. Pogledajmo to na intuitivnoj razini. Ovdje ćemo reći dvije stvari. Prva se tiče **dimenzionalnosti** prostora

značajki. Može se pokazati da je Gramova matrica \mathbf{K} za Gaussovou jezgru matrica punog ranga. To znači da su vektori $\phi(\mathbf{x})$ linearno nezavisni. Efekt toga jest da je prostor značajki (prostor u koji preslikavamo) beskonačno dimenzijski! Točnije, taj prostor ima onoliko dimenzijski koliko imamo primjera, a na broj primjera nema gornje granice!

Druga stvar tiče se sličnosti između vektora u tom beskonačno dimenzijskom prostoru. Već znamo da Gaussova jezgra mjeri sličnost dvaju primjera temeljem njihove udaljenosti u vektorskom prostoru. Za slične primjere vrijedi $\kappa(\mathbf{x}, \mathbf{x}') \rightarrow 1$, pa su ti primjeri očito onda blizu u prostoru značajki, odnosno vektori su im slični nakon preslikavanja, jer je skalarni umnožak najviši mogući (ionako nije više od 1, kada koristimo Gaussovou jezgru). S druge strane, za vrlo različite primjere vrijedi $\kappa(\mathbf{x}, \mathbf{x}') \rightarrow 0$. Budući da je Gaussova jezgra Mercerova, to onda znači da je skalarni umnožak u prostoru značajki teži prema nuli. A što to onda znači? To znači da su primjeri u prostoru značajki međusobno ortogonalni (odnosno ortonormalni, jer su maksimalne duljine 1, budući da sličnost prema Gaussovom jezgri može biti najviše jednaka 1). Pogledajmo parametar Gaussove jezgre σ^2 (širina pojasa) odnosno γ (preciznost). Taj parametar kontrolira kojom brzinom $\kappa(\mathbf{x}, \mathbf{x}')$ teži k nuli u ovisnosti o udaljenosti. Ako je γ malen, $\kappa(\mathbf{x}, \mathbf{x}') \rightarrow 1$ i primjeri su u prostoru značajki grupirani zajedno, što lako dovodi do **podnaučenosti**. S druge strane, ako je γ velik, onda $\kappa(\mathbf{x}, \mathbf{x}') \rightarrow 0$ (izuzev za $\mathbf{x} = \mathbf{x}'$), pa su sve točke u prostoru značajki međusobno ortogonalne, što pak lako dovodi do **prenaučenosti**. Sve u svemu, različite vrijednosti za γ daju različita preslikavanja, pa stoga je taj parametar vrlo važan. Intuitivno, vidimo da, ako odaberemo dovoljno velik γ , moći ćemo preslikati primjere tako da su oni međusobno okomiti.

Zbrojimo li ova dva efekta – **beskonačnu dimenzionalnost** i **ortonormiranost** svih vektora iz skupa primjera za učenje (za dovoljno visok γ) – dobivamo to da je svaki primjer u svojoj dimenziji i da je skroz drugačiji od drugih. To efektivno znači da je svaki primjer “poslan u svoj vrh” duž ortogonalne osi. U takvom prostoru uvijek možemo linearno odvojiti primjere kako god oni bili označeni, a da klasifikacija bude savršeno točna.

Što to znači? To znači da (za dovoljno veliku γ), možemo dobiti **savršenu linearnu odvojivost** u prostoru značajki. Prostor značajki će biti onoliko dimenzijski koliko imamo primjera.

3.5 Izgradnja složenijih jezgara

Ovo je opet vrlo lijepo, i zapravo ne možemo ni tražiti više od beskonačnodimenzijskog prostora. No, što ako ipak trebamo složenije jezgre, zato jer su naši primjeri složeniji? Pritom mislimo na situacije kada naši primjeri nisu vektori, nego imaju složeniju strukturu. Npr., zamislimo da se svaki naš primjeri sastoje od jednog znakovnog niza i od jednog vektora. Ako želimo pomoći jezgrenim funkcijama računati sličnost između takvih primjera, očito nam treba složenija jezgrena funkcija, koja istovremeno može raditi i sa znakovnim nizom i sa vektorom. K tome još želimo da ta jezgra bude Mercerova.

Dobra vijest je da, koristeći standardne jezgrene funkcije koje smo gore naveli kao gradivne blokove, možemo konstruirati složenije jezgrene funkcije. Pritom za kombiniranje Mercerovih jezgri možemo koristiti niz operacija za koje je dokazano da daju jezgre koje su opet Mercerove (engl. *Mercer-preserving operations*). Na primjer, ako su κ_1 i κ_2 Mercerove jezgre, onda će to

biti i jezgra κ dobivena pomoću sljedećih operacija:

$$\begin{aligned}
 \kappa(\mathbf{x}, \mathbf{x}') &= \alpha \kappa_1(\mathbf{x}, \mathbf{x}') & \alpha > 0 \\
 \kappa(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) \kappa_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') & f - \text{bilo koja funkcija} \\
 \kappa(\mathbf{x}, \mathbf{x}') &= q(\kappa_1(\mathbf{x}, \mathbf{x}')) & q - \text{polinom s pozitivnim koeficijentima} \\
 \kappa(\mathbf{x}, \mathbf{x}') &= \exp(\kappa_1(\mathbf{x}, \mathbf{x}')) \\
 \kappa(\mathbf{x}, \mathbf{x}') &= \kappa_1(\phi(\mathbf{x}), \phi(\mathbf{x}')) & \phi : \mathbb{R}^n \rightarrow \mathbb{R}^{m+1} \\
 \kappa(\mathbf{x}, \mathbf{x}') &= \kappa_1(\mathbf{x}, \mathbf{x}') + \kappa_2(\mathbf{x}, \mathbf{x}') \\
 \kappa(\mathbf{x}, \mathbf{x}') &= \kappa_1(\mathbf{x}, \mathbf{x}') \kappa_2(\mathbf{x}, \mathbf{x}')
 \end{aligned}$$

Kombiniranje više jezgri zove naziva se **učenje s više jezgara** (engl. *multiple kernel learning, MKL*). Najjednostavnije je zbrojiti dvije jezgre (i lako je pokazati da zbroj dviju jezgri zapravo odgovara konkatenaciji dvaju vektora u prostoru značajki, što je općenito najjednostavniji način kombiniranja značajki).

17

4 Napomene

4.1 SVM

Prisjetimo se da SVM ima **hiperparametar** C koji određuje njegovu složenost. Manja vrijednost za C znači da će model dozvoljavati više pogrešaka, tj. bit će jednostavniji. Obrnuto, veća vrijednost za C znači da će model više kažnjavati pogreške, pa će biti složeniji. Taj hiperparametar tipično optimiramo unakrsnom provjerom.

Ako koristimo (nelinearnu) jezgrenu funkciju, onda trebamo odabrat i **hiperparametar jezgrena funkcije** (npr., stupanj polinoma p ili preciznost γ). Ovi su parametri povezani s hiperparametrom C . Npr., ako odaberemo visoku vrijednost za γ , dobit ćemo složeniji model, pa će trebati pojačati regularizaciju odabirom manje vrijednosti za C . To znači da unakrsnom provjerom trebamo optimirati dva hiperparametra istovremeno, što tipično činimo iscrpnim pretraživanjem u unaprijed definiranom rasponu, tzv. **pretraživanjem po rešetci** (engl. *grid search*).

4.2 Linearna jezgra

Bismo li ikada želeli koristiti linearnu jezgru? Da, ako želimo **rjetki model**: bit će lakše hiper-ravni prikazati s nekoliko potpornih vektora, nego s hrpom težina. Ta prednost naravno dolazi do izražaja kod predikcije. Kod učenja ionako moramo izračunati jezgrenu funkciju između svih parova primjera.

Druga prednost bi mogla biti da nam se više isplati raditi optimizaciju u dualu, koju algoritmom SMO možemo napraviti sa složenošću $O(N^2)$, dok bi u primarnoj formulaciji morali koristiti generički rješavač kvadratnog programa, pa bi složenost bila $O(n^3)$ (doduše, možemo koristiti gradijentni spust sa pogreškom definiranom na temelju gubitka zglobnice, pa tada nemamo taj problem).

4.3 Aproksimacija jezgrenih funkcija

Jezgredni trik je izvrsna ideja, ali će izračun jezrene matrice biti dosta skup kada imamo puno primjera. S druge strane, optimizacija gradijentnim spustom vrlo je računalno jeftina, međutim tamo ne možemo koristiti jezgre. Ako želimo najbolje od dva svijeta, možemo upotrijebiti **aproksimaciju jezgre**. To znači da ne koristimo jezgredni trik, nego doista primjere stvarno preslikamo u prostor značajki, ali to preslikavanja radimo približno. U tom prostoru onda radimo stohastički gradijentni spust.

4.4 Ujezgravanje algoritama

Jezgreni trik nije primjenjiv samo na SVM. Doduše, kod njega je očit, jer se skalarni umnožak automatski javio kod dualne formulacije optimizacijskog problema. Međutim, kod mnogih drugih modela moguće je također primijeniti jezgreni trik, ako se model preformulira tako da se vektori primjera javljaju kod predikcije (u definiciji modela) i kod optimizacije isključivo u obliku skalarnog umnoška. Ako to uspijemo napraviti, onda znači da možemo **ujezgriti (kernelizirti)** (engl. *kernelize*) algoritam. Npr., i linearna i logistička regresija se mogu vrlo jednostavno ujezgriti.

18

Sažetak

- **Jezgrene funkcije** mjere sličnost dvaju primjera
- Primjeri mogu biti vektori ili neke druge strukture (npr. stringovi, stabla, grafovi)
- Postoji niz standardnih jezgrenih funkcija, npr. **linearna, polinomijalna i Gaussova jezgra**, te operacija za izgradnju složenijih jezgara
- **Jezgreni strojevi** su poopćeni linearni modeli s jezgrenim funkcijama kao baznim funkcijama
- **Jezgreni trik:** skalarni umnožak vektora u prostoru značajki mijenjamo Mercerovom jezgrenom funkcijom. Trik je primjenjiv kod SVM-a, ali i nekih drugih algoritama
- Jezgreni trik omogućava **inverzno oblikovanje**: učinkovitije, jednostavnije, ekspresivnije
- **Gaussova jezgra** je Mercerova i preslikava u beskonačnodimenzijski prostor značajki

Bilješke

1 **Jezgrene metode** (engl. *kernel methods*) jedno su od osnovnih područje strojnog učenja, od velikog teorijskog i praktičnog značaja. Standardni uvodni tekstovi u ovo područje su ([Scholkopf and Smola, 2018](#)), ([Shawe-Taylor et al., 2004](#)) i ([Hofmann et al., 2008](#)).

Što se naziva "jezgra" tiče, on dolazi iz funkcijске analize s početka 20. stoljeća. Isti se naziv počeo koristiti u statistici tek u sedamdesetim godina prošlog stoljeća u kontekstu neparametarskih metoda, konkretno **jezgreni procjenitelji gustoće vjerojatnosti** (engl. *kernel density estimators*). Više ovdje: <https://stats.stackexchange.com/a/341842/93766>

2 Naravno, nije nemoguće riječi jezika na neki smislen način prikazati kao vektor. Npr., riječi bismo mogli prikazati u prostoru razapetom slovima abecede, gdje bi svaka značajka odgovarala broju pojavljivanja određenog slova u toj riječi. U konkretnom primjeru s prepoznavanjem tuđica, mogli bismo smisliti i mnogo pametnije značajke od toga, npr. prisustvo određenih sufiksa ili općenito morfema i slično. U **području obrade prirodnog jezika** (engl. *natural language processing*) riječi se doista prikazuju kao vektori u semantičkom vektorskem prostoru, tako da ti vektori odgovaraju značenju riječi u smislu da semantički slične i srodne riječi imaju bliske vektore (v. [Almeida and Xexéo \(2019\)](#)). Ipak, u nekim je situacijama mnogo lakše razmišljati izravno o sličnosti između riječi, nego o tome kako ih preslikati u vektore. Ovo zapažanje vrijedi općenito: premda je načelno moguće bilo koji apstraktni objekt $\mathbf{x} \in \mathcal{X}$ prikazati kao vektor konačnih dimenzija, nije uvijek jasno kako to najbolje napraviti, i tada je često lakše formalizirati koncepciju sličnosti između tih objekata pomoću jezgrenih funkcija.

3 Nemojte brkati **mjeru sličnosti** (engl. *similarity measure*) s mjerom različitosti odnosno s funkcijom udaljenosti. Objasnimo razlike između ovih pojmljiva, krenuvši od ovog posljednjeg, koji je najspecifičniji. **Funkcija udaljenosti** (engl. *distance function*) ili **udaljenosti** ili **metrika** je funkcija $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_0^+$ za koju vrijede sljedeća svojstva:

$$(1) \quad d(\mathbf{x}^a, \mathbf{x}^b) \geq 0 \text{ (nenegativnost)}$$

- (2) $d(\mathbf{x}^a, \mathbf{x}^b) = 0$ ako i samo ako $\mathbf{x}^a = \mathbf{x}^b$ (strogost)
- (3) $d(\mathbf{x}^a, \mathbf{x}^b) = d(\mathbf{x}^b, \mathbf{x}^a)$ (simetričnost)
- (4) $d(\mathbf{x}^a, \mathbf{x}^b) + d(\mathbf{x}^b, \mathbf{x}^c) \geq d(\mathbf{x}^a, \mathbf{x}^c)$ (nejednakost trokuta)

Svojstva (1) i (2) zajedno nazivaju se pozitivna definitnost. Strikto gledano, svojstvo (1) nije aksiomatsko, tj. može se izvesti iz preostala tri svojstva. Postoji niz standardnih metrika za vektorske prostore \mathcal{X} , npr. euklidska udaljenost ili Manhattanova udaljenost, odnosno općenitije Minkowskijeva udaljenost, koju smo već bili spominjali. Metrika, međutim, nije ograničena isključivo na vektorske prostore. Npr., metriku možemo definirati nad grafovima, nizovima znakova (npr. Levenshteinova udaljenost) ili vjerojatnosnim distribucijama (npr. Wassersteinova metrika).

Općenitiji pojam od udaljenosti je **mjera sličnosti** (engl. *similarity measure*), odnosno njoj komplementarna **mjera različitosti** (engl. *dissimilarity measure*). Udaljenost se može tumačiti kao geometrijska interpretacija sličnosti, odnosno različitosti. Bitna razlika jest u tome što mjera sličnosti (odnosno mjera različitosti) nije metrika. Mjera sličnosti je funkcija $s : \mathcal{X} \times \mathcal{X} \rightarrow [0, 1]$ za koju se obično podrazumijeva da zadovoljava sljedeće:

- (1) $s(\mathbf{x}, \mathbf{x}) = 1$
- (2) $0 \leq s(\mathbf{x}, \mathbf{x}') \leq 1$
- (3) $s(\mathbf{x}, \mathbf{x}') = s(\mathbf{x}', \mathbf{x})$

Iz navedenih je svojstava očigledno da mjera udaljenosti i mjera sličnosti nisu suprotne u smislu da je jedna komplement druge, već su suprotne u smislu da je jednu moguće preslikati u drugu nekom monotono padajućom funkcijom. Na primjer, za preslikavanje udaljenosti d u sličnost s često se koristi:

$$s(\mathbf{x}, \mathbf{x}') = \frac{1}{1 + d(\mathbf{x}, \mathbf{x}')}$$

Obrnuti smjer, pretvorba sličnosti u udaljenosti, nešto je teži zbog uvjeta nejednakosti trokuta. U većini slučajeva, udaljenosti i sličnosti (odnosno različitosti) mogu se koristiti jednakom. U nekim slučajevima međutim sličnosti pružaju dodatnu fleksibilnost.

- 4 Više o jezgrama za grafove možete naći u ([Gärtner, 2003](#)) i ([Kriege et al., 2020](#)).
- 5 Mahalanobisova udaljenost svodi se dakle na euklidsku udaljenost za $\Sigma = \mathbf{I}$. Međutim, možemo pojednostaviti Mahalanobisovu udaljenost, a ne svesti je odmah na euklidsku. Ako kovarijacijsku matricu definiramo kao dijagonalnu matricu, $\Sigma = \text{diag}(\sigma_j^2)$, to znači da između značajki nema kovarijacije, ali dopuštamo da su varijance značajki različite. S dijagonalnom kovarijacijskom matricom dobivamo ovaku jezgrenu funkciju:
$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2} \sum_{j=1}^n \frac{1}{\sigma_j^2} (x_j - x'_j)^2\right)$$
- 6 Prvi primjer preuzet je iz ([Bishop, 2006](#)) (poglavlje 4.3.1), a iduća dva iz ([Murphy, 2012](#)) (poglavlje 14.3.1).
- 7 Pored toga, u visokodimenzionalnim prostorima javlja se problem **prokletstva dimenzionalnosti** (engl. *curse of dimensionality*): podatci postaju rijetki i jednakо udaljeni jedan od drugih, tj. sve sličniji jedni drugima. Više o prokletstvu dimenzionalnosti idući put.
- 8 Pokoji pedantan čitatelj ovdje će primjetiti malu nekonzistentnost u notaciji: ako preslikavanja nema, onda bi funkcija ϕ trebala biti definirana kao $\phi(\mathbf{x}) = (1, \mathbf{x})$, tako da uključi i "dummy"

jedinicu koja množi težinu w_0 . Međutim, kod SVM-a to nije slučaj, budući da je težina w_0 izuzeta i tretiramo ju posebno.

9 Jezgreni trik zapravo radi po principu “što bi bilo kad bi bilo”: nećemo doista preslikati vektore u prostor značajki, ali možemo reći koliko bi iznosio njihov skalarni umnožak kada bismo to doista učinili. U neku ruku, jezgreni je trik prečica koja nam omogućava da dođemo do informacije koja nas zanima, a da si pritom uskratimo muku izračuna. Ovdje do izražaja dolazi razlika između **ekstenzionalne i intenzionalne jednakosti**, koju smo već bili spomenuli u kontekstu jednakosti funkcija hipoteza. Naime, budući da $\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$, to su $\kappa(\mathbf{x}, \mathbf{x}')$ i $\phi(\mathbf{x})^T \phi(\mathbf{x}')$ ekstenzionalno jednakci, međutim nisu intenzionalno jednakci, budući da $\kappa(\mathbf{x}, \mathbf{x}')$ možda možemo izračunati na drugačiji način nego da doista preslikamo vektore i skalarno ih pomnožim. U računarstvu je razlika između intenzionalne i ekstenzionalne jednakosti itekako važna, jer nam nije svejedno na koji način izračunavam funkciju: neki su načini skuplji, a neki jeftiniji. Jednostavan primjer koji ilustrira drastičnost te razlike je suma beskonačnog geometrijskog niza: geometrijski red kovergira i ima konačnu sumu ako je kvocijent manji od jedinice, međutim očito tu sumu nećemo moći izračunati rekurzivnim generiranjem i zbrajanjem svih elemenata niza.

10 Jezgra $\kappa(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$ nije valjana (Mercerova) jezgra. Pretpostavimo suprotno: da postoji preslikavanje $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^m$ takvo da $\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$. Prema definiciji ove jezgre vrijedi $\kappa(\mathbf{x}, \mathbf{x}) = 0$, a to povlači $\phi(\mathbf{x}) = \mathbf{0}$, budući da jedino tako skalarni umnožak svakog vektora sa samim sobom može biti jednak nuli. No, ako $\phi(\mathbf{x}) = \mathbf{0}$, onda $\kappa(\mathbf{x}, \mathbf{x}') = 0$ za bilo koji par vektora \mathbf{x} i \mathbf{x}' , a to je u kontradikciji s definicijom funkcije κ . Stoga ne postoji ϕ takvo $\phi(\mathbf{x})^T \phi(\mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$, pa zaključujemo da $\kappa(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$ nije valjana jezgra.

11 Naziv **Mercerove jezgre** dolazi od **Mercerovog teorema**, koji je jedan od najvažnijih rezultata funkcijске analize za teoriju računalnog učenja. Teorem je 1909. godine dokazao britanski matematičar James Mercer. Opis i dokaz možete pronaći ovdje: <http://homes.cs.washington.edu/~thickstn/docs/mercser.pdf>

12 Gramova matrica je matrica skalarnih umnožaka vektora. Ta je matrica simetrična i pozitivno semidefinitna. Dokaz da je Gramova matrica pozitivno semidefinitna i da je svaka pozitivno semidefinitna matrica Gramova matrica je jednostavan i možete ga pogledati ovdje: https://en.wikipedia.org/wiki/Gramian_matrix#Positive-semidefiniteness.

U ovom predmetu Gramovu matricu susrećemo na dva mesta, i svaki put je definiramo drugačije, što može dovesti do zabune. Prvo mjesto gdje smo susreli Gramovu matricu je kod regresije. Tamo smo Gramovu matricu definirali kao $\mathbf{X}^T \mathbf{X}$, gdje je \mathbf{X} matrica dizajna dimenzija $N \times (n+1)$, odnosno kao $\Phi^T \Phi$, gdje je Φ matrica dizajna u prostoru značajki dimenzija $N \times (m+1)$. Tako definirana Gramova matrica odgovara skalarnome umnošku vektor-stupaca, koji pak odgovaraju značajkama. Dimenzija te Gramove matrice je $(n+1) \times (n+1)$, odnosno $(m+1) \times (m+1)$. U statistici, tu matricu također nazivamo **matrica raspršenja** (engl. *scatter matrix*). Drugo mjesto gdje susrećemo Gramovu matricu jest ova naša današnja tema: naime, jezgrena matrica **Mercerove jezgre** je Gramova matrica. U ovom slučaju, vektori čiji skalarni umnožak računamo su pojedinačni primjeri, dakle vektor-retci matrice dizajna, a ne vektor-stupci kao kod regresije. Gramova je matrica u tom slučaju definirana kao $\Phi \Phi^T$, i njezina je dimenzija $N \times N$. To su, dakle, skalarni umnošci između svih parova primjera iz skupa od N primjera.

Gramova matrica nazvana je po danskom matematičaru s kraja 18. i početka 19. stoljeća, Jorgenu Pedersenu Gramu. Gram se bavio teorijom vjerojatnošću, numeričkom analizom, teorijom brojeva i aktuaristikom (modeliranjem financijskog rizika), a osnovao je i svoju osiguravateljsku tvrtku. Poginuo je 1916. godine kada je na njega u Kopenhagenu naletio bicikl. Ferovcima je Gram vjerojatno poznat po Gram-Schmidtovom postupku ortogonalizacije u linearnoj algebri.

13 Provjeru je li matrica \mathbf{K} pozitivno semidefinitna možemo napraviti tako da izračunamo njezine **svojstvene vrijednosti**. Naime, matrica je pozitivno semidefinitna ako i samo ako su sve njezine svojstvene vrijednosti nenegativne. Za nalaženje svojstvenih vrijednosti matrice u numeričkoj analizi postoji niz algoritama; v. https://en.wikipedia.org/wiki/Eigenvalue_algorithm. Tipično se koristi ili rastav matrice na singularne vrijednosti (SVD) ili neki iterativan algoritam za izračun svojstvenih vrijednosti, npr., Arnoldijev algoritam, koji se temelji na gore spomenutoj Gram-Schmidtovoj ortogonalizaciji. Standardna referenca za numeričke algoritme nad matricama je ([Van Loan and Golub](#))

lub, 1983).

- [14] Još primjera Mercerovih jezgri možete vidjeti ovdje: https://en.wikipedia.org/wiki/Positive-definite_kernel#Examples_of_p.d._kernels

- [15] Dokaz da je **Gaussova jezgra** Mercerova jezgra:

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{x}') &= \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2) \\ &= \exp(-\gamma (\mathbf{x} - \mathbf{x}')^T (\mathbf{x} - \mathbf{x}')) \\ &= \exp(-\gamma (\mathbf{x}^T \mathbf{x}' - 2\mathbf{x}^T \mathbf{x}' + (\mathbf{x}')^T \mathbf{x}')) \\ &= \exp(-\gamma \mathbf{x}^T \mathbf{x}') \exp(2\gamma \mathbf{x}^T \mathbf{x}) \exp(-\gamma (\mathbf{x}')^T \mathbf{x}') \\ &= f(\mathbf{x}) \exp(\alpha \kappa'(\mathbf{x}, \mathbf{x}')) f(\mathbf{x}')\end{aligned}$$

Dobili smo izraz koji je umnožak funkcije koja ovisi samo o vektoru \mathbf{x} , funkcije koje ovisi samo vektoru \mathbf{x}' , te eksponcijalne funkcije od skalarog umnoška (koji jest Mercerova jezgra) pomnoženog s nekim skalarom $\alpha = 2\gamma$. Na ovom mjestu možemo se pozvati na operacije nad Mercerovim jezgrama koje zadržavaju svojstvo Mercerove jezgre (v. glavni tekst u nastavku) te zaključiti da je dobiven izraz Mercerova jezgra.

- [16] Detaljnije u <http://stats.stackexchange.com/a/80405/93766>, <https://math.stackexchange.com/q/130554/273854>, i <http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/svms/RBFKernel.pdf>.

- [17] Pokažimo da zbroj dviju jezgri odgovara **konkatenaciji vektora** u prostoru značajki. Krenimo od konkatenacije vektora. Neka su

$$\begin{aligned}\phi(\mathbf{x}) &= [\phi_1(\mathbf{x}), \phi_2(\mathbf{x})] \\ \phi(\mathbf{z}) &= [\phi_1(\mathbf{z}), \phi_2(\mathbf{z})]\end{aligned}$$

dva vektora u prostoru značajki, svaki dobiven konkatenacijem dvaju vektora. Mercerova jezgra primjenjena na vektore \mathbf{x} i \mathbf{z} je:

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{z}) &= \phi(\mathbf{x})^T \phi(\mathbf{z}) \\ &= [\phi_1(\mathbf{x}), \phi_2(\mathbf{x})]^T [\phi_1(\mathbf{z}), \phi_2(\mathbf{z})] \\ &= \phi_1(\mathbf{x})^T \phi_1(\mathbf{z}) + \phi_2(\mathbf{x})^T \phi_2(\mathbf{z}) \\ &= \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})\end{aligned}$$

- [18] Kako izvesti ujezgenu inačicu L2-regularizirane linearne možete pročitati u (Murphy, 2012) (poglavlje 14.4.3).

Literatura

- F. Almeida and G. Xexéo. Word embeddings: A survey. *arXiv preprint arXiv:1901.09069*, 2019.
- C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
- T. Gärtner. A survey of kernels for structured data. *ACM SIGKDD Explorations Newsletter*, 5(1):49–58, 2003.
- T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.
- N. M. Kriege, F. D. Johansson, and C. Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.
- K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

- B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive Computation and Machine Learning series, 2018.
- J. Shawe-Taylor, N. Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.
- C. F. Van Loan and G. H. Golub. *Matrix computations*. Johns Hopkins University Press Baltimore, 1983.

13. Procjena parametara

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v2.3

Drugu polovicu predmeta započet ćemo razmatranjem probabilističkih (odnosno vjerojatnosnih) modela. Općenito govoreći, to su modeli koji se oslanjaju na **teoriju vjerojatnosti** kako bi modelirali distribuciju primjera i njihovih oznaka, i na temelju toga radili klasifikaciju ili regresiju.

Osnovni mehanizam kod probabilističkih modela jest **procjena parametara** (engl. **parameter estimation**): kako, na temelju označenog skupa podataka, odrediti koji su parametri distribucije podataka. Kod probabilističkih modela, parametri koje tako procijenimo su onda ujedno i parametri našeg modela. Naš označeni skup podataka je ustvari **uzorak**, a procjenom parametara na temelju uzorka zapravo se bavi **statistika**, pa ćemo danas dakle malo pozabaviti statistikom. Danas ćemo objasniti samo osnovnu ideju procjene parametara, a onda ćemo idući put pričati o konkretnim metodama procjene parametara.

Također, u predavanjima koje slijede vratit ćemo se na neke stvari koje smo već bili naučili u prvom dijelu predmeta i pogledati ih ponovo, ali ovog puta gledat ćemo na njih kao na problem procjene parametara. To će dovesti do nekih zanimljivih spoznaja.

1 Motivacija

Prednost probabilističkih modela je trojaka. Prvo, ti su modeli temeljeni na **teoriji vjerojatnosti**, koja je vrlo razrađena, pa dakle imamo dobru teorijsku podlogu koju dobro razumijemo. Druga prednost probabilističkih modela jest što **modeliraju vjerojatnosti**, dakle imat ćemo informaciju koliko je klasifikacija pouzdana. Treća prednost je da su probabilistički modeli idealni u situacijama kada imamo nekakvo **apriorno znanje** o problemu, i želimo to znanje ugraditi u naš model, kako bi model profitirao od tog znanja i radio još točniju klasifikaciju/regresiju. To znanje može biti jednostavno, ali i vrlo složeno, što može dovesti do modela složenih struktura. Nadalje, probabilistički modeli će raditi dosta dobro i onda kada nemamo puno podataka, ako u model uspijemo ugraditi pretpostavke o podatcima koje su točne.

Postoje **parametarski i neparametarski** probabilistički modeli. Mi ćemo se u ovom predmetu fokusirati na parametarske modele, koji se ionako češće koriste. Prisjetimo se: parametarski modeli su modeli koji prepostavljaju da se podatci pokoravaju nekoj vjerojatnosnoj distribuciji, koju moramo unaprijed odabrati.

Za motivaciju, spomenimo jedan konkretan – i najjednostavniji – primjer probabilističkog klasifikatora: **Bayesov klasifikator**. Bayesov klasifikator modelira vjerojatnost oznake za zadani primjer, $P(y|\mathbf{x})$, i to čini na temelju dviju pretpostavljenih vjerojatnosti: **vjerojatnosti primjera za zadalu oznaku** $P(\mathbf{x}|y)$ i **apriorne vjerojatnosti oznaka** $P(y)$. Točnije, vjerojatnost oznake za dani primjer bit će proporcionalna umnošku ovih dviju vjerojatnosti:

$$P(y|\mathbf{x}) \propto P(\mathbf{x}|y)P(y)$$

Dakle, da bismo izračunali vjerojatnost neke oznake y za zadani primjer \mathbf{x} , moramo izračunati dvije vjerojatnosti: vjerojatnost da slučajna varijabla \mathbf{x} poprimi neku konkretnu vrijednost ako znamo da je slučajna varijabla y poprimila neku konkretnu vrijednost (uvjetna vjerojatnost) i

vjerojatnost da slučajna varijabla y poprими neku konkretnu vrijednost. Pitanje je kako ćemo izračunati te vjerojatnosti. To ćemo napraviti tako da ćemo – ovisno o vrsti podataka – odabrati dvije konkretne teorijske distribucije – npr., Gaussovu distribuciju za $P(y|\mathbf{x})$ i Bernoullijevu distribuciju za $P(y)$ – te ćemo pretpostaviti da se podaci pokoravaju tim distribucijama. Onda ćemo **procijeniti parametre** tih distribucija na temelju podataka. Kada to imamo, možemo raditi predikciju za nove primjere, koja se onda svodi na izračun vjerojatnosti $P(y|\mathbf{x})$.

Možda ćete sada uočiti da procjena parametara zapravo znači da određujemo parametre modela na temelju podataka. Nije li to zapravo učenje modela? Jest! Međutim u svijetu statistike, to se zove procjena parametara. Dakle, kada statističar kaže “procjena parametara modela”, to je kao da kaže “učenje/treniranje modela”.

Statistika se temelji na teoriji vjerojatnosti. Hajdemo se malo prisjetiti osnova teorija vjerojatnosti.

1

2 Slučajne varijable

2.1 Slučajna varijabla i distribucija

Kod probabilističkih modela, primjere \mathbf{x} i oznake y modelirat ćemo kao **slučajne varijable**. Slučajna varijabla X ima unaprijed definiran skup vrijednosti $\{x_j\}$ (diskretan ili kontinuiran, konačan ili beskonačan). Ako je skup vrijednosti diskretan, onda govorimo o **diskretnoj slučajnoj varijabli**. Npr., oznaka klase y je diskretna slučajna varijabla.

2

Vrijednost $P(X = x)$ jest **vjerojatnost** da diskretna slučajna varijabla X poprimi vrijednost x , tj. vjerojatnost da se slučajna varijabla X realizira kao x . U nastavku ćemo umjesto $P(X = x)$ pisati kraće $P(x)$. Vrijedi $P(x_i) \geq 0$ i $\sum_i P(x_i) = 1$. Time je zapravo definirana **diskretna distribucija (razdioba) vjerojatnosti**.

3

Navedeno vrijedi za diskretnu slučajnu varijablu. Ako je skup mogućih vrijednosti slučajne varijable kontinuiran, npr., prosjek ocjena studenta, onda govorimo o **kontinuiranoj (neprekidnoj) slučajnoj varijabli**. Za kontinuiranu (neprekidnu) slučajnu varijablu definiramo **funkciju gustoće vjerojatnosti** (engl. *probability density function; PDF*), koju označavamo kao $p(x)$. Za funkciju gustoće vjerojatnosti vrijedi $p(x) \geq 0$ i $\int_{-\infty}^{\infty} p(x) dx = 1$.

4

5

Kažemo da funkcijom gustoće vjerojatnosti $p(x)$ definirana **kontinuirana distribucija (razdioba) vjerojatnosti**. U nastavku ćemo, kao što je tipično u literaturi iz strojnog učenja, koristiti izraz “gustoća $p(x)$ ” ili (pomalo neprecizno) “distribucija $p(x)$ ”, misleći pritom na funkciju gustoće vjerojatnosti $p(x)$.

2.2 Očekivanje, varijanca i kovarijanca

Budući da slučajne varijable mogu poprimiti različite vrijednosti, korisno je da ih pokušamo nekako okarakterizirati. U nastavku ćemo se ograničiti na **numeričke** slučajne varijable (bilo diskretne ili kontinuirane). Numeričke slučajne varijable možemo okarakterizirati preko njihovog očekivanja, varijance i kovarijance.

Krenimo od očekivanja slučajne varijable. Prosječna vrijednost diskretne slučajne varijable X čija je distribucija $P(x)$ naziva se **(matematičko) očekivanje** ili **očekivana vrijednost** varijable X i definira se kao:

$$\mathbb{E}[X] = \sum_x xP(x)$$

To je zapravo težinska suma vrijednosti varijable.

► PRIMJER

Zamislite da student rješava blic s pitanjima koja imaju 4 ponuđena odgovora, i samo jedan odgovor je točan. Točan odgovor nosi 1 bod, a netočan -0.5 bodova (predrastično, kao što ćemo vidjeti).

Prepostavimo da student pojma nema i da odgovara nasumično. To znači da su bodovi koje će dobiti na svakom pojedinačnom zadatku slučajna varijabla. Budući da student odgovara nasumično, vjerojatnost da pogodi 1 točan odgovor od 4 ponuđena je 0.25, a vjerojatnost da promaši je 0.75. Dakle, očekivanje bodova na jednom zadatku je:

$$\mathbb{E}[X] = 1 \cdot 0.25 + (-0.5) \cdot 0.75 = -0.125$$

U slučaju kontinuirane slučajne varijable X s gustoćom vjerojatnosti $p(x)$, očekivanje je

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x p(x) dx$$

Osim očekivanja, drugi način da se okarakterizira slučajna varijabla je **varijanca**. Varijanca slučajne varijable X iskazuje koliko vrijednosti variraju oko očekivane vrijednosti:

$$\text{Var}(X) = \sigma_X^2 = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2.$$

Konačno, zanimljiva će nam biti i **kovarijanca**. Kovarijanca opisuje odnos između **dviju** slučajnih varijabli, odnosno opisuje u kojoj mjeri slučajne varijable zajednički variraju oko svojih očekivanih vrijednosti. Kovarijanca varijabli X i Y definirana je kao

$$\text{Cov}(X, Y) = \sigma_{X,Y} = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$$

Vrijedi $\text{Cov}(X, Y) = \text{Cov}(Y, X)$. Primijetite da je kovarijanca varijable sa samom sobom varijanca, tj. $\text{Cov}(X, X) = \text{Var}(X)$ odnosno $\sigma_{X,X} = \sigma_X^2$.

► PRIMJER

Recimo da $X = \{1, 2\}$ i $Y = \{1, 2\}$. Neka su vjerojatnosti $P(X, Y)$ sljedeće:

$$P(1, 1) = 0.5, P(1, 2) = 0.1, P(2, 1) = P(2, 2) = 0.2$$

Marginalne vjerojatnosti onda su:

$$P(X = 1) = 0.6, P(X = 2) = 0.4, P(Y = 1) = 0.7, P(Y = 2) = 0.3$$

Očekivanja varijabli onda su:

$$\mathbb{E}[X] = 1 \cdot 0.6 + 2 \cdot 0.4 = 1.4$$

$$\mathbb{E}[Y] = 1 \cdot 0.7 + 2 \cdot 0.3 = 1.3$$

Varijance varijabli onda su:

$$\text{Var}(X) = 0.6 \cdot (1 - 1.4)^2 + 0.4 \cdot (2 - 1.4)^2 = 0.24$$

$$\text{Var}(Y) = 0.7 \cdot (1 - 1.3)^2 + 0.3 \cdot (2 - 1.3)^2 = 0.21$$

Korelacija između slučajnih varijabli X i Y onda je:

$$\begin{aligned} \text{Cov}(X, Y) &= \sigma_{X,Y} = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \\ &= \sum_{x \in \{1, 2\}} \sum_{y \in \{1, 2\}} P(x, y)(x - \mathbb{E}[X])(y - \mathbb{E}[Y]) \\ &= 0.5 \cdot (1 - 1.4) \cdot (1 - 1.3) + 0.1 \cdot (1 - 1.4) \cdot (2 - 1.3) + \dots \\ &= 0.08 \end{aligned}$$

6

Za slučajne varijable X i Y za koje vrijedi $\text{Var}(X) \neq 0$ i $\text{Var}(Y) \neq 0$ definiran je **Pearsonov koeficijent korelacije**:

$$\rho_{X,Y} = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}} = \frac{\sigma_{X,Y}}{\sigma_X \sigma_Y}$$

Pearsonov koeficijent upućuje na to koliko su varijable X i Y međusobno **linearno zavisne**. Za savršenu pozitivnu linearu ovisnost vrijedi $\rho_{X,Y} = 1$, dok za savršenu negativnu linearu ovisnost vrijedi $\rho_{X,Y} = -1$.

Npr., visina u centimetrima i visina u metrima će biti savršeno linearne korelirane, starnosna dob i broj godina radnog staža će biti pozitivno korelirane, premda ne savršeno, dok su, npr., starost automobila i njegova cijena negativno korelirane. Pritom primjetite da vrijednost koeficijenta korelacije ne ovisi o skalamu varijabli X i Y (koeficijent korelacije je bezdimenzijska mjeru).

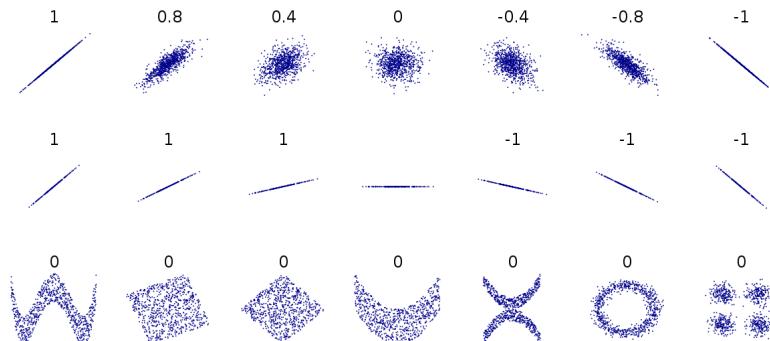
► PRIMJER

Za raniji primjer, koeficijent korelacije je:

$$\rho_{X,Y} = \frac{0.08}{\sqrt{0.24}\sqrt{0.21}} = \frac{0.08}{0.23} = 0.35$$

što je umjerena pozitivna korelacija.

Važno je ovdje primjetiti da Pearsonov koeficijent korelacije mjeri isključivo linearu zavisnost dviju varijable. Varijable mogu biti nelinearno zavisne, a imati nizak Pearsonov koeficijent korelacije. To ilustriraju sljedeći primjeri:



Slika prikazuje grafikone raspršenja (engl. *scatter plots*) za dvije varijable i vrijednosti Pearsonovog koeficijenta korelacije. U srednjem retku varijable su savršeno pozitivno ili negativno korelirane. Vidimo da iznos korelacije ne ovisi o nagibu pravca: nije bitno koliko se varijabla Y poveća ili smanji s povećanjem ili smanjenjem varijable X , već da li se to uvijek konzistentno događa. Slučaj u sredini (kada je varijabla Y konstanta) je slučaj za koji korelacija nije definirana. U prvome retku imamo slučajeve kada je korelacija manja od 1 (umjerena ili slaba) ili kada je jednaka nuli. Korelacija je jednaka nuli ako varijable nisu zavisne, što na grafikonu raspršenja vidimo kao "oblak". No, korelacija može biti jednaka nuli i kada varijable jesu zavisne, ali ta zavisnost nije linearna. To je prikazano u trećem retku. Za sve te slučajeve korelacija je jednaka nuli, premda je očito da između varijabli postoji neka zavisnost, koja se manifestira kao neki uzorak u grafikonu raspšenja. Specifično, u posljednjem (skroz desnom) slučaju u podatcima postoje grupe (klasteri) varijabli, međutim korelacija će tu i dalje biti nula jer je prosječni odmak točke od pravca (rezidual) jednak nuli.

2.3 Višedimenzijska slučajna varijabla

U strojnom učenju, primjeri uobičajeno imaju više značajki, $\mathbf{x} = (x_1, \dots, x_n)$. U teoriji vjerojatnosti takvu višedimenzijsku varijablu modeliramo **slučajnim vektorom**, (X_1, \dots, X_n) . Često će nas zanimati i jesu li i kako značajke međusobno korelirane: npr., sjetimo se modela linearne regresije i problema multikolinearnosti, gdje želimo izbaciti značajke koje su korelirane. Jedan način da dobijemo uvid u korelacije između značajki jest da izračunamo korelacije između svih parova značajki. To nam daje matricu koju zovemo **matrica kovarijacije (kovarijacijska matrica)**, koju označavamo sa Σ . Elementi te matrice su kovarijacije između svih parova varijabli:

$$\Sigma_{ij} = \text{Cov}(X_i, X_j) = \sigma_{ij} = \mathbb{E}[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])]$$

Kovarijacijska matrica je **kvadratna simetrična matrica** (jer $\text{Cov}(X, Y) = \text{Cov}(Y, X)$), koja na glavnoj dijagonali ima varijance varijabli (jer $\text{Cov}(X, X) = \text{Var}(X)$), a izvan dijagonale kovarijance svih parova varijabli:

$$\Sigma = \begin{pmatrix} \text{Var}(X_1) & \text{Cov}(X_1, X_2) & \dots & \text{Cov}(X_1, X_n) \\ \text{Cov}(X_2, X_1) & \text{Var}(X_2) & \dots & \text{Cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}(X_n, X_1) & \text{Cov}(X_n, X_2) & \dots & \text{Var}(X_n) \end{pmatrix}$$

U matričnom računu, kovarijacijska je matrica definirana kao:

$$\Sigma = \mathbb{E}[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^T]$$

Primijetite da ovdje računamo vektorski produkt (vanjski produkt) vektora $(\mathbf{X} - \mathbb{E}[\mathbf{X}])$ sa samim sobom, a ne skalarni produkt, pa je rezultat matrica dimenzija $(n \times 1) \times (1 \times n) = n \times n$, a ne skalar.

Općenito, kovarijacijska matrica će imati ne-nul vrijednosti izvan dijagonale. Međutim, posebno će nam biti zanimljiva dva specifična slučaja:

- (1) Ako su varijable X_1, \dots, X_n međusobno linearno nezavisne, onda $\text{Cov}(X_i, X_j) = 0$ za $i \neq j$ i kovarijacijska je matrica **dijagonalna matrica**, $\Sigma = \text{diag}(\sigma_i^2)$;
- (2) Ako nezavisne varijable X_1, \dots, X_n imaju jednaku varijancu, onda $\sigma_i^2 = \sigma^2$, pa kovarijacijska matrica degenerira u $\Sigma = \sigma^2 \mathbf{I}$, gdje je \mathbf{I} jedinična matrica. Takav slučaj nazivamo **izotropnom kovarijancom**.

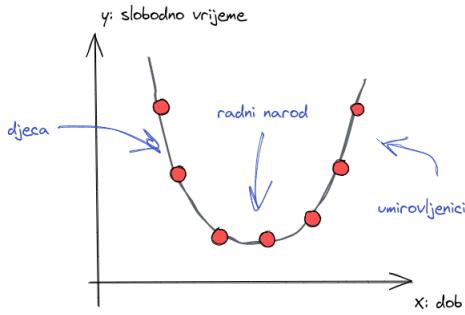
Kasnije će nam trebati i inverz kovarijacijske matrice. Kovarijacijska matrica je **pozitivno semidefinitna**, tj. $\mathbf{x}^T \Sigma \mathbf{x} \geq 0$. To znači da ne mora nužno imati inverz, tj. da može biti singularna. (Kada bi matrica bila **pozitivno definitna**, onda bi nužno imala inverz.) Konkretno, kovarijacijska matrica neće imati inverz akko je neki od elemenata na dijagonali jednak nuli ili ako postoji linearna zavisnost između redaka odnosno stupaca matrice dizajna \mathbf{X} . Sada se možemo pitati kada se to zapravo događa? Kada će neki element dijagonale kovarijacijske matrice biti jednak nuli? Budući da su na dijagonali nalaze varijance za svaku značajku, to se dogoditi samo ako neka značajka ima varijancu jednaku nuli, a to znači da je značajka konstantna. Konstantna značajka očito nije varijabla (jer ne varira) i ona nam je potpuno beskorisna: naime, ako joj je vrijednost jednaka za sve primjere, onda očito nema utjecaja na predikciju pojedinačnog primjera. Što je s linearom zavisnošću redaka odnosno stupaca? Nju ćemo imati ako postoji **savršena multikolinearnost** značajki, tj. ako je neka značajka **redundantna** jer se može predvidjeti iz drugih značajki. Znamo već i da je multikolinearnost problem i kod poopćenih linearnih modela, gdje ona uzrokuje nestabilnost rješenja uslijed loše kondicionirane matrice dizajna.

2.4 Nezavisnost varijabli

Kovarijanca dakle mjeri linearu zavisnost između varijabli. Međutim, već smo napomenuli da varijable mogu biti zavisne, a da ta zavisnost nije linearna.

► PRIMJER

Prisjetimo se primjera ovisnosti slobodnog vremena o dobi:



Premda ovdje očito postoji ovisnost varijable Y o varijabli X (i obrnuto), tu ovisnost nije moguće iskazati korelacijom odnosno kovarijacijom. Naime, ovdje za varijable X i Y vrijedi $\text{Cov}(X, Y) = \rho_{X,Y} = 0$. (Zašto? Zato jer je ova krivulja simetrična po Y osi, pa će se očekivanja $(X - \mathbb{E}[X])$ poništiti. Čim je grafikon raspršenja simetričan po jednoj od osi, korelacija je jednaka nuli.)

Nas će u strojnom učenju zanimati jesu li varijable općenito zavisne, i to bilo linearno ili nelinearno. Zato uvodimo pojam **nezavisnosti slučajnih varijabli**. Dvije slučajne varijable X i Y su **(stohastički) nezavisne** ako i samo ako:

$$P(X, Y) = P(X)P(Y)$$

Intuitivno, varijable X i Y su nezavisne ako je vjerojanost zajedničkog ishoda jednaka umnošku vjerojatnosti pojedinačnih ishoda. Npr., hoće li sutra u Samoboru pasti snijeg i hoće li Kim Kardashian sutra objaviti fotografiju na Instagramu nezavisni su događaji, i vjerojatnost zajedničkog ishoda jednostavno je umnožak vjerojatnosti pojedinačnih ishoda.

Ako su varijable X i Y nezavisne, onda vrijedi $\text{Cov}(X, Y) = \rho_{X,Y} = 0$. Dakle, **nezavisne varijable su nekorelirane**. No, kao što smo već napomenuli i kao što smo vidjeli na gornjem primjeru (ovisnost prihoda o dobi), obrat općenito ne vrijedi: koeficijent korelacijske može biti jednak nuli, a da su varijable ipak nelinearno zavisne (jer koeficijent korelacijske mjeri isključivo linearu zavisnost varijabli).

Ok, sad smo se prisjetili što je to slučajna varijabla i koja su njezina svojstva. Kao što smo već rekli, kod parametarskih probabilističkih modela mi ćemo prepostaviti da se slučajne varijable pokoravaju nekoj distribuciji. Te distribucije nećemo izmišljati, nego ćemo koristiti poznate **teorijske distribucije**. Pa, pogledajmo koje su to konkretno distribucije s kojima ćemo raditi.

3 Osnovne vjerojatnosne distribucije

Odabir vjerojatnosne distribucije prvenstveno ovisi o tome s kakvim podatcima radimo: jesu li podatci diskretni ili kontinuirani, te jesu li jednodimenzionalni ili višedimenzionalni. U strojnom učenju tipično koristimo:

- Diskretna varijabla:
 - Jednodimenzionalna:

- * Binarna: **Bernoullijeva distribucija**
- * Viševrijednosna: **Kategorička (multinulijeva) distribucija**
- Višedimenzijska: Konkatenirani vektor binarnih/viševrijednosnih varijabli
- Kontinuirana varijabla:
 - Jednodimenzijska: **univariatna Gaussova (normalna) distribucija**
 - Višedimenzijska: **multivariatna Gaussova (normalna) distribucija**

Premda su ovo najčešće korištene distribucije, u strojnom učenju susrećemo i neke druge, npr., beta-distribuciju, Dirichletovu distribuciju i Laplaceovu distribuciju.

Ove se distribucije koriste kako za modeliranje primjera \mathbf{x} , tako i za modeliranje oznaka y . Ulagni primjer tipično je **višedimenzijska** slučajna varijabla, a oznaka je **jednodimenzijska** slučajna varijabla.

► PRIMJER

Da vidimo kako biste se snašli s odabirom distribucija.

- Recimo da radimo predviđanje prosjeka ocjena studenta četvrte godine na temelju ocjena iz prethodne tri godine. Očito, riječ je o regresijskom problemu. Koju distribuciju biste iskoristili za vektor značajki \mathbf{x} ? Odgovor je: multivariatna Gaussova distribucija, jer imamo više kontinuiranih značajki.
- Pretpostavite da, umjesto prosjeka prethodnih godina, imate ocjene za svaki predmet (od 2 do 5). Kako biste sada modelirali vektor \mathbf{x} ? Odgovor je: opet multivariatna Gaussova distribucija. Nema veze što su značajke cijeli brojevi, bitno je da su brojevi, a ne kategoričke vrijednosti. Čim radimo s brojevima, bilo cijelim ili realnim, znači da između njih postoji potpuni uređaj (1 je manje od 2, koji je manji od 3 itd.), i taj je uređaj bitan i želimo ga modelirati. Kada bismo koristili kategoričku distribuciju, ne bismo modelirali taj uređaj, jer između kategorija nema nikakvog uređaja.
- Sada pretpostavite da, umjesto ocjena za svaki predmet, imate podatak o tome u koju je srednju školu student/ica išao/la i podatak o tome iz kojeg je grada. Kako biste modelirali značajku \mathbf{x} ? Odgovor je: sa dvije multinulijeve značajke. A koliko mogućih vrijednosti imaju te značajke? Odgovor je: onoliko koliko ima različitih škola odnosno gradova.
- A sad pretpostavite da, umjesto regresije (predviđanje prosjeka ocjena četvrte godine), radite klasifikaciju (tko će pasti godinu?). Kako biste modelirali varijablu oznake y ? Odgovor je: Bernoullijevom varijablom, jer postoje dvije moguće vrijednosti.
- Konačno, pretpostavite da, umjesto predviđanja tko će pasti godinu, radite predviđanje tko će, u godinu dana nakon završetka studija, pronaći posao u HR, tko u inozemstvu, a tko neće pronaći posao. Kako biste sada modelirali varijablu oznake y ? Odgovor je: multinulijeva varijablom s tri moguće vrijednosti.

Pogledajmo sada malo detaljnije svaku od ovih distribucija, premda smo se s većinom njih već susreli.

3.1 Bernoullijeva distribucija

Bernoullijeva distribucija modelira vjerojatnost diskretne slučajne varijable s dva moguća ishoda, $\{0, 1\}$ – dakle, **binarne varijable**. Bernoullijeva distribucija ima svega jedan parametar, μ , koji određuje koja je vjerojatnost da $x = 1$. Očito, vjerojatnost da $\mu = 0$ je onda $1 - \mu$. To pišemo ovako:

$$P(X = x|\mu) = \begin{cases} \mu & \text{ako } x = 1 \\ 1 - \mu & \text{inače} \end{cases} = \mu^x(1 - \mu)^{1-x}$$

Ako uvrstimo ovu distribuciju u definiciju za očekivanje odnosno varijance, dobivamo $\mathbb{E}[X] = \mu$ odnosno $\text{Var}(X) = \mu(1 - \mu)$.

3.2 Kategorička (aka “multinulijeva”) distribucija

Kategorička distribucija (također, u zadnje vrijeme nazivana **“multinulijeva”**, po analogiji s Bernouljevom) modelira slučajnu varijablu koja poprima jednu (i samo jednu) od K mogućih vrijednosti – dakle, **viševrijednosnu diskretnu varijablu**.

Ovu distribuciju smo već sreli kada smo pričali o multinomijalnoj logističkoj regresiji, gdje smo njome modelirali varijablu označe, koja je mogla poprimiti vrijednost jedne od K klasa.

Kategoričku varijablu modeliramo kao binaran vektor **indikatorskih varijabli**:

$$\mathbf{x} = (x_1, x_2, \dots, x_K)$$

Binarnih varijabli ima onoliko koliko kategorička varijabla ima mogućih različitih vrijednosti. Samo jedna binarna varijabla će imati vrijednost jedan, a sve ostale su nula. To se zove **vektor 1-od-K** ili **one-hot encoding**. Npr., trovrijednosna kategorička varijabla koja je poprimila drugu od tri vrijednosti bila bi predstavljena kao $\mathbf{x} = (0, 1, 0)$.

Vjerojatnost da kategorička varijabla poprimi neku vrijednost je općenito različita za svaku vrijednost. Drugim riječima, svaka binarna varijabla iz vektora indikatorskih varijabli ima svoju vjerojatnost da bude jednaka jedinici. To znači da imamo po jedan parametar μ za svaku binarnu varijablu. Te parametre možemo strpati u vektor parametara $\boldsymbol{\mu}$:

$$\boldsymbol{\mu} = (\mu_1, \dots, \mu_K)$$

pri čemu mora vrijediti $\sum_k \mu_k = 1$ i $\mu_k \geq 0$.

Kako sada napisati vjerojatnost da kategorička slučajna varijabla poprimi neku svoju konkretnu vrijednost? To smo već vidjeli kada smo pričali o multinomijalnoj regresiji: ideja je da se matematički napiše primjena binarnog vektora \mathbf{x} kao **maske** nad vektorom $\boldsymbol{\mu}$, tako da izabremo μ_k koji odgovara vrijednosti x_k koja je u vektoru \mathbf{x} postavljena na jedinicu:

$$P(X = \mathbf{x} | \boldsymbol{\mu}) = \prod_{k=1}^K \mu_k^{x_k}$$

► PRIMJER

Četverovrijednosna kategorička varijabla \mathbf{x} neka je poprimila treću vrijednost, tj.

$$\mathbf{x} = x_3 \quad \Rightarrow \quad \mathbf{x} = (0, 0, 1, 0)$$

Vjerojatnosti pojedinačnih vrijednosti neka su:

$$\boldsymbol{\mu} = (0.2, 0.3, 0.4, 0.1)$$

Onda je vjerojatnost da je varijabla poprimila treću vrijednost jednaka:

$$P(\mathbf{x} = (0, 0, 1, 0)) = \prod_{k=1}^4 \mu_k^{x_k} = 1 \cdot 1 \cdot \mu_3 \cdot 1 = \mu_3 = 0.4$$

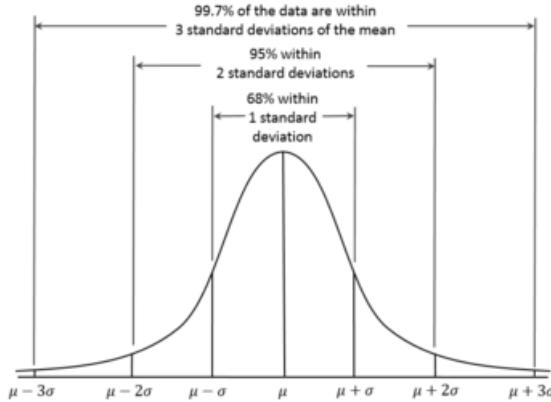
3.3 Gaussova distribucija

Gaussova (normalna) distribucija kontinuirana je distribucija koja se tipično koristi se za modeliranje kontinuirane varijable. Zašto? Prvo, zato što se pokazuje da mnogi prirodni fenomeni slijede Gaussovou distribuciju. Drugi razlog je što je Gaussova distribucija matematički elegantna i jednostavna.

11

Univarijatna (jednodimenzijska) Gaussova distribucija definirana je sljedećom funkcijom gustoće vjerojatnosti (koju smo već vidjeli):

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$$



Iz definicije Gaussove distribucije može se za očekivanje i varijancu slučajne varijable koja se pokorava toj distribuciji izvesti $\mathbb{E}[X] = \mu$ i $\text{Var}(X) = \sigma^2$. Dakle, parametri distribucije μ i σ^2 direktno definiraju očekivanje i varijancu slučajne varijable koja se ravna po Gaussovoj distribuciji.

U strojnom učenju Gaussovu distribuciju koristit ćemo za modeliranje kontinuiranih varijabli uz prisustvo **šuma**. Tu je ideja sljedeća: očekivana vrijednost varijable zapravo je μ , ali zbog šuma dolazi do rasipanja, pa mi u podatcima opažamo neku malo drugačiju vrijednost. Što je šum veći, to je veće rasipanje, odnosno veća je varijanca σ^2 .

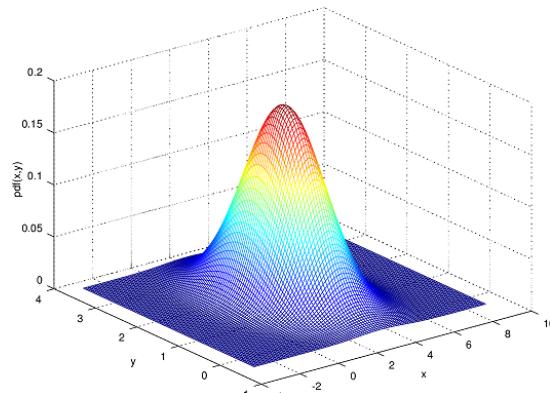
3.4 Multivarijatna Gaussova distribucija

U strojnom učenju često je primjer prikazan kao vektor realnih brojeva, npr., prosjek ocjena kroz četiri razreda srednje škole. Takav primjer modelirat ćemo **multivarijatnom (višedimenzijskom) Gaussovom distribucijom**. Gustoća vjerojatnosti multivarijatne distribucije definirana je ovako:

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right\}$$

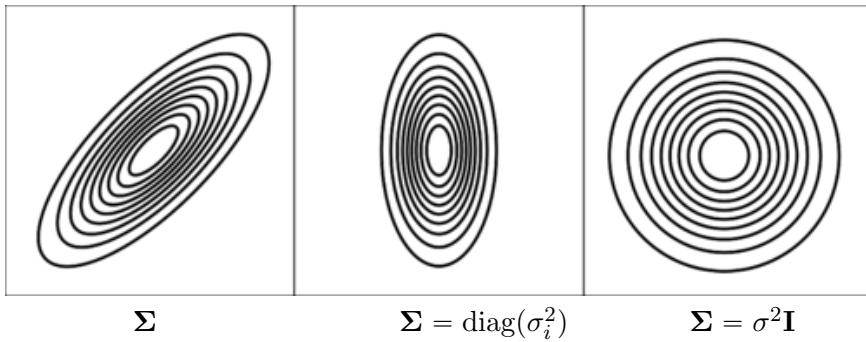
gdje su $\boldsymbol{\mu}$ i $\boldsymbol{\Sigma}$ srednja vrijednost odnosno kovarijacijska matrica, i oni čine parametri multivarijatne Gaussove distribucije.

U dvije dimenzije, imamo dvodimenzijsku odnosno **bivarijatnu** Gaussovou distribuciju, koja izgleda ovako:



Matrica Σ je već spomenuta **kovarijacijska matrica**. Vidimo da nam za izračun gustoće vjerojatnosti treba inverz i determinanta kovarijacijske matrice, te da determinanta ne smije biti nula, jer inače imamo dijeljeje s nulom. Kovarijacijska matrica će imati inverz i determinantu koja je veća od nule samo ako je matrica **pozitivno definitna**. A to, već smo rekli, znači da nema redundantnih značajki i da nema konstantnih značajki (koje su ionako beskorisne). Sjetite se također naših razmatranja u kontekstu invertiranja matrice dizajna kod linearne regresije: čak i ako značajke nisu savršeno multikolinearne, imat ćemo problema s izračunom inverza jer će matrica imati **visok kondicijski broj** i rješenje će biti nestabilno. Vrijede iste napomene kao i ranije: treba eliminirati redundantne značajke.

Eksponent koji se javlja u izrazu za Gaussovou gustoću vjerojatnosti je tzv. **kvadratna forma**: $\Delta^2 = (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})$. Korijen toga je **Mahalanobisova udaljenost** između \mathbf{x} i $\boldsymbol{\mu}$, o kojoj smo već bili pričali kada smo pričali o jezgrenim funkcijama. Udaljenost između primjera bit će, dakle, definirana u ovisnosti o matrici kovarijacije Σ . Isto tako, matrica Σ će određivati kako izgleda Gaussova gustoća vjerojatnosti. Razmotrimo kroz tri konkretna slučaja kako Σ utječe na oblik bivariatne Gaussove gustoće vjerojatnosti:



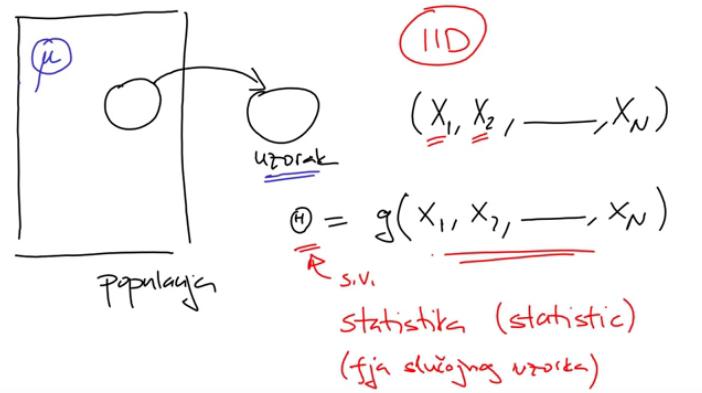
Prva slika pokazuje slučaj kada s porastom vrijednosti varijale X imamo i porast vrijednosti varijable Y , tj. varijable X i Y imaju pozitivnu kovarijaciju (također i: pozitivnu korelaciju). Zbog toga su izokonture Gaussove gustoće vjerojatnosti pozitivno zakošene elipse: gustoća vjerojatnosti za Y raste/pada kako raste/pada gustoća vjerojatnosti za X . Kada bi kovarijacija između X i Y bila negativna, elipse bi bile zakošene u drugu stranu. Taj slučaj, dakle, odgovara situaciji kada imamo punu matricu Σ , tj. matricu koja nema nule izvan glavne dijagonale, što znači da postoje kovarijacije (odnosno korelacije) između varijabli. Srednja slika odgovara situaciji kada je kovarijacijska matrica dijagonalna: na dijagonali su varijance, koje su općenito različite za svaku značajku, a izvan dijagonale su nule, tj. nema kovarijacije. Budući da nema kovarijacije, izokonture su elipse koje nisu zakošene, već su poravnata s osima koordinatnog sustava. No, budući da varijance za svaku varijablu nisu identične, imamo elipse a ne kružnice. U ovom konkretnom slučaju varijanca za Y je veća od varijance za X , pa je elipsa vertikalno izdužena. Kada bi varijanca za X bila veća od varijance za Y , elipsa bi bila horizontalno izdužena. Konačno, na desnoj slici prikazan je slučaj kada je kovarijacijska matrica izotropna: dijagonalna matrica sa jednakovrijednim varijancama na dijagonali. Izokonture gustoće vjerojatnosti sada odgovaraju kružnicama.

Sada smo se upoznali – ili prisjetili, kako tko – osnovnih vjerojatnosnih distribucija. Vidimo da svaka distribucija ima neke svoje **parametre**: npr., Bernoullijeva distribucija ima samo jedan parametar (μ), dok multivariatna Gaussova distribucija ima dva parametra (kovarijacijsku matricu Σ i vektor srednje vrijednosti $\boldsymbol{\mu}$).

Vratimo se sada na osnovno današnje pitanje: ako pretpostavimo da se podatci pokoravaju nekoj teorijskoj distribuciji, kako izračunati njezine parametre na temelju tih podataka? To nas vodi do **procjene parametara**.

4 Procjena parametara

Iz perspektive statistike, podatci koje imamo na raspolaganju jesu **uzorak** iz neke **populacije**. Populacija su svi podatci (kojih može biti konačno ili beskonačno mnogo), a uzorak je samo jedan konačan **podskup**. Taj uzorak treba biti **slučajan**, jer slučajan uzorak (ako je dovoljno velik) je **reprezentativan uzorak**. To onda znači da su podatci u tom uzorku očekivano distribuirani kao i podatci u cijeloj populaciji. Ideja je onda da na temelju tog slučajnog uzorka napravimo **procjenu (estimaciju) parametra** modela koji objašnjava cijelu **populaciju**. Shematski to možemo prikazati ovako:



Što je uzorak? Uzorak je niz slučajnih varijabli: (X_1, X_2, \dots, X_N) – N -torka. Naša pretpostavka su varijable koje čine uzorak **i.i.d. – nezavisno i identično distribuirane** (engl. *independently and identically distributed*). To znači da su slučajne varijable međusobno nezavisne (ishod jedne ne utječe na ishod druge) i da dolaze iz iste distribucije (budući da je cijeli uzorak iz iste populacije). Ova je pretpostavka centralna za mnoge algoritme strojnog učenja, i za sve algoritme koje proučavamo na ovom predmetu.

Na temelju takvog uzorka možemo onda izračunati različite vrijednosti. Označimo to ovako:

$$\Theta = g(X_1, X_2, \dots, X_N)$$

g je dakle neka funkcija koja izračunava nešto na temelju uzorka. Npr., g bi mogla biti funkcija sume, pa bi funkcija računala zbroj vrijednosti u uzorku. Rezultat funkcije g je Θ . Budući da funkcija g radi na slučajnom uzorku, tj. na vektoru slučajnih varijabli, to će rezultat Θ također biti u određenoj mjeri slučajan, tj. Θ je i sama **slučajna varijabla**. Preciznije, Θ je slučajna varijabla čija je vrijednost izračunata na temelju slučajnog uzorka. Takva slučajna varijabla naziva se **statistika** (engl. *statistic*). Dakle, statistika je nekakva **funkcija slučajnog uzorka**. Od beskonačno mnogo različitih statistika koje možemo izmisliti, nas zanimaju one koje odgovaraju nekom **parametru** θ našeg pretpostavljenog modela, koji predstavlja podatke (tj. populaciju). Vrijednost tog parametra nam je nepoznata i nju želimo izračunati, da bismo mogli raditi predikciju. Takvu statistiku, koja nam daje vrijednost parametra populacije, zovemo **procjenitelj (estimator)** parametra θ . Vrijednost procjenitelja $\hat{\theta} = g(x_1, x_2, \dots, x_n)$ naziva se **procjena** (uočite “šešir” na θ). Parametar θ može biti konkretno parametar populacije (npr., srednja vrijednost μ) ili neki drugi parametar koji imamo u našem modelu (npr., vektor težina w), a koji upravlja izgledom distribucije podataka.

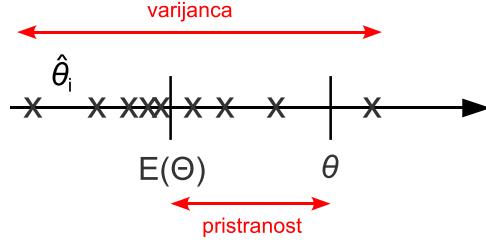
4.1 Pristranost procjenitelja

Budući da je procjenitelj slučajna varijable, on i sam, kao i svaka slučajna varijabla, ima svoje očekivanje i varijancu. Razlika između očekivane vrijednosti procjenitelja i parametra populacije koji želimo procijeniti tim procjeniteljem naziva se **pristranost** (engl. *bias*). Formalno,

pristranost procjenitelja Θ kao procjenitelja parametra θ jednaka je:

$$b_\theta(\Theta) = \mathbb{E}[\Theta] - \theta$$

Graficki to možemo prikazati ovako:



Prava vrijednost parametra populacije je θ , a križići su procjene, odnosno konkretne vrijednosti $\hat{\theta}_i$ procjenitelja Θ na različitim uzorcima iz te populacije. Dakle, to su vrijednosti procjene koje bismo dobivali kada bismo ponavljali izvlačenje uzorka (to je nešto što u praksi ne radimo; u praksi imamo samo jedan uzorak). Očekivano, budući da su uzorci slučajni, dobili bismo svaki puta malo drugačiju procjenu. Srednja vrijednost procjena $\hat{\theta}_i$ jednaka je očekivanju procjenitelja, $\mathbb{E}[\Theta]$ (zapravo, srednja vrijednost procjena je i sama procjena za očekivanje procjenitelja, ali to sad nije bitno). Odstupanje očekivanja procjenitelja $\mathbb{E}[\Theta]$ od prave vrijednosti parametra populacije θ je pristranost procjenitelja Θ .

Procjenitelja parametara populacije ima dobrih i manje dobrih. Željeli bismo procjenitelj koji je što točniji, tj. procjenitelj čija je greška što manja. To često znači da želimo da je procjenitelj čija je pristranost jednaka nuli. Kažemo da je procjenitelj Θ **nepristran procjenitelj** (engl. *unbiased estimator*) parametra θ ako i samo ako:

$$\mathbb{E}[\Theta] = \theta$$

tj. ako je pristranost jednaka nuli. (NB: Pristranost procjenitelja nije ista kao i induktivna pristranost algoritma strojnog učenja, premda je povezana s njom.)

► PRIMJER

Neka je X slučajna varijabla sa vrijednostima iz $x \in \mathbb{R}$. Označimo $\mathbb{E}[X] = \mu$ (srednja vrijednost) i $\text{Var}(X) = \sigma^2$ (varijanca) te varijable. Zanimaju nas parametri μ i σ^2 populacije. Ti parametri su nam nepoznati.

Parametre μ i σ^2 možemo procijeniti na temelju uzorka $\{x^{(i)}\}_{i=1}^N$ pomoću **procjenitelja**. Koje procjenitelje upotrijebiti za ove parametre? Mogli bismo probati s ovima:

$$\hat{\mu} = \frac{1}{N} \sum_i x^{(i)} \quad \hat{\sigma}^2 = \frac{1}{N} \sum_{i=1}^N (x^{(i)} - \hat{\mu})^2$$

Naravno, sad se pitamo jesu li ovo dobri procjenitelji za naše parametre. Drugim riječima, pitamo se je li $\hat{\mu}$ nepristran procjenitelj za μ i je li $\hat{\sigma}^2$ nepristran procjenitelj za σ^2 , tj. je li:

$$\begin{aligned} \mathbb{E}[\hat{\mu}] &= \mu \\ \mathbb{E}[\hat{\sigma}^2] &= \sigma^2 \end{aligned}$$

Može se pokazati, ali to ćemo ostaviti za domaću zadaću, da $\mathbb{E}[\hat{\mu}] = \mu$, tj. $\hat{\mu}$ jest **nepristran** procjenitelj srednje vrijednosti populacije. Međutim, isto se tako može pokazati da $\mathbb{E}[\hat{\sigma}^2] \neq \sigma^2$, tj. $\hat{\sigma}^2$ **nije nepristran** procjenitelj varijance!

$$\mathbb{E}[\hat{\sigma}^2] = \frac{N-1}{N} \sigma^2$$

Vidimo da očekivana vrijednost procjenitelja nije jednaka parametru populacije σ^2 , pa je dakle ovaj procjenitelj pristran. Možemo izračunati njegovu pristranost:

$$b(\hat{\sigma}^2) = \frac{N-1}{N}\sigma^2 - \sigma^2 = -\frac{\sigma^2}{N}$$

Budući da je pristranost negativna, to znači da procjenitelj **podcenjuje** (engl. *underestimates*) pravu varijancu! Primjenom ovog procjenitelja, dobit ćemo varijancu koja je manja od prave varijance populacije.

Možemo li to ispraviti, tj. učiniti procjenitelj nepristranim? Naravno da možemo! Sve što trebamo napraviti jest korigirati vrijednost koju nam daje procjenitelj tako da pristranost svedemo na nulu. Iz izraza za pristranost vidimo da će ona biti nula ako pomnožimo vrijednost procjenitelja sa $\frac{N}{N-1}$. Tako dobivamo **nepristran procjenitelj varijance**:

$$\hat{\sigma}_{\text{nepr.}}^2 = \frac{N}{N-1}\hat{\sigma}^2 = \frac{N}{N-1}\frac{1}{N}\sum_{i=1}^N(x^{(i)} - \hat{\mu})^2 = \frac{1}{N-1}\sum_{i=1}^N(x^{(i)} - \hat{\mu})^2$$

Ova korekcija se treba raditi kada je N (uzorak) malen. Ako je N velik, onda je gotovo svejedno dijelimo li sa N ili $N-1$. Ili, drugim riječima, ako $N \rightarrow \infty$, onda pristranost $-\frac{\sigma^2}{N} \rightarrow 0$.

4.2 Metode za izvođenje procjenitelja

Sada znamo što je procjenitelj i što je nepristrani procjenitelj. Međutim, pitanje je odakle nam uopće formula za procjenitelj? U gornjim primjerima ptičica nam je došapnula formule za procjenitelj srednje vrijednosti i formulu za procjenitelja varijance, koji su se pokazali nepristranim, odnosno pristranim ali ispravljivim. Kako općenito možemo doći do toga? Kako izvesti procjenitelje za bilo koji parametar vjerojatnosne distribucije?

Srećom, u statistici su razvijene općeniti postupci za izvođenje procjenitelja. Konkretno, postoje tri glavne vrste procjenitelja:

- **Procjenitelj najveće izglednosti** (engl. *Maximum Likelihood Estimator; MLE*);
- **Procjenitelj maximum a posteriori (MAP)**;
- **Bayesovski procjenitelj** (engl. *Bayesian estimator*), koji zapravo motivira jedan još općenitiji pristup, naime bayesovsku statistiku.

Mi ćemo na ovom predmetu raditi prva dva procjenitelja: MLE i MAP, dok Bayesovski procjenitelj (i cijelu Bayesovsku statistiku) ostavljamo za bolje dane. Idući sat, dakle, nastavljamo sa MLE i MAP procjeniteljima.

Sažetak

- Učenje probalističkih modela svodi se na **procjenu parametara** distribucije
- Primjere i oznake modeliramo kao **slučajne varijable**
- Slučajne varijable imaju **očekivanje, varijancu i kovarijancu**
- **Nezavisne** slučajne varijable su **nekorelirane**, ali obrat ne vrijedi
- Koristimo teorijske distribucije: Bernoullijevu, Multinulijevu, Gaussovnu
- **Statistika** je funkcija slučajnog uzorka, a **procjenitelj** je statistika koja odgovara parametru distribucije koji nas zanima
- Svaki procjenitelj ima svoju **pristranost i varijancu**

Bilješke

- 1** Prepostavka je, naravno, da suvereno vladate osnovama teorije vjerojatnosti. Premda će nam u ovom predmetu zapravo trebati vrlo malo toga iz teorije vjerojatnosti, dobro je osvijestiti činjenicu da strojno učenje, jednako kao i statistika, ima temelje u teoriji vjerojatnosti. Mnogo je dobrih udžbenika o teoriji vjerojatnosti, međutim ako želite dobiti širu sliku te naučiti nešto o fundamentalnim aspektima te teorije te vezama s drugim znanstvenim disciplinama, posebice logikom, toplo preporučam da pogledate Jaynesov "Probability Theory" (Jaynes, 2003). Usput, Edwin Thompson Jaynes bio je fizičar koji se prvenstveno bavio statističkom mehanikom, ali i temeljima teorije vjerojatnosti i statističkog zaključivanja. Također je poznat po konceptu *pogreške projekcije uma* (engl. *mind projection fallacy*), do koje dolazi kada na stvarnost projiciramo neke karakteristike koje nisu dijelom te stvarnosti. Prema Jaynesu, i teorija vjerojatnosti može dovesti do te pogreške, jer vjerojatnost nije inherentno objektivnoj stvarnosti već manifestacija neizvjesnosti koja proizlazi iz našeg neznanja o toj stvarnosti. Prema Jaynesu, način da se takve pogreške izbjegnu jest usvajanje epistemičke skromnosti (engl. *epistemic humility*).
- 2** Mnogo detaljniju (i mnogo bolju) ekspoziciju ove teme možete naći u (Elezovic, 2010).
- 3** Budući da ćemo, uglavnom, umjesto $P(X = x)$ pisati $P(x)$, to znači da u formulama nećemo razlikovati između slučajne varijable i njezine vrijednosti. Tako će, na primjer, 'x' označavati i slučajnu varijablu primjera (dakle, bilo koji primjer) i vrijednost (tj. realizaciju) te slučajne varijable (dakle, jedan konkretan primjer). U kontekstima gdje je ta distinkcija bitna, vratit ćemo se na notaciju koja razlikuje slučajnu varijablu od njezine vrijednosti (na primjer, pisat ćemo $P(\mathbf{x} = \mathbf{x}^{(i)})$).
- 4** U matematici je **funkcija gustoće vjerojatnosti** tipično označena sa f . Mi ćemo koristiti p , jer je to uobičajeno u literaturi iz strojnog učenja. Dakle, veliko P ćemo koristiti za vjerojatnost, a malo p za gustoću vjerojatnosti. Međutim, često ćemo si pojednostaviti život i u oba slučaja govoriti o "vjerojatnosti" ili "distribuciji" (što nije isto, ali iz konteksta će biti jasno mislimo li na vjerojatnost, gustoću vjerojatnosti, ili distribuciju koje one definiraju).
- 5** Prisjetite se da je **vjerojatnost** da kontinuirana slučajna varijabla X poprimi vrijednost iz intervala $[a, b]$ ($a, b \in \mathbb{R}$, $a \leq b$) jednaka:

$$P(a \leq X \leq b) = \int_a^b p(x) dx.$$

Primijetite da za kontinuiranu varijablu X vrijedi $P(X = a) = 0$, tj. vjerojatnost da kontinuirana varijabla poprimi bilo koju pojedinačnu vrijednost jednaka je nuli.

- 6** **Marginalna vjerojatnost** je vjerojatnost pojedinačne varijable koju dobivamo iz zajedničke vjerojatnosti više varijabli. Marginalnu vjerojatnost dobivamo **marginalizacijom**. Kod diskretnih varijabli marginalizacija se svodi na zbrajanje vrijednosti zajedničke vjerojatnosti po svim preostalim varijablama. Npr., marginalna vjerojatnost $P(X)$ iz zajedničke se vjerojatnosti $P(X, Y)$ dobiva kao:

$$P(X) = \sum_Y P(X, Y)$$

Više o tome u predavanju broj 15, kada ćemo pričati o Bayesovom klasifikatoru.

- 7** Preuzeto sa https://en.wikipedia.org/wiki/Correlation_and_dependence. Primijetite da grafikoni raspršenja prikazuju konkretne vrijednosti varijabli X i Y , tj. prikazujemo njihove realizacije u uzorku (svaka realizacija je jedna točka u grafikonu raspršenja). Međutim, očekivanje, varijanca i kovarijanca (posljedično, i korelacija) definirane su nad slučajnim varijablama (tj. nad njihovim distribucijama), a ne nad njihovim konkretnim realizacijama (tj. nad uzorkom). Međutim, u strojnom učenju mi raspolažemo uzorkom (skupom podataka), a ne distribucijama. Morat ćemo, dakle, nekako od uzorka doći do distribucije. Upravo to radimo pomoću **procjenju parametara**. Ideja je, dakle, da iz podataka kojima raspolažemo procijenimo očekivanje varijancu, korelaciju, kovarijaciju i ostale nama važne veličine koje opisuju varijablu od interesa.
- 8** Ovdje ste možda uočili da je Pearsonov koeficijent korelacije nekako povezan s pogreškom modela linearne regresije, budući da oba modela prepostavljuju linearnu ovisnost varijabli. To je točno.

Naime, za model jednostavne regresije, $h(x) = w_0 + w_1x$, kvadrat Pearsonovog koeficijenta korelacije procijenjenog iz podataka, r^2 , između zavisne varijable y i nezavisne varijable (značajke) x jednak je **koeficijentu determinacije** R^2 . Slično kao i pogreška kvadratnog odstupanja, koeficijent determinacije mjeri koliko regresijski pravac (odnosno općenito hiperravnina) dobro modelira podatke, ali, za razliku od kvadratnog odstupanja, koeficijent determinacije je (na skupu za učenje) normaliziran na interval $[0, 1]$. Veći R^2 odgovara većem r^2 , tj. sve većoj linearnej zavisnosti između varijabli X i Y .

- 9** Primijetite da sve ove distribucije koje ovdje spominjemp pripadaju **eksponencijalnoj familiji distribucija**, koju smo već bili spomenuli u kontekstu poopćenih linearnih modela. To znači da sve ove distribucije imaju neka zajednička svojstva, i da bismo ih mogli promatrati unificirano kroz ta svojstva (i isto tako izvesti unificirane procjenitelje za parametre tih distribucija). Međutim, to je malo komplikiranije, pa nećemo to raditi, nego ćemo svaku distribuciju promatrati zasebno.
- 10** Legitimno pitanje ovdje jest zašto umnožak $\prod \mu_k^{x_k}$ a ne zbroj $\sum x_k \mu_k$? Zato što ćemo kasnije željeti raditi s logaritmima vjerojatnosti, a onda želimo imati umnožak, koji će se logaritmiranjem pretvoriti upavo u ovu sumu.
- 11** Ovdje ponavljam napomenu iz skriptice 3, da pogledate zanimljivu diskusiju o sveprisutnosti i opravdanosti normalne distribucije na <https://stats.stackexchange.com/q/204471/93766>.
- 12** U engleskom jeziku kaže se **statistic** (jednina), dok je **statistics** (množina) naziv za matematičku disciplinu. U hrvatskome jeziku nemamo tu razliku!
- 13** Ovdje gadno pojednostavljujemo stvari. Pristranost je poželjno svojstvo procjenitelja, ali stvari nisu tako jednostavne. Kvaliteta procjenitelja sastoji se od mnogo komponenti, jedna od kojih je pogreška procjenitelja. Ta se može rastaviti na kvadrat pristranosti i varijancu. U pravilu, želimo da je pogreška procjenitelja što manja, ali to onda podrazumijeva **kompromis između pristranosti i varijance** (engl. *bias-variance tradeoff*). Ako je procjenitelj neprištran i k tome još ima najmanju moguću varijancu, onda govorimo o **neprištranom procjenitelju najmanje varijance** (engl. *minimum variance unbiased estimator, MVUE*). Pritom je "najmanja moguća varijanca" definirana Cramér-Raovom ogradiom. Prenda je ovo zapravo vrlo fundamentalno i važno za strojno učenje, pogotovo ideja rastava na pristranost i varijancu, mi ipak u to nećemo ulaziti. Zainteresirane upućujem na <https://en.wikipedia.org/wiki/Estimator>. Ovo je također dobar tekst: <https://stats.stackexchange.com/a/207764/93766>.

Literatura

N. Elezovic. Vjerojatnost i statistika. *Slučajne varijable, Element, Zagreb*, 2010.

E. T. Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

14. Procjena parametara II

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v2.2

Prošli put krenuli smo pričati o procjeni parametara – mehanizmu učenja probabilističkih modela. Malo smo ponovili statistiku, osnovne razdiobe i objasnili što je to procjenitelj. Rekli smo da postoje tri osnovne vrste procjenitelja: **MLE**, **MAP** i **bayesovski**. Mi ćemo raditi MLE i MAP.

Procjenitelj MLE je najjednostavniji i često se koristi. Međutim, vidjet ćemo da s njim imamo jedan problem, a to je **prenaučenost**. To će nam biti motivacija za MAP procjenitelj. MAP procjenitelj će nam omogućiti da u procjenu ugradimo svoje pozadinsko znanje, i na taj način izbjegnemo ili barem ublažimo prenaučenost.

1 Funkcija izglednosti

Naša današnja priča započinje sa **funkcijom izglednosti** (engl. *likelihood function*). Ideja izglednosti jedna je od osnovnih ideja u statistici i, kao što ćemo vidjeti, temelj za procjenu parametara metodama MLE i MAP. Zapravo, mi smo funkciju izglednosti već susreli, kada smo izvodili funkcije empirijske pogreške za poopćene linearne modele, samo što je nismo tako zvali. Prisjetite se: ideja je bila da napišemo vjerojatnost oznaka na temelju predikcije modela, odnosno, iz razloga matematičke jednostavnosti, logaritam te vjerojatnosti. Zatim smo tražili parametre koji maksimiziraju logaritam vjerojatnosti oznaka ili, ekvivalentno, minimiziraju negativan logaritam vjerojatnosti oznaka, a za koji smo onda utvrdili da je jednak ili proporcionalan empirijskoj pogrešci koju želimo minimizirati. Na taj smo način izveli funkciju pogreške i funkciju gubitka nekoliko modela. Vratimo se sada opet na to i pogledajmo detaljnije o čemu se radi.

Kao i uvijek u strojnom učenju, krećemo od skupa podataka. Neka je to neki općeniti skup podataka, $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$. To mogu biti primjeri, ali mogu primjeri i označke, ili samo označke – nije bitno, metoda će vrijedi univerzalno, neovisno o tome što su točno podaci. Skup \mathcal{D} zapravo je **slučajan uzorak** koji smo uzorkovali iz populacije, tj. skupa svih mogućih primjera \mathcal{X} . Prepostavit ćemo da se primjeri $\mathbf{x} \in \mathcal{X}$ ravnaju po nekoj distribuciji. To pišemo ovako:

$$\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\theta})$$

gdje je $\boldsymbol{\theta}$ vektor parametara te distribucije. Primijetite da p ovdje označava funkciju gustoće vjerojatnosti, no distribucija, naravno, može biti i diskretna. Budući da je \mathcal{D} slučajan uzorak iz \mathcal{X} , to je on onda reprezentativan uzorak, pa možemo prepostaviti i da se primjeri $\mathbf{x}^{(i)} \in \mathcal{D}$ također ravnaju po istoj distribuciji kao i primjeri iz populacije \mathcal{X} :

$$\mathbf{x}^{(i)} \sim p(\mathbf{x}|\boldsymbol{\theta})$$

Sada je ideja da napišemo **vjerojatnost uzorka** – vjerojatnost da smo iz dotične distribucije $p(\mathbf{x}|\boldsymbol{\theta})$ izvukli baš taj uzorak \mathcal{D} :

$$p(\mathcal{D}|\boldsymbol{\theta}) = p(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}|\boldsymbol{\theta}) = \prod_{i=1}^N p(\mathbf{x}^{(i)}|\boldsymbol{\theta})$$

Primijetite da smo ovdje (kod druge jednakosti) upotrijebili još jednu pretpostavku: pretpostavku **i.i.d** (nezavisno i identično distribuirani primjeri). Naime, ako su primjeri $\mathbf{x}^{(i)}$ i.i.d., onda vrijedi $p(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = p(\mathbf{x}^{(i)})p(\mathbf{x}^{(j)})$, pa vjerojatnost slučajnog vektora možemo napisati kao produkt vjerojatnosti pojedinačnih slučajnih varijabli.

Sada je bitno da shvatimo da vjerojatnost $p(\mathcal{D}|\boldsymbol{\theta})$ u stvarnosti ovisi samo o parametru $\boldsymbol{\theta}$. Naime, u stvarnosti je uzorak \mathcal{D} **fiksiran** – to je skup podataka kojim raspolažemo, i imamo samo taj jedan skup i ne možemo ga više mijenjati. Dakle, ono što je ovdje varijabilno je parametar $\boldsymbol{\theta}$, dok je \mathcal{D} fiksiran. Kako bismo to naglasili, napisat ćemo eksplisitno da je vjerojatnost $p(\mathcal{D}|\boldsymbol{\theta})$ zapravo **funkcija od parametra $\boldsymbol{\theta}$** , za što uvodimo novu oznaku, \mathcal{L} :

$$p(\mathcal{D}|\boldsymbol{\theta}) = p(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}|\boldsymbol{\theta}) = \prod_{i=1}^N p(\mathbf{x}^{(i)}|\boldsymbol{\theta}) \equiv \mathcal{L}(\boldsymbol{\theta}|\mathcal{D})$$

Funkcija \mathcal{L} zove se **funkcija izglednosti** (engl. *likelihood function*). Ona parametrima $\boldsymbol{\theta}$ pridje luje vjerojatnost da iz populacije s parametrima $\boldsymbol{\theta}$ izvučemo uzorak \mathcal{D} :

$$\mathcal{L} : \boldsymbol{\theta} \mapsto p(\mathcal{D}|\boldsymbol{\theta})$$

Dakle, kako bismo od vjerojatnosti $P(\mathcal{D}|\boldsymbol{\theta})$ ili gustoće vjerojatnosti $p(\mathcal{D}|\boldsymbol{\theta})$ dobili funkciju izglednosti parametra $\boldsymbol{\theta}$, jednostavno tu vjerojatnost odnosno gustoću vjerojatnosti promatramo kao funkciju od $\boldsymbol{\theta}$ a ne od \mathcal{D} . Notacijski, jednostavno zamijenimo argumente: parametar $\boldsymbol{\theta}$ postaje varijabla funkcije, a varijabla \mathcal{D} postaje parametar, pa pišemo $\ln \mathcal{L}(\boldsymbol{\theta}|\mathcal{D})$.

Jako je važno da primijetimo da izglednost parametara $\boldsymbol{\theta}$ nije vjerojatnost parametara $\boldsymbol{\theta}$, nego da je to vjerojatnost skupa podataka \mathcal{D} za distribuciju s parametrima $\boldsymbol{\theta}$. Također kažemo da je to “vjerojatnost podataka pod modelom”. Sljedeći primjer će to malo razjasniti.

► PRIMJER

Pogledajmo kako bi izgledala **izglednost parametra Bernoulliјeve distribucije**.

Neka je naš skup podataka dobiven bacanjem novčića i bilježenjem jesmo li dobili glavu ili pismo. Ishod da dobijemo glavu modeliramo **Bernoulliјevom varijablom**: $x = 1$ znači da smo dobili glavu, a $x = 0$ da smo dobili pismo. Ishod Bernoulliјeve varijable definira **Bernoulliјeva distribucija**, koja ima parametar μ (vjerojatnost da dobijemo $x = 1$):

$$P(X = x|\mu) = \mu^x(1 - \mu)^{1-x}$$

Skup \mathcal{D} čini 10 bacanja novčića ($N = 10$). Glavu smo dobili 8 puta, a pismo 2 puta. Za takav skup podataka, funkcija izglednosti je:

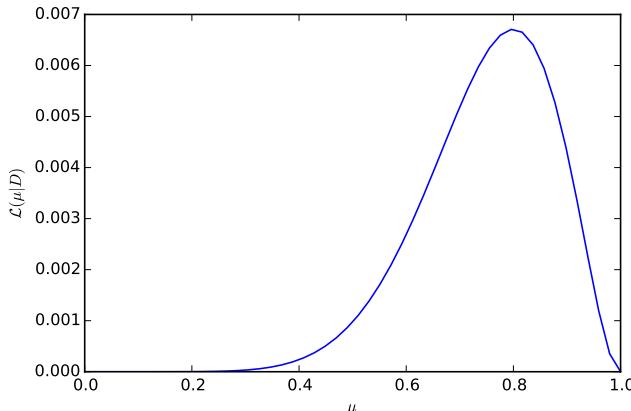
$$\begin{aligned} \mathcal{L}(\mu|\mathcal{D}) &= P(\mathcal{D}|\mu) = P(x^{(1)}, x^{(2)}, \dots, x^{(10)}|\mu) \\ &= \prod_{i=1}^{10} P(x^{(i)}|\mu) = \mu \cdot (1 - \mu) \cdot (1 - \mu) = \mu^8(1 - \mu)^2 \end{aligned}$$

Općenito, ako u N bacanja novčića imamo m glava, onda je funkcija izglednosti jednaka:

$$\mathcal{L}(\mu|\mathcal{D}) = \mu^m(1 - \mu)^{(N-m)}$$

gdje je $m = \sum x^{(i)}$ broj glava.

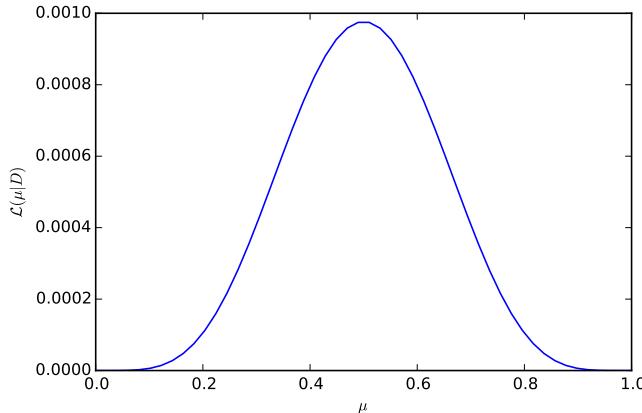
Za konkretni slučaj $m = 8$ i $n = 10$, graf funkcije izglednosti izgleda ovako:



$$m = 8, N = 10$$

Ovdje opet napominjemo da ovo nije vjerojatnost, odnosno, preciznije, funkcija \mathcal{L} nije funkcija gustoće vjerojatnosti. Stoga općenito ne vrijedi $\int_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}|D) d\boldsymbol{\theta} = 1$. Ovaj primjer to dobro pokazuje, budući da je integral ovdje sasvim sigurno manji od jedinice.

Funkcija izglednosti \mathcal{L} je funkcija parametra μ , dok je skup podataka D fiksiran. Skup podataka ovdje smo saželi u dva broja: m i N . Da smo imali drugačiji uzorak D , odnosno da su m i N bili drugačiji, onda bi i funkcija izglednosti bila drugačija. Na primjer, da smo u 10 bacanja dobili 5 puta glavu, onda bi graf funkcije izglednosti bio ovakav:



$$m = 5, N = 10$$

2 Procjenitelj MLE

Izglednost je, dakle, funkcija koja nam za neke parametre $\boldsymbol{\theta}$ kazuje koliko je, uz te parametre, vjerojatno da nam se dogodi skup podatka D kojim raspolažemo. Sada je pitanje: kako to iskoristiti da nađemo parametre modela odnosno parametre vjerojatnosne distribucije?

Ovdje pomaže da se nakratko vratimo na gornja dva primjera s bacanjem novčića. Pogledajte graf funkcije izglednosti za prvi slučaj, za skup D za koji $m = 8$ i $N = 10$. Na temelju tog grafa, za koju biste vrijednost parametra μ rekli da vrijedi za naš novčić, koliko je vjerojatno da dobijemo glavu? Sigurno biste odabrali $\mu = 0.8$, jer je uz tu vrijednost parametra μ vjerojatnost skupa D , tj. vjerojatnost da od deset bacanja dobijemo osam glava, najveća. Drugim riječima, vrijednost 0.8 je *najizglednija* vrijednost parametra μ . Pogledajte sada graf funkcije izglednosti za drugi slučaj, za skup D za koji $m = 5$ i $N = 10$. Ovdje je najizglednija vrijednost parametra

μ jednaka 0.5, tj. vjerojatnosti glave i pisma su jednakе.

Ideja je, dakle, da se postavimo vrlo pragmatično: ako smo dobili uzorak \mathcal{D} , onda mora da je baš on najvjerojatniji mogući, inače ga ni ne bismo dobili! Ako je tako, hajde da nađemo parametre koji naš uzorak čine najvjerojatnjim. Drugim riječima, hajde da nađemo koji parametri **maksimiziraju funkciju izglednosti**. Upravo to je **procjenitelj najveće izglednosti** (engl. *maximum likelihood estimator; MLE*). Dakle, MLE nalazi $\boldsymbol{\theta}$ koji maksimiziraju funkciju izglednosti. Formalno:

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \mathcal{L}(\boldsymbol{\theta} | \mathcal{D})$$

Kao i mnogo puta do sada, pokazat će se da je matematički često jednostavnije maksimizirati logaritam izglednosti, tzv. **log-izglednost**:

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} (\ln \mathcal{L}(\boldsymbol{\theta} | \mathcal{D}))$$

Naravno, činjenica da maksimiziramo logaritam funkcije izglednosti, a ne izravno funkciju izglednosti, ništa ne mijenja na stvari jer je logaritam monotono rastuća funkcija, pa su maksimizatori u oba slučaja identični. Kad god je to moguće i kada se računalno isplati, maksimizaciju ćemo provesti analitički (tj. nalaženjem rješenja u zatvorenoj formi), a kada to nije moguće ili se ne isplati, optimizaciju ćemo provesti iterativnim metodama, npr. gradijentnim spustom.

Sažmimo ovu fantastičnu ideju. Pred sobom imam neki **uzorak podataka**. Činjenica da se baš taj uzorak podataka našao kod mene uzimam kao znak da je upravo taj uzorak bio najvjerojatniji mogući uzorak, inače bih dobio neki drugi uzorak. Ako prepostavim da se podatci pokoravaju nekoj distribuciji, čiji su mi parametri još nepoznati, mogu napisati **funkciju izglednosti**, koja mi daje vjerojatnost uzorka u ovisnosti o parametrima distribucije. Ako je uzorak koji imam najvjerojatniji, onda je moja najbolja procjena za parametre distribucije upravo ona koja taj moj uzorak čini najvjerojatnjim, tj. ona koja **maksimizira funkciju izglednosti** (odnosno log-izglednosti).

Pokažimo sada kako izvesti procjenitelje MLE za osnovne vjerojatnosne distribucije koje smo bili ponovili prošli put: Bernoullijeva, multinulijeva, Gaussova i multivarijatna Gaussova. U svim tim slučajevima MLE ćemo moći izvesti analitički – deriviranjem i izjednačavanjem s nulom (nalaženjem stacionarne točke funkcije izglednosti).

2.1 MLE za Bernoullijevu distribuciju

Već smo rekli da je funkcija log-izglednosti za parametar μ Bernoullijeve distribucije sljedeća:

$$\begin{aligned} \ln \mathcal{L}(\mu | \mathcal{D}) &= \ln \prod_{i=1}^N P(x_i | \mu) = \ln \prod_{i=1}^N \mu^{x^{(i)}} (1 - \mu)^{1-x^{(i)}} \\ &= \sum_{i=1}^N x^{(i)} \ln \mu + \left(N - \sum_{i=1}^N x^{(i)} \right) \ln(1 - \mu) \end{aligned}$$

Maksimizaciju log-izglednosti provodimo nalaženjem nul-točke prve derivacije funkcije:

$$\begin{aligned} \frac{d \ln \mathcal{L}}{d \mu} &= \frac{1}{\mu} \sum_{i=1}^N x^{(i)} - \frac{1}{1-\mu} \left(N - \sum_{i=1}^N x^{(i)} \right) = 0 \\ \Rightarrow \hat{\mu}_{\text{MLE}} &= \frac{1}{N} \sum_{i=1}^N x^{(i)} = \frac{m}{N} \end{aligned}$$

gdje u indeksu pišemo “MLE” kako bismo naznačili da je procjenitelj izведен metodom najveće izglednosti.

Vidimo da je procjena za μ zapravo broj realizacija ishoda 1 podijeljen s veličinom uzorka. Statistički termin za to je **relativna frekvencija**: broj realizacija ishoda 1 podijeljen s veličinom uzorka. Relativna frekvencija je i vrlo intuitivan procjenitelj za parametar μ : naime, ako smo 10 puta bacili novčić i od toga 8 puta dobili glavu, onda bismo rekli da je vjerojatnost glave jednaka $8/10 = 0.8$. Upravo to je MLE procjena za μ . Zapravo, ispada da svakodnevno radimo MLE. Vrijedi $\mathbb{E}(\mu_{\text{MLE}}) = \mathbb{E}[X] = \mu$, pa je ovo je nepristran procjenitelj.

2.2 MLE za kategoricku (multinulijevu) distribuciju

Prisjetite se (od prošlog puta) da vjerojatnost kategoričke varijable sa K mogućih vrijednosti definiramo ovako:

$$P(X = \mathbf{x}|\boldsymbol{\mu}) = \prod_{k=1}^K \mu_k^{x_k}$$

gdje je vektor $\boldsymbol{\mu}$ parametar ove distribucije. Funkcija log-izglednosti za taj parametar je:

$$\ln \mathcal{L}(\boldsymbol{\mu}|\mathcal{D}) = \ln \prod_{i=1}^N P(\mathbf{x}^{(i)}|\boldsymbol{\mu}) = \ln \prod_{i=1}^N \prod_{k=1}^K \mu_k^{x_k^{(i)}} = \sum_{k=1}^K \sum_{i=1}^N x_k^{(i)} \ln \mu_k$$

Kako bismo izrazili MLE, trebamo maksimizirati ovu funkciju. Međutim, za razliku od gornjeg slučaja s Bernoullijevom varijablom, ovdje imamo ograničenje $\sum_{k=1}^K \mu_k = 1$, što znači da ovdje govorimo o problemu **optimizacije uz ograničenje**. Tu optimizaciju možemo provesti (nama već poznatom) metodom **Lagrangeovih multiplikatora**. Ovdje ćemo se toga poštredjeti te dati samo konačan rezultat. MLE za k -tu komponentu vektora $\boldsymbol{\mu}$ je:

$$\hat{\mu}_{k,\text{MLE}} = \frac{1}{N} \sum_{i=1}^N x_k^{(i)} = \frac{N_k}{N}$$

gdje je N_k je broj nastupanja k -te vrijednosti. Vidimo da je ovo posve analogno procjenitelju MLE za Bernullijevu varijablu, jedino što ovdje imamo posebnu procjenu za svaku varijablu k (koje odgovaraju pojedinačnim vrijednostima kategoriskske varijable). Ako je $K = 2$, procjenitelj očekivano degenerira na procjenitelj MLE za Bernullijevu varijablu.

2.3 MLE za Gaussovou distribuciju

Gaussova distribucija je distribucija kontinuirane varijable, dakle ovdje ćemo raditi s funkcijom gustoće vjerojatnosti p , a ne s vjerojatnošću P . Prisjetimo se najprije (od prošlog puta) funkcije Gaussove gustoće vjerojatnosti:

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{(x-\mu)^2}{2\sigma^2} \right\}$$

Ovdje sada imamo dva parametra, μ i σ^2 , pa će funkcija log-izglednost biti funkcija ta dva parametra:

$$\begin{aligned} \ln \mathcal{L}(\mu, \sigma^2 | \mathcal{D}) &= \ln \prod_{i=1}^N \frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{(x^{(i)}-\mu)^2}{2\sigma^2} \right\} \\ &= -\frac{N}{2} \ln(2\pi) - N \ln \sigma - \frac{\sum_i (x^{(i)} - \mu)^2}{2\sigma^2} \end{aligned}$$

Također, budući da imamo dva parametra, maksimizaciju trebamo provesti po oba ta parametra. Dobivamo (izvod preskačemo):

$$\begin{aligned}\nabla \ln \mathcal{L}(\mu, \sigma^2 | \mathcal{D}) &= 0 \\ \Rightarrow \hat{\mu}_{\text{MLE}} &= \frac{1}{N} \sum_{i=1}^N x^{(i)} \\ \Rightarrow \hat{\sigma}_{\text{MLE}}^2 &= \frac{1}{N} \sum_{i=1}^N (x^{(i)} - \hat{\mu}_{\text{MLE}})^2\end{aligned}$$

Primijetite da je procjenitelj $\hat{\sigma}_{\text{MLE}}^2$ izražen pomoću procjenitelja $\hat{\mu}_{\text{MLE}}$, zato jer nam je prava vrijednost parametra μ nepoznata.

Za procjenitelj srednje vrijednosti već smo prošli puta ustanovili da je nepristran. Međutim, ustanovili smo da je ovakav procjenitelj varijance pristran, i da ga možemo korigirati ako podjelimo sa $N - 1$ umjesto s N . Vidimo, dakle, da procjenitelj najveće izglednosti ne mora nužno biti nepristran. Najveća izglednost nije isto što i nepristranost!

2.4 MLE za multivarijatnu Gaussovou distribuciju

Pogledajmo sada multivarijatnu Gaussovou distribuciju. Prisjetimo se, funkcija gustoće multivarijante Gaussove distribucije definirana je ovako:

$$p(\mathbf{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{n/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$

gdje su $\boldsymbol{\mu}$ i $\boldsymbol{\Sigma}$ srednja vrijednost odnosno kovarijacijska matrica. Funkcija log-izglednosti onda je jednaka:

$$\begin{aligned}\ln \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) &= \ln \prod_{i=1}^N p(\mathbf{x}^{(i)} | \boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ &= -\frac{nN}{2} \ln(2\pi) - \frac{N}{2} |\boldsymbol{\Sigma}| - \frac{1}{2} \sum_{i=1}^N (\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu})\end{aligned}$$

Maksimizacija funkcije log-izglednosti (izvod preskačemo) daje:

$$\begin{aligned}\nabla \ln \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathcal{D}) &= 0 \\ \Rightarrow \hat{\boldsymbol{\mu}}_{\text{MLE}} &= \frac{1}{N} \sum_{i=1}^N \mathbf{x}^{(i)} \\ \Rightarrow \hat{\boldsymbol{\Sigma}}_{\text{MLE}} &= \frac{1}{N} \sum_{i=1}^N (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_{\text{MLE}})(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_{\text{MLE}})^T\end{aligned}$$

Procjenitelji $\hat{\boldsymbol{\mu}}_{\text{MLE}}$ i $\hat{\boldsymbol{\Sigma}}_{\text{MLE}}$ analogni su procjeniteljima $\hat{\mu}_{\text{MLE}}$ odnosno $\hat{\sigma}_{\text{MLE}}^2$ univarijatne Gaussove razdiobe. Također vrijede ista zapažanja što se tiče (ne)pristranosti procjenitelja.

2.5 MLE za parametre linearne regresije

Pokazali smo kako izvesti MLE za osnovne distribucije. Vidimo da je rezultat poprilično intuitivan. Međutim, istu tehniku možemo primijeniti kako bismo izveli bilo kakve druge parametre neke distribucije. Zapravo, mi smo to već radili. Prisjetimo se kako smo, na primjer, izveli

kvadratnu pogrešku linearne regresije. Krenuli smo od logaritma vjerojatnosti oznaka \mathbf{y} :

$$\begin{aligned}
\ln p(\mathbf{y}|\mathbf{X}) &= \ln \prod_{i=1}^N p(y^{(i)}|\mathbf{x}^{(i)}) \\
&= \ln \prod_{i=1}^N \mathcal{N}(h(\mathbf{x}^{(i)}; \mathbf{w}), \sigma^2) \\
&= \ln \prod_{i=1}^N \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{w}))^2}{2\sigma^2}\right) \\
&= \underbrace{-N \ln(\sqrt{2\pi}\sigma)}_{\text{konst.}} - \frac{1}{2\sigma^2} \sum_{i=1}^N (y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{w}))^2 \\
&\propto -\frac{1}{2} \sum_{i=1}^N (y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{w}))^2
\end{aligned}$$

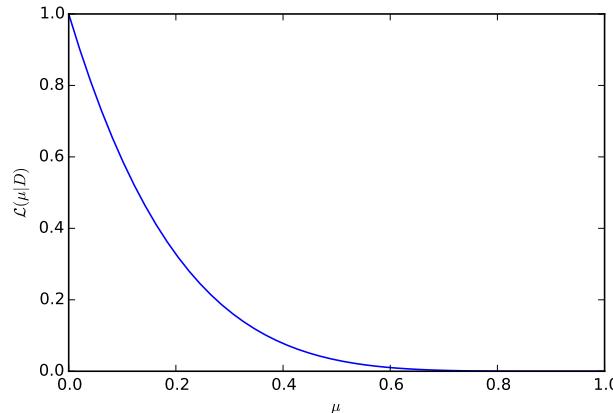
Na osnovu ovoga smo zaključili da, uz pretpostavku Gaussovog šuma u oznakama \mathbf{y} , maksimizacija (logaritma) vjerojatnosti oznaka odgovara minimizaciji pogreške kvadratnog gubitka. No, primijetimo sada da logaritam vjerojatnost oznaka, $\ln p(\mathbf{y}|\mathbf{X})$ nije ništa drugo nego **funkcija log-izglednosti težina**, $\ln \mathcal{L}(\mathbf{w}|\mathcal{D})$. To znači da smo naš zaključak mogli formulirati i ovako: uz pretpostavku da je $p(y|\mathbf{x})$ Gaussova gustoća vjerojatnosti, minimizacija kvadratne pogreške jednaka je MLE procjeni za \mathbf{w} . Tako vidimo da je učenje modela zapravo isto što i procjena parametara neke pretpostavljene distribucije. 4

2.6 MLE i prenaučenost

MLE je najjednostavniji i najinutivniji procjenitelj, međutim ima jedan velik problem – sklon je **prenaučenosti**. Pokažimo prenaučenost MLE-a na konkretnom primjeru.

► PRIMJER

Zamislimo da radimo procjenu parametra μ Bernoullijeve distribucije. Nadalje, zamislimo da se radi o običnom (nemodificiranom) novčiću, za koji očekujemo da je pravedan, tj. vjerojatnost da dobijemo glavu jednaka vjerojatnosti da dobijemo pismo, tj. $\mu = 0.5$. Recimo da bacamo novčić 5 puta. Zamislimo da smo u svih 5 bacanja dobili pismo. Kolika je vjerojatnost glave, tj. kolika je vrijednost parametra μ prema MLE procjenitelju? Da bismo odgovorili na to pitanje, trebamo iskazati funkciju izglednosti, i zatim naći μ koji ju maksimizira. U ovom slučaju, funkcija izglednosti izgleda ovako: 5



Vidimo da funkcija izglednosti doseže svoj maksimum (∞) za $\mu = 0$. To znači da je vjerojatnost da dobijemo glavu jednaka je nuli. No, hoćemo li doista vjerovati takvoj procjeni? Možda je novčić ipak pravedan, ali smo samo imali peh. Na kraju krajeva, uzorak je vrlo malen, i, premda je malo vjerojatno, ipak nije nemoguće da dobijemo pet puta pismo (vjerojatnost za to je $0.5^5 = 0.03125 \approx 3\%$).

Potpuno isti problem bismo imali da smo pet puta dobili glavu. Onda bi MLE dao $\mu = 1$.

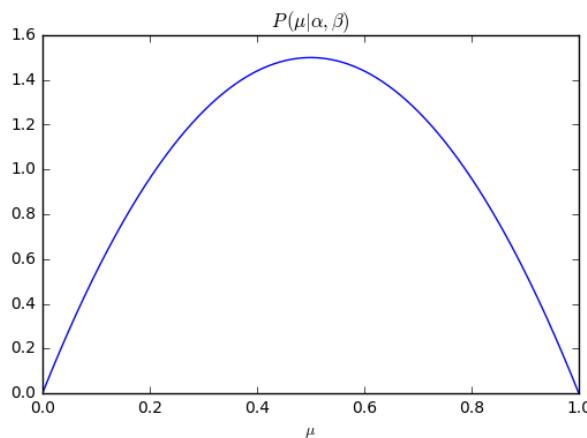
Ovdje je, dakle, problem u tome što, ako u uzorku imamo sve nule odnosno sve jedinice, onda će maksimum funkcije izglednosti biti na $\mu = 0$ odnosno $\mu = 1$.

Uzrok ovog problema jest u tome što se MLE previše oslanja na podatke. To je u redu ako imamo mnogo podataka (ako je uzorak velik). Na primjer, zamislimo da smo 1000 puta bacali novčić. Ako u 1000 bacanja novčića stvarno niti jednom ne dobijemo glavu, onda se čini sasvim razumnim procijeniti da je $\mu = 0$. Međutim, ako nemamo puno podataka, npr., ako imamo samo pet bacanja novčića, onda nije dobro previše se osloniti na takve podatke. Tada se, osim na podatke, trebamo osloniti i na naše znanje (bilo ekspertizu ili zdrav razum). Kada pričamo o procjeniteljima, “znanje” se svodi na **apriornu distribuciju** kojom opisujemo kakve vrijednosti parametara očekujemo. Konkretno, u slučaju novčića, očekujemo da je za normalan (nemodificirani) novčić μ negdje oko 0.5. Ne očekujemo baš $\mu = 0$ ili $\mu = 1$ – kakav bi to novčić bio koji uvijek pada na istu stranu?

Nažalost, MLE nam tu ne može pomoći – ne postoji način da u procjenitelj ugradimo svoje apriorno znanje o parametrima. Za to nam treba neki procjenitelj koji bi bio u stanju kombinirati informacije iz podataka s našim apriornim znanjem. To je **procjenitelj MAP**.

3 Procjenitelj MAP

Procjenitelj **maksimum aposteriori (MAP)** kombinira informacije koje dolaze iz podataka s našim pozadinskim znanjem o mogućim vrijednostima parametra. To naše znanje definiramo kroz **apriornu distribuciju parametra**, koju ćemo označiti sa $p(\theta)$ (formalno, to je gustoća vjerojatnosti, jer općenito θ je kontinuirana varijabla). Apriorna distribucija nam kazuje koje su vrijednosti parametra θ više, a koje manje vjerojatne. Npr., za novčić je više vjerojatno da je $\mu = 0.5$ nego da $\mu = 0.1$, pa bismo apriornu distribuciju mogli definirati ovako:



Primijetite da smo sada parametar θ odjednom počeli tretirati kao slučajnu varijablu koja ima svoju distribuciju. To je veliki napredak u odnosu na ono što smo radili kod procjenitelja MLE, gdje θ nismo eksplicitno tretirali kao slučajnu varijablu (premda on to jest, kad malo razmislite, jer je to procjena na temelju uzorka, međutim kod MLE to potpuno ignoriramo).

Sada nekako trebamo kombinirati izglednost parametra $\mathcal{L}(\boldsymbol{\theta}|\mathcal{D})$, za koju znamo da je zapravo jednaka vjerojatnosti $p(\mathcal{D}|\boldsymbol{\theta})$, i apriornu distribuciju parametra $p(\boldsymbol{\theta})$. Mogli bismo to napraviti na razne načine, ali umjesto da nešto petljamo, bolje je da se zapitamo što zapravo želimo dobiti. Ako već parametar $\boldsymbol{\theta}$ tretiramo kao slučajnu varijablu, onda ima smisla da pokušamo dobiti distribuciju za tu varijablu. Konkretno, ako je $p(\boldsymbol{\theta})$ apriorna vjerojatnost za tu varijablu, onda je ono što mi želimo dobiti zapravo **aposteriorna vjerojatnost parametra** $p(\boldsymbol{\theta}|\mathcal{D})$. To je vjerojatnost da parametar $\boldsymbol{\theta}$ poprimi neku vrijednost, nakon što nam je predložen skup primjera \mathcal{D} . Sada je lako vidjeti da te aposteriorne vjerojatnosti možemo jednostavno doći primjenom uvijek nam dragog **Bayesovog pravila**:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{p(\mathcal{D})}$$

Aposteriorna vjerojatnost parametara (zapravo: aposteriorna gustoća vjerojatnosti parametra) kombinira apriorno znanje i informaciju iz podataka. MAP radi maksimizaciju te vjerojatnosti. Budući da je \mathcal{D} fiksani, to je nazivnik konstanta, pa ga pri maksimizaciji možemo zanemariti. Prema tome, MAP procjenitelj je:

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} p(\boldsymbol{\theta}|\mathcal{D}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta})$$

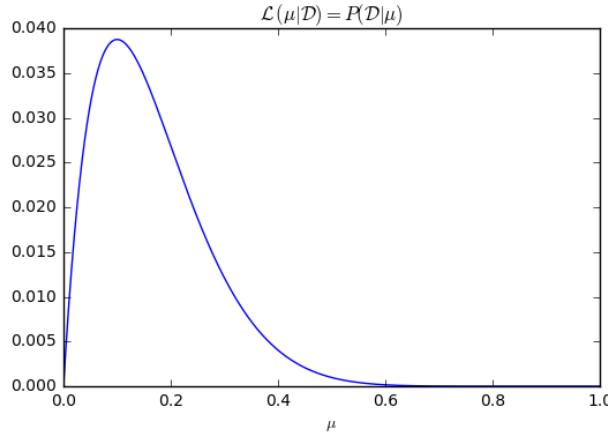
Usporedimo to s procjeniteljem MLE:

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} p(\mathcal{D}|\boldsymbol{\theta})$$

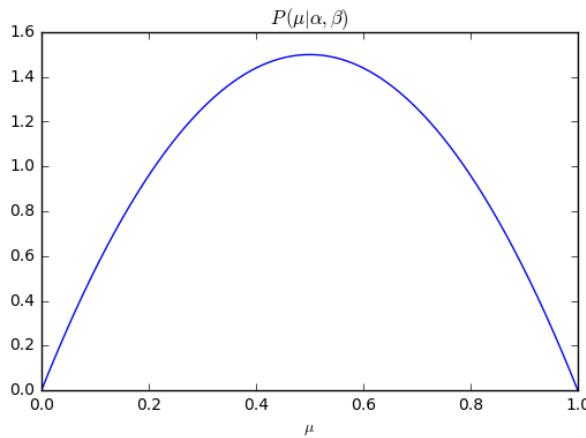
Vidimo da je razlika u tome što MAP zapravo kombinira izglednost parametara $\boldsymbol{\theta}$ (koja je vjerojatnost uzorka \mathcal{D}) s apriornom vjerojatnošću parametara $\boldsymbol{\theta}$. Kombinacija se ostvaruje jednostavnim množenjem tih dviju vjerojatnosti. Ako neke vrijednosti za parametar $\boldsymbol{\theta}$ imaju malu apriornu vjerojatnost, onda će i njihova aposteriorna vjerojatnost biti smanjena! Pogledajmo primjer.

► PRIMJER

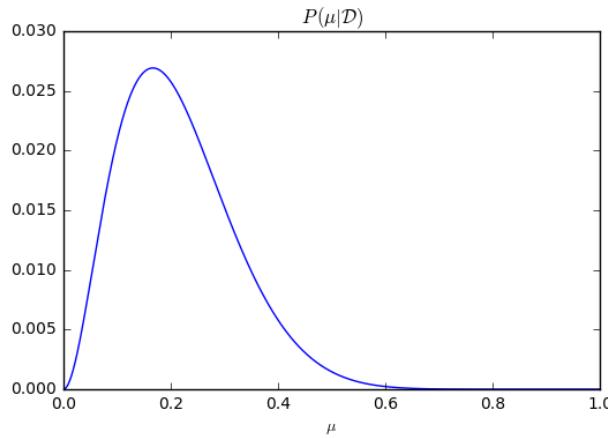
Pogledajmo opet bacanje novčića, dakle procjenu parametra μ Bernoullijeve razdiobe. Recimo da smo od 10 bacanja dobili samo 1 puta glavu. Funkcija izglednosti $\mathcal{L}(\mu|\mathcal{D})$ izgleda ovako:



Najvjerojatnija vrijednost za μ (tj. mod funkcije izglednosti) biti će $\mu = 0.8$. Ali, recimo da mi ipak vjerujemo da je novčić pravedan. To znači da ćemo apriornu distribuciju $p(\mu)$ modelirati ovako:



Mod ove funkcije je $\mu = 0.5$. Kada pomnožimo ove dvije vjerojatnosti, preciznije, kada pomnožimo izglednost $\mathcal{L}(\mu|\mathcal{D})$ i gustoću vjerojatnosti $p(\mu)$, dobivamo aposterioru gustoću vjerojatnosti parametra, $p(\mu|\mathcal{D})$, koja izgleda ovako:



Mod ove funkcije je negdje između 0.1 i 0.2. Dakle, vidimo da se efektivno događa to da naše apriorno znanje pomiče aposterioru vjerojatnost u smjeru koji smatramo više vjerojatnim. Možemo reći i obrnuto: informacije iz podataka utječu na naše apriorno znanje u smjeru vrijednosti parametara koje su izglednije na temelju podataka. Dakle, to je kao nekakav susret teorije i empirije (teorija je naše apriorno znanje, a empirija su konkretni podatci koje imamo).

Toliko o ideji. Pogledajmo sada kako to funkcionira matematički. Mi bismo željeli da se maksimizacija aposteriorne vjerojatnosti $p(\boldsymbol{\theta}|\mathcal{D})$ može provesti analitički, tj. da makimizator možemo izraziti u zatvorenoj formi. Međutim, problem je što umnožak $p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ općenito može poprimiti različite oblike koje nije moguće analitički maksimizirati. Srećom, ovaj problem možemo vrlo pragmatično riješiti. Naime, ništa nas ne sprječava da pokušamo odabrati takve distribucije $p(\mathcal{D}|\boldsymbol{\theta})$ i $p(\boldsymbol{\theta})$ da njihov umnožak daje neku nama poznatu teorijsku distribuciju, s kojom znamo raditi. Istini za volju, distribuciju $p(\mathcal{D}|\boldsymbol{\theta})$ nemamo što birati, jer je ona već definirana vrstom podataka s kojima radimo: to će biti izglednost Bernoullijeve varijable, ili multinulijeve, ili Gaussove. Prema tome, ono što nam preostaje jest da pametno odaberemo distribuciju $p(\boldsymbol{\theta})$, tako da umnožak $p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ bude neka poznata distribucija.

Štoviše, kad već biramo, bilo bi jako dobro da odaberemo tako da aposteriora distribucija $p(\boldsymbol{\theta}|\mathcal{D})$ bude ista vrsta distribucije kao i apriorna distribucija $p(\boldsymbol{\theta})$. Zašto? Zato što bi nam to onda omogućilo da radimo **“online” (pojedinačno) učenje**: da učimo postepeno kako dolaze

novi podatci. Naime, ako je aposteriorna distribucija istoga tipa kao apriorna distribucija, onda, kada izračunamo aposteriornu distribuciju, možemo je u idućoj iteraciji, kada dođu novi podatci, koristiti kao novu apriornu distribuciju. I taj postupak onda možemo ponavljati u svakoj iteraciji "online" učenja: novo znanje koje smo upravo naučili (aposteriorna distribucija) već se u idućoj iteraciji koristi kao staro znanje (apriorna distribucija).

Ako su $p(\theta|\mathcal{D})$ i $p(\theta)$ odabrane tako da su to iste vrste distribucija, onda ih nazivamo **konjugatnim distribucijama**. Kako odabratи apriornu distribuciju $p(\theta)$, a da $p(\theta|\mathcal{D})$ i $p(\theta)$ budu konjugatne? Pokazuje se da, barem za standardne funkcije izglednosti, to i nije neki problem: naime, za svaku izglednost $p(\mathcal{D}|\theta)$ koja je iz **eksponencijalne familije**, postoji apriorna distribucija $p(\theta)$ takva da su apriorna i aposteriorna distribucija konjugatne. Takva apriorna distribucija naziva se **konjugatna apriorna distribucija** za funkciju izglednosti. Sljedeća skica pokazuje te odnose između distribucija:

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta) \cdot p(\theta)$$

konjugatne

konjugatna apriorna distribucija
 za izglednost
 $p(\mathcal{D}|\theta)$

Za nas to onda znači sljedeće: ako za našu funkciju izglednosti odaberemo njoj konjugatnu apriornu distribuciju, onda će aposteriorna i apriorna distribucija biti konjugatne, tj. aposteriorna će distribucija biti ista teorijska razdioba kao i apriorna distribucija. To će nam omogućiti da analitički pronađemo mod (maksimizator) aposteriorne distribucije.

Sada još ostaje da odgovorimo na pitanje koje su apriorne distribucije konjugatne za izglednosti s kojima u strojnom učenju često baratamo. Konjugatna apriorna distribucija za funkciju izglednosti Bernoullijeve varijable je **Beta distribucija**. Za funkciju izglednosti multinulijeve (kategoričke) distribucije to je **Dirichelova distribucija**. Za funkciju izglednosti Gaussove (normalne) varijable to je opet **Gaussova (normalna) distribucija**. Prikažimo to tablično:

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta) p(\theta)$$

Apsteriorna $p(\theta \mathcal{D})$	Izglednost $p(\mathcal{D} \theta)$	Apriorna $p(\theta)$
Beta	Bernoulli	Beta
Dirichlet	Multinuli	Dirichlet
Normal	Normal	Normal
Multivariate normal	Multivariate normal	Multivariate normal

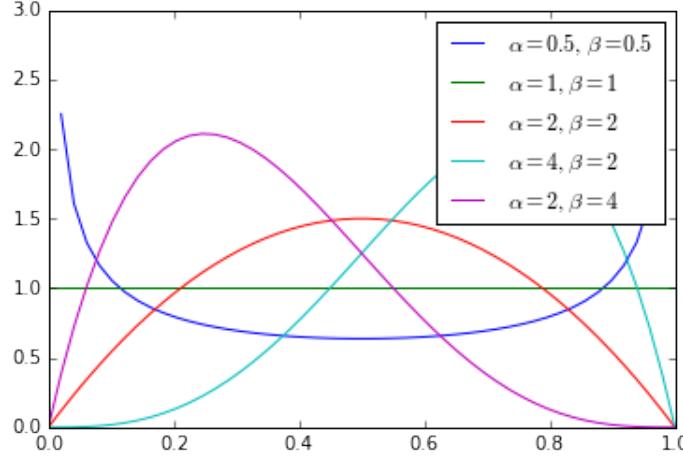
U nastavku ćemo konkretnije pogledati prva dva slučaja, odnosno kako konkretno izračunati MAP procjenitelj za Bernoullijevu varijablu i za kategoričku (multinulijevu) varijablu.

4 Beta-Bernoullijev model

Pogledajmo prvo najjednostavniji slučaj: kada je varijabla Bernoullijeva, a apriorna vjerojatnost parametra μ je beta-distribucija. To je tzv. **Beta-Bernoullijev model**, jer se za modeliranje apriorne distribucije $p(\mu)$ koristi Beta-distribucija, koja je konjugatna Bernoullijevoj izglednosti. Pogledajmo kako je definirana beta-distribucija, točnije funkcija gustoće beta-distribucije:

$$p(\mu|\alpha, \beta) = \frac{1}{B(\alpha, \beta)} \mu^{\alpha-1} (1-\mu)^{\beta-1}$$

Ovdje je B tzv. **beta-funkcija** – normalizirajuća konstanta koja osigurava da je integral funkcije gustoće jednak 1. Parametri distribucije su α i β , za koje mora vrijediti $\alpha, \beta > 0$. Ti parametri određuju oblik distribucije. To prikazuje sljedeći grafikon:



Najčešće želimo postaviti $\alpha > 1$ i $\beta > 1$, kako bismo modelirali veću apriornu gustoću vjerojatnosti za neku vrijednost θ . Što su α i β veći, to je više gustoće vjerojatnosti smješteno oko 0.5. Ako je $\alpha < \beta$, onda je maksimizator manji od 0.5, inače je veći od 0.5. Ako $\alpha = \beta = 1$, onda dobivamo **uniformnu apriornu distribuciju** (engl. *uniform prior*), kojim efektivno modeliramo da nemamo nikakvoga apiorognog znanja o vrijednosti parametra, pa takvu apriornu razdiobu zovemo **neinformativna apriorna distribucija** (engl. *uninformative prior*). 7

Važno je da ovdje primijetimo da je μ parametar koji želimo procijeniti, dok su α i β parametri koje trebamo definirati unaprijed i koji onda definiraju izgled gustoće $p(\mu)$. Drugim riječima, budući da parametre α i β ne procjenjujemo već ih moramo definirati unaprijed, iz perspektive strojnog učenja to su zapravo **hiperparametri**. 8

Vidimo da je beta-distribucija vrlo prikladna jer njome možemo lako modelirati različita apriorna vjerovanja o vrijednosti parametra μ . Drugo lijepo svojstvo beta-distribucije je to što se njezin maksimizator (mod) može jednostavno analitički izraziti na temelju parametara α i β :

$$\frac{\alpha - 1}{\alpha + \beta - 2}$$

pod uvjetom da $\alpha > 1$ i $\beta > 1$.

Pogledajmo sada kako množenjem izglednosti Bernoullijeve varijable i apiorne beta-distribucije dobivamo **aposteriornu beta-distribuciju**. Neka je naš uzorak veličine N , i neka se u tom uzorku Bernoullijeva varijabla realizirala kao 1 u ukupno m puta (npr., bacamo novčić N puta, i od toga smo m puta dobili glavu). Već znamo da je funkcija izglednosti Bernoullijeve varijable sljedeća:

$$p(\mathcal{D}|\mu) = \mu^m (1-\mu)^{N-m}$$

Apriorna vjerojatnost definirat ćemo beta-distribucijom:

$$p(\mu|\alpha, \beta) = \frac{1}{B(\alpha, \beta)} \mu^{\alpha-1} (1-\mu)^{\beta-1}$$

Prema Bayesovom pravilu, aposteriorna vjerojatnost je:

$$p(\mu|\mathcal{D}, \alpha, \beta) = \mu^m (1-\mu)^{N-m} \frac{1}{B(\alpha, \beta)} \mu^{\alpha-1} (1-\mu)^{\beta-1} \frac{1}{p(\mathcal{D})}$$

Nakon što malo presložimo, dobivamo:

$$p(\mu|\mathcal{D}, \alpha, \beta) = \mu^{m+\alpha-1} (1-\mu)^{N-m+\beta-1} \frac{1}{B(\alpha, \beta)p(\mathcal{D})}$$

Znamo da ovo što smo dobili mora biti funkcija gustoće vjerojatnosti, koja se integrira u 1. Budući da izraz na desnoj strani oblikom liči opet na beta-distribuciju, ideja je da izraz pokušamo napisati kao beta-distribuciju. Doista, izraz možemo napisati kao beta-distribuciju sa sljedećim parametrima:

$$p(\mu|\mathcal{D}, \alpha', \beta') = \mu^{\overbrace{m + \alpha - 1}^{\alpha'}} (1 - \mu)^{\overbrace{N - m + \beta - 1}^{\beta'}} \underbrace{\frac{1}{B(\alpha, \beta)p(\mathcal{D})}}_{B(\alpha', \beta')}$$

Vidimo, dakle, da smo dobili novu beta-distribuciju. Parametri te distribucije su α' i β' , koje računamo na temelju parametara α i β iz apriorne distribucije, te parametara N i m funkcije izglednosti. Konkretno:

$$\begin{aligned}\alpha' &= m + \alpha \\ \beta' &= N - m + \beta\end{aligned}$$

Naravno, ovdje nas ne treba čuditi da smo kao aposteriornu distribuciju dobili beta-distribuciju. Naime, to smo i očekivali, budući da smo već rekli da je beta-distribucija konjugatna distribucija za Bernoullijevu izglednost, pa smo zato dobili istu vrstu distribucije kao što je i apriorna distribucija.

Vratimo se na MAP procjenitelj. Prisjetimo se:

$$\hat{\theta}_{\text{MAP}} = \underset{\theta}{\operatorname{argmax}} \ p(\theta|\mathcal{D}) = p(\mathcal{D}|\theta) p(\theta)$$

Dakle, zanima nas **maksimizator aposteriorne distribucije**. Budući da se radi o beta-distribuciji, zna se da je njezin maksimizator jednak:

$$\hat{\mu}_{\text{MAP}} = \frac{\alpha' - 1}{\alpha' + \beta' - 2} = \frac{m + \alpha - 1}{m + \alpha + N - m + \beta - 2} = \frac{m + \alpha - 1}{\alpha + N + \beta - 2}$$

Ovisno o odabiru vrijednosti hiperparametara α i β , i naravno ovisno o vrijednostima m i N koje dobivamo iz podataka, dobivat ćemo različite vrijednosti procjenitelja. Primijetite, očekivano da, MAP procjenitelj degenerira na procjenitelj MLE $\hat{\mu}_{\text{MLE}} = m/N$ za $\alpha = \beta = 1$, tj. kada je apriorna distribucija uniformna. Ako nemamo apriornog znanja, MAP nije ništa bolji od MLE.

► PRIMJER

Recimo da smo novčić bacali 10 puta, a da smo samo 1 dobili glavu. Pretpostavljamo da je novčić ipak pravedan, što smo odlučili modelirati sa $\alpha = \beta = 2$. Onda za MAP procjenitelj dobivamo:

$$\hat{\mu}_{\text{MAP}} = \frac{1 + 2 - 1}{2 + 10 + 2 - 2} = \frac{2}{12} = 0.167$$

Što bismo dobili procjeniteljem MLE? Dobili bismo $\mu = 1/10 = 0.1$. Vidimo dakle da smo s MAP procjeniteljem dobili procjenu koja kombinira naše apriorno znanje ($\mu = 0.5$ je najvjerojatnije) i podatke (dobili smo samo 1 glavu u 10 bacanja).

Fantastična stvar sa MAP procjeniteljem je da “automatski” balansira između podataka (empirije) i apriornog znanja (teorije). To se lijepo vidi kod MAP procjenitelja za Bernoullijevu distribuciju. Naime, iz razlomka MAP procjenitelja vidi se da, što je podataka više (N je veći), to više vjerujemo podatcima. Suprotno, što je podataka manje (N je manji), to više vjerujemo apriornom znanju. Drugim riječima, ako je skup za učenje velik, onda N dominira, inače dominiraju α i β .

U strojnog učenju najčešće se koristi MAP procjenitelj s apriornom razdiobom $\alpha = \beta = 2$. To konkretno daje:

$$\hat{\mu}_{\text{MAP}} = \frac{m+1}{N+2}$$

Ovaj se procjenitelj još naziva i **Laplaceov procjenitelj**. Primijetite da ovaj procjenitelj efikasno rješava problem **prenaučenosti** do koje dolazi kada je $m = 0$ ili $m = N$. Naime, ono što se događa je da, premda je $m = 0$, ipak nećemo reći da je $\mu = 0$, nego će μ biti malo veći (analogno za $m = N$). To je kao da smo ukupnu masu vjerojatnosti malo preraspodijelili, tako da vjerojatnost pozitivnog ishoda Bernoullijeve varijable ipak nije nula, nego ima neku malu vrijednost. Taj efekt raspodjele dijela mase vjerojatnosti s vrlo vjerojatnih događaja na manje vjerojatne događaje općenito se u strojnog učenju naziva **zaglađivanje** (engl. *smoothing*). MAP procjenitelj je jedan (principijelan) način da **zagladimo procjene parametara** i tako smanjimo mogućnost prenaučenosti probabilističkih modela.

Sada znamo kako zaglađivati, odnosno kako izračunati MAP procjenitelj kada je varijabla binarna. No, što ako je varijabla multinomialna, tj. kategorijalna? To nas vodi do Dirichlet-kategoričkog modela.

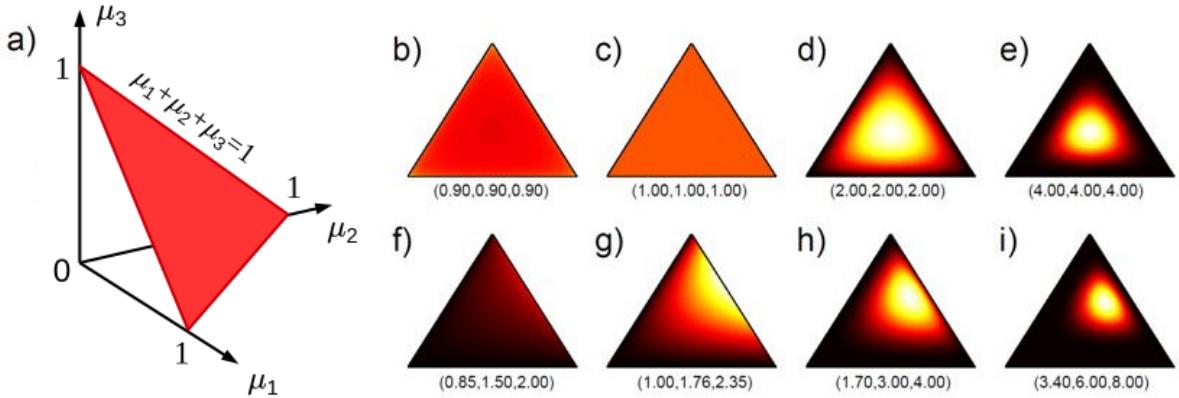
5 Dirichlet-kategorički model

Nećemo to izvoditi, ali slično kao što smo izveli MAP procjenitelj za Bernullijevu distribuciju može se izvesti procjenitelj za **multinulijevu (kategoričku) distribuciju**, koristeći **Dirichletovu distribuciju** kao konjugatnu apriornu distribuciju multinulijevoj izglednosti. To daje tzv. **Dirichlet-kategorički model**.

Dirichletova distribucija definirana je ovako:

$$P(\boldsymbol{\mu}|\boldsymbol{\alpha}) = P(\mu_1, \dots, \mu_K | \alpha_1, \dots, \alpha_K) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{k=1}^K \mu_k^{\alpha_k - 1}$$

Dirichletova distribucija poopćenje je beta distribucije na više μ_k . Hiperparametri $\alpha_1, \dots, \alpha_K$ određuju vjerojatnosti parametara μ_1, \dots, μ_K . Međutim, ovdje lako dolazi do zabune: nije slučaj da pojedinačni α_k direktno određuje vjerojatnost pojedinačnog parametra μ_k , nego je ta veza indirektna, na način da cijeli vektor $\boldsymbol{\alpha}$ određuje vjerojatnosti parametara $\boldsymbol{\mu}$. Nadalje, budući da mora vrijediti $\sum_{k=1}^K \mu_k = 1$, to znači da se parametri μ_k nalaze na tzv. $(K-1)$ -dimezijskom **standardnom simpleksu**. Npr., za $K = 3$, to je trokut u trodimenijskom prostoru:



Svaka točka na ovom trokutu odgovara jednoj kombinaciji (μ_1, μ_2, μ_3) te vrijedi $\mu_1 + \mu_2 + \mu_3 = 1$. Nadalje, svakoj takvoj točki pridružena je određena vrijednost funkcije gustoće vjerojatnosti, a integral po gustoći vjerojatnosti po cijelom trokutu je jednak 1. Vektor $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3)$

određuje kako je vjerojatnost distribuirana između (μ_1, μ_2, μ_3) . Primjeri za konkretnе vrijednosti vektora $\boldsymbol{\alpha}$ prikazani su na slikama b–f ispod trokuta. Ako $\alpha_1 = \alpha_2 = \alpha_3 = 1$, onda je svaka kombinacija μ_1, μ_2, μ_3 jednako vjerojatna (slika c). Ako $\alpha_1 = \alpha_2 = \alpha_3 > 1$, onda su vjerojatnije kombinacije koje daju jednaku vjerojatnosti za μ_1, μ_2 i μ_3 , tj. one koje su u sredini trokuta (slike c–e). Asimetrične vrijednosti za α_1, α_2 i α_3 davat će veću masu vjerojatnosti kombinacijama kod kojih μ_k nisu jednaki (npr. slike f–i).

Lako bismo mogli izvesti Dirichlet-kategorički model, ali nećemo. Kada bismo to napravili, za procjenitelj za μ_k dobili bismo opet da je jednak modu aposteriorne (Dirichletove) distribucije:

$$\hat{\mu}_{k,\text{MAP}} = \frac{\alpha'_k - 1}{\sum_{k=1}^K \alpha'_k - K}$$

gdje

$$\alpha'_k = N_k + \alpha_k$$

gdje je N_k broj nastupanja k -te vrijednosti.

Ako želimo modelirati da je najvjerojatnije da su sve pojedinačne vrijednosti kategoriskske varijable jednako vjerojatne, tj. $\mu_k = 1/K$, onda možemo staviti $\alpha_k = 2$. Za MAP procjenitelj ćemo onda dobiti:

$$\hat{\mu}_{k,\text{MAP}} = \frac{N_k + 2 - 1}{\sum_k (N_k + 2) - K} = \frac{N_k + 1}{N + K}$$

gdje je $N = \sum_k N_k$ ukupan broj primjera.

► PRIMJER

Npr., ako kategoriskska varijabla ima $K = 3$ moguće vrijednosti u skupu od $N = 10$ primjera, ali prva vrijednost se nikada nije pojavila, $N_1 = 0$, MLE procjena za tu vrijednost bi bila $\hat{\mu}_1 = 0$, dok je MAP procjena $\hat{\mu}_1 = \frac{0+1}{10+3} = 0.077$.

Sažetak

- **Procjenitelj najveće izglednosti (MLE)** odabire parametre koji maksimiziraju vjerojatnost realizacije uzorka (tj. izglednost)
- Kod poopćenih linearnih modela, MLE je istovjetan minimizaciji empirijske pogreške
- MLE je jednostavan, ali lako daje **prenaučene modele**
- **MAP-procenitelj** dodatno koristi apriornu razdiobu parametara i maksimizira aposteriornu vjerojatnost parametara, efektivno provodeći **zaglađivanje**
- MAP-procenitelj za binarnu varijablu koristi **Beta-Bernoullijev model**
- MAP-procenitelj za kategorisksku varijablu koristi **Dirichlet-kategorički model**

Bilješke

1 Već smo to bili napomenuli, ali napomenimo za svaki slučaj ponovo, da je \mathbf{x} u izrazu $p(\mathbf{x}|\boldsymbol{\theta})$ slučajna varijabla, dok $\boldsymbol{\theta}$ to nije, već je to samo parametar distribucije. Bilo bi, stoga, korektnije pisati $p(\mathbf{x}; \boldsymbol{\theta})$, jer ako pišemo $p(\mathbf{x}|\boldsymbol{\theta})$ to izgleda kao uvjetna vjerojatnost gdje su i \mathbf{x} i $\boldsymbol{\theta}$ varijable. Međutim, u literaturi se uvriježilo pisati $p(\mathbf{x}|\boldsymbol{\theta})$, pa ćemo i mi ostati na tome. Imajte, međutim, na umu da je

θ parametar, a ne slučajna varijabla. Ova napomena ne vrijedi za bayesovsku statistiku, kod koje su i \mathbf{x} i θ slučajne varijable. Više o tome pri kraju današnjeg predavanja.

- [2] Jasnoće radi, u nastavku ćemo često koristiti naziv **procjentielj MLE**, premda je to pleonazam, budući da “E” već stoji za “procjenitelj”.
- [3] Primjenimo **metodu Lagrangeovih multiplikatora** na maksimizaciju funkcije log-izglednosti kategoričke (multinulijeve) varijable. Želimo maksimizirati sljedeću funkciju log-izglednosti:

$$\ln \mathcal{L}(\boldsymbol{\mu}|\mathcal{D}) = \ln \prod_{i=1}^N P(\mathbf{x}^{(i)}|\boldsymbol{\mu}) = \ln \prod_{i=1}^N \prod_{k=1}^K \mu_k^{x_k^{(i)}} = \sum_{k=1}^K \sum_{i=1}^N x_k^{(i)} \ln \mu_k$$

uz ograničenje $\sum_{k=1}^K \mu_k = 1$. Prisjetite se (iz skriptice 8) kako se definira Lagrangeova funkcija. U ovom slučaju, Lagrangeova je funkcija sljedeća:

$$\sum_{i=1}^N \sum_{k=1}^K x_k^{(i)} \ln \mu_k + \lambda \left(\sum_{k=1}^K \mu_k - 1 \right)$$

gdje je λ Lagrangeov multiplikator. Deriviranjem po μ_k i izjednačavanjem s nulom dobivamo:

$$\mu_k = -\frac{1}{\lambda} \sum_{i=1}^N x_k^{(i)}$$

Kako bismo izračunali vrijednost multiplikatora λ , dobiveni izraz uvrštavamo u ograničenje $\sum_k \mu_k = 1$ i tako dobivamo:

$$\sum_{k=1}^K \mu_k = -\frac{1}{\lambda} \underbrace{\sum_{k=1}^K \sum_{i=1}^N x_k^{(i)}}_{=N} = 1$$

Budući da svaka multinomijalna varijabla \mathbf{x} ima jedinicu postavljenu na samo jednoj komponenti, zbroj komponenata svih varijabli jednak je broju primjera N . Vrijedi dakle $\lambda = -N$, pa uvrštavanjem u gornju jednadžbu dobivamo:

$$\hat{\mu}_{k,\text{ML}} = \frac{1}{N} \sum_{i=1}^N x_k^{(i)} = \frac{N_k}{N}$$

- [4] Preciznije, $\ln p(\mathbf{y}|\mathbf{X})$ nije logaritam vjerojatnosti oznaka, nego logaritam funkcije gustoće vjerojatnosti. Osim toga, ta je funkcija implicitno i funkcija težina \mathbf{w} , dakle trebali bismo pisati $\ln p(\mathbf{y}|\mathbf{X}, \mathbf{w})$. Iz toga je onda jasnije da je odgovarajuća funkcija log-izglednosti $\ln \mathcal{L}(\mathbf{w}|\mathbf{X}, \mathbf{y})$. Par (\mathbf{X}, \mathbf{y}) čini skup označenih primjera \mathcal{D} , pa je, dakle, funkcija log-izglednosti $\ln \mathcal{L}(\mathbf{w}|\mathcal{D})$, tj. to je funkcija težina \mathbf{w} uz fiksirani skup primjera \mathcal{D} .

- [5] Za pravedan novčić očekujemo $\mu = 0.5$. Međutim, Jaynes smatra: “...anyone familiar with the law of conservation of angular momentum can, after some practice, cheat at the usual coin-toss game and call his shots with 100 per cent accuracy. You can obtain any frequency of heads you want; and the bias of the coin has no influence at all on the results!” (Jaynes, 2003). Ipak, nisam uvjeren da se *znanje* o očuvanje kutne količine gibanja doista, pa čak i nakon “nešto vježbe”, može pretočiti u *vještinu* bacanja novčića tako da se uvijek dobije željeni ishod, jednako kao što nisam uvjeren da bi netko znanje o kinematici krutog tijela mogao iskoristiti da uvijek obrani jedanaesterac, koliko god marljivo vježbao. Ali, u duhu epistemičke skromnosti, ostajem otvoren za takvu mogućnost (ovo se ne računa: <https://www.youtube.com/watch?v=MZc3VD214OU>).
- [6] Ovdje postoji mala terminološka zbrka, jer se pridjev **konjugatan** koristi na dva različita načina: za opis toga da su apriorna i aposterioridna distribucije iste vrste i za opis toga da je apriorna distribucija distribucija takva da, kada se pomnoži s izglednošću, daje distribuciju koja je iste vrste kao i aposteriorna distribucija. Upamtite: ako je $p(\theta)$ **konjugatna distribucija za izglednost** $p(\mathcal{D}|\theta)$, onda su $p(\theta)$ i $p(\theta|\mathcal{D})$ **konjugatne distribucije**, i obrnuto.
- [7] Također, za $\alpha = \beta = 1$ to je ujedno i tzv. **nepravilna apriorna distribucija** (engl. *improper prior*), jer to onda nije prava funkcija gustoće vjerojatnosti budući da je funkcija veća od nule na intervalu od

minus do plus beskonačno, pa njezin integral nije jednak. Međutim, u praksi nas to ne smeta, sve dok je rezultirajuća aposteriorna funkcija dobro definirana distribucija (a bit će, jer ju normaliziramo u nazivniku Bayesovog pravila).

- [8] Također, u tom smislu možemo reći da je gustoća $p(\mu)$ zapravo **distribucija distribucije** ili **distribucija drugog reda**. Zašto? Zašto što za svaku vrijednost od μ imamo jednu Bernoullijevu distribuciju, pa onda distribucija $p(\mu)$ opisuje kako su distribuirane te Bernoullijeve distribucije. Npr., za $\alpha = \beta = 2$, najverovatnije Bernoullijeve distribucije su one za koje je $\mu = 0.5$.

Literatura

E. T. Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

15. Bayesov klasifikator

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v3.1

Prošli puta pričali smo o statističkoj procjeni parametara, što je ključni mehanizam učenja probabilističkih modela. Prisjetili smo se osnovnih vjerojatnosnih distribucija, koje se koriste u strojnem učenju te smo zatim pričali o **procjeniteljima**. Objasnili smo procjenitelj **najveće izglednosti (MLE)**, a zatim smo razmotrili procjenitelj **maksimum aposteriori (MAP)**, koji nam omogućava da u procjenu ugradimo naše apriorno znanje o parametrima.

Danas ćemo razmotriti najjednostavniji probabilistički model – **Bayesov klasifikator**. Prvo ćemo pričati o Bayesovom klasifikatoru za kontinuirane značajke, odnosno **Gaussovom Bayesovom klasifikatoru**. U idućem predavanju pogledat ćemo onda Bayesov klasifikator za diskretne značajke.

Bayesov klasifikator ujedno će biti naš prvi **generativni model** – svi modeli koje smo razmatrali do sada bili su diskriminativni. Reći ćemo nešto više o toj podjeli na generativne i diskriminativne modele.

1 Pravila vjerojatnosti

Prije nego što zaronimo u Bayesov klasifikator, prisjetit ćemo se nekih jednostavnih ideja iz teorije vjerojatnosti, a koje se sveprisutne kod probabilističkih modela. Cijela algebra teorije vjerojatnosti svodi se na dva jednostavna pravila: pravilo zbroja i pravilo umnoška. **Pravilo zbroja** je:

$$P(x) = \sum_y P(x, y)$$

Vjerojatnost $P(x, y)$ je vjerojatnost zajedničke realizacije x i y . U strojnem učenju, to je vjerojatnost da primjer \mathbf{x} ima oznaku y . Tu vjerojatnost zovemo **zajednička (združena) vjerojatnost** (engl. *joint probability*). Pravilo zbroja nam govori da iz zajedničke vjerojatnosti možemo dobiti vjerojatnost pojedinačnih varijabli ili podskupa varijabli. Ta vjerojatnost se onda zove **marginalna vjerojatnost**. Zove se marginalna jer se dobiva **marginalizacijom** – izračunom vjerojatnosti podskupa varijabli. Marginalizacija, naravno, vrijedi i kada imamo više varijabli. Tada možemo marginalizirati po, npr., samo jednoj varijabli:

$$P(x, y) = \sum_z P(x, y, z)$$

ili možemo marginalizirati po više varijabli:

$$P(x) = \sum_y \sum_z P(x, y, z)$$

Drugo pravilo teorije vjerojatnosti je **pravilo umnoška**. Najprije, prisjetimo se da je **uvjetna vjerojatnost** definirana kao:

$$P(y|x) = \frac{P(x, y)}{P(x)} \quad \text{ili} \quad P(x|y) = \frac{P(x, y)}{P(y)}$$

To je vjerojatnost vrijednosti y , ako znamo da je ostvarena vrijednost x (ili obrnuto, za drugu formulu). Pravilo umnoška samo je drugi pogled na definiciju uvjetne vjerojatnosti:

$$P(x, y) = P(y|x)P(x) = P(x|y)P(y)$$

Ovo su, dakle, ta dva osnovna pravila. Oba pravila vrijede i ako vjerojatnost P zamijenimo s gustoćom vjerojatnosti p . Sve što ćemo mi raditi u nastavku svodit će se na pametnu primjenu ova dva pravila. Zapravo, odmah ćemo upotrijebiti ova dva pravila kako bismo izveli osnovno pravilo koje nam treba za Bayesov klasifikator, a to je **Bayesovo pravilo**:

$$\begin{aligned} P(x, y) &= P(y|x)P(x) = P(x|y)P(y) / P(x) \\ P(y|x) &= \frac{P(x|y)P(y)}{P(x)} \end{aligned}$$

Snaga bayesovog pravila leži u tome što nam omogućava da obrnemo smjer zaključivanja: iz vjerojatnosti $P(x|y)$ možemo zaključiti o vjerojatnosti $P(y|x)$. U logici bi se to zvalo **abduktivno zaključivanje**. U klasičnoj logici abduktivno zaključivanje je “persona non grata”, zato jer je to pravilo evidentno neispravno (zaključak nije logička posljedica premisa), no u strojnog učenju abdukcija je dragi gost koji nosi lijepo darove (mogućnost zaključivanja od podataka prema klasi, tj. od posljedice prema uzroku).

Sada kada znamo osnovna dva pravila vjerojatnosti (pravilo zbroja i pravilo umnoška) te iz njih izvedeno Bayesovo pravilo, spremni smo za Bayesov klasifikator.

2 Bayesov klasifikator

Bayesov klasifikator izravno primjenjuje Bayesov teorem kako bi izračunao vjerojatnost oznake y za zadani ulazni primjer \mathbf{x} :

$$P(y|\mathbf{x}) = \frac{p(\mathbf{x}, y)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|y)P(y)}{p(\mathbf{x})}$$

Primijetite da se ovdje pojavljuju **vjerojatnosti** (veliko “ P ”) i **gustoće vjerojatnosti** (malo “ p ”), ovisno o tome je li varijabla diskretna ili kontinuirana. Konkretno, varijabla y je diskretna oznaka klase, dok je vektor primjera \mathbf{x} općenito vektor značajki, diskretnih ili kontinuiranih, pa pišemo gustoću vjerojatnosti jer je to općenitiji slučaj. To što množimo gustoću vjerojatnosti p s vjerojatnošću P nije nikakav problem (rezultat će biti gustoća vjerojatnosti).

Vjerojatnost $P(y|\mathbf{x})$ nazivamo **aposteriorna vjerojatnost oznake** (engl. *posterior*): to je vjerojatnost oznake y za zadani primjer \mathbf{x} . I to je ono što nas zapravo zanima: koliko je vjerojatno da primjer \mathbf{x} pripada klasi y . Gustoću vjerojatnosti $p(\mathbf{x}|y)$ nazivamo **izglednost klase** (engl. *class likelihood*). To je gustoća vjerojatnosti primjera uz zadanu klasu, odnosno distribucija primjera unutar dotične klase. Intuitivno, izglednost klase nam govori kolika je vjerojatnost primjera za zadanu klasu (“ako gledam samo primjere iz klase y , koliko je vjerojatno da vidim \mathbf{x} ”). Primijetite da je to točno obrnuto od aposteriorne vjerojatnosti, koja nam govori koja je vjerojatnost klase za zadani primjer. Vjerojatnost $P(y)$ naziva se **apriorna vjerojatnost klase** (engl. *class prior*). To je vjerojatnost klase neovisno o primjerima. Npr., ako radimo klasifikaciju treba li klijentu banke odobriti kredit, i ako je situacija takva da se većina (npr. 80%) kredita odobrava (a vjerojatno jest jer inače ne bi bili tu gdje jesmo), onda je $P(y = 1) = 0.8$. Konačno, gustoća vjerojatnosti $p(\mathbf{x})$ je gustoća vjerojatnosti primjera neovisno o klasi (“koliko je vjerojatno da vidim \mathbf{x} iz bilo koje klase”).

Možda ste se pitali – ili, ako niste, možda da se sada pitate – zašto uopće rastavljamo zajedničku gustoću vjerojatnosti $p(\mathbf{x}, y)$ na izglednost $p(\mathbf{x}|y)$ i apriornu vjerojatnost $P(y)$? Naime, matematički gledano, očito je to opet jednako zajedničkoj vjerojatnosti, pa ispada da zapravo ništa nismo napravili. No, problem je u tome što je zajednička vjerojatnost (odnosno

1

2

zajednička gustoća vjerojatnosti) općenito može biti vrlo složena distribucija. Npr., \mathbf{x} može biti vektor kontinuiranih vrijednosti, dok je y diskretna varijabla (npr. klasifikacija učenika po uspjehu u srednjoj školi). Takve složene distribucije ne možemo modelirati nekom standardnom teorijskom distribucijom. Međutim, ako zajedničku distribuciju rastavimo, dobivamo dvije jednostavnije distribucije, koje možemo mnogo jednostavnije modelirati, koristeći standardne distribucije. Štoviše, gustoću $p(\mathbf{x}|y)$ možemo modelirati *zasebno* za svaku pojedinu klasu y , što vrlo pojednostavljuje stvari. Rastavljanje neke složene distribucije na umnožak jednostavnijih distribucija naziva se **faktorizacija**.

Jedini preostali problem je $p(\mathbf{x})$ u nazivniku Bayesovog pravila. To je gustoća vjerojatnosti primjera \mathbf{x} , neovisno o klasi. Zašto nam je ona problematična? Zato jer je i ta distribucija također općenito vrlo složena. Zato ćemo i nju faktorizirati, i to ovako:

$$p(\mathbf{x}) = \sum_y p(\mathbf{x}, y) = \sum_y p(\mathbf{x}|y)P(y)$$

Dakle, faktorizaciju smo napravili tako da smo najprije primijenili pravilo zbroja (po svim mogućim oznakama klase), a zatim pravilo umnoška. Uvjerite se da je to doista jednako $p(\mathbf{x})$. Ovako modelirati $p(\mathbf{x})$ jednostavnije je iz istog razlog koji smo spomenuli gore: modeliramo zasebno izglednost klase i zasebno apriornu vjerojatnost klase, i te dvije distribucije možemo modelirati jednostavnim teorijskim distribucijama. To što je $p(\mathbf{x})$ onda složena distribucija koja se dobiva po gornjem izrazu nas više ne zanima.

Nakon svih ovih intervencija, dolazimo napokon do **modela Bayesovog klasifikatora**:

$$h_j(\mathbf{x}; \boldsymbol{\theta}) = P(y = j|\mathbf{x}) = \frac{p(\mathbf{x}|y)P(y)}{\sum_{y'} p(\mathbf{x}|y')P(y')}$$

Ovako definirana hipoteza daje nam vjerojatnost klasifikacije u klasu j . Primijetite da Bayesov klasifikator bez problema može raditi s više klasa (formalno, imamo po jednu hipotezu h_j za svaku od K klasa).

Ako nas, međutim, ne zanimaju vjerojatnosti klasifikacije u pojedine klase, nego samo želimo odrediti oznaku primjera, onda ćemo ga klasificirati u klasu čija je vjerojatnost najveća. To je tzv. **maksimum a posteriori hipoteza (MAP)**. Model tada definiramo kao:

$$h(\mathbf{x}; \boldsymbol{\theta}) = \operatorname{argmax}_y p(\mathbf{x}|y)P(y)$$

► PRIMJER

Radimo klasifikaciju u $K = 3$ klase. Recimo da smo naučili model (tj. procijenili parametre modela). Apriorne vjerojatnosti klase su $P(y = 1) = P(y = 2) = 0.3$, $P(y = 3) = 0.4$. Za neki konkretni primjer \mathbf{x} , izglednosti klase su $p(\mathbf{x}|y = 1) = 0.9$, $p(\mathbf{x}|y = 2) = p(\mathbf{x}|y = 3) = 0.4$. U koju klasu klasificiramo primjer \mathbf{x} ?

$$p(x_1|y = 1)P(y = 1) = 0.9 \cdot 0.3 = 0.27$$

$$p(x_1|y = 2)P(y = 2) = 0.4 \cdot 0.3 = 0.12$$

$$p(x_1|y = 3)P(y = 3) = 0.4 \cdot 0.4 = 0.16$$

$$p(x) = \sum_{j=1}^3 p(x|y = j)P(y = j) = 0.55$$

$$P(y = 1|x) = 0.27/0.55 = 0.49 \quad \Leftarrow \text{MAP (primjer klasificiramo u ovu klasu)}$$

$$P(y = 2|x) = 0.12/0.55 = 0.22$$

$$P(y = 3|x) = 0.16/0.55 = 0.29$$

Vektor θ je vektor parametara apriorne distribucije i izglednosti klase. Koji su to točno parametri i koliko ih ukupno ima ovisi o tome koje smo distribucije odabrali. Što se apriorne vjerojatnosti klase tiče, ako je klasifikacija binarna ($K = 2$), koristit ćemo **Bernoullijevu distribuciju**, dok ćemo za višeklasnu ($K > 2$) klasifikaciju koristiti **kategoričku (multinulijevu distribuciju)**. Što se izglednosti klase tiče, ako su značajke diskretne, koristit ćemo **Bernoullijevu ili kategoričku distribuciju**, ovisno o tome je li značajka binarna ili viševrijednosna. Za kontinuirane značajke koristit ćemo **Gaussovnu (normalnu) distribuciju**. Zapravo, budući da ćemo uvjek imati više od jedne značajke, koristit ćemo **multivarijatnu Gaussovnu distribuciju**.

Bayesov klasifikator je **parametarski model**. Što to znači? Općenito, to znači da broj parametara modela ne ovisi o broju primjera. No, budući da se ovdje konkretno radi o probabilističkom modelu, to također znači da model pretpostavlja da se primjeri x i oznake y pokoravaju nekoj teorijskoj vjerojatnosnoj distribuciji. Naravno, broj parametara tih distribucija ne ovisi o broju primjera, pa zato ni broj parametara cjelokupnog modela ne ovisi o broju primjera.

Ovime smo definirali model Bayesovog klasifikatora. Što je s druge dvije komponente algoritma strojnog učenja: funkcijom pogreške i optimizacijskim postupkom? Drugim riječima, kako ćemo trenirati Bayesov klasifikator? Prisjetimo se da kod probabilističkih modela trenirati model znači **procijeniti parametre**. Za procjenu parametara upotrijebit ćemo ono što smo usvojili prošli tjedan: MLE procjenitelje ili MAP procjenitelje. Dakle, optimizacijski postupak bit će maksimizacija izglednosti parametara (MLE) ili maksimizacija aposteriorne vjerojatnosti parametara (MAP). Sukladno tome, funkcija pogreške bit će jednostavno negativna izglednost parametara (MLE) ili negativna apostериорna vjerojatnost parametara (MAP).

Bayesov klasifikator naš je prvi **generativni model**. Sada je dobar trenutak da kažemo nešto više o toj familiji modela.

3 Generativni modeli

Generativni modeli modeliraju **zajedničku vjerojatnost** (odnosno zajedničku gustoću vjerojatnosti) $p(x, y)$. Kao što smo vidjeli, na temelju te vjerojatnosti može se vrlo jednostavno, primjenom Bayesovog pravila, izračunati aposteriorna vjerojatnost $P(y|x)$, tj. vjerojatnost da primjer x pripada klasi y .

Ovakav pristup, koji modelira zajedničku distribuciju primjera x i klase y , nazivamo generativnim jer modelira postupak **generiranja (nastanka) podataka**. Da bismo razumijeli što pod time mislimo, pogledajmo brojnik Bayesovog klasifikatora:

$$P(x, y) = p(x|y)P(y)$$

Želimo li opisati način nastanka označenih primjera $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_i$, koji se ravnaju po zajedničkoj distribuciji $p(x, y)$, možemo reći da primjeri nastaju **stohastičkim procesom** koji se sastoji od dva koraka: u prvome koraku odabrana je oznaka y po distribuciji $P(y)$, a u drugome je koraku za odabrani y odabran primjer x po distribuciji $p(x|y)$. Takva priča, koja objašnjava stohastički postupak generiranja podataka, naziva se **generativna priča** (engl. *generative story*).

Kod Bayesovog klasifikatora generativna je priča dosta kratka i pomalo dosadna, jer se sastoji od svega dva koraka, ali općenito, kod složenijih generativnih modela, priča može biti dulja i uzbudljivija. Takvi složeniji generativni modeli su, npr., **Bayesove mreže, mješavina Gausso-vih distribucija** (engl. *Gaussian mixture model, GMM*), **skriven Markovljev model** (engl. *Hidden Markov model, HMM*), i **latentna Dirichletova alokacija** (engl. *latent Dirichlet allocation, LDA*). Prva dva modela ćemo pogledati u narednim tjedima.

Generativna priča također se može upotrijebiti za generiranje sintetičkih podataka. Jednom kada je model naučen, možemo slijediti generativnu priču kako bismo uzorkovali primjere iz zajedničke distribucije. Kod Bayesovog klasifikatora, za svaki primjer prvo bismo uzorkovali klasu y prema distribuciji $P(y)$, a onda bismo uzorkovali primjer iz distribucije $P(x|y)$ koja odgovara izglednosti za dotičnu klasu y .

3.1 Generativno vs. diskriminativno

Usporedimo sada generativne modele sa **diskriminativnim modelima**. Svi modeli s kojima smo se do sada bavili (linearna regresija, logistička regresija, SVM, k-NN) bili su diskriminativni, a ne generativni. U čemu je razlika?

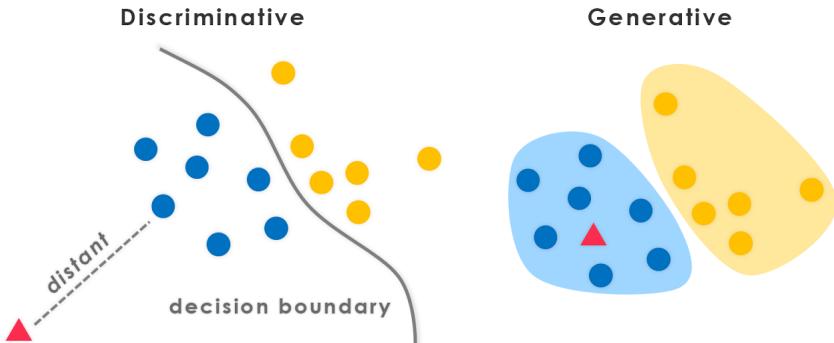
Diskriminativni modeli izravno modeliraju aposteriornu vjerojatnost $P(y|\mathbf{x})$. Prisjetimo se logističke regresije. Model je tamo bio definiran ovako:

$$h(\mathbf{x}; \mathbf{w}) = P(y|\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$$

Primijetite da tu niti u jednom trenutku nismo modelirali zajedničku vjerojatnost, nego smo direktno modelirali aposteriornu vjerojatnost. To je drugačije od generativnih modela, kod kojih aposteriornu vjerojatnost modeliramo indirektno, preko zajedničke vjerojatnosti.

Logistička regresija je probabilistički diskriminativni model – modelira vjerojatnost oznake za neki primjer. Međutim, u porodicu diskriminativnih modela pripadaju i oni modeli koji uopće ne modeliraju vjerojatnost, npr. SVM. Zapravo, većina diskriminativnih modela nisu probabilistički (npr., SVM, perceptron, višeslojni perceptron, stabla odluke, k-NN). Ti modeli doduše ne modeliraju aposteriornu vjerojatnost, ali, slično kao i logistička regresija, direktno modeliraju granicu između klasa (ni u kojem trenutku SVM ne modelira zajedničku vjerojatnost primjera i oznaka).

Razliku između generativnih i diskriminativnih modela ilustrira sljedeća slika:



Prikazan je problem binarne klasifikacije (plavi vs. žuti primjeri). Na lijevoj je slici prikazan rezultat dobiven nekim diskriminativnim modelom. Sve što ovdje dobivamo je granica između klase (decizijska granica), koja je općenito nekakva hiperpovršina u ulaznom prostoru za koju $h(\mathbf{x}) = 0.5$ (kod logističke regresije) ili $h(\mathbf{x}) = 0$ (kod SVM-a). Koncept pouzdanosti klasifikacije povezan je s konceptom udaljenosti primjera od granice. Na desnoj slici prikazan je rezultat na istom skupu primjera dobiven nekim generativnim modelom. Ovdje modeliramo vjerojatnosne distribucije primjera unutar svake od dviju klasa $p(\mathbf{x}|y)p(y)$, što, nakon množenja s apriornom vjerojatnošću klase $P(y)$, daje distribuciju $p(\mathbf{x}, y)$, tj. zajedničku vjerojatnost (odnosno gustoću vjerojatnosti) primjera i oznaka. Kada želimo raditi klasifikaciju primjera, iz te zajedničke vjerojatnosti možemo izračunati aposteriornu vjerojatnost $p(y|\mathbf{x})$, i klasificirati primjere u klasu za koju je ta vjerojatnost najveća (MAP hipoteza). Time je onda implicitno definirana i granica između dviju klasa kao $h(\mathbf{x}) = p(y|\mathbf{x}) = 0.5$.

3.2 Prednosti i nedostatci

Ovdje se sad, naravno, postavlja neizbjegljivo pitanje: jesu li bolji generativni ili diskriminativni modeli? Odgovor je, kako to često biva: ovisi. Prisjetite se **teorema o nebesplatnom ručku** (engl. *no free lunch theorem*): niti jedan algoritam nije univerzalno najbolji. U nekim situacijama generativni modeli mogu biti bolji, a u nekim diskriminativni.

Da bismo znali ocijeniti koji je model bolji za koji problem, trebamo znati njihove prednosti i nedostatke. Krenimo od prednosti generativnih modela:

- U generativne modele relativno je lako ugraditi **pozadinsko/stručno znanje o problemu**. To znanje je u obliku apriornih distribucija (apriorna distribucija klase, apriorna distribucija parametara) ili čak specifičnih struktura modela. Takvo znanje može se onda elegantno kombinirati sa znanjem dobivenim na temelju podataka – naznaku te ideje već smo vidjeli kod procjenitelja MAP;
- Druga prednost je **interpretabilnost rezultata i mogućnost različitih analiza** – generativni modeli nude vrlo intuitivnu interpretaciju podataka temeljenu na teoriji vjerojatnosti. Također, budući da modeliramo zajedničku distribuciju, možemo raditi različite analize ovisnosti između varijabli (tj. između značajki), a također možemo predviđati vrijednosti značajki na temelju drugih značajki, itd. Dakle, nismo ograničeni samo na izračun aposteriorne vjerojatnosti $p(y|x)$, nego možemo izračunati razne vjerojatnosti koje nas zanimaju. Naime, zajednička distribucija sadrži sve informacije – sve ostale vjerojatnosti (uvjetne, marginalne) mogu se iz nje izvesti i sadrže manju informaciju.

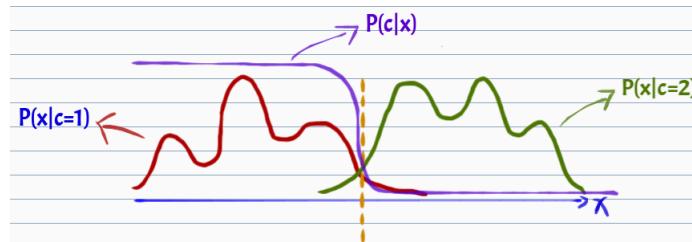
Druge (manje važne) prednosti generativnih modela su:

- **Ocjena pouzdanosti klasifikacije** – izlaz klasifikatora može se tumačiti kao vjerojatnost ili pouzdanost da primjer \mathbf{x} pripada klasi y – ovo je doduše prednost svih modela koji imaju probabilistički izlaz, a koji ne moraju nužno biti generativni, npr. logistička regresija;
- **Odbijanje klasifikacije** – ako je za neki primjer \mathbf{x} izlaz klasifikatora manji od unaprijed zadanog praga, klasifikator može odbiti klasificirati primjer \mathbf{x} i tako smanjiti broj pogrešnih klasifikacija. Primjeri koje klasifikator odbije klasificirati mogu se proslijediti na ručnu klasifikaciju. Ovo je također prednost koja nije ekskluzivna za generativne modele;
- Nalaženje **stršćih vrijednosti** (engl. *outliers*) – marginalizacijom vjerojatnosti $P(\mathbf{x}, y)$ možemo odrediti vjerojatnost primjera $P(\mathbf{x})$ i tako detektirati vrijednosti koje odskaču.

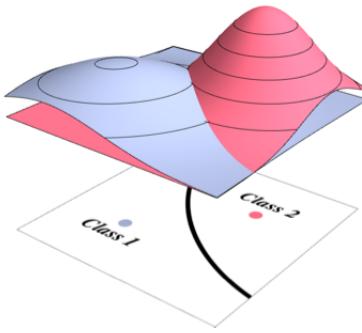
Naravno, generativni modeli u odnosu na diskriminativne modele imaju i neke nedostatke.

Glavni nedostatci su:

- **Potreba za velikim brojem primjera** – modeliranje zajedničke vjerojatnosti $P(\mathbf{x}, y)$ iziskuje velik broj primjera, a da bi procjena parametara bila pouzdana. To je pogotovo problem kada je ulazni prostor visoke dimenzije, dakle kada vektor \mathbf{x} sadrži mnogo značajki.
- **Nepotrebna složenost modeliranja** – ako je naš konačni cilj klasifikacija, onda je zapravo nepotrebno modelirati zajedničku distribuciju $P(\mathbf{x}, y)$, koja može biti vrlo složena i za čiju procjenu treba mnogo primjera. U tom slučaju dovoljno je izravno modelirati samo aposteriornu vjerojatnost $P(y|\mathbf{x})$, kao što to čine diskriminativni modeli. Ovaj nedostatak lijepo ilustrira sljedeća slika:



Ako je sve što nas zanima samo granica između klasa, onda je logistička regresija (sigmoïda na slici) puno jefiniji model (u smislu složenosti modeliranja, tj. broja parametara) od generativnog modela koji je ovdje poprilično složen (složene višemodalne izglednosti klase). Isto vrijedi i u ulaznom prostoru s više značajki, npr. dvije značajke:

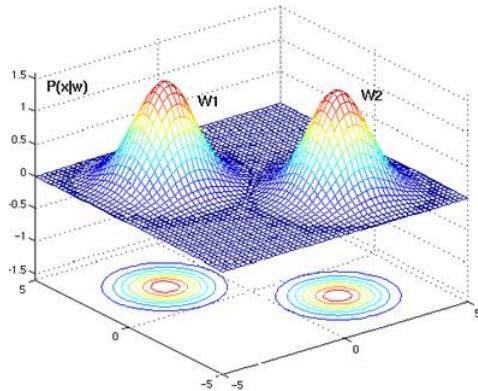


Budući da složenost modela korelira s brojem parametara (složeniji modeli imaju više parametara), to će generativni model u pravilu imati više parametara nego diskriminativni model koji otprilike radi isti posao u smislu klasifikacije. (Vidjet ćemo primjer toga na idućem predavanju.) Vjerojatno upravo zbog te veće složenosti, u praksi se pokazuje da diskriminativni modeli – ako nas zanima samo točnosti klasifikacije – rade bolje od generativnih. Ipak, imajte na umu da to ne treba uvijek biti slučaj (nebesplatan ručak). Također, imajte na umu sve prednosti generativnih modela: nekad nam treba više od same klasifikacije – nekad je ključno da u model ugradimo pozadinsko znanje, ili želimo dubinski analizirati značajke i tumačiti rezultate. Onda su generativni modeli bolji izbor od diskriminativnih.

6

4 Gaussov Bayesov klasifikator

Pogledajmo sada napokon kako definirati Bayesov klasifikator, i to Bayesov klasifikator za kontinuirane značajke, tzv. **Gaussov Bayesov klasifikator**. Kod tog modela primjer \mathbf{x} predstavljen je kao vektor brojeva, tj. značajke su numeričke, što znači da ćemo izglednosti klase $P(\mathbf{x}|y)$ modelirati **Gaussovom distribucijom**. Npr., za dvije značajke (dvodimenzionalni ulazni prostor) to izgleda ovako:



7

Ideja je da izglednost klase svake modeliramo kao jednu zasebnu multivarijatnu Gaussovou distribuciju. Srednja vrijednost te distribucije, dakle vektor μ , predstavlja **prototipni primjer** te klase, i taj primjer ima najveću gustoću vjerojatnosti. Primjeri koji su udaljeni od središta su manje vjerojatni da pripadaju toj klasi. Idealno, svi primjeri koji pripadaju ovoj klasi bi bili jednak prototipnom primjeru, međutim zbog **šuma** to nije slučaj. Dakle, Gaussova distribucija ovdje modelira odstupanje od idealnog uslijed šuma.

8

4.1 Univarijatni Gaussov Bayesov klasifikator

Jednostavnosti radi, za početak ćemo pogledati (nerealan) slučaj kada je prostor primjera jednodimenzionalni, tj. kada imamo samo jednu značajku. U tom slučaju izglednost klase $P(x|y)$

modeliramo univarijatnom Gaussovom distribucijom:

$$x|y \sim \mathcal{N}(\mu_j, \sigma_j^2)$$

Prisjetimo se Gaussove distribucije:

$$p(x|y = j) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp \left\{ -\frac{(x - \mu_j)^2}{2\sigma_j^2} \right\}$$

Model je:

$$h(x) = \underset{y}{\operatorname{argmax}} p(x, y) = \underset{y}{\operatorname{argmax}} p(x|y)P(y)$$

Ako nas zanima samo **pouzdanost** za svaku klasu j , a ne vjerojatnost, možemo definirati model čije je izlaz jednak zajedničkoj vjerojatnosti:

$$h_j(x) = p(x, y = j) = p(x|y = j)P(y = j)$$

Radi matematičke jednostavnosti, prelazimo u logaritamsku domenu:

$$\begin{aligned} h_j(x) &= \ln p(x|y = j) + \ln P(y = j) \\ &= -\frac{1}{2} \cancel{\ln 2\pi} - \ln \sigma_j - \frac{(x - \mu_j)^2}{2\sigma_j^2} + \ln P(y = j) \end{aligned}$$

Uklanjanje konstante (ne utječe na maksimizaciju):

$$h_j(x|\theta_j) = -\ln \sigma_j - \frac{(x - \mu_j)^2}{2\sigma_j^2} + \ln P(y = j)$$

gdje je vektor parametara jednak

$$\theta_j = (\mu_j, \sigma_j, \mu'_j)$$

pri čemu smo sa μ'_j označili parametar distribucije $P(y = 1)$, koji je jednak apriornoj vjerojatnosti klase $y = 1$. Ovdje se radi o poznatim distribucijama (Gaussova i Bernoullijeve). Njihove parametre možemo procijeniti pomoću procjenitelja MLE (kojeg smo izveli prošli put):

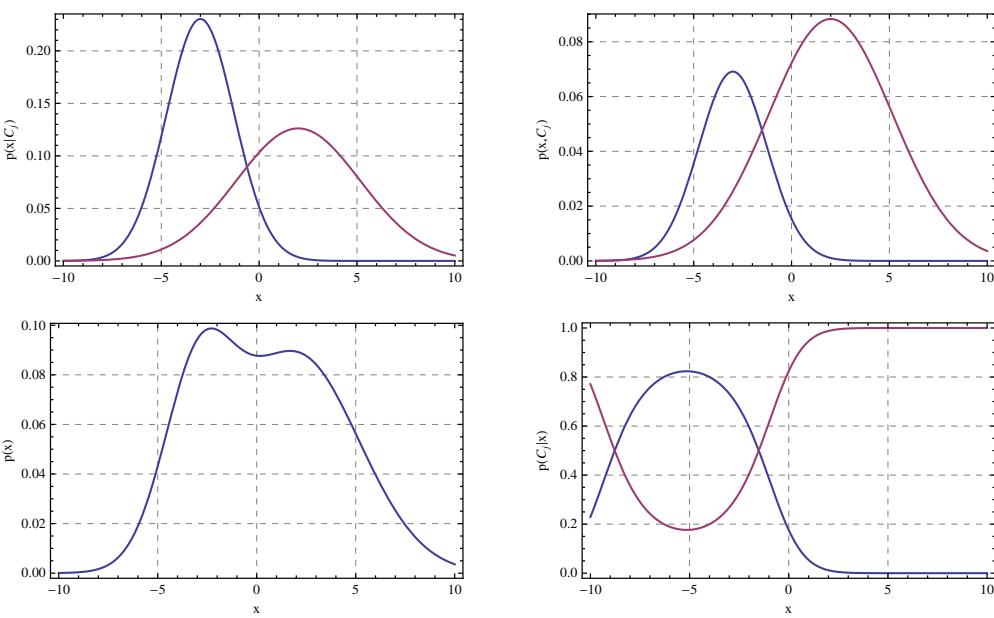
$$\begin{aligned} \hat{\mu}_j &= \frac{1}{N_j} \sum_{i=1}^N \mathbf{1}\{y^{(i)} = j\}x^{(i)} \\ \hat{\sigma}_j^2 &= \frac{1}{N_j} \sum_{i=1}^N \mathbf{1}\{y^{(i)} = j\}(x^{(i)} - \hat{\mu}_j)^2 \\ P(y = j) &= \hat{\mu}'_j = \frac{1}{N} \sum_{j=1}^N \mathbf{1}\{y^{(i)} = j\} = \frac{N_j}{N} \end{aligned}$$

► PRIMJER

Klasificiramo u dvije klase, $y = 1$ i $y = 0$. Izglednosti tih klasa definirane su sljedećim Gaussovovim gustoćama vjerojatnosti:

$$\begin{aligned} p(x|y = 1) &= \mathcal{N}(-3, 3) \\ p(x|y = 0) &= \mathcal{N}(2, 10) \end{aligned}$$

Apriorne vjerojatnosti klasa neka su $P(y = 1) = 0.3$ i $P(y = 0) = 0.7$. Gustoće vjerojatnosti onda izgledaju ovako:



Krenimo od gornje lijeve slike. Ona prikazuje izglednosti za dvije klase $y = 1$ i $y = 0$, tj. gustoće vjerojatnosti $p(x|y = 1)$ (plava krivulja) i $p(x|y = 0)$ (ljubičasta krivulja). Budući da gustoća vjerojatnosti za $y = 1$ ima manju varijancu od gustoće vjerojatnosti za $y = 0$, to je ova prva "uža i viša". Na gornjoj desnoj slici prikazane su zajedničke gustoće vjerojatnosti $p(x, y = 1)$ i $p(x, y = 0)$ (to je funkcija dviju varijabli, ali ju prikazujemo odvojeno, kao dvije krivulje, za $y = 1$ i $y = 0$). Te se gustoće vjerojatnosti dobivaju iz $p(x|y)$ množenjem sa $P(y)$, pa je dakle razlika između ove slike i one prethodne u tome što su gustoće vjerojatnosti pomnožene s vrijednošću apriorne vjerojatnosti za $y = 1$ odnosno $y = 0$. Budući da je $P(y = 0) > P(y = 1)$, to je ljubičasta krivulja sada puno viša od plave. Na donjoj lijevoj slici prikazana je marginalna gustoća vjerojatnosti $p(x)$, koja se dobiva marginalizacijom zajedničke vjerojatnosti po oznaci y , tj. $p(x) = p(x, y = 0) + p(x, y = 1)$, što odgovara zbrajanju plave i ljubičaste krivulje. Konačno, na donjoj desnoj slici prikazane su aposteriorne gustoće vjerojatnosti $p(y = 1|x)$ i $p(y = 0|x)$. One su, prema Bayesovom teoremu, dobivene dijeljenjem zajedničke gustoće vjerojatnosti $p(x, y)$ (slika gore desno) s gustoćom vjerojatnosti primjera $p(x)$ (donja lijeva slika). Primijetite da je zbroj $p(y = 0|x) + p(y|x)$ nužno jednak jedinici. Npr., za primjer $x = -5$ vjerojatnost klasifikacije u klasu $y = 1$ iznosi $p(y = 1|x = 0.5) \approx 0.8$, a vjerojatnost klasifikacije u klasu $y = 0$ iznosi $p(y = 0|x = 0.5) \approx 0.2$. Također primijetite da je granica između klasa mjesto u ulaznom prostoru (ovdje jednodimenzionalnom) gdje $p(y = 1|x) = p(y = 0|x) = 0.5$. To je u ovom slučaju na dva mesta (jer izglednost za drugu klasu ima veću varijancu od izglednosti za prvu klasu, pa se Gaussove krivulje sijeku na dva mesta; i slici gore lijevo to se ne vidi najbolje).

4.2 Multivariatni Gaussov Bayesov klasifikator

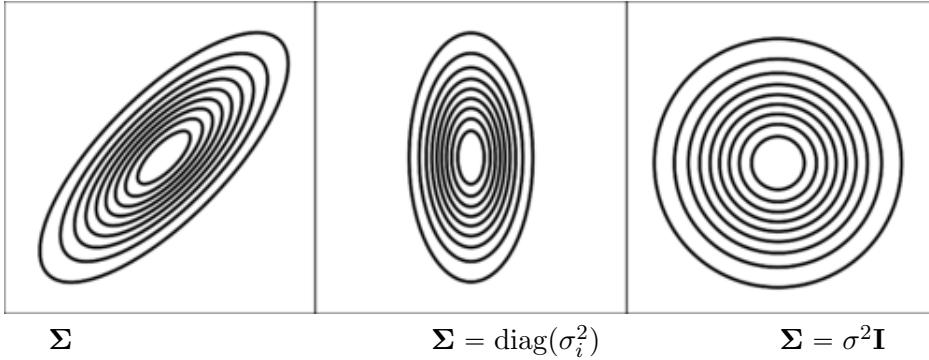
Fokusirajmo se sada na realističan slučaj: primjer \mathbf{x} je vektor realnih brojeva. Tada izglednost svake klase j modeliramo **multivarijatnom Gaussovom distribucijom**:

$$p(\mathbf{x}|y = j) = \frac{1}{(2\pi)^{n/2} |\Sigma_j|^{1/2}} \exp \left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_j)^T \Sigma_j^{-1} (\mathbf{x} - \boldsymbol{\mu}_j) \right)$$

gdje je Σ_j matrica kovarijacije, a μ_j je vektor srednjih vrijednosti za klasu j . Nakon logaritmanja, model za svaku klasu j pojedinačno je:

$$\begin{aligned} h_j(\mathbf{x}) &= \ln p(\mathbf{x}|y=j) + \ln P(y=j) \\ &= -\frac{n}{2} \ln 2\pi - \frac{1}{2} \ln |\Sigma_j| - \frac{1}{2} (\mathbf{x} - \mu_j)^T \Sigma_j^{-1} (\mathbf{x} - \mu_j) + \ln P(y=j) \\ &\Rightarrow -\frac{1}{2} \ln |\Sigma_j| - \frac{1}{2} (\mathbf{x} - \mu_j)^T \Sigma_j^{-1} (\mathbf{x} - \mu_j) + \ln P(y=j) \end{aligned}$$

Interpretacija za μ i Σ analogna je kao i za jednodimenzionalni slučaj: μ_j je prototipna vrijednost primjera u klasi j , dok je Σ_j količina šuma i korelacija između izvora šuma unutar klase j . Ovisno o tome kako definiramo kovarijacijsku matricu, možemo na različite načine modelirati korelaciju između izvora šuma, npr. (slika koju smo već bili diskutirali prošli put):



Koliko ovaj model ukupno ima parametara? Odgovor je: $\frac{n}{2}(n+1)K + K \cdot n + K - 1 \Rightarrow \mathcal{O}(n^2)$ (primijetite da je kovarijacijska matrica simetrična, pa je potrebno pohraniti samo dijagonalu i jedan trokut matrice). Ovdje, dakle, imamo **kvadratnu ovisnost** parametara o broju značajki (zbog kovarijacijske matrice). Procijeniti tolike parametre bi mogao biti problem, pogotovo onda kada je n velik a N malen.

Za procjenu parametara opet možemo koristiti MLE:

$$\begin{aligned} \hat{\mu}_j &= \frac{1}{N_j} \sum_{i=1}^{N_j} \mathbf{1}\{y^{(i)} = j\} \mathbf{x}^{(i)} \\ \hat{\Sigma}_j &= \frac{1}{N_j} \sum_{i=1}^{N_j} \mathbf{1}\{y^{(i)} = j\} (\mathbf{x}^{(i)} - \hat{\mu}_j)(\mathbf{x}^{(i)} - \hat{\mu}_j)^T \\ \hat{\mu}_j &= \frac{1}{N} \sum_{j=1}^J \mathbf{1}\{y^{(i)} = j\} = \frac{N_j}{N} \end{aligned}$$

Što mislite, je li ovo **linearan ili nelinearan model**? Tj., je li granica između klasa linearna (hiperravnina) ili nelinearna (hiperpovršina)? I zašto je to uopće bitno? Bitno je, jer nelinearnost granice direktno upućuje na složenost (kapacitet) modela. Ako model ima linearnu granicu, onda je manje složenosti, i moguće prejednostaviti za probleme koji iziskuju nelinearnu granicu između klasa. Da bismo utvrdili je li granica linearna ili ne, najbolje je da je izračunamo, odnosno izrazimo kao jednadžbu. Granica između klasa $y = 1$ i $y = 0$ jesu točke (odnosno hiperpovršina) za koje:

$$h_1(\mathbf{x}) = h_0(\mathbf{x})$$

tj. točke za koje vrijedi:

$$h_1(\mathbf{x}) - h_0(\mathbf{x}) = 0$$

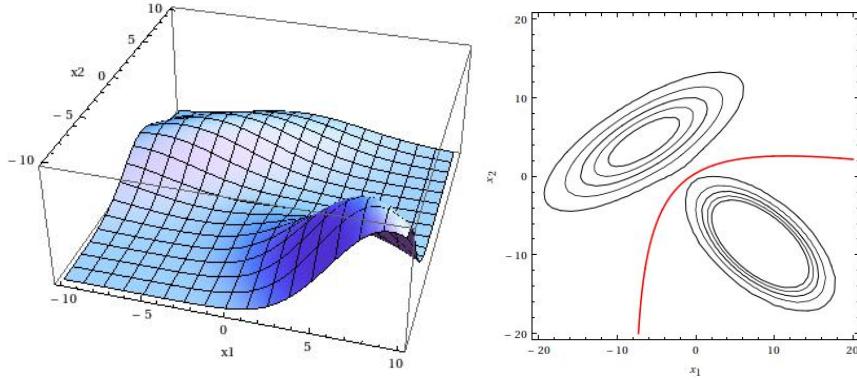
Model za klasu j definirali smo kao (ekspandiramo kvadrat):

$$\begin{aligned} h(\mathbf{x})_j &= -\frac{1}{2} \ln |\Sigma_j| - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_j)^T \Sigma_j^{-1} (\mathbf{x} - \boldsymbol{\mu}_j) + \ln P(y = j) \\ &= -\frac{1}{2} \ln |\Sigma_j| - \frac{1}{2} (\mathbf{x}^T \Sigma_j^{-1} \mathbf{x} - 2\mathbf{x}^T \Sigma_j^{-1} \boldsymbol{\mu}_j + \boldsymbol{\mu}_j^T \Sigma_j^{-1} \boldsymbol{\mu}_j) + \ln P(y = j) \end{aligned}$$

Granica između dva modela, za klase $y = 1$ i $y = 0$, bila bi onda:

$$\begin{aligned} h_{10}(\mathbf{x}) &= h_1(\mathbf{x}) - h_0(\mathbf{x}) \\ &= -\frac{1}{2} \ln |\Sigma_1| - \frac{1}{2} (\mathbf{x}^T \Sigma_1^{-1} \mathbf{x} - 2\mathbf{x}^T \Sigma_1^{-1} \boldsymbol{\mu}_1 + \boldsymbol{\mu}_1^T \Sigma_1^{-1} \boldsymbol{\mu}_1) + \ln P(y = 1) \\ &\quad - \left(-\frac{1}{2} \ln |\Sigma_0| - \frac{1}{2} (\mathbf{x}^T \Sigma_0^{-1} \mathbf{x} - 2\mathbf{x}^T \Sigma_0^{-1} \boldsymbol{\mu}_0 + \boldsymbol{\mu}_0^T \Sigma_0^{-1} \boldsymbol{\mu}_0) + \ln P(y = 0) \right) \\ &= \dots \mathbf{x}^T (\Sigma_1^{-1} - \Sigma_0^{-1}) \mathbf{x} \dots \end{aligned}$$

Vidimo da će granica biti **nelinearna** (točnije: kvadratna, tj. opisana parabolom) jer postoji član koji kvadratno ovisi o \mathbf{x} . To izgleda ovako: 10



Ljeva slika prikazuje gustoće dviju izglednosti klase. Desna slika prikazuje to isto, ali pomoću izokonture gustoće vjerojatnosti, i prikazuje odgovarajuću granicu između dviju klasa (crvena parabola). To su točke za koje $h_1(\mathbf{x}) = h_0(\mathbf{x})$.

Ovo je lijepo, međutim problem bi moglo biti to što kvadratni model ima puno parametara: $\mathcal{O}(n^2)$. To bi onda značilo da se lako može dogoditi da model bude prenaučen. Zato ćemo sada razmotriti neka pojednostavljenja modela, koja će dovesti do **linearnosti**, a time i do manjeg broja parametara. Kako možemo pojednostaviti model? Moramo uvesti dodatne **induktivne pretpostavke**. Moguća su razna pojednostavljenja ovog modela, a mi ćemo pokazati tri inkrementalna pojednostavljenja. Dakle, krenuvši od ovog modela, uvodit ćemo dodatne pretpostavke i izvesti tri sve jednostavnija modela.

5 Varijante Gaussovog Bayesovog klasifikatora

Prvo pojednostavljenje koje ćemo napraviti jest da pretpostavimo da sve klase imaju identičnu kovarijacijsku matricu. Takvu kovarijacijsku matricu zovemo **dijeljena (vezana) kovarijacijska matrica** (engl. *shared (tied) covariance matrix*). Kod procjene, tu dijeljenu matricu računamo kao težinski prosjek pojedinačnih kovarijacijskih matrica za pojedinačne klase: 11

$$\hat{\Sigma} = \sum_j \hat{\mu}_j \hat{\Sigma}_j$$

gdje je $\hat{\mu}_j$ apriorna vjerojatnost klase j (odnosno parametar multinulijeve distribucije za varijablu y). Ideja je da klase čija je apriorna vjerojatnost manja manje doprinose dijeljenoj kovarijacijskoj matrici.

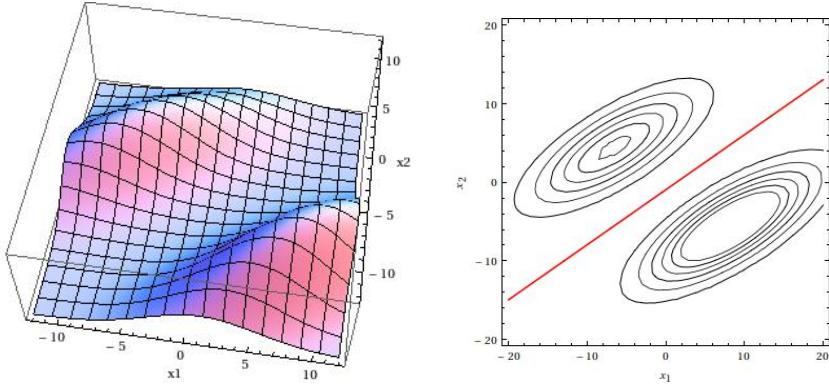
Uz dijeljenu kovarijacijsku matricu, model za klasu j izgleda ovako:

$$\begin{aligned} h_j(\mathbf{x}) &= \ln p(\mathbf{x}|y=j) + \ln P(y=j) \\ &= -\frac{n}{2} \ln 2\pi - \frac{1}{2} \ln |\Sigma| - \frac{1}{2} (\mathbf{x}^T \Sigma^{-1} \mathbf{x} - 2\mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu}_j + \boldsymbol{\mu}_j^T \Sigma^{-1} \boldsymbol{\mu}_j) + \ln P(y=j) \end{aligned}$$

Prva dva člana smo izbacili jer su konstante, tj. isti su za sve klase, pa nikako ne utječu na klasifikacijsku odluku. Granica između dviju klasa sada je:

$$\begin{aligned} h_{10}(\mathbf{x}) &= h_1(\mathbf{x}) - h_0(\mathbf{x}) \\ &= -\frac{1}{2} \cancel{\mathbf{x}^T \Sigma^{-1} \mathbf{x}} + \mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu}_1 - \frac{1}{2} \boldsymbol{\mu}_1^T \Sigma^{-1} \boldsymbol{\mu}_1 + \ln P(y=1) \\ &\quad - \frac{1}{2} \cancel{\mathbf{x}^T \Sigma^{-1} \mathbf{x}} - \mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu}_0 + \frac{1}{2} \boldsymbol{\mu}_0^T \Sigma^{-1} \boldsymbol{\mu}_0 - \ln P(y=0) \\ &= \mathbf{x}^T \underbrace{\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0}_{\mathbf{w}} - \underbrace{\frac{1}{2} \boldsymbol{\mu}_1^T \Sigma^{-1} \boldsymbol{\mu}_1 + \frac{1}{2} \boldsymbol{\mu}_0^T \Sigma^{-1} \boldsymbol{\mu}_0 + \ln \frac{P(y=1)}{P(y=0)}}_{w_0} \\ &= \mathbf{w}^T \mathbf{x} + w_0 \end{aligned}$$

ovdje smo samo vektorom \mathbf{w} zamijenili izraz $\Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$, koji je n -dimenzijski vektor parametara, te smo skalarom w_0 zamijenili preostali dio izraza, koji je doista skalar. Iz toga je postalo jasno da se granica između klase može opisati **linearnim modelom** $\mathbf{w}^T \mathbf{x} + w_0$. Dakle, zaključujemo da, ako je kovarijacijska matrica dijeljena između klasa, onda je granica između klasa linearna. Pogledajmo kako bi to izgledalo u dvodimenzijskome ulaznom prostoru:



Izokonture izglednosti klase su elipse s istim nagibom. Zbog toga je granica između klasa linearna.

Koliko ovaj model ima parametara? Odogovor je: $\frac{n}{2}(n+1) + nK + K - 1$. Imamo, dakle, K puta manje parametara nego kod modela s nedijeljenom kovarijacijskom matricom. Nažalost, to je i dalje $\mathcal{O}(n^2)$. Je li nam se to isplatilo? Ovisi. Nekad će se možda isplatiti: ako su izvori šumova u svim klasama slični, a imamo puno klasa, onda je ovakav model možda bolji (bolje generalizira).

5.1 Drugo pojednostavljenje

Sada ćemo napraviti dodatno pojednostavljenje: osim što ćemo prepostaviti da je kovarijacijska matrica dijeljena, prepostaviti ćemo i da je **dijagonalna**:

$$\Sigma = \text{diag}(\sigma_i^2)$$

To znači da su kovarijacije između značajki jednake nuli, odnosno da nema linearne zavisnosti između značajki. (Pazite to ne znači nužno da nema nikakvih zavisnosti! Sjetite se: ako je

kovarijacija jednaka nuli, to znači da nema linearne zavisnosti, ali može biti da ima nelinearne zavisnosti.) Ako je kovarijacijska matrica dijagonalna, onda je lako izračunati njezinu determinantu i inverz:

$$|\Sigma| = \prod_i \sigma_i^2$$

$$\Sigma^{-1} = \text{diag}(1/\sigma_i^2)$$

Uvrstimo to u izraz za izglednost klase:

$$\begin{aligned} p(\mathbf{x}|y=j) &= \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_j)\right) \\ &= \frac{1}{(2\pi)^{n/2} \prod_{i=1}^n \sigma_i} \exp\left(-\frac{1}{2} \sum_{i=1}^n \left(\frac{x_i - \mu_{ij}}{\sigma_i}\right)^2\right) \\ &= \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma_i} \prod_{i=1}^n \exp\left(-\frac{1}{2} \left(\frac{x_i - \mu_{ij}}{\sigma_i}\right)^2\right) \\ &= \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left\{-\frac{1}{2} \left(\frac{x_i - \mu_{ij}}{\sigma_i}\right)^2\right\} \\ &= \prod_{i=1}^n \mathcal{N}(\mu_{ij}, \sigma_i^2) \end{aligned}$$

Što smo dobili? Dobili smo umnožak univarijatnih Gaussovih distribucija, po jednu za svaku značajku:

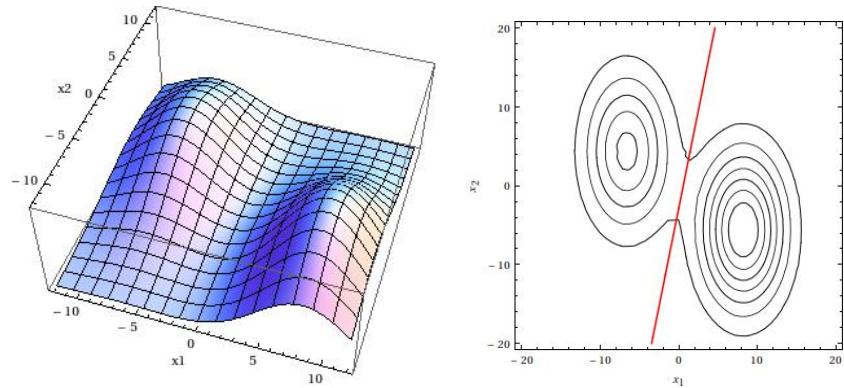
$$p(\mathbf{x}|y) = \prod_{i=1}^n p(x_i|y)$$

Ovo zapravo znači da su značajke x_i međusobno **uvjetno nezavisne** uz zadanu klasu y . Bayesov klasifikator s takvom pretpostavkom naziva se **naivan Bayesov klasifikator**. O tome ćemo više drugi put. Izveli smo, dakle, naivan Bayesov klasifikator za kontinuirane značajke. Model tog klasifikatora je:

$$\begin{aligned} h_j(\mathbf{x}) &= \ln p(\mathbf{x}|y=j) + \ln P(y=j) \\ &= \ln \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{1}{2} \left(\frac{x_i - \mu_{ij}}{\sigma_i}\right)^2\right) + \ln P(y=j) \\ &= \sum_{i=1}^n \ln \frac{1}{\sqrt{2\pi}\sigma_i} + \sum_{i=1}^n \left(-\frac{1}{2} \left(\frac{x_i - \mu_{ij}}{\sigma_i}\right)^2\right) + \ln P(y=j) \\ &\Rightarrow -\frac{1}{2} \sum_{i=1}^n \left(\frac{x_i - \mu_{ij}}{\sigma_i}\right)^2 + \ln P(y=j) \end{aligned}$$

Primijetimo da će predikcija modela za klasu j biti proporcionalna normiranoj euklidskoj udaljenosti između primjera \mathbf{x} i vektora srednjih vrijednosti $\boldsymbol{\mu}_j$.

Kako ovdje izgleda granica između klasa? Granica je i dalje linearna, čim je matrica kovarijacijske dijeljena. To se malo slabije vidi iz gornjeg izraza za model, jer se u izrazu pojavljuje x_i , koji kvadriramo, međutim ti kvadratni članovi se poništavaju jer su identični za svaki par klasa (za isti ulaz). U dvodimenzionskom ulaznom prostoru to izgleda ovako (izokonture su elipse poravnate s osima):



Ovaj model ima $n + n \cdot K + K - 1$ parametara, što je $\mathcal{O}(n)$. Drugim riječima, napokon smo dobili linearnu ovisnost broja parametara o broju značajki.

5.2 Treće pojednostavljenje

Konačno, treće pojednostavljenje jest prepostaviti dijeljenu i **izotropnu kovarijacijsku matricu**:

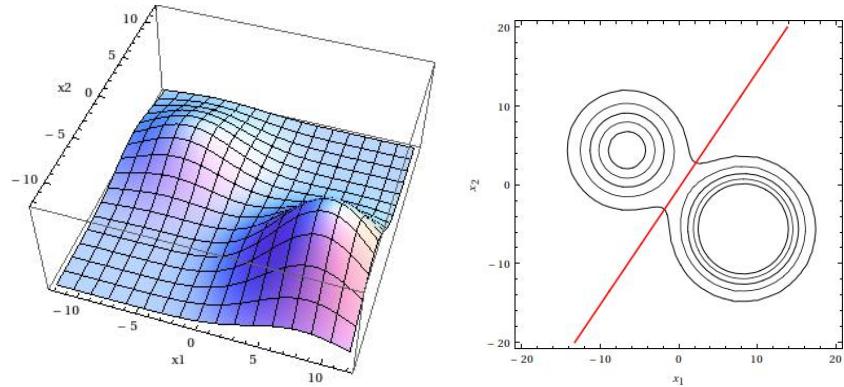
$$\Sigma = \sigma^2 \mathbf{I}$$

Dakle, prepostavljamo da je kovarijacijska matrica za sve klase identična, da je dijagonalna (značajke nisu linearno zavisne) i da su varijance za svaku značajku identične, što znači da prepostavljamo da je raspon svih značajki isti. Ima li to smisla? Da, ako smo skalirali značajke izvan modela (kao dio predobrade podataka). Inače baš i ne.

Model je sada:

$$h_j(\mathbf{x}) = -\frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu_{ij})^2 + \ln P(y = j)$$

U dvodimenzionskom ulaznom prostoru to izgleda ovako (izokonture su kružnice):



Koliko ovaj model ima parametara? Odgovor je: $1 + Kn + K - 1$, dakle broj parametara linearan je u broju značajki. (Usput, primijetite da ne možemo odbaciti parametar σ^2 , premda je identičan za svaku klasu, jer apriorne vjerojatnosti $P(y = j)$ nisu identične za svaku klasu.)

Sada se postavlja pitanje: od svih ovih modela koje smo izveli, koji bismo odabrali? Model s punom kovarijacijskom matricom ili jedno od tri pojednostavljenja? Ili neka druga pojednostavljenja, npr.:

Pretpostavka na kov. matricu	Kov. matrica	Broj parametara
Različite, hiperelipsoidi	Σ_j	$Kn(n+1)/2 + Kn$
Dijeljena, hiperelipsoidi	Σ	$n(n+1)/2 + Kn$
Različite, poravnati hiperelipsoidi	$\Sigma_j = \text{diag}(\sigma_{i,j}^2)$	$2Kn$
Dijeljena, poravnati hiperelipsoidi	$\Sigma = \text{diag}(\sigma_i^2)$	$n + Kn$
Različite, hipersfere	$\Sigma_j = \sigma_j^2 \mathbf{I}$	$K + Kn$
Dijeljena, hipersfere	$\Sigma = \sigma^2 \mathbf{I}$	$1 + Kn$

Kao i uvijek: najjednostavniji način jest napraviti **unakrsnu provjeru**: ispitati svaku varijantu modela na izdvojenom skupu označenih primjera te odabratи onu varijantu koja ima najmanju ispitnu pogrešku. Ta varijanta očekivano najbolje generalizira na neviđene primjere.

Sažetak

- Osnovna pravila teorije vjerojatnosti su **pravilo umnoška** i **pravilo zbroja**, pomoću kojih smo izveli **Bayesovo pravilo**
- Bayesov klasifikator pomoću Bayesovog pravila izračunava **aposteriornu vjerojatnost** označake primjera
- Bayesov klasifikator je **generativan** i **parametarski** model
- Generativni modeli opisuju nastanak podataka i omogućuju uključivanje **pozadinskog znanja** i **interpretabilnost**
- **Gaussov Bayesov klasifikator** izglednost klase modelira pomoću multivarijatnog Gaussa i taj model koristimo za kontinuirane značajke
- Uvođenjem različitih pretpostavki na **kovarijacijsku matricu** dobivamo **jednostavnije** varijante Gaussovog Bayesovog klasifikatora

Bilješke

- [1] Kao što smo vidjeli prošli put, ako je $p(x|y)$ općenito neka vjerojatnost od x za fiksirani y , onda je to ujedno i izglednost od y za fiksirani x . Zato gustoću vjerojatnosti $p(\mathbf{x}|y)$ zovemo **izglednost klase** (ili označke) y . Naravno, to je ujedno i vjerojatnost od \mathbf{x} za danu klasu (oznaku) y .
- [2] Naravno, reći da dekompozicija nekog X na dijelove X_i nema smisla jer je kompozicija dijelova X_i jednaka X sama po sebi nema smisla. Dekompozicija je jedan od osnovnih alata znanosti, posebno računarstva. V. [https://en.wikipedia.org/wiki/Decomposition_\(computer_science\)](https://en.wikipedia.org/wiki/Decomposition_(computer_science)).
- [3] Ovdje postoji opasnost da **maximum a posteriori hipotezu (MAP)** pobrkamo s **procjeniteljem maksimum a posteriori (MAP)**. Zabuna je očekivana, jer se radi o potpuno istom mehanizmu: maksimiziramo aposteriornu vjerojatnost. Međutim, hipoteza MAP i procjenitelj MAP su ipak dvije različite stvari jer pripadaju različitim razinama. Procjenitelj MAP je način procjene parametara modela (npr. Bayesovog klasifikatora). Jednom kada smo procijenili parametre (tj. naučili model), možemo raditi klasifikaciju. Način na koji onda odlučujemo u koju klasu klasificirati primjer određuje nam hipoteza MAP. Dakle, prvo za treniranje modela koristimo procjenitelj MAP, a onda pomoću naučenog modela radimo klasifikaciju primjera pomoću hipoteze MAP. Također, ništa nas ne sprječava da parametre procijenimo nekom drugom metodom (npr. MLE), a onda za predikciju koristimo MAP hipotezu. Vrijedi i obrnuto: možemo koristiti procjenitelj MAP, ali za predikciju ne koristiti MAP hipotezu nego neko drugo pravilo odlučivanja. Međutim, tipično ipak koristimo MAP hipotezu, jer je ona optimalna u smislu **bayesovske teorije odlučivanja** (naime, hipoteza MAP minimizira vjerojatnost pogreške; v. poglavlje 4.1.1 u skripti).
- [4] **Generativni modeli** ponekad se u literaturi nazivaju **modeli zajedničke vjerojatnosti** (engl. *joint*

models), dok se probabilistički diskriminativni modeli nazivaju i **uvjetni modeli** (engl. *conditional models*), budući da modeliraju uvjetnu (aposteriornu) vjerojatnost.

- 5 **Teorem o nebesplatnom ručku** (engl. *no free lunch theorem*, *NFL*) su zapravo dva teorema: jedan za strojno učenje, a drugi za postupke optimizacije. Oba je predložio američki matematičar David Wolpert (drugi teroem u suautorstvu sa Williamom Macreadyjem) 1996. odnosno 1997. godine. NFL za strojno učenje opisan je u [Wolpert \(1996\)](#). Neformalni opis možete naći na <http://www.no-free-lunch.org/>. Lijep opis i dokaz možete naći u [Shalev-Shwartz and Ben-David \(2014\)](#) (poglavlje 5.1). Argument NFL-a u osnovi je povezan s Humovim problemom indukcije; <https://plato.stanford.edu/entries/induction-problem/>.
- 6 Rivalstvo između generativnih i diskriminativnih modela otvoreno je pitanje. Pogledajte, na primjer, ([Ng and Jordan, 2001](#)) i ([Xue and Titterington, 2008](#)).
- 7 Ovaj dio uglavnom prati poglavlja 5.4–5.6 iz ([Alpaydin, 2020](#)).
- 8 Primijetite da izglednost svake klase modeliramo jednom multivarijatnom Gaussovom gustoćom vjerojatnosti. To znači da su gustoće vjerojatnosti za svaku klasu unimodalnu. To zapravo znači da pretpostavljamo da je klasa homogena, u smislu da ima jedan prototipni primjer oko kojega se svi drugi primjeri rasipaju zbog šuma. Složeniji model bi bio onaj koji omogućio modeliranje nehomogenih klasa kao kombinaciju (superpoziciju) više multivarijatnih Gaussova gustoća vjerojatnosti. Nehomogene klase u stvarnosti nisu rijetkost. Npr., klasifikacija rukom pisanih znachenki ima nehomogenu klasu za znachenku 7, jer se ona može pisati na dva načina (s horizontalnom crticom ili bez nje, uglavnom ovisno o tome jeste li Europljanin ili Amerikanac). Bayesov klasifikator podrazumijeva da su klase homogene. Međutim, postoje modeli koji mogu modelirati nehomogene klase koje se sastoje od kombinacije Gaussova gustoća vjerojatnosti. To su **modeli Gaussove mješavine** (engl. *mixture of Gaussians, MoG*). O tim modelima pričat ćemo u kontekstu grupiranja podataka.
- 9 Prisjetimo se napomene u vezi kovarijacijske matrice Σ : ta matrica je uvijek **pozitivno semidefinitna**, što znači da $\mathbf{x}^T \Sigma \mathbf{x} \geq 0$, a isto tako $\Delta^2 = \mathbf{x}^T \Sigma^{-1} \mathbf{x} \geq 0$, tj. za Mahalanobisovu udaljenost vrijedi $\Delta \geq 0$ Ali, da bi PDF bila dobro definirana, Σ mora biti **pozitivno definitna**: $\Delta^2 = \mathbf{x}^T \Sigma^{-1} \mathbf{x} > 0$ za ne-nul vektor \mathbf{x} . Ako je Σ pozitivno definitna, onda je nesingularna: $|\Sigma| > 0$ i postoji Σ^{-1} . Ako Σ nije pozitivno definitna, uzroci su: $\text{Var}(x_i) = 0$ (beskorisna značajka) ili $\text{Cov}(x_i, x_j) = 1$ (multikolinearnost – redundantan par značajki). Također, imajte na umu da matrica može biti skoro linearna (što će biti indicirano njezinim visokim **kondicijskim brojem**).
- 10 Model **Gaussovog Bayesovog klasifikatora** s kvadratnom granicom istovjetan je modelu **kvadratne diskriminantne analize** (engl. *quadratic discriminant analysis, QDA*). Kvadratna diskriminantna analiza je počeoje **linearne diskriminantne analize** (engl. *linear discriminant analysis, LDA*), kod koje je granice između klasa linearna. Linearnu diskriminantnu analizu predložio je 1936. godine predložio Ronald Fisher, jedan od najvećih statističara. U nastavku ćemo uvesti pojednostavljenja Gaussovog Bayesovog klasifikatora, koja će nas dovesti do modela istovjetnog linearnej diskriminativnoj analizi.
- 11 U statistici, pretpostavka da je kovarijacijska matrica identična za sve klase naziva se **homoske-dastičnost** (homogenost varijance). Izračun kovarijacijske matrice kao težinske kombinacije kovarijacijski matrica pojedinačnih klasa naziva se **udružena varijanca** (engl. *pooled variance*).
- 12 Gaussov Bayesov klasifikator s dijeljenom kovarijacijskom matricom, kod kojega je granica između klasa linearna, istovjetan je ranije spomenutoj **linearnoj diskriminativnoj analizi (LDA)**. Linearu diskriminativnu analizu nemojte brkati s također ranije spomenutom **Latentnom Dirichleovom alokacijom (LDA)**. Kratice su iste, ali metode su posve različite. Jedina zajedničkost im je da to što su obje metode generativne.
- 13 Preciznije, ovo znači da značajke **nisu linearne zavisne** za zadalu klasu y . Međutim, kao što smo već napomenuli, između značajki može postojati nelinearna zavisnost, premda nema linearne zavisnosti. Primijetite da i u tom slučaju – kada kovarijacija jest jednaka nuli ali varijable su ipak nelinearno zavisne – dobivamo naivan Bayesov model. To je zato što u multivarijatnoj Gaussovoj distribuciji kroz kovarijacijsku možemo modelirati samo linearu zavisnost između varijabli – i to samo između parova varijabli. Ako su varijable nelinearno zavisne, to ne možemo modelirati. Mogli bismo reći da

je kod kontinuiranog Bayesa dio “tereta naivnosti” preuzeo i sama Gaussova distribucija, jer njenim odabirom implicitno prepostavljam da značajke mogu biti samo linearne zavisne.

Literatura

- E. Alpaydin. *Introduction to machine learning*. MIT press, 2020.
- A. Ng and M. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 14:841–848, 2001.
- S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.
- D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.
- J.-H. Xue and D. M. Titterington. Comment on “on discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes”. *Neural processing letters*, 28(3):169, 2008.

16. Bayesov klasifikator II

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v3.2

Prošli puta pričali smo o Bayesovom klasifikatoru, koji modelira vjerojatnost da primjer pripada nekoj klasi i to čini pomoću Bayesovog pravila. Zapravo, rekli smo da Bayesov klasifikator modelira **zajedničku vjerojatnost** primjera i označaka, i da se takvi modeli nazivaju **generativni modeli**. Zatim smo govorili o Gaussovom Bayesovom klasifikatoru, odnosno Bayesovom klasifikatoru kod kojeg su značajke kontinuirane i izglednosti klase modelirane su Gaussovim gustoćama vjerojatnosti. Na kraju smo razmotrili nekoliko varijanti tog klasifikatora, koje se razlikuju u pretpostavkama o linearnoj zavisnosti značajki, kodirane matricom kovarijacije.

Danas nastavljamo s Bayesovim klasifikatorom. Zadržat ćemo se još malo na **Gaussovom Bayesovom klasifikatoru** i usporediti ga s **logističkom regresijom**. Ta usporedba otkrit će nam da su tva dva modela zapravo povezana, čime ćemo uspostaviti izravnu vezu između generativnog i diskriminativnog strojnog učenja. Nakon toga razmotrit ćemo **naivan Bayesov klasifikator** za diskrete značajke, koji pretpostavlja uvjetnu nezavisnost između značajki unutar svake klase. Na kraju ćemo razmotriti **polunaivan Bayesov klasifikator**, koji relaksira pretpostavku o nezavisnosti, čime dobivamo složenije modele.

1

1 Bayesov klasifikator vs. logistička regresija

Na ovom predmetu volimo uočavati veze između naoko nepovezanih stvari. Zašto je to važno? Zato što su bitna načela, a načela nadilaze pojedinačne algoritme. Ako možete uočavati te veze, onda možete vidjeti što su zajedničkosti i različitosti, a to znači da vidite suštinu. Mi smo tako već uočavali razne veze između modela i algoritama. Zadnje što smo uočili jest da postoji veza između MLE-a i minimizacije pogreške, dakle između procjene parametara kao tipične tehnikе za učenje generativnih modela i minimizacije pogreške kao tipične tehnikе za učenje poopćenih linearnih modela.

Sada ćemo otkriti još jednu takvu zanimljivu povezanost. Pokazat ćemo da je logistička regresija zapravo kontinuirani Bayesov klasifikator, ili, obrnuto, da je kontinuirani Bayesov klasifikator zapravo poopćeni linearni model. Drugim riječima, naći ćemo točku na kojoj se spajaju dva svijeta iz strojnog učenja: generativni i diskriminativni.

Nakon ovog pretencioznog uvoda, pogledajmo o čemu se radi. Prisjetimo se najprije modela **logističke regresije**:

$$h(\mathbf{x}; \mathbf{w}) = P(y = 1 | \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$$

Ideja je, da krenuvši od kontinuiranog Bayesovog klasifikatora, pokušamo doći do modela logističke regresije. Ako to uspijemo, znači da su ovi modeli zapravo identični. Očito, budući da oba modela izražavaju aposteriornu vjerojatnost $P(y|\mathbf{x})$, ta vjerojatnost će nam biti pivotna točka usporedbe.

Krenimo od toga da napišemo kako Bayesov klasifikator izračunava aposteriornu vjerojatnost. Razmotrimo slučaj dvije klase, $y = 1$ i $y = 0$, budući da želimo uspostaviti vezu s binarnom logističkom regresijom. Aposteriorna vjerojatnost koju izračunava Bayesov klasifikator je sljedeća:

$$P(y = 1 | \mathbf{x}) = \frac{p(\mathbf{x}|y = 1)P(y = 1)}{p(\mathbf{x}|y = 1)P(y = 1) + p(\mathbf{x}|y = 0)P(y = 0)}$$

Iz ovoga nekako želimo doći do logističke funkcije $1/(1 + \exp(-\alpha))$, jer bi to onda uspostavilo vezu prema logističkoj regresiji. S tim ciljem sada ćemo u nazivniku izlučiti prvi pribrojnik i pokratiti ga s brojnikom. Nadalje, u nazivniku želimo imati funkciju \exp , pa ćemo u tu svrhu na drugi pribrojnik u nazivniku primijeniti funkciju $\exp \circ \ln$ (kompoziciju funkcija \exp i \ln), koja je funkcija identiteta. Tako dobivamo:

$$\begin{aligned} P(y = 1|\mathbf{x}) &= \frac{1}{1 + \frac{p(\mathbf{x}|y=0)P(y=0)}{p(\mathbf{x}|y=1)P(y=1)}} \\ &= \frac{1}{1 + \exp\left(\ln \frac{p(\mathbf{x}|y=0)P(y=0)}{p(\mathbf{x}|y=1)P(y=1)}\right)} \\ &= \frac{1}{1 + \exp(-\alpha)} = \sigma(\alpha) \end{aligned}$$

gdje onda definiramo:

$$\alpha = \ln \frac{p(\mathbf{x}|y = 1)P(y = 1)}{p(\mathbf{x}|y = 0)P(y = 0)} = \ln \frac{P(y = 1|\mathbf{x})}{P(y = 0|\mathbf{x})} = \ln \frac{P(y = 1|\mathbf{x})}{1 - P(y = 1|\mathbf{x})}$$

Dobili smo definiciju modela koja oblikom odgovara onoj za logističku regresiju. Međutim, da bi korespondencija bila potpuna, mora vrijediti:

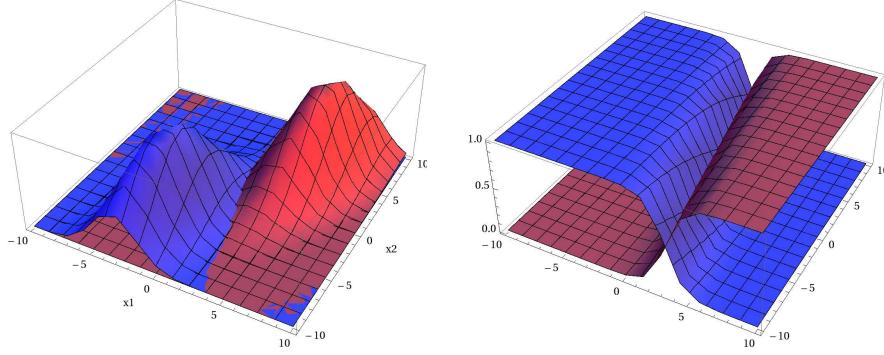
$$\alpha = \ln \frac{p(\mathbf{x}|y = 1)P(y = 1)}{p(\mathbf{x}|y = 0)P(y = 0)} = \underbrace{\ln p(\mathbf{x}|y = 1)P(y = 1)}_{h_1(\mathbf{x})} - \underbrace{\ln p(\mathbf{x}|y = 0)P(y = 0)}_{h_0(\mathbf{x})} = \mathbf{w}^T \mathbf{x} + w_0$$

tj. razlika $h_1(\mathbf{x}) - h_0(\mathbf{x})$ mora biti linearna funkcija od \mathbf{x} . Drugim riječima, granica između klase $y = 1$ i $y = 0$ mora biti linearna!

Sada se prisjetimo što smo naučili prošli put: tada smo utvrdili da uz će, uz određenu pretpostavku na kovarijacijsku matricu Σ za gustoću vjerojatnosti izglednosti klase $p(\mathbf{x}|y)$, Gaussov Bayesov klasifikator dati linearu granicu, tj. da će iz kvadratnog modela degenerirati u linearan model. Koja je to bila pretpostavka? Odgovor je: pretpostavka **dijeljene kovarijacijske matrice**. Naime, prisjetimo se, ako je kovarijacijska matrica dijeljena, granica između dviju klasa je:

$$\begin{aligned} h_{10}(\mathbf{x}) &= h_1(\mathbf{x}) - h_0(\mathbf{x}) \\ &= \mathbf{x}^T \underbrace{\Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)}_{\mathbf{w}} - \underbrace{\frac{1}{2}\boldsymbol{\mu}_1^T \Sigma^{-1} \boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_0^T \Sigma^{-1} \boldsymbol{\mu}_0 + \ln \frac{P(y = 1)}{P(y = 0)}}_{w_0} \\ &= \mathbf{w}^T \mathbf{x} + w_0 \end{aligned}$$

Krenuli smo od Gaussovog Bayesovog klasifikatora za dvije klase te smo, uz pretpostavku dijeljene kovarijacijske matrice stigli do logističke regresije. Time smo pokazali da je Gaussov Bayesov klasifikator s dijeljenom kovarijacijskom matricom zapravo ista stvar kao i (neregularizirana) binarna logistička regresija. Što to zapravo znači? Znači da, ako radimo binarnu klasifikaciju, istu granicu možemo dobiti (neregulariziranom) logističkom regresijom ili Gaussovim Bayesovim klasifikatorom! To lijepo ilustrira sljedeći primjer u dvodimenzijском ulaznom prostoru:



Ljeva slika prikazuje zajedničku gustoću vjerojatnosti, $p(\mathbf{x}, y)$, za dvije klase (plava i crvena), dok desna slika prikazuje aposteriornu vjerojatnost, $p(y|\mathbf{x})$, za iste te dvije klase. Aposteriorna se vjerojatnost može izračunati iz zajedničke vjerojatnosti pomoću Bayesovog pravila, i tako to radi Bayesov klasifikator. Međutim, logistička regresija izravno izračunava aposteriornu vjerojatnost. U konačnici, međutim, granica između ovih dviju klasa je identična.

Ovime smo povezali generativni model (Gaussov Bayesov klasifikator) s njemu odgovarajućim diskriminativnim modelom (logistička regresija). U strojnem učenju ima više takvih **generativno-diskriminativnih parova** modela. Gaussov Bayesov klasifikator i logistička regresija jedan su primjer. Drugi primjeri generativno-diskriminativnog para jesu **skriveni Markovljev model** (koji ćemo spomenuti idući put) i **model uvjetnih slučajnih polja** (engl. *conditional random field*), koji nećemo raditi.

Osim povezivanja generativnog i diskriminativnog, ovime smo zapravo napokon dobili i potpuno vjerodostojano opravdanje za probabilističku interpretaciju izlaza logističke regresije. Sjetite se da smo, kada smo pričali o logističkoj regresiji, rekli da koristimo sigmoidnu funkciju kako bi izlaz modela bio ograničen na interval $(0, 1)$, i kako bismo onda tu vrijednost mogli tumačiti kao vjerojatnost oznake $y = 1$ za primjer \mathbf{x} . Međutim, nije bilo jasno uz koje zapravo pretpostavke možemo tako tumačiti izlaz modela (samo zato što je neki broj u intervalu $[0, 1]$ ne znači automatski da odgovara vjerojatnosti nekog događaja). No, sada je to jasno: ako pretpostavimo da su (1) primjeri iz obiju klasa normalno distribuirani oko srednje, prototipne vrijednosti (tj. izglednost je Gaussova gustoća vjerojatnosti) i (2) da postoji linearna zavisnost između izvora šuma koja je u obje klase identična (tj. kovarijacijska matrica je dijeljena), onda izlaz logističke regresije doista odgovara aposteriornoj vjerojatnosti oznake y za primjer \mathbf{x} . S druge strane, ako ove pretpostavke ne vrijede, onda nemamo teorijski model uz koji bi izlaz logističke regresije odgovarao aposteriornoj vjerojatnosti. Međutim, u praksi se time previše ne zamaramo, tj. izlaz logističke regresije tumačimo kao vjerojatnost neovisno o tome koliko podatci doista odgovaraju navedenim pretpostavkama. No, to također u praksi znači da, ako postoji veliko odstupanje od ovih pretpostavki (npr. kovarijacije su bitno različite u dvjema klasama), onda će naš model loše raditi (bit će podnaučen).

Da sažmemo: logistička regresija je **diskriminativni model** koji izravno modelira aposteriornu vjerojatnost (ovdje smo w_0 uključili u vektor \mathbf{w}):

$$P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$$

dok je Bayesov klasifikator njoj odgovarajući **generativni model** koji tu istu vjerojatnost modelira neizravno:

$$P(y = 1|\mathbf{x}) = \frac{p(\mathbf{x}|y = 1)P(y = 1)}{p(\mathbf{x}|y = 1)P(y = 1) + p(\mathbf{x}|y = 0)P(y = 0)}$$

Zadnje opažanje koje ćemo napraviti tiče se broja parametara modela. Koliko parametara imaju ovi modeli za n -dimenzijski ulazni prostor? Logistička regresija ima samo vektor \mathbf{w} kao parametre, pa dakle logistička regresija ima $n+1$ parametar. S druge strane, Bayesov klasifikator

ima $\frac{n}{2}(n+1) + 2n + 1$ parametara (kovarijacijsku matricu Σ , dva vektora srednjih vrijednosti μ_j te jedan parametar apriorne Bernoullijeve vjerojatnosti). Ovo ilustrira tipičnu situaciju: generativni modeli koji ostvaruju istu složenost (dakle isti oblik granice u ulaznom prostoru) općenito imaju više parametara od njihovog diskriminativnog para. Zbog toga generativni modeli općenito trebaju više primjera za učenje nego diskriminativni modeli, odnosno, zbog toga diskriminativni modeli općenito rade bolje od generativnih na istom skupu podataka.

Nakon ove nešto šire slike na strojno učenje, vratimo se opet na Bayesov klasifikator. Sada kada znamo kako radi Bayesov klasifikator za kontinuirane značajke, razmotrimo Bayesov klasifikator za diskretne značajke, krenuvši od **naivnog Bayesovog klasifikatora**.

2 Naivan Bayesov klasifikator

2.1 Model Naivnog Bayesovog klasifikatora

Prisjetimo se, općenit model Bayesovog klasifikatora je:

$$P(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)P(y)}{\sum_{y'} p(\mathbf{x}|y')P(y')} = \frac{p(x_1, \dots, x_n|y)P(y)}{\sum_{y'} p(x_1, \dots, x_n|y')P(y')}$$

Kod diskretnog klasifikatora, značajke su diskretne, dakle varijable x_k su **kategoričke (multinuličeve)** varijable (ili **Bernoullićeve**, ako imaju samo dvije moguće vrijednosti).

Sad je pitanje: kako ćemo točno modelirati izglednost klase $p(\mathbf{x}|y)$, ako znamo da su x_k kategoričke/Bernoullićeve varijable? To će zapravo biti jedina razlika u modelu u odnosu na Gaussov Bayesov klasifikator, kod kojega smo, prisjetite se, izglednosti klasa modelirali multivarijatnom Gaussovom distribucijom. Prva mogućnost koja bi nam mogla pasti na pamet jest da $p(\mathbf{x}|y)$ tretiramo kao **kategoričku razdiobu**, čije su vrijednosti sve moguće kombinacije pojedinačnih kategoričkih varijabli, tj. pojedinačnih značajki. Pogledajmo primjer.

► PRIMJER

Imamo tri značajke: prve dvije značajke imaju tri moguće vrijednosti, $K_1 = 3$ i $K_2 = 3$, dok je treća značajka binarna, $K_3 = 2$. Budući da su to kategoričke varijable, prikazat ćemo ih kao binarne vektore indikatorskih varijabli duljine 3, 3, odnosno 2. Te vektore konkaténiramo u jedan vektor, \mathbf{x} . Npr., vektor:

$$\mathbf{x} = (\underbrace{0, 1, 0}_{x_1}, \underbrace{0, 0, 1}_{x_2}, \underbrace{1, 0}_{x_3})$$

odgovara kombinaciji vrijednosti $x_1 = 1$ (od mogućih vrijednosti $\{0, 1, 2\}$), $x_2 = 2$ (od mogućih vrijednosti $\{0, 1, 2\}$) i $x_3 = 0$ (od mogućih vrijednosti $\{0, 1\}$). Vektor \mathbf{x} sada možemo tretirati kao jednu kategoričku varijablu, koja ima $3 \times 3 \times 2 = 18$ mogućih različitih vrijednosti. Tomu odgovarajuća kategorička distribucija ima onoliko parametara μ_k koliko ima različitih vrijednosti. Da bismo naučili takvu distribuciju, trebamo dakle procijeniti svih 18 parametara μ_k . Zapravo, dovoljno je da procijenimo njih 17, jer znamo da mora vrijediti $\sum \mu_k = 1$. Međutim, ovu procjenu moramo napraviti za svaku označku klase y . Ako imamo npr. $K = 3$ klase, onda moramo procijeniti ukupno $17 \times 3 = 51$ parametar. Te procjene za μ_k mogu se prikazati kao **tablica uvjetne vjerojatnosti** (engl. *conditional probability table; CPT*). U ovom konkretnom slučaju CPT bi izgledala ovako:

k	x_1	x_2	x_3	y	$\mu_k = p(\mathbf{x} y)$
1	0	0	0	0	...
2	0	0	1	0	...
3	0	1	0	0	...
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
17	2	2	0	0	...
18	0	0	0	1	...
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
34	2	2	1	1	...
35	0	0	0	2	...
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
51	2	2	1	2	...

gdje bismo umjesto “...” imali neke konkretnе vrijednosti, koje se za svaku klasu y zbrajaju u 1. Primijetite da smo zbog toga za svaku klasu uštedjeli jedan redak, znajući da se vjerojatnosti $p(\mathbf{x}|y)$ za svaku klasu y moraju zbrajati u 1.

Općenito, ako pojedinačne kategoričke varijable x_k imaju svaka K_k mogućih vrijednosti, ukupno imamo $\prod_{k=1}^n K_k$ različitih vrijednosti. Trebamo procijeniti te vrijednosti za svaku od K klasa. Ukupan broj parametara distribucije, odnosno broj redaka u tablici uvjetne vjerojatnosti, jednak je:

$$K \cdot \left(\prod_{k=1}^n K_k - 1 \right)$$

($K_k - 1$ jer se vjerojatnosti $P(\mathbf{x}|y)$ za svaki pojedini y moraju sumirati na jedinicu.)

Ovdje sad odmah vidimo i što je problem s ovakvim modeliranjem izglednosti klase. Zapravo, postoje dva problema. Prvi je problem očit: **velik broj parametara**. Naime, broj parametara raste eksponencijalno s brojem mogućih vrijednosti pojedinačnih značajki.

► PRIMJER

Neka su varijable x_k **binarne**. Npr., radimo klasifikaciju crno-bijelih slika, pa je svaka varijabla x_k jedan piksel. Imamo n značajki, tj. n piksela. Recimo da je riječ o binarnoj klasifikaciji, tj. imamo $K = 2$ klase. Koliko parametara moramo procijeniti za modeliranje distribucije $P(x_1, x_2, \dots, x_n|y)$, tj. koliko će redaka imati CPT?

k	x_1	\dots	x_{n-1}	x_n	y	$\mu_k = p(\mathbf{x} y)$
1	0	\dots	0	0	0	...
2	0	\dots	0	1	0	...
3	0	\dots	1	0	0	...
\vdots	\vdots	\dots	\vdots	\vdots	\vdots	\vdots
2^{n-1}	0	\dots	1	0	0	...
2^n	0	\dots	1	0	1	...
\vdots	\vdots	\dots	\vdots	\vdots	\vdots	\vdots
$2 \cdot (2^n - 1)$	0	\dots	1	0	1	...

Ukupno imamo $2 \cdot (2^n - 1)$ redaka. Drugim riječima, broj parametara **eksponencijalno** ($\mathcal{O}(2^n)$) ovisi o broju značajki. To je u stvarnosti potpuno neprihvatljivo! Npr., za klasifikaciju binarnih slika dimenzija 100×100 u dvije klase, trebalo bi nam 19998 parametara, što je neprihvatljiva složenost. Trebalo bi nam vrlo mnogo primjera da dobro naučimo takav model. (Zašto? Zato što bismo za svaku kombinaciju vrijednosti značajki x_1, \dots, x_n , tj. za svaki redak CPT-a, morali imati dovoljno primjera s točno takvom kombinacijom vrijednosti značajki, a da bismo mogli dovoljno dobro procijeniti parametar μ_k kategoričke distribucije.)

Dakle, prvi problem, ako $p(\mathbf{x}|y)$ modeliramo kao kategoričku distribuciju gdje jednostavno sve vektore pojedinačnih varijabli konkateniramo u jedan golemi vektor, jest da imamo previše parametara. Drugi je problem povezan s time, ali je zapravo fundamentalniji: takav model uopće **ne može generalizirati**. Zašto? Zato što ćemo vjerojatnosti moći procijeniti samo za one primjere koje smo vidjeli (koje imamo u skupu za učenje). Ukupna masa vjerojatnosti bit će raspodijeljena samo na te primjere. Svi ostali primjeri – a to su neviđeni primjeri – imat će $\mu_k = 0$. To znači da je vjerojatnost neviđenog primjera jednaka nuli. Ako pogledate Bayesovo pravilo, to znači da će nam i brojnik i nazivnik za takve primjere biti jednaki nuli, pa aposteriorna vjerojatnost uopće neće biti definirana. Očito, to je vrlo loša generalizacija na neviđene primjere!

4

Kako riješiti ovaj problem? Ako model ne može dobro generalizirati, to znači da trebamo pojačati **induktivnu pristranost**, tj. uvesti još prepostavki. Kod generativnih modela to možemo učiniti tako da **pojednostavimo** gustoću vjerojatnosti $p(\mathbf{x}|y)$ (odnosno vjerojatnost $P(\mathbf{x}|y)$, za slučaj diskretnih značajki). To ćemo ostvariti tako da tu vjerojatnost na prikidan način **faktoriziramo**.

Pogledajmo sada što znači faktorizirati vjerojatnost. Prošli smo puta uveli dva osnovna pravila teorije vjerojatnosti: pravilo zbroj i pravilo umnoška. Zatim smo upotrijebili ta dva pravila da bismo izveli Bayesovo pravilo. Još jedno pravilo koje možemo izvesti, i to jednostavnom primjenom pravila umnoška, jest **pravilo lanca** (engl. *chain rule*). Za zajedničku vjerojatnost sa tri varijable, to pravilo izgleda ovako:

$$P(x, y, z) = \underbrace{\underbrace{P(x)P(y|x)}_{P(x,y)}}_{P(x,y,z)} P(z|x, y)$$

Vidimo da zajedničku vjerojatnost od tri varijable možemo napisati kao umnožak triju vjerojatnosti. Umnožak se sastoji od marginalne vjerojatnosti jedne varijable (ovdje je to varijabla x) te uvjetnih vjerojatnosti za preostale dvije varijable, gdje u uvjetni dio dodajemo sve više varijabli. Pritom je redoslijed kojim to radimo proizvoljan, tj.:

$$P(x, y, z) = P(x)P(y|x)P(z|x, y) = P(z)P(y|z)P(x|y, z) = P(z)P(x|z)P(y|x, z) = \dots$$

(ima ukupno $3! = 6$ mogućnosti). Ovo je bio primjer s tri varijable, ali princip naravno vrijedi i općenito, za n -dimenzijski slučajan vektor. Općenito, pravilo lanca za zajedničku vjerojatnost od n varijabli je:

$$P(x_1, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \cdots P(x_n|x_1, \dots, x_{n-1}) = \prod_{k=1}^n P(x_k|x_1, \dots, x_{k-1})$$

Pravilo lanca nam je vrlo korisno kada zajedničku vjerojatnost sastavljenu od mnogo varijabli želimo dekomponirati na umnožak više jednostavnijih uvjetnih vjerojatnosti. Kada zajedničku vjerojatnost dekomponiramo na umnožak više vjerojatnosti, onda se pojedinačne vjerojatnosti zovu **faktori**, a “rastavljanje” zajedničke vjerojatnosti na faktore zove se, očekivano, **faktorizacija**.

Sada kada ovo znamo, primjenom pravila lanca, izglednost klase mogli bismo faktorizirati na sljedeći način:

$$P(x_1, \dots, x_n|y) = \prod_{k=1}^n P(x_k|x_1, \dots, x_{k-1}, y)$$

Ovo je identično gore navedenom pravilu lanca, samo što smo cijelu vjerojatnost uvjetovali varijablom y , tj. oznakom klase. To samo znači da u gornjoj jednakosti varijablu y jednostavno dodajemo u uvjetni dio vjerojatnosti i na lijevoj i na desnoj strani.

Sada znamo kako izglednost klase $P(x_1, \dots, x_n|y)$ raspisati kao umnožak više uvjetnih vjerojatnosti. Međutim, nažalost moramo utvrditi da ovime nismo napravili baš nikakvo pojednostavljenje: to je samo drugačiji način zapisa zajedničke vjerojatnosti za \mathbf{x} . Broj parametara je isti kao i ranije. Kako bismo pojednostavili model, moramo uvest dodatne pretpostavke o **uvjetnoj nezavisnosti varijabli**. Konkretno, uvest ćemo vrlo radikalnu pretpostavku:

$$P(x_k|x_1, \dots, x_{k-1}, y) = P(x_k|y)$$

tj. za svaki faktor prepostaviti ćemo da je uvjetovan samo oznakom y . Drugim riječima, to znači da smo pretpostavili da su značajke x_k za primjere unutar klase y međusobno nezavisne. Pogledat ćemo uskoro detaljnije što to točno znači. Međutim, već sada vidimo da se, uz ovu pretpostavku, izglednost faktorizira ovako:

$$P(x_1, \dots, x_n|y) = \prod_{k=1}^n P(x_k|x_1, \dots, x_{k-1}, y) = \prod_{k=1}^n P(x_k|y)$$

I to nas onda konačno dovodi do modela **naivnog Bayesovog klasifikatora** (engl. *naïve Bayes classifier*):

$$h(x_1, \dots, x_n) = \operatorname{argmax}_y P(y) \prod_{k=1}^n P(x_k|x_1, \dots, x_{k-1}, y) = \operatorname{argmax}_y P(y) \prod_{k=1}^n P(x_k|y)$$

gdje smo, dakle, kod druge jednakosti iskoristili pretpostavku o uvjetnoj nezavisnosti značajki za zadanu klasu. Primijetimo još da nam ovako definiran model ne daje vjerojatnost (jer nedostaje nazivnik Bayesovog pravila), pa, ako želimo vjerojatnost, trebamo normalizirati sukladno Bayesovom pravilu.

Dakle, uz pretpostavku uvjetne nezavisnosti značajki za zadanu klasu, izglednost smo faktorizirali na n faktora, svaki od kojih se bavi samo jednom značajkom. Je li tako faktorizirani model jednostavniji od nefaktoriziranog modela (ili, što je ekvivalentno, od modela gdje smo faktorizaciju proveli bez dodatnih pretpostavki, primjenjujući samo pravilo lanca)? Naravno da je! Pogledajmo koliko nam parametara treba za jedan faktor $P(x_k|y)$. Ako kategorička varijabla x_k poprima K_k mogućih vrijednosti, a klasificiramo u K klasa, faktor ćemo prikazati tablicom uvjetne vjerojatnosti (CPT) s ovoliko parametara:

$$(K_k - 1) \cdot K$$

Za ukupno n značajki imamo n faktora, pa je ukupan broj parametara modela naivnog Bayesovog klasifikatora, kada se još uračunaju parametri apriorne distribucije klasa, jednak:

$$\sum_{k=1}^n (K_k - 1) \cdot K + K - 1$$

Primijetite razliku u odnosu na nefaktoriziranu razdiobu: tamo smo morali imati po jedan parametar za svaku kombinaciju, kojih je bilo eksponencijalno mnogo u broju značajki. Ovdje imamo **linearnu ovisnost** u broju značajki!

2.2 Učenje naivnog Bayesovog klasifikatora

Izvršno, definirali smo model naivnog Bayesovog klasifikatora! Sad je pitanje kako ga naučiti. Prisjetimo se: (naivan) Bayesov klasifikator je probabilistički model. Što znači naučiti probabilistički model? To znači **procijeniti parametre**. Dakle, trebamo procijeniti parametre za sve distribucije koje se koriste u modelu. Konkretno, trebamo procijeniti parametre apriorne distribucije $P(y)$ i parametre za izglednosti klasa, i to, budući da smo izglednost faktorizirali,

tu procjenu trebamo napraviti posebno za svaki faktor $P(x_k|y)$. Procjena svih ovih parametara je vrlo jednostavna. Za apriornu distribuciju možemo koristiti procjenitelj MLE:

$$P(y = j) = \hat{\mu}_j = \frac{1}{N} \sum_{i=1}^N \mathbf{1}\{y^{(i)} = j\} = \frac{N_j}{N}$$

tj. to je **relativna frekvencija** klase j u skupu svih primjera. Slično, za faktor $P(x_k|y)$, procjena MLE je:

$$P(x_k|y = j) = \hat{\mu}_{k,j} = \frac{\sum_{i=1}^N \mathbf{1}\{x_k^{(i)} = x_k \wedge y^{(i)} = j\}}{\sum_{i=1}^N \mathbf{1}\{y^{(i)} = j\}} = \frac{N_{kj}}{N_j}$$

tj. to je relativna frekvencija vrijednosti x_k u svim primjerima označenima sa $y = j$.

Međutim, za procjenu parametara faktora $P(x_k|y)$ nam nije pametno koristiti procjenitelj MLE. Zašto? Problem je u tome što se lako može dogoditi da u nekoj od klasa neka značajka baš nikada ne poprimi neku vrijednost. Sličan problem imali smo kada smo sve značajke konkatenirali u jedan vektor. Ovdje doduše imamo samo jednu varijablu, a ne cijeli vektor, pa je vjerojatnost da nam se dogodi da se dotična kombinacija značajke i oznake klase ne pojavljuje u skupu za učenje sigurno manja nego da nam se to dogodi za cijeli vektor značajki, međutim ipak se to u praksi događa. U tom slučaju će procjena MLE za tu kombinaciju biti jednaka nula. To efektivno znači da tu kombinaciju smatramo nemogućom. Što se onda događa kad kod predikcije dođe primjer koji ima baš tu kombinaciju? Onda je vjerojatnost tog faktora jednaka nula. Budući da se faktori međusobno množe, to će aposteriorna vjerojatnost klase za taj primjer biti jednaka nuli!

Kako ćemo riješiti taj problem? Tako da ne dopustimo da vjerojatnosti budu jednake nuli. Kako? Tako da radimo **zaglađivanje** (engl. *smoothing*) procjena, što znači da zapravo umjesto procjenitelja MLE koristimo procjenitelj MAP. Najjednostavnije je da koristimo **Laplaceov procjenitelj** (za koji, dakako, još od prošlog tjedna znamo da je MAP procjenitelj s Dirichletovom distribucijom kao apriornom distribucijom i hiperparametrima $\alpha_k = 2$):

$$P(x_k|y = j) = \hat{\mu}_{k,j} = \frac{\sum_{i=1}^N \mathbf{1}\{x_k^{(i)} = x_k \wedge y^{(i)} = j\} + 1}{\sum_{i=1}^N \mathbf{1}\{y^{(i)} = j\} + K_k} = \frac{N_{kj} + 1}{N_j + K_k}$$

Ovime smo definirali algoritam naivnog bayesovog klasifikatora: definirali smo model, koji koristi pretpostavku o uvjetnoj nezavisnosti značajki unutar zadane klase, te optimizacijski postupak, a to je MLE (za apriorne vjerojatnosti) odnosno MAP (za izglednosti klase). Funkciju gubitka nismo eksplisitno definirali: ona je implicitna u postupku MLE odnosno MAP.

Naivan Bayesov klasifikator jednostavan je i učinkovit algoritam. Zovemo ga naivnim zbog pretpostavke o uvjetnoj nezavisnosti značajki unutar klase. Ta pretpostavka je nekada prenaivna, i onda umjesto naivnog želimo koristiti **polunaivan Bayesov klasifikator** (engl. *semi-naïve Bayes classifier*), koji relaksira neke od pretpostavki uvjetne nezavisnosti. U nastavku ćemo razmotriti polunaivni Bayesov klasifikator. Međutim, prije nego što to napravimo, pogledajmo detaljnije što je to uopće uvjetna nezavisnost.

3 Uvjetna nezavisnost

Što, zapravo, znači da su varijable **uvjetno nezavisne**? Prisjetimo se najprije što znači da su varijable **(marginalno) nezavisne**:

$$P(X, Y) = P(X) \cdot P(Y)$$

što se može napisati kao:

$$\begin{aligned} P(X|Y) &= P(X) \\ P(Y|X) &= P(Y) \end{aligned}$$

Ovaj drugi oblik puno je intuitivniji: varijabla X nezavisna je od varijable Y ako poznavanje ishoda varijable Y ne utječe na vjerojatnosti ishoda varijable X (i obrnuto). Da su varijable X i Y nezavisne (marginalno nezavisne) označavamo s $X \perp Y$.

Uvjetna nezavisnost znači da dvije varijable, X i Y , postaju nezavisne ako nam je poznat ishod neke treće varijable, Z . To znači da vrijedi:

$$\begin{aligned} P(X|Y, Z) &= P(X|Z) \\ P(Y|X, Z) &= P(Y|Z) \end{aligned}$$

što je ekvivalentno sa:

$$P(X, Y|Z) = P(X|Z)P(Y|Z)$$

Da su dvije varijable, X i Y , uvjetno nezavisne, uvjetovano na varijablu Z , označavat ćeemo s $X \perp Y|Z$. Pogledajmo primjer.

► PRIMJER

Razmotrimo situaciju upisa studenata na elitne fakultete. Neka:

$$\begin{aligned} X &= \text{"studentica je primljena na FER"} \\ Y &= \text{"studentica je primljena na PMF-MO"} \end{aligned}$$

U stvarnom životu, iskustveno, ako znamo da se ostvario X , onda to mijenja vjerojatnost našeg znanja da se ostvario Y . Tj.:

$$P(Y|X) \neq P(Y)$$

odnosno varijable nisu marginalno nezavisne, što pišemo kao $X \not\perp Y$. S druge strane, neka:

$$Z = \text{"studentica je sudjelovala na matematičkim olimpijadama"}$$

Ako znamo da se ostvario Z , onda to objašnjava prijem na oba faksa, pa spoznaja o X više ne utječe na Y , tj. vrijedi:

$$P(Y|X, Z) = P(Y|Z)$$

Drugim riječima, varijable X i Y su uvjetno nezavisne uz Z , što pišemo kao $X \perp Y|Z$.

Vratimo se naivnom Bayesovom klasifikatoru. Što mislite, vrijedi li općenito ta uvjetna nezavisnost, npr. za dvije značajke: $x_i \perp x_k | y$? Pogledajmo jedan primjer koji će nam pomoći rasvijetliti to pitanje.

► PRIMJER

Radimo klasifikaciju novinskog teksta u tematske rubrike. Želimo izgraditi naivan Bayesov klasifikator koji novinski tekst, sastavljen od riječi, klasificira u jednu od tri rubrike: sport, politika, kriminal (to ionako pokriva većinu tema dnevnog tiska). Imamo, dakle, primjer \mathbf{x} , koji je sastavljen od riječi, i označke y iz skupa {sport, politika, kriminal}. Značajke neka indiciraju prisustvo pojedine riječi u novinskom tekstu. Konkretno, pogledajmo ove tri značajke:

$$\begin{aligned} x_1 &= \mathbf{1}\{\text{"rezultat"} \in \mathbf{x}\} \\ x_2 &= \mathbf{1}\{\text{"lopta"} \in \mathbf{x}\} \\ x_3 &= \mathbf{1}\{\text{"gol"} \in \mathbf{x}\} \end{aligned}$$

To jest, značajka x_1 će biti jednaka 1 ako u novinskom tekstu negdje pojavljuje riječ "rezultat", a inače će biti jednaka 0. Značajke x_2 i x_3 funkcioniraju identično, za riječi "lopta" odnosno "gol" (ova zadnja kao imenica, ne kao pridjev).

Naivan Bayesov klasifikator prepostavlja $x_1 \perp x_2 | y$ i $x_2 \perp x_3 | y$. Provjerimo bi li ove prepostavke doista vrijedile u praksi. Pogledajmo prvo je li $x_1 \perp x_2 | y$. Za riječi "rezultat" i "lopta" možemo očekivati da će se pojavljivati za $y = \text{sport}$, pa nas dakle zanima vrijedi li:

$$P(\text{rezultat} | \text{sport}) = P(\text{rezultat} | \text{lopta}, \text{sport})$$

Intuitivno, čini se da ovo vrijedi, jer čim se u novinskom tekstu spominje sport, onda to određuje i vjerojatnost da se spomene rezultat, a spominjanje lopte ne utječe na tu vjerojatnost. Dakle, zaključujemo da bi ove varijable u praksi doiste mogле biti uvjetno nezavisne, dakle:

$$x_1 \perp x_2 | y$$

Primijetimo, međutim, da ove dvije varijable nisu marginalno nezavisne. Naime:

$$P(\text{rezultat}) \neq P(\text{rezultat} | \text{lopta})$$

jer spominjanje lopte u tekstu općenito povećava vjerojatnost spominjanja rezultata u tekstu.

Pogledajmo sada vrijedi li $x_2 \perp x_3 | y$? Pitamo se, vrijedi li:

$$P(\text{lopta} | \text{sport}) = P(\text{lopta} | \text{gol}, \text{sport})$$

Čini se da ovo ne vrijedi: ako je $y = \text{sport}$, onda vjerojatnost da se u tekstu spomene "lopta" ovisi o tome je li se u tekstu spomeno "gol" (konkretno, vjerojatnost za "lopta" raste, ako se u tekstu spomeno "gol", i obrnuto). Prema tome, varijable x_2 i x_3 nisu uvjetno nezavisne, tj.:

$$x_2 \not\perp x_3 | y$$

Vidimo, dakle, da uvjetna nezavisnost značajki unutar neke klase nekada vrijedi, a nekada ne vrijedi. Istina je da u praksi **savršena uvjetna nezavisnost rijetko vrijedi**. Ipak, unatoč tome što ne vrijedi, pokazuje se da ta prepostavka daje modele koji sasvim dobro funkcioniraju.

Ali ipak, što ako doista postoji jaka uvjetna zavisnost između varijabli, kao u ovom primjeru između riječi "lopta" i "gol"? Ako znamo da su neke varijable jako uvjetno zavisne, onda je bolje da budemo manje naivni i da ne prepostavljamo uvjetnu nezavisnost. Naravno, ekstrem bi bio da ništa ne prepostavimo, ali to smo već vidjeli da ne funkcionira (dobivamo prenaučen model koji nikako ne generalizira). Umjesto toga, ideja bi bila da samo za neke parove varijabli ne prepostavimo uvjetnu nezavisnost. Tako dobivamo **polunaivan Bayesov klasifikator**. Pogledajmo to malo detaljnije.

4 Polunaivan Bayesov klasifikator

Motivirajmo polunaivan Bayesov klasifikator našim ranijim primjerom. Ako, na primjer, ne vrijedi $x_2 \perp x_3 | y$, jer, npr., pojavljivanje riječi "lopta" i "gol" nije nezavisno za neku klasu, onda bi nam bilo pametnije da zajedničku vjerojatnost ne faktoriziramo kao

$$P(x_1, x_2, x_3, y) = P(x_1 | y)P(x_2 | y)P(x_3 | y)P(y)$$

nego kao

$$P(x_1, x_2, x_3, y) = P(x_1 | y)P(x_2, x_3 | y)P(y)$$

Ovdje je potcrтан "združeni faktor", koji nismo do kraja faktorizirali, tj. faktor kojim modeliramo zajedničku vjerojatnost varijabli x_2 i x_3 (uvjetovano na y).

Što bi bila prednost da model definiramo na ovakav način? Prednost je to što bolje modeliramo zavisnost koja postaje među varijablama x_2 i x_3 , pa će model biti točniji. Što bi bio nedostatak? To što model postaje **složeniji**, tj. broj parametara raste. Naime, broj parametara

za združeni faktor je $(K_3 \cdot K_2 - 1) \cdot K$, te on dakle ovisi o ukupnom broju kombinacija mogućih vrijednosti varijabli x_2 i x_3 .

Iz ovoga se nekako nameće zaključak da ima smisla združiti neke varijable, kako bismo dobili složeniji model, ali opet ne sve varijable, jer onda dobivamo presložen model. Ključno je, dakle, pitanje koje varijable združiti? To možemo formulirati kao **problem pretraživanja prostora stanja**: razmatramo sva moguća združivanja i po nekom kriteriju odaberemo optimalno združivanje. Pogledajmo najprije koliko mogućnosti za združivanje uopće postoje? Broj mogućnosti jednak je broju svih **particija**. Npr., za tri varijable, a , b i c , moguće particije su:

$$\begin{aligned} & \{\{a\}, \{b\}, \{c\}\} \\ & \{\{a\}, \{b, c\}\} \\ & \{\{b\}, \{a, c\}\} \\ & \{\{c\}, \{a, b\}\} \\ & \{\{a, b, c\}\} \end{aligned}$$

Ukupan broj particija je **Bellov broj**. Bellov broj za skup od tri elementa je $B_3 = 5$, od četiri $B_4 = 15$, od pet $B_5 = 52$, a od deset $B_{10} = 115975$. Očito, to je previše mogućnosti za iscrpno pretraživanje. Dakle, treba nam **heurističko pretraživanje**. Kod tog pretraživanja, svaka particija odgovara jednom stanju. Pretraga kreće od stanja s potpuno odvojenim varijablama, u svakom koraku združujemo neke varijable, i na taj način pretražujemo prostor stanja u potrazi za optimalnim združivanjem. Treba nam još kriterij pretraživanja (odnosno heurstika), kojim ćemo ocijeniti koliko je neko stanje (odnosno particija) dobra. Taj kriterij treba nam odgovoriti na pitanje je li je li bolje združiti varijable x_j i x_k u zajednički faktor $P(x_j, x_k|y)$ ili ih je bolje ostaviti odvojenima kao dva zasebna faktora, $P(x_j|y)$ i $P(x_k|y)$? Za kriterij združivanja imamo dvije mogućnosti:

1. Koristimo unakrsnu provjeru te isprobavamo **točnost** modela na skupu za provjeru i združujemo one varijable koje povećavaju točnost. Primjer takvog algoritma je algoritam **FSSJ (Forward Sequential Selection and Joining)**;
2. Mjerimo **zavisnost** varijabli i združujemo one varijable koje su najviše zavisne. Primjeri algoritama koji tako funkcioniraju su **TAN** i **k -DB**.

U nastavku je pseudokod algoritma FSSJ.

► Algoritam FSSJ

1. Inicijaliziraj $X = \emptyset$. Početna faktorizacija:

$$P(x_1, \dots, x_n, y) = P(x_1) \cdots P(x_n)P(y)$$

$$P(y|x_1, \dots, x_n) = P(y)$$

Klasificiraj primjere iz skupa za provjeru: $y^* = \operatorname{argmax}_j P(y = j)$

2. Za svaku varijablu $x_k \notin X$ koja još nije uključena u model, razmotri:
 - (a) Ukljući x_k kao uvjetno nezavisnu u odnosu na ostale varijable za danu klasu j
 - (b) Ukljući x_k tako da se ona doda u zajednički faktor s nekom već uključenom varijablom
3. Izaberi x_k i opciju koja minimizira pogrešku generalizacije
4. Ponavljam od koraka (2) do konvergencije pogreške

Drugi pristup je da nekako mjerimo koje su varijable uvjetno zavisne, pa da onda njih združimo. Ovdje se postavlja pitanje kako općenito mjeriti (ne)zavisnost varijabli? Zavisnost između varijabli mogli bismo pokušati izmjeriti Pearsonovim koeficijentom korelacije, kojeg smo se prisjetili prošli tjedan, no znamo da Pearsonov koeficijent korelacije mjeri samo linearu zavisnost između varijabli. Ako postoji nelinearna zavisnost između varijabli – što je u stvarnosti itekako moguće – to nećemo moći otkriti Pearsonovim koeficijentom korelacije. Treba nam, dakle, nešto općenitije, nešto što mjeri bilo kakvu zavisnost između varijabli. Rješenje trebamo potražiti u samoj definiciji stohastičke nezavisnosti. Pogledajmo marginalnu zavisnost (ista zapažanja vrijedit će i za uvjetnu nezavisnost). Kada su varijable (marginalno) nezavisne? Onda kada vrijedi:

$$P(X, Y) = P(X)P(Y)$$

U stvarnosti, rijetko kada ćemo imati savršenu nezavisnost. Čak i da varijable jesu savršeno nezavisne, sjetimo se da mi procjenjujemo parametre njihovih distribucija na temelju uzorka, i te procjene nikada nisu savršene. Dakle, ne možemo očekivati da će u praksi situacija biti tako čista: nećemo imati savršenu nezavisnost. Umjesto toga, zavisnost je pitanje stupnja, tj. varijable će biti u određenoj mjeri zavisne. To znači da, što su varijable više zavisne, to ćemo više odstupati od gornje jednakosti. A to onda znači da, kako bismo mjerili zavisnost varijable, trebamo mjeriti koliko $P(X, Y)$ odstupa od $P(X)P(Y)$. Ako je to odstupanje blizu nule, onda možemo reći da su varijable X i Y nezavisne. Ako je odstupanje vrlo malo, možemo reći da su varijable malo zavisne. Ako je odstupanje veliko, onda znamo da gornja jednakost sigurno ne vrijedi, tj. znamo da su varijable zavisne (bilo linearno ili nelinearno).

Sada se postavlja pitanje kako možemo mjeriti koliko $P(X, Y)$ odstupa od $P(X)P(Y)$? Znamo da su $P(X, Y)$ i $P(X)P(Y)$ dvije distribucije, pa se dakle pitanje svodi na to kako mjeriti odstupanje jedne distribucije od druge. U statistici se za to koristi **divergencija** – funkcija koja mjeri udaljenost između dviju distribucija. Jedna takva mjera divergencije, često korištena u strojnog učenju, jest **Kullback-Leiblerova divergencija** – divergencija između distribucije $P(x)$ u odnosu na $Q(x)$:

$$D_{\text{KL}}(P||Q) = \sum_x P(x) \ln \frac{P(x)}{Q(x)}$$

Ova se mjera može lako izvesti iz **relativne entropije**. Što je vrijednost ove mjere veća, to su distribucije međusobno različitije. Nas ovdje konkretno zanima KL-divergencija između $P(X, Y)$ i $P(X)P(Y)$, koja je jednaka:

$$D_{\text{KL}}(P(x, y)||P(x)P(y)) = \sum_{x,y} P(x, y) \ln \frac{P(x, y)}{P(x)P(y)} = I(x, y)$$

Ovako definirana KL-divergencija naziva se **uzajamna informacija** (engl. *mutual information*) i označava sa $I(x, y)$. Dakle, uzajamna informacije $I(x, y)$ je zapravo KL-divergencija između zajedničke distribucije $P(X, Y)$ i zajedničke distibucije uz pretpostavku nezavisnosti (koja je onda jednaka umnošku marginalnih distribucija, $P(X)P(Y)$). Što je $I(x, y)$ veća, to su distribucije $P(X, Y)$ i $P(X)P(Y)$ različitije, tj. to su varijable X i Y **više zavisne**, jer smo sve udaljeniji od jednakosti distribucija $P(X, Y)$ i $P(X)P(Y)$. S druge strane, ako $X \perp Y$, onda je $I(x, y) = 0$ (vidimo da ćemo tada imati $\log 1 = 0$).

Dva popularna algoritma za polunaivni Bayesov klasifikator koji koriste mjeru uzajamne informacije su **TAN** (engl. *tree augmented naive Bayes*) i **k -DB** (engl. *k -limited dependence Bayesian classifier*). Ti algoritmi koriste mjeru uzajamne informacije kako bi heuristički pretražili prostor stanja mogućih particija varijabli, s ciljem da se združe one varijable koje su najviše međusobno zavisne, čime se dobiva polunaivni Bayesov klasifikator. Ovdje nećemo ići u detalje tih algoritama. Dovoljno je da znamo da, ako trebamo nekako odrediti koje su varijable najviše međusobno zavisne, kako bismo ih združiti u zajednički faktor, za to možemo koristiti mjeru uzajamne informacije.

Sažetak

- Gaussov Bayesov klasifikator sa dijeljenom kovarijacijskom matricom daje istu granicu kao i **logistička regresija**, no ima više parametara
- Naivan Bayesov klasifikator faktorizira izglednost na temelju prepostavke o **uvjetnoj nezavisnosti** značajki unutar klase, čime se smanjuje broj parametara i omogućava generalizaciju
- Parametre naivnog Bayesovog klasifikatora možemo procijeniti pomoću MLE ili MAP
- **Polunaivni Bayesov klasifikator** modelira zavisnost između odabranih varijabli, čime dobivamo **složeniji model**

Bilješke

1 Prva tri poglavlja današnjeg predavanja slijede poglavlja 3.1–2 i 5.4–6 iz ([Alpaydin, 2020](#)).

2 Ovo je zapravo **logaritam omjera šansi** (engl. *log odds*):

$$\ln \frac{p}{1-p}$$

gdje je p neka vjerojatnost. Omjer šansi (engl. *odds*) alternativni je način da se iskaže izglednost nekog događaja, definiran jednostavno kao omjer vjerojatnosti da se događaj dogodi i vjerojatnosti da se događaj ne dogodi (ili, ako želimo izbjegći vjerojatnost, a shvatimo je frekventistički, kao broj pozitivnih realizacija u ponavljanju pokusa, onda je to omjer broja pozitivnih ishoda i broja negativnih ishoda). Logaritam omjera šansi je logaritam tog omjera, i to je zapravo inverzna funkcija sigmoidne funkcije. Drugačije rečeno, ako $p = \sigma(\alpha)$, onda je $\alpha = \ln \frac{p}{1-p}$. Logaritam omjera šansi naziva se i **logit** funkcija. Kolokvijalno, “logit” je broj koji dovodimo na ulaz sigmoidalne (također i softmax) funkcije. Logiti su važni u statističkoj analizi efekata pomoću logističke regresije.

3 Prisjetite se da smo prošli puta spomenuli da je Gaussov Bayesov klasifikator s linearom granicom između klasa istovjetan modelu **linearne diskriminantne analize (LDA)**.

4 Preciznije, ako se primjer \mathbf{x} nije pojavio u skupu za učenje niti sa jednom oznakom y , onda će aposteriorna vjerojatnost $P(y|\mathbf{x})$ biti 0/0, tj. nedefinirana. Ako se, međutim, primjer pojavio u skupu za učenje, ali se nije pojavio u nekoj od klasa, tj. $P(\mathbf{x}) \neq 0$, ali za neku oznaku y vrijedi $P(\mathbf{x}|y) \neq 0$, onda će aposteriorna vjerojatnost $P(y|\mathbf{x})$ za klase u kojima se primjer pojavio biti različita od nule, a za klase u kojima se primjer nije pojavio bit će jednaka nuli. No, u oba slučaja model loše generalizira.

5 Važan tehnički detalj koji smo ovdje zanemarili jest **podljev** (engl. *underflow*) koji se može dogoditi pri izračunu zajedničke vjerojatnosti u brojniku ili nazivniku Bayesovog pravila. Naime, množenjem pojedinačnih faktora, koji će u pravilu biti vrlo mali brojevi, dobivamo ekstremno male brojeve koji su manji od onoga što je prikazivo u zapisu s pomičnim zarezom. Rješenje za to je tzv. **log-sum-exp trik**; v. <https://stats.stackexchange.com/q/105602/93766>.

6 Legitimno pitanje je zašto ne bismo procjenitelj MAP koristili za procjenu parametara apriorne vjerojatnosti $P(y)$, a ne samo izglednosti $P(x_k|y)$? Odgovor je da za to nema potrebe. Kod faktora $P(x_k|y)$ lako se može dogoditi da se neka kombinacija (x_k, y) nikada nije pojavila u skupu za učenje, stoga, kako bismo izbjegli da vjerojatnost te kombinacije bude jednak nuli (i time efektivno bude proglašena nemogućom), radimo zaglađivanje, tj. koristimo procjenitelj MAP. Kod vjerojatnosti $P(y)$, međutim, vjerojatnost će za neku vrijednost oznake y biti jednak nuli samo onda kada u skupu za učenje ne postoji niti jedan primjer koji pripada dotičnoj klasi. A ako je to slučaj, onda nema smisla pokušavati izgraditi klasifikator za tu klasu. Zato nema potrebe zagladiti procjenu za $P(y)$.

7 Jamačno ste se upitali kako bismo u tekstu automatski odredili je li riječ “gol” imenica (u značenju vratiju sastavljenih od dviju stativa, prečke i mreže, ili u značenju pogotka postignutog ubacivanjem

lopte) ili pridjev (u značenju onoga koji na sebi nema odjeće ili nečega što je bez svog svojstva). Taj zadatak u domeni je područja **obrade prirodnog jezika** (engl. *natural language processing, NLP*), i poznat je pod nazivom **označavanje vrste riječi** (engl. *part-of-speech tagging, PoS tagging*), i smatra se za većinu jezika zadovoljavajuće riješenim problemom. Algoritmi za označavanje vrste riječi temelje se na, dakako, na strojnem učenju, i to su uglavnom modeli već spomenutog **uvjetnog slučajnog polja** (engl. *conditional random field*), ili, u novije vrijeme, povratne neuronske mreže. Više: https://en.wikipedia.org/wiki/Part-of-speech_tagging i http://nlpprogress.com/english/part-of-speech_tagging.html.

- 8** **Bellov broj** može se izračunati različitim metodama, v. <http://fredrikj.net/blog/2015/08/computing-bell-numbers/>. Vjerojatno najjednostavnija metoda je **Bellov trokut**, v. <https://code.sololearn.com/cxt4qe0Dy7hp/#py>. Inače, Bellovi brojevi nazivaju se tako po matematičaru Ericu Bellu, koji se bavio teorijom brojeva, pa je tako pisao i o Bellovim brojevima, ali su ti brojevi matematičarima bili poznati i ranije. Korisno je upamtiti da $B_n \leq n!$, tj. broj particija nekog skupa nije veći od broja permutacija elemenata tog skupa (jer redoslijed elemenata u skupu ne igra ulogu).
- 9** Pokažimo kako je **Kullback-Leiblerova divergencija** izvedena iz relativne entroprije. Prisjetimo se, **entropija** je definirana kao

$$H(P) = - \sum_x P(x) \ln P(x)$$

i to je prosječan minimalan broj bitova potreban za kodiranje događaja iz distribucije P . **Unakrsna entropija** definirana je kao

$$H(P, Q) = - \sum_x P(x) \ln Q(x)$$

i to je prosječan broj bitova potreban za kodiranje događa, ako se upotrijebi shema kodiranja koja je optimalna za drugu distribuciju, Q . **Relativa entropija** $P(x)$ u odnosu na $Q(x)$ definirana je kao razlika unakrsne entropije i entropije:

$$\begin{aligned} H(P, Q) - H(P) &= - \sum_x P(x) \ln Q(x) - \left(- \sum_x P(x) \ln P(x) \right) \\ &= - \sum_x P(x) \ln Q(x) + \sum_x P(x) \ln P(x) \\ &= \sum_x P(x) \ln \frac{P(x)}{Q(x)} = D_{\text{KL}}(P||Q) \end{aligned}$$

i to je Kullback-Leiblerova divergencija.

- 10** Uzajamnu informaciju lako možemo proširiti na **uvjetnu uzajamnu informaciju** (engl. *conditional mutual information*):

$$I(x, y|z) = \sum_z P(z_k) I(x, y|z) = \sum_z \sum_x \sum_y P(x, y, z) \ln \frac{P(x, y|z)}{P(x|z)P(y|z)}$$

- 11** Detalje o algoritmima **TAN** i **k-DB**, popraćene primjerima, možete naći u poglavlju 4.3 u skripti.

Literatura

E. Alpaydin. *Introduction to machine learning*. MIT press, 2020.

17. Probabilistički grafički modeli

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v2.1

Prošli put bavili smo se **Bayesovim klasifikatorom**. Bayesov klasifikator je probabilistički i generativni model. Danas nastavljamo s probabilističkim modelima. Bavit ćemo se jednom širom porodicom probabilističkih modela koju zovemo **probabilistički grafički modeli** (engl. *probabilistic graphic models*), ili kraće **PGM-ovi**. PGM-ovi uključuju Bayesov klasifikator, ali i druge složenije generativne modele, a također uključuju i neke diskriminativne modele. Možda ste čuli za skrivene Markovljeve modele (HMM), ili Latentnu Dirichletovu alokaciju (LDA), ili pak za uvjetna slučajna polja (CRF). Sve su to PGM-ovi koji se vrlo često koriste u praksi i na mnogim klasifikacijskim zadatcima daju vrlo dobre rezultate. S konceptualne strane, PGM-ovi su zanimljivi jer predstavljaju jedan dobar (možda najbolji?) radni okvir za modeliranje **kauzalnosti**, a kauzalnost je ono što bi nas u znanosti trebalo najviše zanimati.

PGM-ovi su ogromno područje unutar strojnog učenja: mnogo se ljudi bavi PGM-ovima i postoje mnoge korisne primjene. Zapravo, PGM-ovi su bili dominantna paradigma u strojnom učenju do negdje 2010. godine, kada se fokus prebacio na duboko učenje. Budući da se radi o velikom području, a da je naše vrijeme ograničeno, radit ćemo na dvije razine: temeljne ideje ćemo temeljito obraditi, dok ćemo neke napredne koncepte samo spomenuti.

1 Uvod

Kao uvod u PGM-ove poslužit će nam **naivan Bayesov klasifikator**. Prisjetimo se najprije tog modela. Npr., model naivnog Bayesovog klasifikatora za tri značajke jest:

$$h(\mathbf{x}) = P(x_1|y)P(x_2|y)P(x_3|y)P(y)$$

Također, prisjetimo se **polunaivnog Bayesovog klasifikatora**, kod kojega ne prepostavljamo da su sve značajke uvjetno nezavisne za zadanu klasu, nego dopuštamo da su neke značajke možda zavisne, pa ih modeliramo zajedničkim faktorom. Npr., ako ne želimo prepostaviti uvjetnu nezavisnost značajki x_2 i x_3 , onda ćemo model definirati ovako:

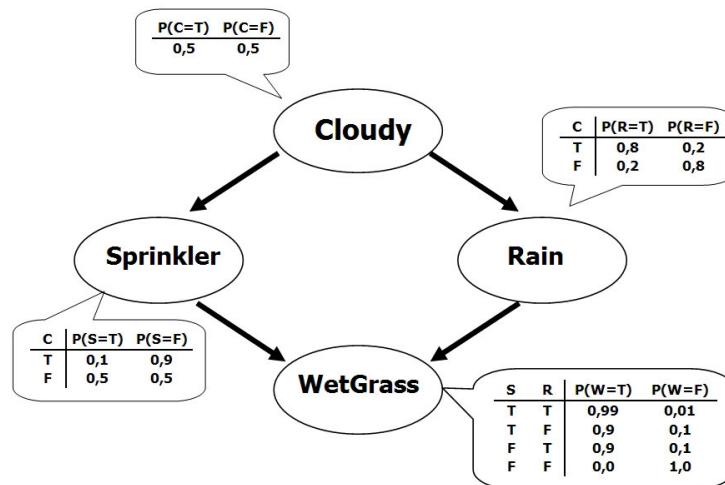
$$h(\mathbf{x}) = P(x_1|y)P(x_2, x_3|y)P(y)$$

U oba ova slučaja radi se o tome da smo na neki način (potpuno ili manje potpuno) faktorizirali zajedničku distribuciju $P(x_1, x_2, x_3)$. Ispostavlja se da su oba ova modela samo posebni slučajevi probabilističkog grafičkog modela (PGM).

PGM je **sažet način zapisa zajedničke distribucije pomoću grafa**. Ta zajednička distribucija može biti vrlo složena i može biti definirana nad visokodimenzijskim prostorima. Pritom graf služi kao kostur za faktorizaciju zajedničke distribucije: zasniva se na ideji da svaka varijabla interagira samo s manjim brojem drugih varijabli, tj. zavisnosti između varijabli su **lokalne**. Čvorovi u grafu su varijable, dok bridovi u grafu kodiraju zavisnosti između varijabli. Budući da se graf može prikazati grafički, to se ovi modeli zovu **grafički modeli**. To je malo nesretno ime, ali sad je gotovo.

► PRIMJER

Evo jednog jednostavog motivirajućeg primjera, koji se pojavljuje u svoj literaturi o PGM-ovima:



Ova mreža prikazuje zajedničku vjerojatnost na temelju zavisnosti između varijabli. Možemo reći da te zavisnosti zapravo modeliraju **kauzalnost** (uzročnost). Svaki čvor odgovara jednoj slučajnoj varijabli i pohranjuje uvjetne vjerojatnosti za tu varijablu – to su **uvjetne vjerojatnosne distribucije** (engl. *conditional probability distributions*, *CPDs*). Parametri uvjetnih vjerojatnosti za svaki čvor zapravo su parametri modela, koje možemo naučiti iz podataka, npr. procjeniteljem MLE. Ako su varijable diskretne, kao što je to ovdje slučaj, onda su parametri njihovi distribucija zapravo parametri μ_k kategoričke razdiobe, i njih možemo prikazati **tablicom uvjetne vjerojatnosti** (engl. *conditional probability table*, *CPT*), koju smo već bili spomenuli prošli put. U ovom primjeru, graf je izgrađen ručno, na temelju našeg znanja o kauzalnosti između događaja, no postoje postupci da se i sam graf uči na temelju podataka.

Svrha PGM-a jest omogućiti **probabilističko zaključivanje** o vrijednostima jedne ili više varijabli, moguće uz fiksirane vrijednosti nekih drugih varijabli. Na primjer, na temelju mreže iz gornjeg primjera možemo zaključiti da, ako vidimo da je trava mokra, a ne znamo je li bilo oblačno, koliko je vjerojatno da je radila prskalica, a koliko da je padala kiša (konkretno, ispada da je vjerojatnost da je radila prskalica 43%, a vjerojatnost da je padala kiša 70%).

Postoje tri aspekta PGM-a o kojima trebamo pričati: **prikazivanje (reprezentacija)**, **zaključivanje** (engl. *inference*) i **učenje**.

Što se reprezentacije tiče, tu postoje dvije osnovne porodice modela: **Bayesove mreže** (engl. *Bayesian networks*), koje koriste usmjerene grafove, i **Markovljeve mreže** (engl. *Markov networks*), koje koriste neusmjerene grafove. Ova dva pristupa razlikuju se po načinu kako prikazuju zavisnosti između varijabli. Ta razlika nije samo estetske prirode: svaki od ovih modela može prikazati zavisnosti koje onaj drugi ne može. Ovaj gornji primjer je Bayesova mreža.

Zaključivanje znači da, na temelju kompaktног zapisa zajedničke distribucije, odredimo vrijednosti nepoznatih varijabli na temelju poznatih varijabli – vrijednosti značajki. To se također naziva **upiti nad distribucijom** (engl. *querying the distribution*): upit su varijable koje su nam poznate, a odgovor su vrijednosti varijabli koje su nam prethodno bile nepoznate. (Primijetite da ovo nije isto kao "zaključivanje" u statistici, koje se zapravo svodi na procjenu parametara distribucije na temelju uzorka iz populacije.)

Konačno, htjeli bismo moći učiti ovakve modele na temelju podataka. Učenje modela može biti **generativno** ili **diskriminativno**. Bayesove mreže (usmjereni grafovi) tipično učimo generativno, a Marovljeve mreže (neusmjereni grafovi) tipično učimo diskriminativno. Osim učenja parametara, možemo učiti i samu strukturu modela (graf).

Danas ćemo se fokusirati na Bayesove mreže, poput ove u gornjem primjeru. Bayesove mreže također se nazivaju **usmjereni grafički modeli**. Također se nazivaju **mreže vjerovanja** (engl. *belief networks*) i **kauzalne mreže**, jer lijepo mogu opisati naše vjerovanje o kauzalnim vezama između događaja. (Teorija vjerojatnosti je dobar radni okvir za opisivanje kauzalnosti.)

3

Kao što smo rekli, postoje tri aspekta o kojima možemo pričati kod PGM-ova, pa tako i kod Bayesovih mreža: reprezentacija, zaključivanje i učenje. Danas ćemo pričati samo o reprezentaciji, dok ćemo o zaključivanju i učenju pričati idući put.

2 Bayesove mreže: reprezentacija

2.1 Usmjereni graf i uvjetne (ne)zavisnosti

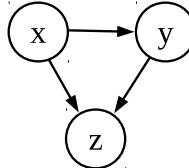
Kao primjer, razmotrit ćemo zajedničku distribuciju triju varijabli:

$$p(x, y, z)$$

Općenitosti radi, ovdje pišemo gustoće vjerojatnosti p , ali sve što ćemo dalje raditi vrijedi i za vjerojatnosti P , tj. za slučaj kada su varijable diskretne. Primjenom **pravila umnoška** ovu distribuciju možemo napisati kao:

$$p(x, y, z) = p(x)p(y|x)p(z|x, y)$$

Iz ovoga sada crtamo usmjereni graf: čvorovi grafa odgovaraju varijablama (ili skupovima varijabli), a bridovi odgovaraju zavisnostima, i to tako da crtamo brid od varijable koja uvjetuje prema varijabli koja je uvjetovana:



Npr., za faktor $p(z|x, y)$ imamo bridove iz x i y u z , dok za faktor $p(x)$ nemamo ulaznih bridova.

Primijetite da smo faktorizaciju mogli napraviti i na neki drugi način. Mi smo odabrali redoslijed varijabli x, y, z . Međutim, mogli smo odabratи неки drugi redoslijed varijabli (od ukupno $3! = 6$ mogućih), npr.:

$$p(x, y, z) = p(y)p(z|y)p(x|y, z)$$

Ovo naravno funkcioniра i kad imamo više od tri varijable. Naime, prisjetimo se da svaku zajedničku distribuciju uvijek možemo faktorizirati primjenom **pravila lanca** (koji je zapravo samo višestruka primjena pravila umnoška):

$$\begin{aligned} p(x_1, \dots, x_n) &= p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)\cdots p(x_n|x_1, \dots, x_{n-1}) \\ &= \prod_{k=1}^n p(x_k|x_1, \dots, x_{k-1}) \end{aligned}$$

Kako izgleda Bayesova mreža koja odgovara takvoj faktorizaciji? Svaki čvor ima ulazne bridove iz svih čvorova koji mu u odabranom poretku prethode (svih čvorova s manjim rednim brojem indeksa). Drugim riječima, svaki čvor x_i povezan je sa svim čvorovima x_{i-1}, \dots, x_1 . To zapravo znači da je Bayesova mreža **potpuno povezan graf** (svaki par čvorova je povezan).

Međutim, iz naših prošlotjednih razmatranja Bayesovog klasifikatora, već znamo što je problem s takvim modelima: presloženi su. Broj kombinacija vrijednosti varijabli (a tome odgovara i broj parametara modela koje moramo procijeniti iz podataka) raste eksponencijalno s

brojem varijabli. Nadalje, takvi modeli uopće ne mogu generalizirati: za kombinacije vrijednosti značajki koje se nisu pojavile u skupu za učenje vjerojatnost je jednaka nuli. Da bi model mogao generalizirati, moramo uvesti neke **induktivne pristranosti**, odnosno neke pretpostavke. Kod probabilističkih grafičkih modela, jednako kao i kod Bayesovog klasifikatora, te pretpostavke dolaze u obliku **uvjetnih nezavisnosti**. Prisjetimo se, slučajne varijable X i Y uvjetno su nezavisne uz varijablu Z , ako:

$$\begin{aligned} X \perp\!\!\!\perp Y | Z &\Leftrightarrow P(X, Y | Z) = P(X|Z)P(Y|Z) \\ &\Leftrightarrow P(X|Y, Z) = P(X|Z) \\ &\Leftrightarrow P(Y|X, Z) = P(Y|Z) \end{aligned}$$

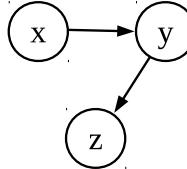
Uvođenjem uvjetnih nezavisnosti između parova varijabli, zajednička distribucija može se jednostavnije faktorizirati, tj. faktorizirati u faktore koji imaju manji broj varijabli (a time i parametara) nego što bi to imali faktori bez pretpostavki o uvjetnoj nezavisnosti. Npr., ako u ranijem primjeru pretpostavimo da vrijedi:

$$x \perp\!\!\!\perp z | y$$

onda $p(z|x, y) = p(z|y)$, pa dobivamo:

$$p(x, y, z) = p(x)p(y|x)p(z|x, y) = p(x)p(y|x)p(z|y)$$

što nam daje i jednostavniju Bayesovu mrežu:



Nadalje, ako bismo još pretpostavili da $x \perp\!\!\!\perp y$ (marginalna nezavisnost), onda $p(y|x) = p(y)$, pa se faktorizacija i pripadna Bayesova mreža još više pojednostavljuju:

$$p(x, y, z) = p(x)p(y|x)p(z|y) = p(x)p(y)p(z|y)$$

Vidimo da uvođenje dodatnih pretpostavki o nezavisnosti zapravo pojednostavljuje Bayesovu mrežu – i to uklanjanjem bridova. Gdje god između para varijabli nedostaje neki brid (u odnosu na potpuno povezan graf), tu su varijable međusobno uvjetno nezavisne. Konstatiramo, dakle, da Bayesove mreže – a to vrijedi i za PGM-ovove općenito – sažeto prikazuju zajedničku distribuciju na temelju pretpostavki o **uvjetnoj nezavisnosti**. Te pretpostavke **pojednostavljuju** model i u konačnici omogućavaju **generalizaciju**.

Vidimo da iz faktorizacije zajedničke vjerojatnosti uvijek možemo konstruirati pripadnu Bayesovu mrežu. Također je očito da možemo napraviti i obrnuto: iz zadane Bayesove mreže možemo iščitati faktorizaciju zajedničke distribucije.

2.2 Uredajno Markovljevo svojstvo

Definirajmo sada malo formalnije taj odnos između Bayesove mreže (usmjerenog grafa) i njoj odgovarajuće zajedničke distribucije. Za Bayesovu mrežu sa n čvorova (varijabli), zajednička distribucija dana je sa:

$$p(\mathbf{x}) = \prod_{k=1}^n p(x_k | \text{pa}(x_k))$$

gdje $\text{pa}(x_k)$ označava čvorove roditelje čvora x_k . Da bi ova faktorizacija bila moguća, bitno je da se čvorovi mogu poredati u **potpuni (linearni) uređaj** tako da roditelji dolaze prije djece

(po indeksima). Inače bi se jedna te ista varijabla trebala u faktorizaciji koristiti više puta, što ne bi dalo faktorizaciju. Ovaj uvjet u Bayesovoj mreži svodi se na to da u grafu ne smije biti usmjerenih ciklusa (diciklusa), tj. Bayesova mreža je **usmjereni aciklički graf** (engl. *directed acyclic graph, DAG*). Uređaj čvorova koji možemo iščitati iz DAG-a zapravo je tzv. **topološki uredaj**: potpuni (linearni) uređaj kod kojeg čvorovi roditelji dolaze prije čvorova djece. Svaki DAG ima barem jedan topološki uredaj, ali ih može imati i više.

Za zadani topološki uredaj, faktorizacija koja odgovara Bayesovoj mreži zapravo pretpostavlja da svaki čvor x_k ovisi samo o svojim **direktnim roditeljima**, a ne i o svim svojim prethodnicima. To možemo formalno zapisati ovako:

$$x_k \perp \text{pred}(x_k) \setminus \text{pa}(x_k) \mid \text{pa}(x_k)$$

gdje je $\text{pred}(x_k)$ skup prethodnika čvora x_k po topološkom uredaju. Dakle, ako su poznate vrijednosti čvorova roditelja, vrijednost čvora x_k nezavisna je od vrijednosti čvorova koji prethode roditeljskim čvorovima. Navedeno svojstvo naziva se **uredajno Markovljevo svojstvo** (engl. *ordered Markov property*).

Razmotrima sada jedan primjer koji će nam pomoći da bolje shvatimo povezanost između grafa Bayesove mreže i uvjetnih nezavisnosti koje su kodirane tom mrežom.

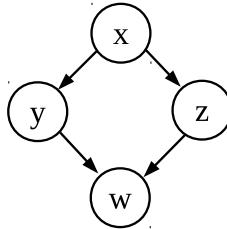
► PRIMJER

Prepostavimo da se zajednička vjerojatnost od četiri varijable može faktorizirati na sljedeći način:

$$p(x, y, z, w) = p(x)p(y|x)p(z|x)p(w|y, z)$$

Vidimo da, kada bismo pokušali pomnožiti ove faktore primjenom pravila umnoška, ne bismo uspjeli dobiti vjerojatnost na lijevoj strani jednakosti. Drugim riječima, ova faktorizacija nije dobivena izravnom primjenom pravila lanca, a to onda znači da su u ovu faktorizaciju ugrađene neke uvjetne nezavisnosti.

Bayesova mreža koja odgovara ovoj faktorizaciji je sljedeća:



Koji bridovi ovdje nedostaju (u odnosu na potpuno povezan graf)? Nedostaju dva brida: od x do w i od y do z (ili obrnuto, ovisno za koji se topološki uredaj odlučimo). Topološki uredaji čvorova su x, y, z, w i x, z, y, w . Odaberimo ovaj prvi.

Možemo li iz Bayesove mreže rekonstruirati zajedničku distribuciju $p(x, y, z, w)$? To znači da želimo napisati zajedničku distribuciju kao umnožak faktora, poštujući pritom pretpostavke o uvjetnoj nezavisnosti koje su implicitno (kroz odsustvo bridova) kodirani u Bayesovoj mreži.

Prvi način na koji to možemo napraviti jest da krenemo od faktorizacije Bayesove mreže i primjenjujemo **pravilo umnoška**, proširujući faktore gdjegod je to potrebno. Gdjegod proširujemo faktore, zapravo otkrivamo uvjetnu nezavisnost koja je bila kodirana u tom faktoru dok on nije bio proširen. Konkretno:

$$\begin{aligned} p(x)p(y|x)p(z|x)p(w|y, z) &= p(x, y)p(z|x)p(w|y, z) \\ \textcolor{red}{y \perp\!\!\! \perp z|x} \Rightarrow p(x, y)p(\underline{z|x}, \textcolor{red}{y})p(w|y, z) &= p(x, y, z)p(w|y, z) \\ \textcolor{red}{x \perp\!\!\! \perp w|y, z} \Rightarrow p(x, y, z)p(\underline{w|y, z}) &= p(x, y, z, w) \end{aligned}$$

Potrtani su faktori koje proširujemo i koje smo dobili proširivanjem. To su faktori kod kojih smo dodajemo varijable u uvjetni dio uvjetne vjerojatnosti, kako bismo mogli primijeniti pravilo umnoška. Npr., u prvom koraku faktor $p(z|x)$ proširili smo u $p(z|x,y)$, tj. dodali smo varijablu y u uvjetni dio, jer tek tako prošireni faktor možemo pomnožiti s faktorom $p(x,y)$ kako bismo, na temelju pravila umnoška, dobili faktor $p(x,y,z)$. Budući da smo faktor $p(z|x)$ napisali kao faktor $p(z|x,y)$, to znači da mi zapravo prepostavljam da vrijedi $p(z|x) = p(z|x,y)$, a to, prema definiciji uvjetne nezavisnosti, znači da prepostavljam da vrijedi uvjetna nezavisnost $y \perp z|x$. Slično smo postupili kod faktora $p(w|y,z)$, koji smo proširili varijablom x u uvjetnom dijelu. Na koncu smo doista uspijeli doći do zajedničke vjerojatnosti primjenom pravila umnoška. Pritom smo uveli dvije prepostavke o uvjetnoj nezavisnosti: $y \perp z|x$ i $x \perp w|y,z$. To su prepostavke koje su implicitno ugrađene u graf Bayesove mreže. Primijetite da te prepostavke ne možemo samo tako iščitati iz Bayesove mreže: naime, u Bayesovoj mreži nedostaju bridovi između varijabli x i w i varijabli y i z , pa bismo mogli zaključiti da su ti parovi varijabli nezavisni, međutim nije jasno čime je ta nezavisnost uvjetovana. (Vidjet ćemo nešto kasnije da postoje pravila pomoću kojih bismo to ipak mogli zaključiti.)

Alternativno, umjesto da pokušamo rekonstruirati zajedničku distribuciju i pritom implicitno otkrivamo uvjetne nezavisnosti, uvjetne nezavisnosti možemo izvesti izravno iz Markovljevog svojstva:

$$x_k \perp \text{pred}(x_k) \setminus \text{pa}(x_k) \mid \text{pa}(x_k)$$

To ćemo napraviti tako da izrazimo Markovljevo svojstvo za svako od četiri varijable:

$$\begin{aligned} y \perp \{x\} \setminus \{x\} \mid \{x\} \\ z \perp \{x, y\} \setminus \{x\} \mid \{x\} \Rightarrow y \perp z|x \\ w \perp \{x, y, z\} \setminus \{y, z\} \mid \{y, z\} \Rightarrow x \perp w|y, z \end{aligned}$$

Ovdje smo koristili topološki uređaj x, y, z, w , pa je tako, na primjer, $\text{pred}(z) = \{x, y\}$. Primijetite da vrijedi komutativnost (uvjetne) nezavisnosti, pa je $z \perp y|x$ isto što i $y \perp z|x$. Također primijetite da u prvom izrazu nismo uspjeli izvesti nezavisnost za y , ali u drugom jesmo. To ovisi o tome koji smo topološki uređaj odabrali. Da smo se odlučili za drugi topološki uređaj (x, z, y, w) , dobili bismo u konačnici iste dvije uvjetne nezavisnosti.

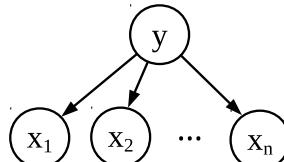
Pogledajmo sada nekoliko primjera Bayesovih mreža.

3 Primjeri Bayesovih mreža

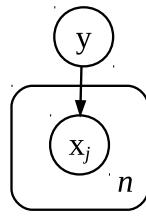
Prošli tjedan pričali smo o **naivnom Bayesovom klasifikatoru**. Kod naivnog Bayesovog klasifikatora prepostavljam da su značajke uvjetno nezavisne za danu klasu. Ta nam je prepostavka omogućila ovakvu faktorizaciju zajedničke vjerojatnosti:

$$p(\mathbf{x}, y) = p(y) \prod_{j=1}^n p(x_j|y)$$

Ovoj faktorizaciji odgovara sljedeća Bayesova mreža:



Čvorova x_j ima onoliko koliko ima značajki, odnosno možemo reći da je čvor x_j multipliciran n puta. Često se događa da se neki čvor tako multiplicira: za svaku značajku ili za svaki primjer ili nešto treće. Kako bi se u takvim slučajevima pojednostavio grafički izgled Bayesove mreže, uvodimo malo sintaktičkog šećera: **pladnjeve** (engl. *plate notation*). Varijablu koja se ponavlja crtamo u okviru, varijabli dodajemo indeks, i u kutu pladnja pišemo broj ponavljanja:



Prošli put rekli smo da, u slučajevima kada između nekih varijabli postoji zavisnost (točnije: kada ne postoji uvjetna nezavisnost), bolje je ne prepostaviti takvu nezavisnost, odnosno ne faktorizirati u potpunosti. To nas je dovelo do **polunaivnog Bayesovog klasifikatora**. Npr., ako znamo (ili ako utvrdimo, npr., mjerom uzajamne informacije) da varijable x_2 i x_3 nisu uvjetno nezavisne za klasu y , tj. $x_2 \perp\!\!\! \perp x_3 | y$, onda nam je bolje da umjesto

$$p(x_1, x_2, x_3, y) = p(x_1|y)p(x_2|y)p(x_3|y)p(y)$$

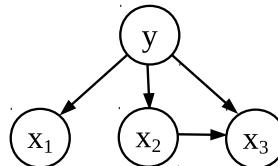
zajedničku vjerojatnost faktoriziramo kao:

$$p(x_1, x_2, x_3, y) = p(x_1|y)\underline{p(x_2, x_3|y)}p(y)$$

što je jednako kao (primjenom **pravila umnoška**):

$$p(x_1, x_2, x_3, y) = p(x_1|y)\underline{p(x_2|y)}p(x_3|x_2, y)p(y)$$

Ovoj drugoj faktorizaciji odgovara sljedeća Bayesova mreža:

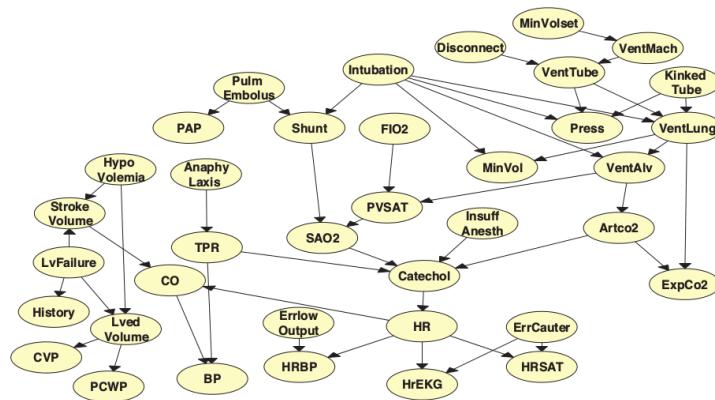


(Prvoj faktorizaciji odgovara Bayesova mreža koja u jednom čvoru ima kombinaciju dvije varijable, x_2, x_3 , tj. ima čvor za zajedničku vjerojatnost $p(x_2, x_3|y)$. Ovo razlika je samo grafičke prirode; modeli su zapravo istovjetni te imaju isti broj parametara).

A evo sada i nekoliko primjera složenijih Bayesovih mreža iz literature.

► PRIMJER

“Alarm network” dijagnostički je model za nadzor pacijenata, temeljen na Bayesovoj mreži. Mreža ima 37 varijabli, od čega 8 varijabli predstavlja moguće dijagnoze, 16 varijabli predstavljaju simptome, a ostalih 13 varijabli su dodatne varijable:

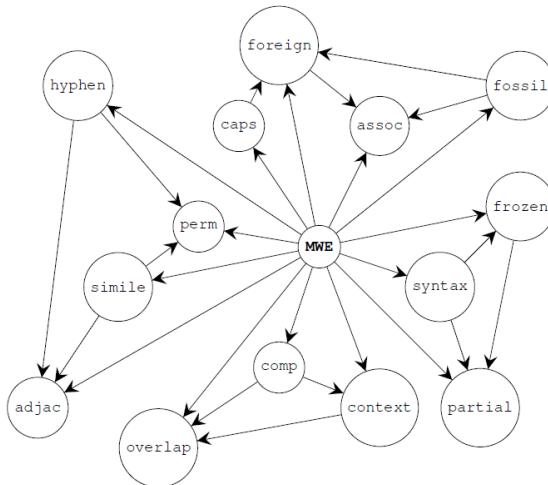


Varijable simptoma očitavaju fiziološke podatke pacijenta, kao što su krvni tlak, frekvenciju srca, itd. Sve varijable su kategoričke (kontinuirane varijable su diskretizirane). Mreža ukupno ima 504 parametra.

► PRIMJER

Drugi primjer je model iz područja obrade prirodnog jezika. Model je razvijen u svrhu prepoznavanja višerječnih izraza (engl. *multiword expressions*, *MWE*) hrvatskoga jezika, npr. izraza poput "godišnji odmor", "povišica plaće", "uputstva za uporabu" ili "Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave". Višerječni izrazi uključuju i tzv. semantički neprozirne izraze, kod kojih je značenje izraza nadilazi jednostavnu kombinaciju značenja riječi od kojih su ti izrazi sastavljeni, npr., izrazi poput "ležeći policajac" ili "medeni mjesec". Prepoznavanje višerječnih izraza važno je u obradi prirodnog jezika, kako bi se takvi izrazi mogli obrađivati kao cjeline, umjesto da ih se rastavlja na pojedinačne riječi. Prepoznavanje se tipično provodi analizom velikih količina tekstova (tzv. korpusa), gdje se analiziraju statistička stvojstva nizova riječi, kako bi se utvrdilo čini li neki niz riječi višerječnu jedinicu.

Bayesova mreža funkcioniра kao binarni klasifikator: za zadani niz od n riječi mreža treba odlučiti je li taj niz višerječni izraz hrvatskoga jezika, ili je naprsto slučajna kombinacija od dvije ili više riječi. U tu svrhu za nizove riječi iz korpusa izračunali smo niz značajki, poput frekvencije zajedničkog pojavljivanja riječi, pojavljuju li se te riječi u permutiranom redoslijedu, sadrži li neka od riječi spojnicu, je li neka od riječi strana riječ, itd. To su varijable u našoj Bayesovoj mreži. Ciljna varijabla je varijabla MWE, koja određuje je li zadani niz riječi doista višerječni izraz hrvatskoga jezika. Mreža izgleda ovako:



Varijabla MWE (u sredini mreže) je varijabla klase koja nas zanima, a ona uvjetuje sve druge varijable, bilo direktno ili indirektno. Ova je mreža izrađena ručno, na temelju lingvističke intuicije (postupci za učenje strukture mreže nisu dali dobre rezultate). Parametri mreže naučeni su na ručno označenom skupu podataka (skupu koji za nizove riječi ima označeno jesu li to doista višerječni izrazi ili nisu).

► PRIMJER

Microsoftov Office je od 1997. godine imao ugrađenog virtualnog asistenta Clippy, koji je pružao podršku korisnicima pri izvođenju određenih akcija, ovisno o ciljevima korisnika. Clippy je bio temeljen na Bayesovoj mreži. Varijable te mreže bile su akcije korisnika (npr., gdje je trenutačno pozicioniran pokazivač miša, je li korisnik u prošlosti odbijao ponudu za pomoć i sl.).

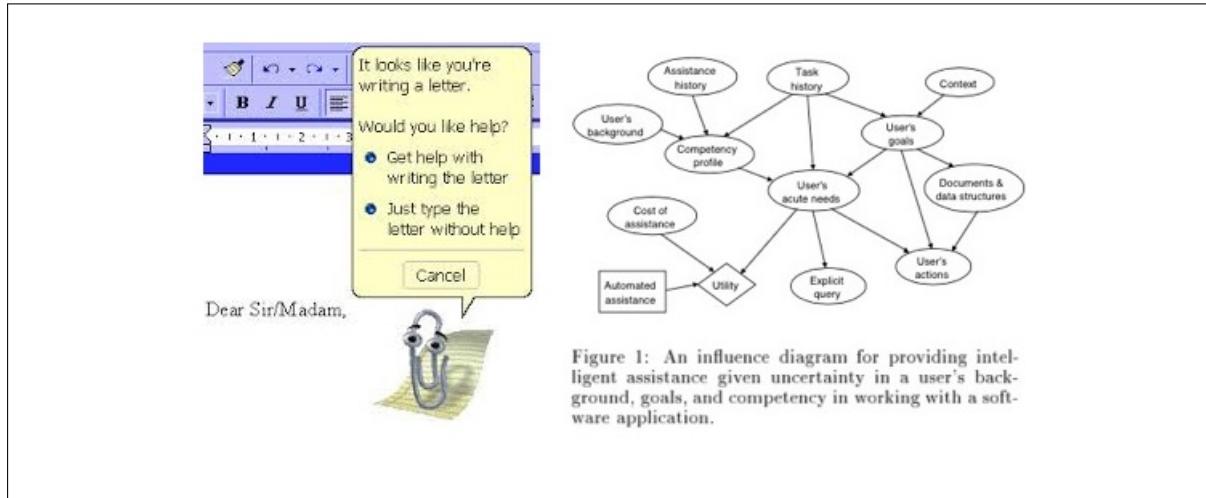
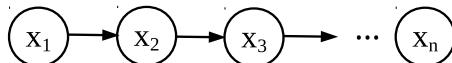


Figure 1: An influence diagram for providing intelligent assistance given uncertainty in a user's background, goals, and competency in working with a software application.

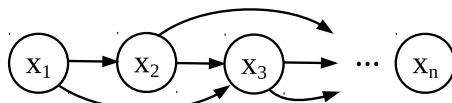
Još jedna važna vrsta PGM-ova su modeli za **slijedne podatke**. To mogu biti vremenski nizovi ili, npr., riječi u rečenici (kod obrade jezika) ili fonemi u riječi (kod prepoznavanja govora). Tipično se za slijedne podatke koriste Bayesove mreže koje nazivamo **Markovljev model**. Specifično, **Markovljev model prvog reda** zajedničku distribuciju faktorizira u lanac, na ovakav način:

$$p(\mathbf{x}) = p(x_1) \prod_{k=2}^n p(x_k|x_{k-1})$$

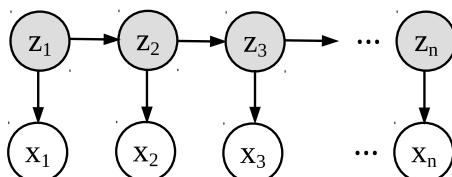


Lanac može odgovarati točkama u vremenu ili općenito bilo kakvom slijedu (npr., niz riječi u rečenici). Npr., ako su u pitanju riječi, ovim modelom možemo modelirati da vjerojatnost da se u rečenici pojavi neka riječ ovisi o riječi koja joj neposredno prethodi. Ako postoji zavisnosti koje idu dalje od samo jedne prethodne varijable, npr., ako želimo modelirati da riječ u rečenici ovisi o dvije prethodne riječi, možemo koristiti **Markovljev model drugog reda**:

$$p(\mathbf{x}) = p(x_1)p(x_2|x_1) \prod_{k=3}^n p(x_k|x_{k-1}x_{k-2})$$



No, što ako postoje još dalje zavisnosti? To onda postaje problematično: ne možemo u nedogled povećavati red modela, jer to, kao što znamo, brzo dovodi do eksplozije broja parametara. Rješenje je **skriveni Markovljev model** (engl. *Hidden Markov model, HMM*). Kod tog modela stanja procesa razdvojena su od podataka koje opažamo. Pretpostavljamo da je proces koji generira podatke doista Markovljev model prvog reda, no mi ne opažamo direktno stanja tog procesa, nego opažamo podatke koje taj proces generira i na koje utječe šum. Bayesova mreža HMM-a je ovakva:



Sam proces koji generira opažene podatke x_j nije opažen, tj. skriven je, i taj proces prolazi kroz stanja z_1, z_2, \dots, z_n (variabile koje nisu odgovaraju u gornjem prikazu sivo obojanim čvorovima). Ono što mi opažamo jesu podatci koje taj proces generira, tj. opažamo slijed x_1, x_2, \dots, x_n . Ovoj mreži odgovara sljedeća faktorizacija zajedničke distribucije:

$$p(\mathbf{x}, \mathbf{z}) = p(z_1)p(x_1|z_1) \prod_{k=2}^n p(z_k|z_{k-1})p(x_k|z_k)$$

HMM se, dakle, sastoji od **modela prijelaza** $p(z_k|z_{k-1})$ i **modela opažanja** $p(x_k|z_k)$. Model prijelaza opisuje prijelaz skrivenog procesa iz jednog stanja u drugi. Budući da variabile z_k ne opažamo u podatcima, već opažamo samo variablu x_k , to malo komplificira postupak zaključivanja i učenja. O tome više idući put.

4 D-odvajanje

Uvjetne nezavisnosti glavni su sastojak Bayesovih mreža. Iz zadane Bayesove mreže možemo izračunati zajedničku distribuciju. No, što ako nas zanima jesu li neke dvije variabile uvjetno nezavisne? Npr., u Bayesovoj mreži s travom i prskalicom, jesu li variabla "cloudy" i "wet grass" nezavisne? A jesu li nezavisne ako opažamo variablu "sprinkler" i "rain"? Sada se možda pitate zašto bi nam bilo bitno znati da su neke variabla uvjetno nezavisne. To je bitno s praktične strane, jer ako to znamo, možemo pri zaključivanju ukloniti one variabla koje su nezavisne od upita i time smanjiti broj variabli koje moramo analizirati. No, važnije, znati koje su variabla nezavisne bitno je s konceptualne strane, jer na taj način možemo zaključivati o tome koje statistička svojstva (zavisnost/nezavisnost variabli) proizlaze iz određene kauzalne strukture (definirane Bayesovom mrežom). Na primjer, ako variabla "cloudy" i "wet grass" nisu uvjetno nezavisne, onda znamo da između njih postoji neka kauzalna veza. S druge strane, ako su variabla "cloudy" i "wet grass" nezavisne uz opažene variablu "sprinkler" i "rain", onda znamo da su prskalica ili kiša već same po sebi dovoljne da uzrokuju mokru travu, tj. da se nalaze negdje u kauzalnom lancu između oblaka i mokre trave. Za ovaku vrstu zaključivanja o kauzalnim odnosima nužno je da možemo ispitati nezavisnosti i uvjetne nezavisnosti variabli i skupova variabli u Bayesovoj mreži.

Očito, kako smo pokazali u ranijem primjeru, primjenom **uređajnog Markovljevog svojstva** (ili višestrukom primjenom pravila umnoška i pravila zbroja) mogli bismo iz zadane Bayesove mreže izvesti sve uvjetne nezavisnosti koje su u njoj kodirane. Međutim, to je vrlo nepraktično. Osim toga, na taj način ne možemo odrediti zavisnost ili nezavisnost dviju proizvoljnih variabli iz Bayesove mreže, koje u toj mreži mogu biti vrlo udaljene jedna od druge. Srećom, Bayesove mreže nude jednu elegantnu mogućnost da se uvjetne nezavisnosti izvedu direktno iz strukture usmjerjenog grafa. Taj se postupak temelji na principu tzv. **d-odvajanja** (engl. *directed separation*).

Ideja d-odvajanja jest da analiziramo **staze** u grafu između čvorova x i y , koji odgovaraju varijablama čiju nezavisnost želimo ispitati, i da onda na temelju toga odredimo jesu li variabla uvjetno nezavisne. Staze između čvorova mogu biti **povezane** ili **odvojene**, već ovisno o tome koje su variabla u mreži opažene. Ako su sve staze između dva čvora odvojene, onda su čvorovi uvjetno nezavisni. Naravno, da bi ovo funkcionalo, moramo definirati kada su staze povezane a kada su odvojene. I to ćemo napraviti na temelju uređajnog Markovljevog svojstva. Prema tome, na d-odvajanje možemo gledati kao na proširenje uređajnog Markovljevog svojstva, koje opisuje lokalnu uvjetnu nezavisnost variabli, na nezavisnost između variabli koje su u mreži proizvoljno udaljene, a koje su povezane nekom stazom.

8

9

4.1 Pravila d-odvajanja

Konkretno, definirat ćemo tri jednostavna pravila. Ona opisuju tri osnovna slučaja koja mogu nastupiti između tri čvora u mreži: **račvanje**, **lanac** i **sraz**. Pravila ćemo izvesti razmatrajući sljedeće tri faktorizacije:

- | | | |
|--------------------------------------|---------------------------------|----------|
| (1) $p(x, y, z) = p(x z)p(y z)p(z)$ | $x \leftarrow z \rightarrow y$ | račvanje |
| (2) $p(x, y, z) = p(x)p(z x)p(y z)$ | $x \rightarrow z \rightarrow y$ | lanac |
| (3) $p(x, y, z) = p(x)p(y)p(z x, y)$ | $x \rightarrow z \leftarrow y$ | sraz |

Ideja je da promatramo kada je par varijabli x i y zavisan ili nezavisan, ovisno o tome je li varijabla u sredini, z , poznata (opažena) ili nije. Pogledajmo najprije **račvanje**. Iz uređajnog Markovljevog svojstva za varijablu y slijedi:

$$y \perp x | z \Leftrightarrow x \perp y | z$$

Iz ovoga sad možemo izvesti pravilo za račvanje na čvorove x i y od čvora z : ako je varijabla z **opažena**, onda su varijable x i y uvjetno nezavisne uz varijablu z , tj. čvorovi su **odvojeni**. Inače, ako varijabla z nije opažena, onda varijable x i y nisu nezavisne i njihovu su čvorovi povezani.

Pogledajmo sada **lanac**. Prema uređajnom Markovljevom svojstvu za varijablu y ponovo dobivamo:

$$y \perp x | z \Leftrightarrow x \perp y | z$$

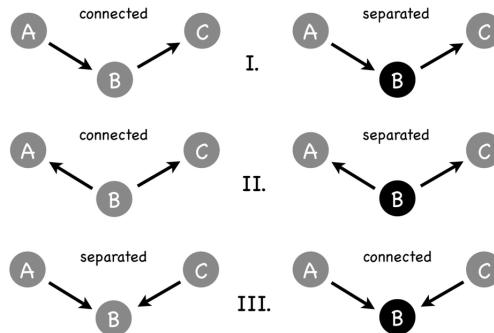
Iz toga izvodimo pravilo za lanac: ako čvorovi x i y čine lanac preko čvora z , i ako je varijabla z opažena, onda su varijable x i y uvjetno nezavisne uz varijablu z , tj. čvorovi su **odvojeni**. Inače, ako varijabla z nije opažena, varijable x i y nisu nezavisne i njihovi su čvorovi povezani.

Konačno, pogledajmo **sraz**. Uređajno Markovljevo svojstvo za varijablu y daje nam:

$$y \perp x | \emptyset$$

tj. varijable x i y su marginalno nezavisne (primijetimo da ti čvorovi nemaju roditelja, a da je čvor x topološki prethodnik čvora y). Dakle, za razliku od prethodna dva slučaja (račvanje i lanac), u ovom trećem slučaju (sraz) nismo dobili da su varijable x i y uvjetno nezavisne uz z . Umjesto toga, dobili smo da su varijable marginalno nezavisne. Dakle, slučaj sraza je upravo suprotan slučaju račvanja i slučaju lanca: opažanje srednje varijable u lancu čini varijable na krajevima lanca zavisnim, dok su inače nezavisne. To nas onda dovodi do našeg trećeg pravila: ako su čvorovi x i y u srazu preko čvora z , i ako je varijabla z **neopažena**, onda su varijable x i y nezavisne i njihovi su čvorovi **odvojeni**. Inače, ako je varijabla z opažena, varijable x i y su zavisne i njihovi su čvorovi povezani.

Sažeto, pravila su dakle ova:



Kako god, moramo priznati da je pravilo za sraz malo čudno. Pogledajmo to malo detaljnije.

4.2 Efekt objašnjavanja

O srazu je korisno razmišljati kao o situaciji u kojoj se dvije varijable **natječu** za objašnjavanje (odnosno uzrokovanje) treće varijable. Budući da i varijabla x i varijabla y svaka samostalno može objasniti/uzrokovati varijablu z , ono što se događa jest to da, ako znamo da se ostvarila z , onda se naše vjerovanje o tome je li se ostvario x mijenja ovisno o tome je li se ostvario y ili nije. Formalno:

$$x \not\perp y | z \Leftrightarrow p(x|z) \neq p(x|y, z)$$

Ako opazimo da su se ostvarili i y i z , to će smanjiti vjerojatnost da se ostvario x u odnosu na situaciju kada se ostvario samo z . (Također, vrijedi i obrnuto: ako opazimo da su se ostvarili i x i z , to će smanjiti vjerojatnost da se ostvario y u odnosu na situaciju kada se ostvario samo z). Ovaj se fenomen naziva **efekt objašnjavanja** (engl. *explaining away*). Efekt je zapravo vrlo intuitivan, kao što će pokazati sljedeći primjer.

► PRIMJER

Bacamo dva novčića. Neka glava odgovara vrijednosti 1, a pismo vrijednosti 0. Opažamo zbroj tih novčića. Zbroj može biti 0, 1 ili 2. Neka su novčići x i y , a njihov zbroj je z . Bayesova mreža odgovara strukturi sraza:

$$x \rightarrow z \leftarrow y$$

Očito, oba novčića utječu na zbroj. Apriorno, novčići su nezavisni. No, čim vidimo zbroj, oni postaju zavisni: npr., ako je zbroj 1, a prvi novčić je 0, onda drugi mora biti 1. Dakle, vrijedi $x \not\perp y | z$, i to je efekt objašnjavanja (opažanje zbroja novčića i jednog od dva novčića objašnjava vrijednost drugog novčića).

Efekt objašnjavanja samo je jedan primjer tzv. **međukauzalnog zaključivanja**, koje ljudi zapravo vrlo često (nesvjesno) koriste. Ipak, u nekim situacijama efekt se manifestira na ne-intuitivan način, i u takvim je situacijama poznat pod nazivom **Berksonov paradoks**. Sljedeći primjer ilustrira takvu situaciju.

► PRIMJER

Ekstreman primjer, koji se često navodi u literaturi, je sljedeći. Zamislimo da fakultet prima studente koji su visokointelligentni ili sportaši (ili oboje!). Neka P označava da je netko primljen na fakultet, što će biti istina ako je netko visokointelligentan (V) ili sportaš (S). Prepostavimo da su u općenitoj populaciji V i S nezavisni. Odnose možemo modelirati strukturu sraza:

$$V \rightarrow P \leftarrow S$$

U općenitoj populaciji $V \perp S$. Međutim, ako pogledamo samo studente koji su primljeni na fakultet (dakle one za koje opažamo $P = 1$), naći ćemo da su visokointelligentni studenti manje vjerojatno sportaši i obrnuto:

$$P(S = 1 | P = 1, V = 1) \leq P(S = 1 | P = 1)$$

To je zato što je svako svojstvo od ova dva samo po sebi dovoljno da objasni/uzrokuje P . Zbog toga ćemo u studentskoj populaciji imati negativnu koreliranost visokointelligentnih i sportaša.

4.3 D-odvajanje čvorova

Sada kada smo se upoznali sa sva tri slučaja koja se mogu dogoditi na stazi od tri čvora, možemo to poopćiti na slučaj dvaju čvorova koja se nalaze bilo gdje u grafu. To nas dovodi do ideje **d-odvajanja čvorova**.

11

12

► D-odvajanje čvorova

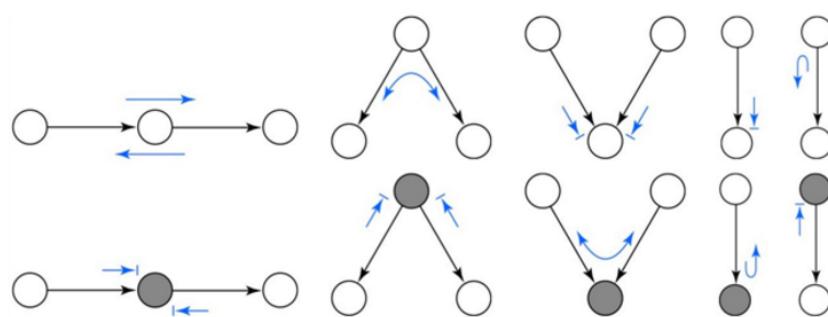
Raspolažemo skupom varijabli E koje su opažene. Za **stazu** P od čvora x do čvora y kažemo da je **d-odvojena** (engl. *d-separated*) ako i samo ako vrijedi barem jedno od sljedećeg:

- (1) P sadrži **lanac** $x \rightarrow z \rightarrow y$ ili $x \leftarrow z \leftarrow y$ i $z \in E$
- (2) P sadrži **račvanje** $x \leftarrow z \rightarrow y$ i $z \in E$
- (3) P sadrži **sraz** $x \rightarrow z \leftarrow y$ i varijabla z nije u E i niti jedan sljedbenik od z nije u E

Za **par čvorova** x i y kažemo da su čvorovi x i y **d-odvojeni** za dani E ako su sve staze između ta dva čvora d-odvojene za dani E . Čvorovi x i y su uvjetno nezavisni za dani E ako i samo ako su d-odvojeni za dani E .

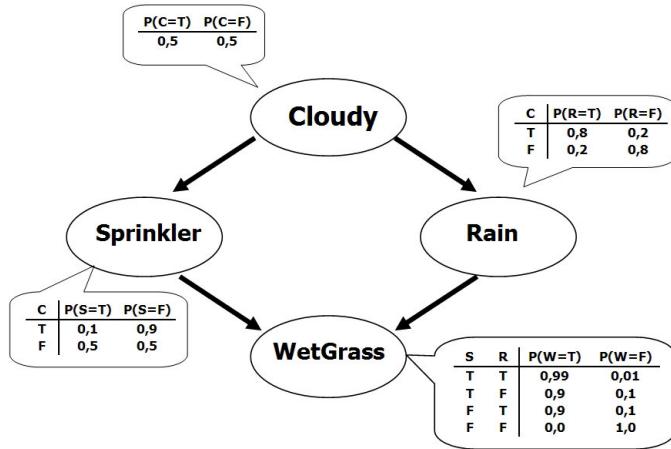
Dakle, da bismo utvrdili jesu li, za neko zadano opažanje varijabli, dva čvora uvjetno nezavisna, trebamo analizirati jesu li sve staze između njih d-odvojene. Ako jesu, onda su čvorovi uvjetno nezavisni, inače to nisu.

Ispitivanje d-odvojenosti lako se može pretočiti u algoritam. Naivan algoritam bio bi da ispitamo sve moguće staze u mreži između dva čvora. Međutim, za velike mreže to bi moglo biti skupo, pa su razvijeni učinkovitiji algoritmi čija složenost linearno ovisi o broju čvorova. Jedan takav algoritam je **Bayesova kuglica** (engl. *Bayes-Ball*). Algoritam funkcioniра tako da “aktiviramo” sve čvorove čije varijable opažamo stavimo kuglice u jedan skup varijabli, i onda ih pustimo da se kreću po mreži (neovisno o usmjerenju bridova) prema pravilima koja odgovaraju d-odvajanju. Ukupno postoji 10 pravila:



Pravila definiraju kroz koje čvorove kuglica može proći (plave strelice), a kroz koje ne može (plave strelice s vertikalnom crtom), u ovisnosti o tome koje su varijable opažene (sivi čvorovi odgovaraju opaženim varijablama). Ako ijedna kuglica dosegne drugi skup čvorova, to onda znači da odgovarajuće varijable nisu d-separirane, tj. da nisu uvjetno nezavisne. Nećemo ići u detalje tog algoritma. Umjesto toga, pogledajmo opet primjer sa travom i prskalicom, i pokažimo na toj mreži kako funkcioniра d-odvajanje.

► PRIMJER



Analizirajmo kada će varijabla "cloudy" biti uvjetno nezavisna od varijable "wet grass". Između te dvije varijable, odnosno između njihovih čvorova, imamo dva lanca. Prema pravilu d-odvajanja, oba će ova lanca biti odvojena ako opažamo i varijablu "sprinkler" i varijablu "rain". Dakle, varijable "cloudy" i "wet grass" su uvjetno nezavisne ako opažamo varijable "sprinkler" i "rain" (obje!). Intuitivno, to znači da možemo zaključiti da je trava mokra a da pritom ne znamo je li oblačno samo ako znamo radi li prskalica za travu i je li pada kiša. Drugačije rečeno, prskalica i kiša su mogući uzroci (objašnjenja) za mokru travu, i ako opažamo i jedno i drugo, onda imamo dovoljnu informaciju da zaključimo i njihovoj kauzalnoj posljedici (mokroj travi).

Pogledajmo sada jesu li varijable "sprinkler" i "rain" nezavisne? Ove su varijable povezane dvjema stazama: jednim račvanjem (preko varijable "cloudy") i jednim srazom (preko varijable "wet grass"). Sukladno pravilima d-odvajanja, ako opažamo "cloudy", onda je ta staza odvojena. Suprotno, ako ne opažamo varijablu "wet grass", onda je i ta staza odvojena. Dakle, varijable "sprinkler" i "rain" uvjetno su nezavisne ako opažamo varijablu "cloudy", a ne opažamo varijablu "wet grass". S druge strane, ako opažamo varijablu "wet grass", varijable "sprinkler" i "rain" postaju uvjetno nezavisne. To je efekt objašnjavanja: premda su varijable "sprinkler" i "rain" nezavisne (za opažanu varijablu "cloudy"), ako opažamo njihov zajednički čvor-dijete, "wet grass", onda te varijable postaju uvjetno zavisne. Npr., ako je trava mokra a znamo da pada kiša, onda se smjenuje vjerojatnost da je trava mokra zbog prskalice.

Sažetak

- **Probabilistički grafički modeli (PGM-ovi)** na sažet način prikazuju **zajedničku distribuciju** kao graf u kojem su čvorovi varijable a prijelazi **zavisnosti** između njih
- Tri aspekta PGM-a: **reprezentacija, zaključivanje i učenje**
- **Bayesove mreže** su PGM-ovi s usmjerenim acikličkim grafovima, a **Markovljeve mreže** s neusmjerenim grafovima
- Bayesove mreže kodiraju uvjetne nezavisnosti između varijabli preko **uredajnog Makov-ljevog svojstva**
- Pomoću pravila **d-odvajanja** možemo odrediti je li par varijabli u mreži (uvjetno) nezavisan

Bilješke

¹ Glavna i doista sveobuhvatna referenca za probabilističke grafičke modele je ([Koller and Friedman, 2009](#)), koju toplo preporučam da razmotrite kao jedan cijelogodišnji projekt. Izvrsna referenca je i ([Darwiche, 2009](#)), koja je mnogo sažetija. Kraće pregledi Bayesovih mreža i PGM-ova, po kojima

je uglavnom i strukturirano ovo predavanje, možete naći u (Alpaydin, 2020) (poglavlje 14), (Bishop, 2006) (poglavlja 8.1–2, 11.2–3 i 13.1) te (Murphy, 2012) (poglavlje 10).

- [2] Za PGM-ove se često kaže da predstavljaju (sretan) brak teorije vjerojatnosti i teorije grafova. Osnovna ideja je **modularizacija**: složeni sustav gradi se kombinacijom jednostavnijih. Teorija vjerojatnosti služi kao “ljepilo” koje povezuje dijelove sustava i povezuje model s podatcima. S druge strane, teorija grafova nudi sučelje prema čovjeku, koje omogućava modeliranje na visokoj razini apstrakcije.
- [3] Ovdje jedna terminološka napomena o nazivu **Bayesove mreže** (na engleskom “Bayes networks”, ali češće “Bayesian networks”), i alternativnim nazivima **mreže vjerovanja** (engl. *belief networks*) i **kauzalne mreže** (engl. *causal networks*). Uglavnom je prihvaćen stav da ovi nazivi nisu baš najsretniji. Kevina Murphy u Murphy (2012) vrlo jasno objašnjava zašto: *“A directed graphical model or DGM is a GM whose graph is a DAG. These are more commonly known as Bayesian networks. However, there is nothing inherently ‘Bayesian’ about Bayesian networks: they are just a way of defining probability distributions. These models are also called belief networks. The term ‘belief’ here refers to subjective probability. Once again, there is nothing inherently subjective about the kinds of probability distributions represented by DGMs. Finally, these models are sometimes called causal networks, because the directed arrows are sometimes interpreted as representing causal relations. However, there is nothing inherently causal about DGMs.”* Murphy stoga predlaže korisiti naziv **directed graphical model (DGM)**. Mi ćemo ipak koristiti naziv **Bayesove mreže**, primjećujući, usput, da time radimo manju pogrešku nego da koristimo naziv **Bayesovske mreže**, budući da Bayesove mreže ne impliciraju da koristimo bayesovsku statistiku (koju i nećemo koristiti).
- [4] Bayesove mreže mogu se koristiti za modeliranje **kauzalnosti**, no one same po sebi, bez dodatnih prepostavki i mehanizama, nisu nužno kauzalni model. Ako vas zanima veza između Bayesovih mreža i kauzalnosti, preporučam pročitati radeve izraelsko-američkoga znanstvenika i filozofa Judea Pearla, koji je najpriznatiji stručnjak u području kauzalnog zaključivanja. O vezi između Bayesovih mreža i kauzalnih mreža možete pročitati u (Pearl, 1995), a o kauzalnim modelima općenito u (Pearl, 2000). Pristupačan uvod u Pearlov rad je (Pearl and Mackenzie, 2018). Više možete naći na njegovoj web-stranici: http://bayes.cs.ucla.edu/jp_home.html. Kauzalnost je važan koncept u strojnem učenju, pogotovo u kontekstu izgradnje pravednih (nepristranih) i tumačivih (interpretabilnih) modela. O kauzalnosti u kontekstu strojnog učenja možete pročitati u (Schölkopf, 2019).
- [5] Model “alarm network” opisan je u (Beinlich et al., 1989).
- [6] Model za prepoznavanje višerječnih jedinica u tekstovima na hrvatskome jeziku opisan je u (Buljan and Šnajder, 2017).
- [7] Clippy je bio razvijen u okviru Microsoftovog projekta Lumière, čiji je fokus bio primjena Bayesovog zaključivanja za automatiziranu pomoć korisnicima. Voditelj tima bio je američki znanstvenik Eric Horvitz, danas vrlo poznat u AI krugovima, posebno po svojim radovima u području zaključivanja uz ograničene resurse. Unatoč početnom entuzijazmu, deset godina nakon uvođenja u Office 97, Clippy je u tišini eliminiran, jer je uglavnom živcirao korisnike. Razlozi za to su razni, uključivo i loša predikcija cilja korisnika. Za to krivnju ne treba svaljivati na Bayesove mreže, već na prejednostavan model. Naime, neke od očitih nedostataka modela bile su što model nije u obzir uzimao profil korisnika (npr. stupanj stručnosti, kao i činjenicu da korisnik kroz vrijeme poboljšava svoju vještina korištenja Officea) te je akcije modelirao kao atomičke cjeline, a ne kao složenje nizove atomičkih akcija. Službeno Microsoftovo objašnjenje bilo je, vrlo mudro, da je “Office XP toliko jednostavan za korištenje da Clippy više nije niti potreban niti koristan” (<https://tinyurl.com/yxqbnqw2>). A ovdje je detaljna analiza zašto korisnici nisu voljeli Clippya: <http://xenon.stanford.edu/~lswartz/paperclip/>.
- [8] Modeliranje kauzalnih struktura kompleksnih sustava (bioloških, društvenih, psiholoških, tehničkih) osnovna je motivacija za razvoj Bayesovih mreža, odnosno PGM-ova općenito. Kod takvih je modela ključno imati na raspolaganju metode pomoću kojih možemo izvesti “statističke posljedice” zadane kauzalne strukture. Znanstvenici su se ovim metodama bavili otprilike razvoja umjetne inteligencije; važni su u tom kontekstu radovi Herberta Simona i Huberta Blalocka s kraja šezdesetih godina. Međutim, problem je adekvatno riješen tek uvođenjem koncepta **d-odvajanja** sredinom osamdesetih

godina, u radovima Judea Pearla, Dana Geigera i Thomasa Verma sa Sveučilišta u Kaliforniji. Ovo je njihov izvorni rad: ([Geiger et al., 1990](#)).

- 9** Već smo rekli da Bayesove mreže općenito ne moraju nužno modelirati stvarnu kauzalnost. Je li neki odnos između dvaju fenomena, kojima odgovaraju dvije slučajne varijable, doista kauzalan je svojstvo koja ne proizlazi iz same Bayesove mreže već dolazi izvana, već je pretpostavka koju unosimo mi sami, kod analize mreže, a na temelju našeg postojećeg znanja ili pretpostavki o kauzalnim odnosima u stvarnom svijetu. Da je Bayesovo mreži doista svejedno kako modeliramo odnose između varijabli postaje jasno kada shvatimo da je su sljedeće dvije mreže ekvivalentne:

$$\begin{aligned} x &\rightarrow y \rightarrow z \\ x &\leftarrow y \leftarrow z \end{aligned}$$

Naime, obje mreže modeliraju zajedničku vjerojatnost $p(x, y, z)$ i obje mreže imaju isti broj parametara. Vidimo da je smjer bridova nevažan, međutim kod modeliranja kauzalnosti smjer je, naravno, bitan. To znači da, ako pokažemo da su dvije varijable duž neke staze u Bayesovoj mreži zavisne, jesu li te varijable kauzalno povezane i koji je smjer kauzalnosti možemo reći samo ako se oslonimo na dodatno znanje ili pretpostavke.

- 10** Tri **pravila d-odvajanja** izveli smo primjenom uredajnog Markovlјeg svojstva. Ista smo pravila, međutim, mogli izvesti i primjenom pravila umnožaka i pravila zbroja. Pokažimo to. Krenimo od pravila za **račvanje**. Podijelimo lijevu i desnu stranu izraza za zajedničku vjerojatnost za $p(z)$:

$$\begin{aligned} \frac{p(x, y, z)}{p(z)} &= \frac{p(x|z)p(y|z)p(z)}{p(z)} \\ p(x, y|z) &= p(x|z)p(y|z) \end{aligned}$$

Iz ovoga, po definiciji uvjetne nezavisnosti, slijedi $x \perp\!\!\!\perp y|z$. Primijetite međutim da ne vrijedi marginalna nezavisnost, tj. ne vrijedi $x \perp\!\!\!\perp y$. U to se možemo uvjeriti marginalizacijom lijeve i desne strane po varijabli z (tj. primjenom pravila zbroja):

$$\begin{aligned} \sum_z p(x, y, z) &= \sum_z p(x|z)p(y|z)p(z) \\ p(x, y) &= \sum_z p(x|z)p(y|z)p(z) \end{aligned}$$

Desna strana se općenito ne faktorizira u umnožak $p(x)p(y)$, pa zaključujemo $x \not\perp\!\!\!\perp y$. Slično možemo izvesti i za druga dva slučaja. Za slučaj lanca, prema uredajnom Markovlјevom svojstvu dobivamo:

$$y \perp\!\!\!\perp x|z \Leftrightarrow x \perp\!\!\!\perp y|z$$

I opet smo to mogli izvesti tako da lijevu i desnu stranu podijelimo sa $p(z)$:

$$\begin{aligned} p(x, y, z) &= p(x)p(z|x)p(y|z) \\ \frac{p(x, y, z)}{p(z)} &= \overbrace{\frac{p(x)p(z|x)}{p(z)}}^{p(x,z)} p(y|z) \\ p(x, y|z) &= p(x|z)p(y|z) \end{aligned}$$

Slično kao i u prvom slučaju, marginalizacijom po z nećemo moći izvesti faktorizaciju $p(x)p(y)$, iz čega zaključujemo da varijable x i y nisu marginalno nezavisne, $x \not\perp\!\!\!\perp y$. Konačno u trećem slučaju, uredajno Markovlјevu svojstvo daje nam:

$$y \perp\!\!\!\perp x|\emptyset$$

tj. x i y su marginalno nezavisni (primijetimo da oni nemaju roditelja, a da je x prethodnik od y). Isto smo mogli dobiti marginalizacijom po varijabli z :

$$\begin{aligned} \sum_z p(x, y, z) &= \sum_z p(x)p(y)p(z|x, y) \\ p(x, y) &= p(x)p(y) \sum_z p(z|x, y) = p(x)p(y) \end{aligned}$$

S druge strane, ako uvjetujemo na varijablu z :

$$\frac{p(x, y, z)}{p(z)} = \frac{p(x)p(y)p(z|x, y)}{p(z)}$$

Ovo se dalje ne faktorizira na $p(x|z)p(y|z)$, pa zaključujemo da varijable x i y nisu uvjetno nezavisne za zadani z , tj. $x \not\perp\!\!\! \perp y | z$.

- [11] Razmatranje **efekta objašnjavanja** s aspekta **međukausalnog zaključivanja** možete naći u ([Wellman and Henrion, 1993](#)). No, evo možda uvjerljivijeg primjera koji ilustrira da u svakodnevnom zaključivanju često koristimo efekt objašnjavanja. Ako imate visoku temperaturu, i bojite se da možda imate COVID-19, bit će vam jako draga kada vam liječnik kaže da imate običnu upalu grla. Očito, to što imate upalu grla ne sprječava da imate i COVID-19. Međutim, to što imate upalu grla dobro objašnjava vaše simptome, i time značajno smanjuje vjerojatnost da imate COVID-19. Svejedno, često perite ruke.
- [12] Prema američkom fizičaru i statističaru Josephu Barksonu, koji je taj fenomen prvi primijetio. Berkson je također uveo naziv **logit** (logaritam omjera šansi), koji smo bili spomenuli prošli put.
- [13] Algoritam **Bayesove kuglice** osmislio je 1990. godine Ross D. Shachter sa Sveučilišta Stanford ([Shachter, 1998](#)). Za alternativne algoritme, također temeljene na d-odvajanju, pogledajte ([Koller and Friedman, 2009](#)) (poglavlje 3.3.3) i ([Darwiche, 2009](#)) (poglavlje 4.5).

Literatura

- E. Alpaydin. *Introduction to machine learning*. MIT press, 2020.
- I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. In *AIME 89*, pages 247–256. Springer, 1989.
- C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
- M. Buljan and J. Šnajder. Combining linguistic features for the detection of croatian multiword expressions. In *Proceedings of the 13th Workshop on Multiword Expressions (MWE 2017)*, pages 194–199, 2017.
- A. Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge university press, 2009.
- D. Geiger, T. Verma, and J. Pearl. d-separation: From theorems to algorithms. In *Machine Intelligence and Pattern Recognition*, volume 10, pages 139–148. Elsevier, 1990.
- D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
- K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.
- J. Pearl. From bayesian networks to causal networks. In *Mathematical models for handling partial knowledge in artificial intelligence*, pages 157–182. Springer, 1995.
- J. Pearl. *Causality: Models, reasoning and inference*. 2000.
- J. Pearl and D. Mackenzie. *The book of why: the new science of cause and effect*. Basic Books, 2018.
- B. Schölkopf. Causality for machine learning. *arXiv preprint arXiv:1911.10500*, 2019.

R. Shachter. Bayes-ball: The rational pastime for determining irrelevance and requisite information in belief networks and influence diagrams. In *Uncertainty in Artificial Intelligence*, 1998.

M. P. Wellman and M. Henrion. Explaining 'explaining away'. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(3):287–292, 1993.

18. Probabilistički grafički modeli II

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

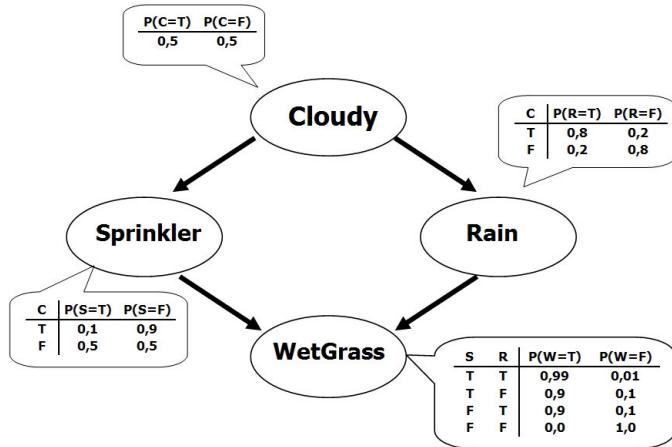
Jan Šnajder, predavanja, v2.1

Prošli smo puta pričali općenito o **probabilističkim grafičkim modelima** (PGM-ovima). Rekli smo da su to modeli koji na sažet način prikazuju zajedničku distribuciju, preko faktorizacije nad strukturom grafa. Rekli smo i da su tri aspekta PGM-ova **reprezentacija, zaključivanje i učenje**. Zatim smo se fokusirali na Bayesove mreže i pričali o reprezentaciji kod Bayesove mreže. Vidjeli smo kako Bayesova mreža kodira uvjetne nezavisnosti, vidjeli smo neke primjere Bayesovih mreža i zatim smo vidjeli kako možemo ispitati uvjetnu nezavisnost para varijabli pomoću metode d-odvajanja, što je pogotovo važno ako Bayesovom mrežom želimo modelirati i analizirati kauzalne odnose.

Danas nastavljamo pričati o Bayesovim mrežama, i to o njihova druga dva aspekta: zaključivanju i učenju. Kao što smo bili napomenuli prošli put, PGM-ovi su ogromno područje unutar strojnog učenja, i nemamo naravno ambiciju sve pokriti ovim predavanjem. U neke ćemo stvari ući malo više, no većina toga bit će ipak na razini informacije, u nadi da oni koje to zanima mogu dalje više informacija potražiti sami.

1 Zaključivanje

Rekli smo već da je najčešći zadatak koji želimo raditi s probabilističkim grafičkim modelima, a tako i s Bayesovom mrežom, **probabilističko zaključivanje**. Prisjetimo se primjera Bayesove mreže s prskalicom za travu i kišom.



Na primjer, ako opažamo da je trava mokra ($w = 1$), što možemo zaključiti o prskalici (s) i kiši (r) – što je od tog dvoje vjerojatnije? To, zapravo, znači da nas zanimaju **uvjetne vjerojatnosti** $P(s = 1|w = 1)$ i $P(r = 1|w = 1)$. A njih, po definiciji uvjetne vjerojatnosti, možemo izračunati na sljedeći način:

$$P(s = 1|w = 1) = \frac{P(s = 1, w = 1)}{P(w = 1)}$$

$$P(r = 1|w = 1) = \frac{P(r = 1, w = 1)}{P(w = 1)}$$

Vjerojatnost koja se pojavljuje u nazivniku, $P(w = 1)$, naziva se **vjerojatnost dokaza** (u našem slučaju dokaz je činjenica da je trava mokra).

U ovim se izrazima javlja **zajednička vjerojatnost** dvije varijable (u brojniku razlomka) i marginalna vjerojatnost (u nazivniku razlomka). Sve te vjerojatnosti možemo izračunati iz Bayesove mreže. Prisjetimo se, naime, da Bayesova mreža upravo služi tome da na sažet način kodira zajedničku vjerojatnost. Konkretno, u ovom našem primjeru imamo (iščitavanjem iz Bayesove mreže):

$$P(c, s, r, w) = P(c)P(s|c)P(r|c)P(w|r, s)$$

Nadalje, prisjetimo se da iz ove zajedničke vjerojatnosti možemo izračunati bilo koju drugu vjerojatnost s ove četiri varijable, tako da napravimo prikladnu **marginalizaciju**. Konkretno, kako bismo izračunali tražene dvije uvjetne vjerojatnosti, zajedničku vjerojatnost $P(c, s, r, w)$ marginalizirat ćemo u brojniku i nazivniku izraza za uvjetnu vjerojatnost na sljedeći način:

$$\begin{aligned} P(s = 1|w = 1) &= \frac{P(s = 1, w = 1)}{P(w = 1)} = \frac{\sum_{c,r} P(c, s = 1, r, w = 1)}{\sum_{c,r,s} P(c, s, r, w = 1)} \\ P(r = 1|w = 1) &= \frac{P(r = 1, w = 1)}{P(w = 1)} = \frac{\sum_{c,s} P(c, s, r = 1, w = 1)}{\sum_{c,r,s} P(c, s, r, w = 1)} \end{aligned}$$

Iščitavanjem konkretnih vrijednosti ovih vjerojatnosti iz Bayesove mreže i uvrštavanjem u gornje formule, dobivamo:

$$\begin{aligned} P(w = 1) &= \sum_{c,r,s} P(c, s, r, w = 1) = 0.6471 \\ P(s = 1|w = 1) &= \frac{\sum_{c,r} P(c, s = 1, r, w = 1)}{\sum_{c,r,s} P(c, s, r, w = 1)} = \dots = 0.2781/0.6471 = 0.43 \\ P(r = 1|w = 1) &= \frac{\sum_{c,s} P(c, s, r = 1, w = 1)}{\sum_{c,r,s} P(c, s, r, w = 1)} = \dots = 0.4851/0.6471 = 0.708 \end{aligned}$$

Možemo zaključiti da je, ako opažamo mokru travu, vjerojatnije da pada kiša nego da je upaljena prskalica. Uključivanjem pozadinskog znanja o kauzalnim odnosima mogli bismo zaključiti da je vjerojatnije da je trava mokra zbog kiše nego zbog prskalice. Također smo izračunali da je vjerojatnost dokaza jednaka otprilike 0.6, što znači da mokru travu opažamo u 60% slučajeva.

Iz ovog smo primjera vidjeli da sve vjerojatnosti koja nas zanimaju možemo izračunati iz zajedničke vjerojatnosti prikladnom marginalizacijom i uvjetovanjem na opažane varijable. U tom smislu možemo reći da zajednička distribucija sadrži potpunu informaciju, iz koje se mogu derivirati sve druge distribucije koje nas zanimaju.

Općenito, **problem zaključivanja** kod PGM-ova može se definirati ovako: postoje varijable koje su vidljive odnosno **opažene** (engl. *observed*) i varijable koje nisu opažene odnosno koje su **skrivene** (engl. *hidden*). Npr., u našem primjeru, varijabla w bila je opažena, dok su sve druge varijable bile skrivene. Nadalje, od varijabli koje su skrivene, za neke nas baš zanima koja je njihova vjerojatnost, dok nas je za neke baš briga. Ove prve, koje nas zanimaju, nazivamo **varijable upita** (engl. *query variables*), a ove druge, koje nas ne zanimaju, nazivamo **varijable smetnje** (engl. *nuisance variables*). U našem primjeru, varijable upita bile su s (u prvom upitu) i r (u drugom upitu), dok su preostale varijable bile varijable smetnje.

Definirajmo ovo malo općenitije. Za neki upit, skup varijabli možemo razdijeliti u tri disjunktna skupa: **skup opaženih varijabli x_o** , **skup varijabli upita x_q** i **skup varijabli smetnje x_n** . Nakon što smo tako razdijelili varijable, možemo raditi različite upite. Dvije osnovne vrste upita su **aposteriorni upiti** i **MAP-upiti**.

Kod **aposteriornih upita**, zanima kakva je aposteriorna distribucija skrivenih varijabli za zadane opažene varijable. Dakle, neke varijable smo opazili, i sada nas zanima koja je vjerojatnost

vrijednosti preostalih varijabli. To jest, zanima nas koliko iznosi sljedeća uvjetna vjerojatnost:

$$p(\mathbf{x}_q | \mathbf{x}_o) = \frac{\sum_{\mathbf{x}_n} p(\mathbf{x}_q, \mathbf{x}_o, \mathbf{x}_n)}{p(\mathbf{x}_o)} = \frac{\sum_{\mathbf{x}_n} p(\mathbf{x}_q, \mathbf{x}_o, \mathbf{x}_n)}{\sum_{\mathbf{x}_n, \mathbf{x}'_q} p(\mathbf{x}'_q, \mathbf{x}_o, \mathbf{x}_n)}$$

Primijetite kako smo u brojniku izmarginalizirali varijable smetnje \mathbf{x}_n . U nazivniku smo, uz varijable smetnje, izmarginalizirali i varijable upita \mathbf{x}_q , kako bismo dobili vjerojatnost dokaza. Vidimo da je ovo upravo ono što smo bili radili u našem ranijem primjeru. Pritom je u prvoj upitu s bila varijabla upita, w je bila opažena varijabla, a c i r su bile varijable smetnje. U drugom upitu, r je bila varijabla upita, w je opet bila opažena varijabla, a c i s varijable smetnje.

Druga vrsta upita koja bi nas mogla zanimati jesu **MAP-upiti**, tj. upiti **maksimum aposteriori**. MAP-upiti također se nazivaju **najvjerojatnije objašnjenje** (engl. *most probable explanation, MPE*). Kod MAP-upita, zanima nas koje su najvjerojatnije vrijednosti za sve varijable upita, uz dane opažene varijable. Formalno:

$$\mathbf{x}_q^* = \operatorname{argmax}_{\mathbf{x}_q} \sum_{\mathbf{x}_n} p(\mathbf{x}_q, \mathbf{x}_o, \mathbf{x}_n)$$

Dakle, kod ove vrste upita ne zanima nas distribucija vjerojatnosti varijabli upita, nego nas zanima najvjerojatnije pridjeljivanje vrijednosti tim varijablama. Primijetite da najvjerojatnija vrijednost za vektor \mathbf{x}_q nije nužno ista ona kao i kombinacija zasebno najvjerojatnijih vrijednosti za svaku pojedinačnu varijablu iz \mathbf{x}_q , kao što pokazuje sljedeći primjer.

► PRIMJER

Najvjerojatniji vektor \mathbf{x} ne mora biti isto kao i kombinacija zasebno najvjerojatnijih komponenti tog vektora. Npr., ako je zajednička vjerojatnost $P(x, y)$ ovakva:

	x_1	x_2
y_1	0.4	0.3
y_2	0	0.3

onda je vektor (x_1, y_1) najvjerojatnija kombinacija para varijabli x i y , premda vrijednost x_1 zasebno gledano nije najvjerojatnija ($P(x_1) = 0.4$, ali $P(x_2) = 0.6$).

U našem primjeru s prskalicom i travom, MAP-upit bi na primjer mogao glasiti: ako opažamo da je trava mokra i da ne pada kiša, što je najvjerojatnija vrijednost za par skrivenih varijabli (c, s) , tj. što je aposteriorno najvjerojatnije stanje oblačnosti i prskalice, ako apriorno znamo (jer to opažamo) da je trava mokra i da kiša ne pada.

2 Zaključivanje: Eliminacija varijabli

Kao što smo vidjeli, u načelu se obje ove vrste upita mogu odgovoriti tako da se najprije izgradi zajednička distribucija $p(\mathbf{x}_q, \mathbf{x}_o, \mathbf{x}_n)$, a potom se ili marginalizira pa normira (aposteriorni upiti) ili se nađe kombinacija s najvećom vjerojatnošću (MAP-upiti).

Međutim, takav je pristup naivan. Što je problem s takvim pristupom? Problem je što će izgradnja pune zajedničke distribucije pa njezina marginalizacija neminovno dovesti do **kombinatorne eksplozije**. Naime, u realnim je problemima broj varijabli vrlo velik, pa postoji mnogo kombinacija njihovih vrijednosti. Općenito, ako je distribucija diskretna i n -dimenzijska, a svaka varijabla ima K mogućih vrijednosti, imat ćemo ukupno K^n kombinacija, tj. mogućih vrijednosti n -dimenzijskoga slučajnog vektora. Marginalizacija zajedničke distribucije iziskivat će onda množenje (da dobijemo zajedničku vjerojatnost) i zbrajanje (da provedemo marginalizaciju) velikog broja faktora, što će biti računalno zahtjevno. Zapravo, kada bismo to tako radili,

u potpunost bismo poništili korist koju imamo od Bayesovih mreža – sažet zapis zajedničke distribucije!

Budući da je broj kombinacija eksponencijalan, pokazuje se da je zaključivanje kod Bayesovih mreža **NP-težak** problem. Međutim, postoje mudriji pristupi koji izbjegavaju eksplizitnu konstrukciju zajedničke vjerojatnosti. Alternative dolaze u dva okusa: **egzaktno zaključivanje** i **približno zaključivanje**.

Osnovni postupak egzaktnog zaključivanja je tzv. **eliminacija varijabli**. Eliminacija varijabli koristi **dinamičko programiranje**. Ideja je da se, umjesto da se prvo konstruira zajednička distribucija pa da se marginalizira (odnosno da se radi “izvana prema unutra”), da se ona gradi postepeno i da se pohranjuju međurezultati (to je onda “iznutra prema van”). Ovo je moguće zato što varijable u Bayesovoj mreži imaju lokalne zavisnosti: većina varijabli zavisi o manjem broju varijabli koje odgovaraju susjednim čvorovima. To znači da, kada računamo zajedničku vjerojatnost na temelju Bayesove mreže, mnogi će se izrazi ponavljati, pa je dovoljno da ih izračunamo samo jednom i pohranimo (memoiziramo) tu izračunatu vrijednost kako je poslije ne bismo morali ponovo računati.

► PRIMJER

Pogledajmo kako bi egzaktno zaključivanje izgledalo na našem primjeru s prskalicom za travu. Recimo da želimo izračunati distribuciju $p(w)$, dakle vjerojatnosti $P(w = 0)$ i $P(w = 1)$. (Iz didaktičkog razloga, zanemarimo sada činjenicu da je dovoljno da izračunamo jedno od tog dvoje pa da znamo ono drugo. Primjer bi imao više smisla kada bi varijabla w imala više od dvije vrijednosti.) Na temelju naše Bayesove mreže, vjerojatnost $P(w)$ možemo računati ovako:

$$\begin{aligned} P(w) &= \sum_c \sum_s \sum_r P(c, s, r, w) \\ &= \sum_c \sum_s \sum_r P(c)P(s|c)P(r|c)P(w|s, r) \end{aligned}$$

Problem kada radimo ovako jest u složenosti izračuna, odnosno u broju računskih operacija. Koliko računskih operacija ovdje imamo? Prvo, prisjetimo se da su sve varijable ovdje binarne. Ove tri sume generirat će, dakle, $2 \times 2 \times 2 = 8$ pribrojnika. Nadalje, svaki je pribrojnik umnožak od četiri faktora. Dakle, računanje svakog pribrojnika iziskuje 3 operacije množenja, i to ponavljamo 8 puta, dakle imamo 24 operacije množenja. Zatim zbrajamo te pribrojниke, što je još 7 operacija zbrajanja, pa je ukupan broj operacija 31. Na koncu to još trebamo ponoviti dva puta, za $w = 0$ i $w = 1$, što dakle daje ukupno 62 operacije.

Izračun postaje jednostavniji ako izlučimo zajedničke faktore. Na primjer:

$$\begin{aligned} P(w) &= \sum_c \sum_s \sum_r P(c, s, r, w) \\ &= \sum_c \sum_s \sum_r P(c)P(s|c)P(r|c)P(w|s, r) \\ &= \sum_c P(c) \sum_s P(s|c) \sum_r P(r|c)P(w|s, r) \end{aligned}$$

Iz jednostavnog razloga što se izlučeni članovi javljaju u izrazu samo jednom, to dovodi do smanjenja broja operacija koje moramo izračunati. Konkretno, ovdje bi to bile 42 operacije. Međutim, na ovaj način nismo ostvarili uštedu koju možemo ostvariti ako pohranjujemo izračune koji se ponavljaju. U gornjem izrazu niti jedan se izračun zapravo ne ponavlja.

Veću uštedu u broju izračuna možemo napraviti ako najprije presložimo redoslijed fakora, na sljedeći način:

$$\begin{aligned} P(w) &= \sum_s \sum_r \sum_c P(c, s, r, w) \\ &= \sum_s \sum_r P(w|s, r) \sum_c P(c)P(s|c)P(r|c) \end{aligned}$$

Prednost ovakvog izračuna jest u tome što izraz unutar \sum_c ne ovisi o varijabli c (jer po njoj marginaliziramo) ni o varijabli w (zbog lokalnosti, odnosno toga što faktori unutar sume ne ovise o varijabli w). Posljednja suma je, dakle, funkcija varijabli s i r . Uvedimo eksplisitno tu funkciju, i nazovimo ju $t_1(s, r)$. Dakle, uz $t_1(s, r) = \sum_c P(c)P(s|c)P(r|c)$, imamo:

$$P(w) = \sum_s \sum_r P(w|s, r)t(s, r)$$

Slično, marginalizacija varijabli r eliminira tu varijablu. Izraz koji je preostao ovisi samo o varijablama s i w , pa možemo uvesti funkciju $t_2(s, w) = \sum_r P(w|c, r)t_1(s, r)$. Onda imamo:

$$P(w) = \sum_s t_2(s, w)$$

Konačno, možemo uvesti funkciju $t_3(w) = \sum_s t_2(s, w)$, pa onda:

$$P(w) = t_3(w)$$

Izračunavanje $P(w)$ sada se svodi na izračun sljedećih funkcije za $w = 0$ i $w = 1$:

$$\begin{aligned} t_3(w) &= \sum_s t_2(s, w) \\ t_2(s, w) &= \sum_r P(w|c, r)t_1(s, r) \\ t_1(s, r) &= \sum_c P(c)P(s|c)P(r|c) \end{aligned}$$

Uštedu ovdje ostvarujemo pri izračunu funkcije $t_1(s, r)$, koja ne ovisi o w , pa je dakle dovoljno da je izračunamo samo jednom (za $w = 0$), pohranimo rezultat te ga iskoristimo idući put (za $w = 1$). Pogledajmo broj računskih operacija. Za izračun funkcije t_1 trebamo 5 operacija. Za izračun funkcije t_2 treba nam 8 operacija (od toga 5 za izračun funkcije t_1 , ali njega ćemo napraviti samo jednom). Konačno, za izračun funkcije t_3 treba nam 17 operacija (od toga dva puta po 8 za dva izračuna funkcije t_2). Za izračun $P(w)$ ovo trebamo ponoviti za $w = 0$ i $w = 1$, pa je dakle ukupan broj operacija 34. To je manje nego 42 operacije, koliko smo imali u prethodnom slučaju.

Ovaj konceptualno jednostavan postupak zaključivanja naziva se **eliminacija varijabli zbroj-umnožak** (engl. *sum-product variable elimination*). Glavna je ideja da varijable marginaliziramo jednu po jednu, i svaki puta marginalizaciju radimo nad produktom faktora koji sadržavaju tu varijablu. Učinkovitost koja se pritom ostvaruje ovisi o konkretnoj strukturi Bayesove mreže. Što je manje bridova u mreži, to su varijable manje zavisne i faktori su to “lokalniji”, pa je to više izraza koji se ponavljaju.

Inače, specifično za skrivene Markovljeve modele (HMM), algoritam eliminacije varijabli naziva se **algoritam unaprijed-unazad** (engl. *forward-backward algorithm*). Varijanta tog algoritma specifično za MAP-upite za HMM naziva se **Viterbijev algoritam**, i taj se algoritam vrlo često koristi u praksi (npr., u raspoznavanju govora i obradi prirodnog jezika).

Nažalost, za općenite grafove egzaktno je zaključivanje presloženo i traje eksponencijalno u broju čvorova (tj. varijabli). Konkretno, algoritamska složenost eliminacije varijabli izravno ovisi o veličini najvećeg faktora (jer po njemu trebamo obaviti sumaciju). U takvim slučajevima, alternativa egzaktnom zaključivanju je **približno zaključivanje** (engl. *approximate inference*). Tu postoje dvije glavne grupe algoritama. Prva se grupa temelji na ideji da se konstruira neka jednostavna distribucija koja što bolje aproksimira ciljanu distribuciju, i zatim da se optimizira sličnost između tih dviju distribucija. Takvi postupci uključuju **propagacijske algoritme** (engl. *belief propagation*) i **varijacijsko zaključivanje** (engl. *variational inference*). Nažalost, nemamo vremena ulaziti u ove metode. Drugu grupu algoritama za približno zaključivanje čine algoritmi temeljeni na **uzorkovanju**. U nastavku ćemo malo pogledati neke od tih metoda.

3

4

3 Zaključivanje: Metode uzorkovanja

Metoda uzorkovanja metoda je, dakle, za približno zaključivanje. Prisjetimo se da kod zaključivanja mi zapravo želimo izračunati vjerojatnost varijabli upita, gdje su neke varijable možda već opažene, a neke nas ne zanimaju i to su varijable smetnje. Sad je pitanje kakve to ima veze s uzorkovanjem? Kako pomoću uzorkovanja možemo izračunati neku vjerojatnost? Zapravo, vrlo jednostavno. Ideja je da jednostavno uzorkujemo varijablu \mathbf{x} iz njezine distribucije $P(\mathbf{x})$, tj. $\mathbf{x} \sim P(\mathbf{x})$, i to ponavljamo više puta kako bismo dobili uzorak vrijednosti te varijable iz dotične distribucije. Jednom kada imamo takav uzorak veličine N , onda da iz tog uzorka možemo **procijeniti** parametre distribucije pomoću procjenitelja, npr., MLE ili MAP.

Na primjer, ako nas zanima koja je vjerojatnost da je trava mokra, $P(w = 1)$, onda ćemo najprije generirati uzorak iz zajedničke distribucije $P(c, s, r, w)$. Nakon što dobijemo takav uzorak, možemo napraviti procjenu za $P(w = 1)$, što je zapravo procjena parametra μ za Bernoullijevu varijablu w . MLE procjena tog parametra, kao što znamo, je jednostavno relativna frekvencija realizacije $w = 1$ u uzorku:

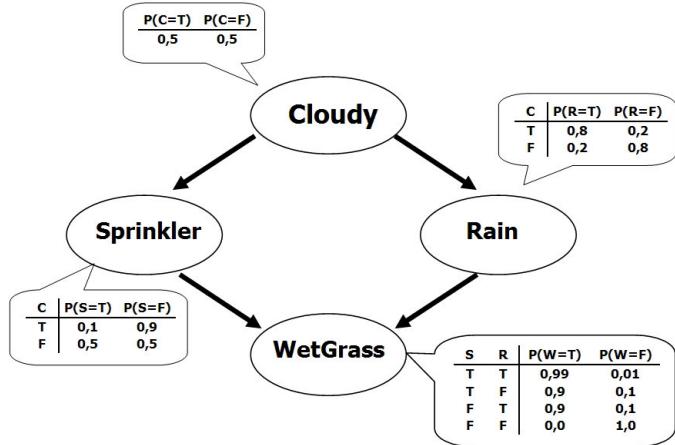
$$P(w = 1) = \frac{1}{N} \sum_{i=1}^N \mathbf{1}\{w_i = 1\}$$

To je, dakle, osnovna ideja – uzorkovati varijablu \mathbf{x} iz distribucije $P(\mathbf{x})$, a onda na tom uzorku procijeniti parametar od interesa. U našem slučaju $P(\mathbf{x})$ je definirana pomoću Bayesove mreže. Prisjetimo se ovdje da je Bayesova mreža **generativan model**, a to znači da možemo generirati uzorce iz distribucije koju ta mreža prikazuje. No, kako to točno možemo napraviti?

3.1 Unaprijedno uzorkovanje

Prva ideja koja odmah svima padne na pamet jest da jednostavno uzorkujemo po strukturi Bayesove mreže, krenuvši od roditeljskih čvorova prema čvorovima djeci. Drugim riječima, zapravo slijedimo **generativnu priču** Bayesove mreže. Konkretno, krenemo od prvog čvora u mreži (prvog po topološkom uređaju) i slučajno generiramo, sukladno distribuciji tog čvora, jednu vrijednost za tu varijablu. Zatim idemo na iduēu varijablu po topološkom uređaju i generiramo slučajnu vrijednost za tu varijablu. Ako je ta varijabla uvjetovana prvom varijablom, onda naravno uzimamo to u obzir kod generiranja, tj. kod čvorova djece generiramo iz uvjetne distribucije s pravilno postavljenim vrijednostima varijabli u čvorovima roditelja. Budući da idemo topološkim uređajem, zajamčeno je da ćemo u svakom čvoru do kojeg dođemo već imati generirane vrijednosti za sve varijable roditelja. Naposlijetu, kada tako obiđemo cijelu mrežu, imat ćemo jedan slučajan vektor iz zajedničke distribucije. Ponavljanjem ovog postupka generiramo niz ovakvih vektora, i oni onda čine naš uzorak. Ovaj se postupak naziva **unaprijedno uzorkovanje** (engl. *forward sampling*). Pogledajmo unaprijedno uzorkovanje na primjeru prskalice i trave.

► PRIMJER



U prvom čvoru (“cloudy”) uzorkovali bismo varijablu c iz distribucije za tu varijablu. Vjerojatnost da $c = 1$ je 0.5, ista kao i vjerojatnost za $c = 0$. Recimo da smo uzorkovali $c = 1$. U drugom koraku uzorkujemo iz čvora “sprinkler” iz uvjetne distribucije $p(s|c = 1)$. Tu je veća vjerojatnost da uzorkujemo $s = 0$, pa recimo da smo tako i dobili. U trećem koraku uzorkujemo u čvoru “rain” iz uvjetne distribucije $p(r|c = 1)$, i recimo da smo tu dobili $r = 1$. Konačno, u četvrtom koraku, kada već imamo $c = 1$, $s = 0$ i $r = 1$, uzorkujemo varijablu w u čvoru “wet grass” iz uvjetne distribucije $p(w|s = 0, r = 1)$. Recimo da smo tu dobili $w = 1$, jer je to vjerojatnije nego $w = 0$. Naš konačni slučajni vektor je dakle $\mathbf{x} = (c = 1, s = 0, r = 1, w = 1)$. Ponavljanjem ovog postupka dobili bismo uzorak vektora iz zajedničke distribucije $P(c, s, r, w)$.

3.2 Uzorkovanje s odbacivanjem

Unaprijedno uzorkovanje je superjednostavno, ali postoji problem. Nas u principu ne zanima uzorkovanje iz zajedničke distribucije, nego iz uvjetne distribucije! Naime, kada radimo upite nad mrežom, zanima nas aposteriorna vjerojatnost varijabli upita uz opažene varijable:

$$P(\mathbf{x}_q | \mathbf{x}_o)$$

Dakle, sada je pitanje kako možemo **uzorkovati iz uvjetne distribucije**? Bismo li za to mogli upotrijebiti postupak unaprijednog uzorkovanja? Problem je u tome što nekako moramo osigurati da su opažene varijable \mathbf{x}_o postavljene na željene vrijednosti. Jedna mogućnost je da uzorkujemo kao i ranije, a onda da, iz uzorka koje dobijemo, uzmemo samo one vektore kod kojih su varijable \mathbf{x}_o postavljene na točne vrijednosti, a sve ostale da odbacimo. Takav se postupak naziva **uzorkovanje s odbacivanjem** (engl. *rejection sampling*).

Vidite li problem s ovakvim pristupom? Problem je što je ćemo možda dobiti vrlo malo iskoristivih vektora. To će pogotovo biti slučaj kada je vjerojatnost dokaza $P(\mathbf{x}_o)$ jako malena. Tada ćemo, naime, vjerojatno zeznuti već kod uzorkovanja u početnim čvorovima. Npr., u našem primjeru, ako je $\mathbf{x}_o = (c = 1, s = 1)$, vjerojatnost da uzorkujemo takvu kombinaciju je samo $0.5 \cdot 0.1 = 0.05$. U složenijim Bayesovim mrežama lako se može dogoditi da za neke kombinacije vrijednosti varijabli jednostavno ne uspijevamo skupiti uzorak dovoljne veličine za pouzdanu procjenu.

3.3 Uzorkovanje po važnosti

Alternativa koja nam ovdje može pasti na pamet je da lijepo fiksiramo vrijednosti opažanih varijabli na željene vrijednosti, pa da onda uzorkujemo iz svih preostalih varijabli. Npr., ako

je $w = 1$, idemo tako fiksirati vrijednost varijable w , a onda ćemo uzorkovati iz preostalih varijabli. Međutim, tu je problem da ćemo tako dobiti uzorke koji ne poštuju pravu aposteriornu vjerojatnost (točnije: nećemo dobiti uzorke iz željene aposteriorne distribucije). Naime, ako fiksiramo $w = 1$, može nam se dogoditi da uzorkujemo $s = 0$ i $r = 0$, međutim prema tablici uvjetne vjerojatnosti u čvoru “wet” imamo $P(w = 1|s = 0, r = 0) = 0$, pa se to dakle uopće ne bi smjelo dogoditi. Slično, ako uzorkujemo uz opažanje $s = 1$, pa u vektoru postavimo $s = 1$, ali međutim nezavisno uzorkujemo c , onda možemo očekivati da ćemo dobiti 50% uzoraka sa $c = 0$ i 50% uzoraka sa $c = 1$, neovisno o tome što je $s = 1$. Međutim, to ne odgovara distribuciji definiranoj mrežom. Za tu distribuciju vidimo iz tablice uvjetnih vjerojatnosti da je pet puta manje vjerojatno da je $s = 1$ ako je $c = 1$ nego ako je $c = 0$.

Ovo bismo mogli korigirati, tako da damo težine uzorcima. Na primjer, ako fiksiramo $s = 1$, onda ćemo reći da su uzorci za koje $c = 1$ pet puta manje težine nego ovi za $c = 0$, budući da za izglednosti varijable c imamo $p(s = 1|c = 1) = 0.1$ i $p(s = 1|c = 0) = 0.5$. I doista, možemo to tako napraviti: možemo fiksirati vrijednosti svim opaženim varijablama, napraviti unaprijedno uzorkovanje, izračunati ovakve težine u svim čvorovima u kojima imamo opažene varijable, i onda pri izračunu očekivanja koristiti te težine. Takav postupak naziva se **uzorkovanje otežano izglednostima** (engl. *likelihood weighted sampling*), i jedan je od postupaka iz šire porodice postupka koji se nazivaju **uzorkovanje po važnosti** (engl. *importance sampling*).

Ovaj se postupak čini sasvim razumnim. Međutim, kada bismo analizirali njegova matematička svojstva (što ne budemo radili), isпадa da postupak nije uvijek tako dobar. Konkretno, zanima nas kvaliteta procjenitelja (jer mi zapravo procjenjujemo parametre distribucije iz uzorka). Pokazuje se da kvaliteta ovisi o tome koliko je distribucija iz koje uzorkujemo (ona koja odgovara unaprijednom uzorkovanju iz Bayesove mreže uz fiksirane vrijednosti opaženih varijabli) različita od aposteriorne distribucije. Ako su te dvije distribucije vrlo slične, onda nema problema i naša će procjena biti dosta dobra. Međutim, ako su te dvije distribucije dosta različite, onda procjena neće biti dobra. Primijetimo da će distribucija iz koje uzorkujemo i aposteriona distribucija biti potpuno iste (i da nema potrebe za težinama) ako se sve opažene varijable nalaze u početnim čvorovima mreže (po topološkom uređaju), tj. ako su svi roditelji opaženih varijabli također opaženi. Tada, naime, zapravo uzorkujemo direktno iz aposteriorne distribucije. Suprotno tome, ako su sve opažene varijable u listovima, mi zapravo uzorkujemo iz apriorne distribucije, i ona može biti poprilično različita od aposteriorne distribucije. Tada se moramo osloniti na težine kako bismo dobili uzorak što sličniji onome kao da smo doista uzorkovai iz aposteriorne distribucije. Nažalost, to onda ipak smanjuje kvalitetu procjene.

3.4 Gibbsovo uzorkovanje

Uzorkovanje po važnosti je dobar postupak, ali možemo mi i bolje. Alternativu predstavljaju tzv. **Monte Carlo metode s Markovljevim lancem** (engl. *Markov chain Monte Carlo, MCMC*). Te metode uzorak generiraju slučajnim procesom, koji čini **Markovljev lanac**: niz slučajnih varijabli kod kojega iduća varijabla ovisi samo o trenutačnoj varijabli, ali ne i prethodnim varijablama u lancu. Lanac se konstruira tako da konvergira k stacionarnoj distribuciji koja je upravo ona distribucija iz koje želimo uzorkovati. Najpoznatije varijante algoritma MCMC su **Metropolis-Hastings** i **Gibbsovo uzorkovanje** (engl. *Gibbs sampling*). U nastavku ćemo ukratko opisati Gibbsovog uzorkovanja, koje se u praksi najviše koristi (također i za bayesovske modele, ali njih nećemo raditi).

Ideja Gibbsovog uzorkovanja je ova: krenemo od nekog slučajnog početnog vektora iz zajedničke distribucije, a onda ciklički uzorkujemo varijablu po varijablu tog vektora, tako da su uvijek sve varijable osim jedne fiksirane. Dakle, uvijek uzrokujemo samo jednu varijablu, iz distribucije koja je uvjetovana vrijednostima svih preostalih varijabli. Pogledajmo primjer.

► PRIMJER

Želimo uzorkovati iz zajedničke distribucije $p(x_1, x_2, x_3)$. Krenemo sa slučajnim vektorom \mathbf{x}^0 iz uzorkovanim iz distribucije $p(x_1^0, x_2^0, x_3^0)$, npr., unaprijednim uzorkovanjem (broj u eksponentu označava broj iteracija). Zatim uzrokujemo varijable redom, i svaki puta prethodnu vrijednost te varijable nadomjestimo vrijednošću koju smo upravo dobili uzorkovanjem. Konkretno, najprije uzorkujemo x_1^1 iz uvjetne distribucije $p(x_1|x_2^0, x_3^0)$. Dobivamo x_1^1 . Zatim uzorkujemo x_2^1 iz uvjetne distribucije $p(x_2|x_1^1, x_3^0)$ te dobivamo x_2^1 . Zatim uzorkujemo x_3^1 iz uvjetne distribucije $p(x_3|x_1^1, x_2^1)$. Nakon ova tri koraka, završili smo prvu iteraciju uzorkovanja te imamo naš prvi slučajni vektor, $\mathbf{x}^1 = (x_1^1, x_2^1, x_3^1)$. Sada bismo ovaj postupak ponavljali i generirali daljne slučajne vektore. Dakle, postupak uzorkovanja je ovaj:

$$\begin{aligned}\mathbf{x}^0 &\sim p(x_1^0, x_2^0, x_3^0) \quad \Rightarrow \text{prvi vektor (npr., unaprijednim uzorkovanjem)} \\ x_1^1 &\sim p(x_1|x_2^0, x_3^0) \\ x_2^1 &\sim p(x_2|x_1^1, x_3^0) \\ x_3^1 &\sim p(x_3|x_1^1, x_2^1) \quad \Rightarrow \text{vektor } \mathbf{x}^1 = (x_1^1, x_2^1, x_3^1) \\ x_1^2 &\sim p(x_1|x_2^1, x_3^1) \\ x_2^2 &\sim p(x_2|x_1^1, x_3^1) \\ x_3^2 &\sim p(x_3|x_1^2, x_2^1) \quad \Rightarrow \text{vektor } \mathbf{x}^2 = (x_1^2, x_2^2, x_3^2) \\ &\vdots\end{aligned}$$

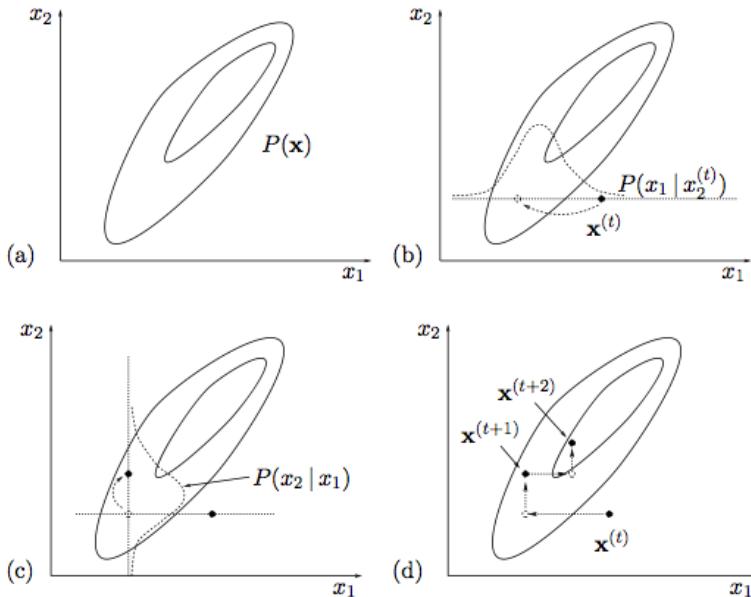
Budući da Gibbsovo uzorkovanje započinjemo od nekog slučajnog vektora, koji može biti poprilično nereprezentativan (tj. malo vjerljatan) za distribuciju iz koje uzorkujemo, općenito treba proći određeni broj iteracija prije nego što uzorkovanje dođe do područja veće gustoće vjerojatnosti i vektori postanu reprezentativni za distribuciju. Da bismo to ostvarili, lanac uzorkovanja treba biti dovoljno dug. Isto tako se preporuča odbaciti određeni broj početno dobivenih vektora, dok se lanac ne “završi” u području veće gustoće vjerojatnosti, odnosno dok ne završi tzv. **period zagrijavanja** (engl. *burn-in period*).

Očito, Gibbsovo uzorkovanje pretpostavlja da možemo jednostavno izračunati uvjetnu distribuciju jedne varijable uvjetovane na sve ostale varijable te da možemo egzaktно uzorkovati iz te distribucije. Ovdje nam u prilogu ide činjenica da kod Bayesovih mreža svaka varijabla uvjetno ovisi tek o manjem broju drugih varijabli. Preciznije, svaka uvjetna distribucija neke varijable u Bayesovoj mreži ovisi samo o varijablama u **Markovljevom omotaču** (engl. *Markov blanket*) čvora koji odgovara toj varijabli. Markovljev omotač čine roditeljski čvorovi, čvorovi djece te roditelji čvora djece. Zbog tako lokalizirane ovisnosti, općenito je jednostavno napisati distribuciju neke varijable uvjetovanu na sve ostale varijable. Ipak, budući da varijable u Bayesovoj mreži općenito mogu imati različite distribucije (ne moraju nužno biti kategoričke), izračun uvjetne distribucije, a time i egzaktno uzorkovanje iz takve distribucije, općenito mogu biti netrivijalni.

► PRIMJER

Pogledajmo Gibbsovo uzorkovanje iz bivarijatne Gaussove distribucije:

$$p(x_1, x_2) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$



Na slici (a) gore lijevo prikazane su izokonture bivarijatne Gaussove distribucije $p(x_1, x_2)$. Na slici (b) gore desno prikazan je prvi korak Gibbsovog uzorkovanja, gdje inicijalno krećemo od slučajnog vektora (točke) \mathbf{x}^t i zatim uzorkujemo prvu varijablu x_1 iz uvjetne distribucije $p(x_1|x_2^{(t)})$. Ta uvjetna distribucija bivarijatne Gaussove distribucije je također Gaussova distribucija. Naime, svaka distribucija podskupa varijabli multivarijatne Gaussove distribucije uvjetovane na podskup preostalih varijabli je i sama multivarijatna Gaussova distribucija, a u slučaju Gibbsovog uzorkovanja iz Gaussove distribucije to je uvijek univarijatna Gaussova distribucija budući da su sve varijable osim jedne fiksirane. Dakle, varijablu x_1 uzorkujemo iz uvjetne distribucije (odnosno gustoće vjerojatnosti) $p(x_1|x_2^{(t)})$, i to je univarijatna Gaussova distribucija koja je na slici (b) prikazana iscrtkano. Recimo da smo uzorkovali vrijednost koja je naznačena na slici, pa smo napravili pomak ulijevo u smjeru osi $-x_1$, kako je indicirano strelicom. Zatim, u drugom koraku, uzorkujemo varijablu x_2 iz uvjetne distribucije (odnosno gustoće vjerojatnosti) $p(x_2|x_1)$, što je opet univarijatna Gaussova distribucija, te se pomičemo, npr., u smjeru osi $+x_2$ kako je prikazano na slici (c) dolje lijevo. Time smo dobili prvi vektor (točku) \mathbf{x}^{t+1} . Sada ponavljamo postupak. Kroz niz koraka, kao što ilustrira slika (d) dolje desno, generiramo niz točaka, većina kojih će biti smještena u području navječe gustoće naše bivarijatne Gaussove distribucije, odnosno točke će biti distribuirane po toj distribuciji.

U gornja dva primjera uzorkovali smo iz zajedničke distribucije. Međutim, kao što smo rekli, tipično nas zanima uzorkovanje iz aposterione distribucije $p(\mathbf{x}_q|\mathbf{x}_o)$, gdje su neke varijable \mathbf{x}_o već opažene. U tom slučaju, kod Gibbsovog uzorkovanja jednostavno ćemo te opažene varijable fiksirati na njihove opažene vrijednosti te nećemo uzorkovati vrijednosti za te varijable. Uzorkovanjem iz varijabli upita \mathbf{x}_q , ciklički kako je opisano gore, dobit ćemo upravo uzorak iz aposteriorne distribucije uvjetovane na opažene varijable \mathbf{x}_o .

► PRIMJER

Pogledajmo opet primjer s prskalicom za travu. Ako znamo da je trava mokra ($w = 1$), što možemo zaključiti o kiši (r)? Drugim riječima, zanima nas aposteriorna distribucija $P(r|w = 1)$. Budući da je varijabla w opažena s vrijednošću 1, fiksirat ćemo $w = 1$, dok ćemo vrijednosti ostalih varijabli uzorkovati. Krećemo od nekog slučajnog vektora, generiranog unaprijednim uzorkovanjem. Neka je to vektor $\mathbf{x}^0 = (c = 0, s = 0, r = 1, w = 1)$. U prvom koraku uzorkujemo varijablu c iz uvjetne distribucije $P(c|s = 0, r = 1, w = 1)$, zanemarujući inicijalnu vrijednost varijable c . Tu uvjetnu

distribuciju možemo lako izračunati na temelju naše Bayesove mreže:

$$\begin{aligned} P(c|s=0, r=1, w=1) &= \frac{P(c, s=0, r=1, w=1)}{P(s=0, r=1, w=1)} \\ &= \frac{P(c)p(s=0|c)p(r=1|c)P(w=1|s=0, r=1)}{\sum_c P(c)P(s=0|c)P(r=1|c)P(w=1|s=0, r=1)} \end{aligned}$$

Vrijednost za varijablu c uzorkovali bismo iz ove distribucije (u ovom slučaju to je Bernoullijeva distribucija jer je c binarna varijabla). U idućem koraku fiksiramo varijablu c na dobivenu vrijednost, dok ostale varijable ostavljamo na njihovim prethodnim vrijednostima, osim varijable s , koju bismo iduću uzorkovali. Zatim bismo isto napravili za varijablu r . Nakon toga, ciklus se ponavlja od varijable c , pa s , pa opet r , itd. Varijablu w ne bismo uzorkovali jer je ona opažena, i njezina je vrijednost fiksirana na $w = 1$. Na taj bismo način dobili uzorak iz distribucije $P(c, s, r|w=1)$, te iz tog uzorka možemo procijeniti parametre te distribucije.

Ovdje možda izgleda kao da nismo puno dobili Gibbsovim uzorkovanjem u odnosu na egzaktno zaključivanje, jer svaki puta moramo izračunati uvjetnu distribuciju iz koje uzorkujemo, a da bismo to napravili moramo množiti faktore i marginalizirati, baš kao i kod egzaktnog zaključivanja. Doista, kod ovako male Bayesov mreže egzaktno zaključivanje zapravo nije problematično. Međutim, kod većih mreža, koje reprezentiraju visokodimenzijske zajedničke distribucije, egzaktno je zaključivanje, kako smo već napomenuli, netraktabilno. Gibbsovo uzorkovanje onda predstavlja učinkovitu alternativu. Naime, premda za Gibbsovo uzorkovanje moramo izračunati uvjetnu distribuciju za varijablu po kojoj uzorkujemo, ta će distribucija uvijek ovisiti o manjem broju susjednih varijabli u mreži (već spomenuti Markovljev omotač), pa dakle nemamo problem s netraktabilnošću kao kod egzaktnog zaključivanja.

Konačno, budući da nas zapravo zanima distribucija $P(r|w=1)$, nju možemo izračunati iz distribucije $P(c, s, r|w=1)$ tako da marginaliziramo po varijablama s i r (koje su u ovom upitu varijable smetnje):

$$\begin{aligned} P(r|w=1) &= \sum_s \sum_r P(c, s, r|w=1) \\ &= P(c=0, s=0, r|w=1) + P(c=0, s=1, r|w=1) + \\ &\quad P(c=1, s=0, r|w=1) + P(c=1, s=1, r|w=1) \end{aligned}$$

I ove četiri vjerojatnosti mogu se procijeniti iz našeg uzorka izvučenog iz $P(c, s, r|w=1)$, pa dakle ovo nije problem izračunati.

4 Učenje

PGM-ovi su probabilistički modeli. Kako ih učimo, odnosno na što se svodi učenje kod probabilističkih modela? Učenje se svodi na **procjenu parametara θ** distribucije koja opisuje podatke. Kako ćemo izračunati procjenu parametara te distribucije? Kao i uvijek, na raspolaganju imamo tri alata: **MLE, MAP i bayesovsku procjenu** (ovu treću nećemo raditi).

Prošli put spomenuli smo **skriveni Markovljev model** (engl. *Hidden Markov Model, HMM*). Karakteristika tog modela bila je da sadrži varijable koje su skrivene, tj. koje **nisu opažene u podatcima**. Postoje mnogi drugi PGM-ovi koji imaju skrivene varijable, budući da uvođenje skrivenih varijabli može znatno pojednostaviti modeliranje. No, pokazuje se da se postupak učenja bitno razlikuje ovisno o tome ima li PGM skrivenih varijabli ili ne. Naime, ako model ima skrivenih varijabli, onda te varijable nikada nisu opažene u podatcima (te varijable postoje samo u modelu, ali ne i u podatcima), pa nećemo moći samo tako procijeniti parametre njihovih distribucija. U tom slučaju govorimo o učenju iz **nepotpunih podataka** (engl. *incomplete data*) – u podatcima nemamo sve varijable koje imamo u modelu. Suprotno, ako model nema skrivenih varijabli, onda to znači da sve varijable opažamo iz podataka, i u tom slučaju govorimo u učenju iz **potpunih podataka** (engl. *complete date*). Učenje iz potpunih podataka je jednostavnije, pa

ćemo njega prvog razmotriti.

4.1 Potpuni podatci

Najjednostavniji način procjene parametara je MLE. Znamo MLE procjenitelje za standardne teorijske distribucije. Međutim, Bayesova mreža odgovara nekoj složenijoj distribuciji, za čije parametre ne znamo kako glasi MLE procjenitelj. Kako ćemo onda izračunati MLE procjenu za parametre te distribucije? Napraviti ćemo to kao i ranije: izvest ćemo MLE procjenitelj **maksimizacijom log-izglednosti**. Trebamo, dakle, napisati log-izglednost parametara distribucije koja odgovara Bayesovoj mreži, a to je jednostavno. Najprije, prisjetimo se, od prošlog puta, da je zajednička vjerojatnost $p(\mathbf{x})$ pomoću Bayesove mreže faktorizirana na sljedeći način:

$$p(\mathbf{x}) = \prod_{k=1}^n p(x_k | \text{pa}(x_k))$$

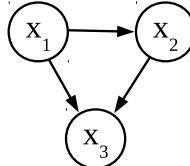
gdje je $\text{pa}(x_k)$ skup roditeljskih čvorova čvora x_k . Parametri ove distribucije neka su $\boldsymbol{\theta}$. Log-izglednost parametara $\boldsymbol{\theta}$ onda je:

$$\begin{aligned} \ln \mathcal{L}(\boldsymbol{\theta} | \mathcal{D}) &= \ln p(\mathcal{D} | \boldsymbol{\theta}) = \ln p(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)} | \boldsymbol{\theta}) \\ &= \ln \prod_{i=1}^N p(\mathbf{x}^{(i)} | \boldsymbol{\theta}) = \ln \prod_{i=1}^N \prod_{k=1}^n p(x_k^{(i)} | \text{pa}(x_k^{(i)}), \boldsymbol{\theta}_k) \\ &= \ln \prod_{k=1}^n \prod_{i=1}^N p(x_k^{(i)} | \text{pa}(x_k^{(i)}), \boldsymbol{\theta}_k) = \sum_{k=1}^n \sum_{i=1}^N \ln p(x_k^{(i)} | \text{pa}(x_k^{(i)}), \boldsymbol{\theta}_k) \end{aligned}$$

Ovdje primijetite da imamo dva produkta: jedan ide po svim primjerima ($i = 1, \dots, N$), jer izglednost računamo kao umnožak vjerojatnosti za sve primjere iz skupa \mathcal{D} , a drugi ide po svim značajkama ($k = 1, \dots, n$), jer za izračun zajedničke vjerojatnosti moramo pomnožiti sve faktore, a u Bayesovoj mreži imamo po jedan faktor za svaku značajku. Budući da produkti komutiraju, možemo zamijeniti njihov redoslijed. Nakon što primijenimo logaritam, produkti se pretvaraju u sume, pa tako dobivamo sumu po faktorima (vanjska suma), gdje za svaki faktor sumiramo po primjerima (unutarnja suma). Kažemo da se log-izglednost **dekomponirala** prema strukturi grafa Bayesove mreže. Pod time mislimo da smo dobili po jedan pribrojnik za svaki čvor Bayesove mreže. Pogledajmo primjer.

► PRIMJER

Razmotrimo sljedeću Bayesovu mrežu:



Pripadna faktorizacija je:

$$p(x_1, x_2, x_3) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)$$

Log-izglednost parametara ove distribucije je:

$$\begin{aligned}
\ln \mathcal{L}(\boldsymbol{\theta} | \mathcal{D}) &= \ln p(\mathcal{D} | \boldsymbol{\theta}) \\
&= \sum_{k=1}^n \sum_{i=1}^N \ln p(x_k^{(i)} | \text{pa}(x_k^{(i)}), \boldsymbol{\theta}_k) \\
&= \sum_{i=1}^N \ln p(x_1^{(i)} | \text{pa}(x_1^{(i)}), \boldsymbol{\theta}_1) + \sum_{i=1}^N \ln p(x_2^{(i)} | \text{pa}(x_2^{(i)}), \boldsymbol{\theta}_2) + \sum_{i=1}^N \ln p(x_3^{(i)} | \text{pa}(x_3^{(i)}), \boldsymbol{\theta}_3) \\
&= \sum_{i=1}^N \ln p(x_1^{(i)} | \boldsymbol{\theta}_1) + \sum_{i=1}^N \ln p(x_2^{(i)} | x_1^{(i)}, \boldsymbol{\theta}_2) + \sum_{i=1}^N \ln p(x_3^{(i)} | x_1^{(i)}, x_2^{(i)}, \boldsymbol{\theta}_3)
\end{aligned}$$

Dekompozicija log-izglednosti je dobra stvar, jer to znači da možemo za svaki čvor mreže nezavisno procijeniti parametre $\boldsymbol{\theta}_k$ uvjetne distribucije za varijablu x_k . Konkretno, MLE procjena parametara uvjetne distribucije za k -tu varijablu je:

$$\boldsymbol{\theta}_k^* = \underset{\boldsymbol{\theta}_k}{\operatorname{argmax}} \sum_{i=1}^N \ln p(x_k^{(i)} | \text{pa}(x_k^{(i)}), \boldsymbol{\theta}_k)$$

Slično, MAP procjena parametra uvjetne distribucije za k -tu varijablu je:

$$\boldsymbol{\theta}_k^* = \underset{\boldsymbol{\theta}_k}{\operatorname{argmax}} \left(\sum_{i=1}^N \ln p(x_k^{(i)} | \text{pa}(x_k^{(i)}), \boldsymbol{\theta}_k) + \ln p(\boldsymbol{\theta}_k) \right)$$

gdje je $p(\boldsymbol{\theta}_k)$ apriorna gustoća vjerojatnosti parametra $\boldsymbol{\theta}_k$. Prisjetimo se: ako želimo rješenje u zatvorenoj formi, onda $p(\boldsymbol{\theta}_k)$ treba biti konjugatna za izglednost $\sum_i p(x_k^{(i)} | \text{pa}(x_k^{(i)}), \boldsymbol{\theta}_k)$. Prisjetimo se, također, da ako je apriorna vjerojatnost parametara $p(\boldsymbol{\theta}_k)$ uniformna, MAP degenerira na MLE.

Naravno, sad je pitanje koja je to konkretno distribucija u pitanju, koju pojedini čvorovi modeliraju. Distribucija je najčešće diskretna tj. **kategorička** (ali ne mora biti, može, npr., biti Gaussova). Kao što znamo, za kategoričku varijablu MLE procjena za μ_k je jednostavno relativna frekvencija za realizaciju vrijednosti svake vrijednosti. Malo preciznije, ako je k indeks varijable (čvora), j je vrijednost uvjetovanih varijabli (roditeljskih čvorova), a l je vrijednost varijable, onda:

$$N_{kjl} = \sum_{i=1}^N \mathbf{1}\{\mathbf{x}_{\text{pa}(x_k)}^{(i)} = j \wedge x_k^{(i)} = l\}$$

Broj primjera za koje je vrijednost varijabli roditelja čvora k jednaka j :

$$N_{kj} = \sum_l N_{kjl}$$

MLE procjena je onda relativna frekvencija:

$$\hat{\mu}_{k,j,l} = \frac{N_{k,j,l}}{N_{k,j}}$$

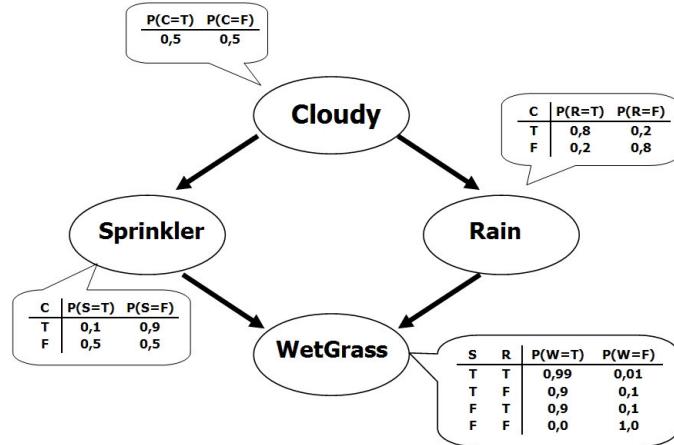
Ako želimo raditi MAP-procjenu onda (Dirichlet-kategorički model uz $\alpha = 2$):

$$\hat{\mu}_{k,j,l} = \frac{N_{kjl} + 1}{N_{kj} + K_k}$$

gdje je K_k broj mogućih vrijednosti varijable x_k .

► PRIMJER

Prisjetimo se Bayesove mreže s prskalicom za travu:



Recimo da želimo procijeniti, pomoću MAP, parametre za čvor w . Čvor w odgovara uvjetnoj vjerojatnosti $P(w|s, r)$. MAP procjena je:

$$\hat{\mu}_{w,(s,r),1} = P(w=1|s, r) = \frac{\sum_{i=1}^N \mathbf{1}\{x_s^{(i)} = s \wedge x_r^{(i)} = r \wedge x_w^{(i)} = w\} + 1}{\sum_{i=1}^N \mathbf{1}\{x_s^{(i)} = s \wedge x_r^{(i)} = r\} + 2}$$

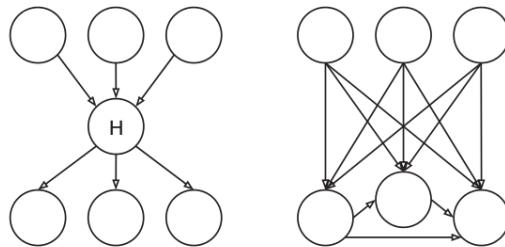
4.2 Nepotpuni podatci

Kako što smo već napomenuli, PGM možemo definirati tako da uključuje skrivene varijable. Tipično su to varijable koje smo uveli u model kao "posredničke varijable" (engl. *mediator variables*) između varijabli koje opažamo. Npr., ako modeliramo vezu između uzorka bolesti (npr., pušenje) i simptoma (npr., bol u prsima), skrivena varijabla koju ne opažamo je sama bolest (srčana bolest). Skrivene varijable nikada ne opažamo u podatcima (one u podatcima ne postoje), i zato u takvom slučaju govorimo u učenju iz **nepotpunih podataka**.

Sljedeći primjer ilustrira pogodnosti uvođenja skrivene varijable u model.

► PRIMJER

Razmotrimo dvije Bayesove mreže:



Obje Bayesove mreže opisuju vezu između primarnih uzroka bolesti (gornji čvorovi) i simptoma (donji čvorovi). Mreža na lijevoj slici sadrži skrivenu varijablu H (čvor u sredini) koji modelira bolest kao zajedničku posljedicu primarnih uzroka i zajednički uzrok svih simptoma. Ta je varijabla skrivena jer bolest, za razliku od primarnih uzroka i simptoma, ne možemo izravno opažati. Desna slika prikazuje Bayesovu mrežu koja modelira iste odnose između primarnih uzroka bolesti i simptoma, ali

7

8

bez posredovanja skrivene varijable H . U ovom modelu imamo izravno povezane čvorove primarnih uzroka i simptoma, te dodatne veze između simptoma. Lijevi je model znatno intuitivniji jer uvažava postojanje posredničkog konstrukta (bolesti), unatoč tome što taj konstrukt ne možemo izravno opažati (općenito, skrivene varijable mogu biti, i zapravo često i jesu, nekakvi hipotetski i apstraktни konstrukti). No, osim konceptualne jednostavnosti, lijevi je model znatno jednostavniji u smislu broja parametara. Naime, uz pretpostavku da su sve varijable binarne, lijevi model ukupno ima 17 parametara, dok desni model ukupno ima 59 parametara (uvjerite se u to!). Čak i ako bismo u desnom modelu uklonili veze između simptoma – takav model bolje bi odgovarao lijevom modelu sa skrivenom varijablom – i dalje bi broj parametara bio veći nego kod modela sa skrivenom varijablom (imali bismo ukupno 27 parametara).

Ranije smo vidjeli da, kada učimo Bayesovu mrežu iz potpunih podataka, log-izglednost se dekomponira po strukturi Bayesove mreže, i to nam omogućava da parametre modela procjenimo zasebno za svaki čvor. Nadalje, ako su distribucije za pojedinačne varijable neke teorijske distribucije (iz eksponencijalne familije), onda MLE i MAP procjene dobivamo u zatvorenoj formi. Međutim, kod učenja modela sa skrivenim varijabla, tj. učenja iz nepotpunih podataka, log-izglednost se neće dekomponirati po čvorovima mreže, i to komplikira postupak učenja.

Pokažimo u čemu je problem. Mogli bismo to pokazati na bilo kojem modelu sa skrivenim varijablama, npr., skrivenom Markovljevom modelu. No, izabrat ćemo jedan jednostavniji model sa skrivenim varijablama: **mješavinski model** (engl. *mixture model*). O tom ćemo modelu pričati u jednom od narednih predavanja u kontekstu nenadziranog učenja, točnije grupiranja (klasteringa). Mješavinski modeli zajedničku vjerojatnost modeliraju na sljedeći način:

$$P(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = P(\mathbf{z}|\boldsymbol{\theta})p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})$$

Ovo na prvi pogled izgleda kao naivan Bayesov klasifikator (gdje bi \mathbf{z} bila oznaka klase). No, problem je što kod nenadziranog učenja \mathbf{z} nije opažena varijabla, jer kod nenadziranog učenja ne znamo kojoj klasi primjer pripada. Dakle, varijable \mathbf{z} nisu opažene varijable, tj. to nisu označke \mathbf{y} . U tom smislu naši su podatci nepotpuni: raspolažemo samo opažanjima za varijablu \mathbf{x} , ali ne i opažanjima za skrivenu varijablu \mathbf{z} .

Pogledajmo kako bismo izrazili log-izglednost parametara $\boldsymbol{\theta}$ ovog modela:

$$\begin{aligned} \ln \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) &= \ln p(\mathcal{D}|\boldsymbol{\theta}) = \ln p(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}|\boldsymbol{\theta}) \\ &= \ln \prod_{i=1}^N p(\mathbf{x}^{(i)}|\boldsymbol{\theta}) = \ln \prod_{i=1}^N \sum_{\mathbf{z}} p(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}|\boldsymbol{\theta}) \\ &= \ln \prod_{i=1}^N \sum_{\mathbf{z}} p(\mathbf{z}^{(i)}|\boldsymbol{\theta})p(\mathbf{x}^{(i)}|\mathbf{z}^{(i)}, \boldsymbol{\theta}) = \sum_{i=1}^N \ln \sum_{\mathbf{z}} p(\mathbf{z}^{(i)}|\boldsymbol{\theta})p(\mathbf{x}^{(i)}|\mathbf{z}^{(i)}, \boldsymbol{\theta}) \end{aligned}$$

Primijetite da smo proveli marginalizaciju po varijabli \mathbf{z} , jer tu varijablu imamo u modelu, ali je ne opažamo u skupu \mathcal{D} , pa je moramo marginalizirati kako bismo izrazili vjerojatnost skupa \mathcal{D} . No, kao što vidimo, za razliku od log-izglednosti potpunih podataka, ovdje logaritamsku funkciju ne možemo potisnuti unutar sume (logaritamska je funkcija “zapela” između dvije sume), pa se log-izglednost **ne dekomponira** po čvorovima mreže. Posljedica toga je da ne postoji rješenje u **zatvorenoj formi**, niti za MLE, MAP niti bayesovski procjenitelj. Umjesto toga, moramo koristiti neki iterativni optimizacijski algoritam. Jedna mogućnost je **gradijentni uspon** (engl. *gradient ascent*) (uspon, a ne spust, jer maksimiziramo log-izglednost, ali princip je identičan kao i kod gradijentnog spusta). Međutim, za mješavinske modele i skrivene Markovljeve modele češće se koristi **algoritam maksimizacije očekivanje (EM-algoritam)** (engl. *expectation maximization algorithm*), o kojem ćemo pričati u kontekstu algoritama grupiranja. Ukratko, ideja EM-algoritma je sljedeća: kada bi sve varijable bile opažene, onda bismo

lako izračunali MLE/MAP. Ideja je onda da izačunamo **očekivane vrijednosti** svih varijable (zapravo, aposteriorne distribucije za sve varijable), koristeći postupak **zaključivanja**, a zatim te očekivane vrijednosti koristimo kao da su opažene vrijednosti. Na temelju očekivanih vrijednosti varijabli zatim izračunamo MLE ili MAP procjene, tj. provodimo maksimizaciju izglednosti (za MLE) odnosno vjerojatnosti (za MAP). Nakon toga, iznova izračunamo očekivane vrijednosti varijabli, i zatim taj postupak iterativno ponavljamo do konvergencije, alternirajući između koraka izračuna očekivanja i maksimizacije izglednosti/vjerojatnosti.

Osim što rješenje za MLE/MAP ne postoji u zatvorenoj formi, drugi problem s učenjem iz nepotpunih podataka jest što log-izglednost (za MLE) odnosno aposteriorna vjerojatnost (za MAP) više nisu konveksne funkcija parametara θ . Te funkcija niti su konveksne niti konkavne, već imaju **lokalne maksimume**, što znači da optimizacija ne mora dati globalno optimalne parametre. EM-algoritam i gradijentni uspon konvergirat će u neki lokalni optimum funkcije izglednosti (ovisno o početnim uvjetima), pa je dobro da se postupak optimizacije više puta ponovi, sa slučajno odabranim početnim parametrima.

Sažetak

- **Zaključivanje** kod probabilističkih grafičkih modela (PGM-ova) svodi se na određivanje vjerojatnosti **varijabli upita** uvjetovane na **opažene varijable**, uz marginalizaciju **varijabli smetnje**
- Postoje **aposteriorni upiti** i **MAP-upiti**
- **Eliminacija varijabli** je egzaktan postupak zaključivanja koji tehnikom **dinamičkog programiranja** izbjegava problem kombinatorne eksplozije pri konstrukciji zajedničke distribucije
- Za općenite PGM-ove egzaktni su postupci presloženi, pa koristimo **približno zaključivanje**
- **Metode uzorkovanja** (npr., **uzorkovanje s odbijanjem** i **Gibbsovo uzorkovanje**) služe za približno zaključivanje pomoću procjene parametara iz uzorka generiranog iz PGM-a
- **Gibbsovo uzorkovanje** generira uzorke pomoću Markovljevog lanca, uzorkujući varijablu po varijablu
- **Učenje** Bayesove mreže svodi se na MLE ili MAP procjenu parametara svake varijable zasebno, jer se izglednost dekomponira po čvorovima mreže
- Ako model ima **skrivenе varijable**, onda učimo iz **nepotpunih podataka**, za što moramo koristiti iterativne algoritme, npr., **gradijentni uspon** ili **algoritam maksimizacije očekivanja**

Bilješke

- 1 Ovo se predavanje dominantno oslanja na (Koller and Friedman, 2009) (poglavlja 9 i 12). Također preporučam (Bishop, 2006) (poglavlja 8.4 i 11.1–3) i (Murphy, 2012) (poglavlja 10.4, 20.3, 23.1–4, 24.1–2).
- 2 Vjerujem da vam je koncept **dinamičkog programiranja** poznat. Meni se svidjela knjižica (Dennardo, 2012), koja daje iscrpan opis tog područja. Za kratak uvod, pogledajte <http://20bits.com/article/introduction-to-dynamic-programming>. Dobro je ne brkati **memoizaciju** (pohranjivanje jednom izračunatih rezultata) i dinamičko programiranje (način rješavanja problema koji iskorištava preklapanje potproblema, a tipično se ostvaruje rekurzijom s memoizacijom ili tabuliranjem rješenja). Dinamičko programiranje razvio je u ranim 1950. godinama američki matematičar Richard Bellman. Bellman je uveo pojam **prokletstva dimenzionalnosti**, koji smo spomenuli u kontekstu izračuna udaljenosti u visokodimenzijskom prostoru.
- 3 Konkretno, učinkovitost algoritma eliminacije varijabli ovisi o strukturi Bayesove mreže i o redoslijedu kojim eliminiramo varijable, kao što je pokazao naš primjer. Broj računalnih operacija ograničen

je odozgo tzv. **širinom stabla** (engl. *tree width*) Bayesove mreže. Neformalno, širina stabla govori nam koliko je graf različit od stabla. Malo formalnije, širina stabla je broj čvorova u najvećoj kliki (potpuno povezanom podgrafu) grafa umanjena za jedan. Tako je stablasta širina grafa koji je stablo, a također i grafa koji je lanac, jednaka 1. Nažalost, pokazuje se da je nalaženje optimalnog redoslijeda eliminacije varijabli, koji bi dao minimalan broj računalnih operacija, NP-težak problem. U praksi stvari nisu tako crne: razvijeni su postupci za neke specifične strukture Bayesovih mreža, kao i heuristički postupci. Više u ([Koller and Friedman, 2009](#)) (poglavlje 9.4).

- 4** **Varijacijsko zaključivanje** (engl. *variational inference*) jedan je od osnovnih alata strojnog učenja i dio mnogih naprednih algoritama strojnog učenja. Posebno su popularne kod generativnih dubokih modela, npr. **varijacijski autoenkoder** ([Doersch, 2016](#)). Varijacijske metode problem zaključivanja svode na **problem optimizacije** u visokodimenzijskome prostoru – zato se te metode često opisuju sloganom “zaključivanje kao optimizacija” (engl. *inference as optimization*). Cilj takve optimizacije jest iz ograničene familije jednostavnijih distribucija pronaći onu distribuciju q koja je trakabilna (što će reći da se da izraziti u zatvorenoj formi), a opeć što sličnija pravoj (ciljnoj) aposteriornoj distribuciji p . Distribucija q naziva se **prijedložna distribucija** (engl. *proposal distribution*). Sličnost između prijedložne i ciljne distribucije tipično se mjeri Kullback-Leibler divergencijom, koju smo bili spomenuli prošli put. Nakon optimizacije, probabilistički se upiti onda provode nad prijedložnom (i jednostavijom) distribucijom q , a ne nad ciljnom (i mnogo složenijom) distribucijom p . Dobar uvod u varijacijske metode možete naći u ([Fox and Roberts, 2012](#)) i ([Blei et al., 2017](#)). Za razliku od (također popularnih) metoda uzorkovanja, o kojima ćemo pričati u nastavku, varijacijske su metode u načelu brže te računalno atraktivnije u smislu da se mogu implementirati tehnikama gradijentnog spusta i da se daju paralelizirati, što je i razlog zašto su ove tehnike danas posebno popularne u dubokom učenju. S aspekta računalne teorije učenja, varijacijske metode također imaju prednost nad metodama uzorkovanja jer su prikladnije za teorisku analizu ograda na točnost. S druge strane, za razliku od metoda uzorkovanja, varijacijske metode rijetko nalaze globalno optimalno rješenje zbog ograničenja da prijedložna distribucija mora biti jednostavna. Zbog toga su predloženi različiti pristupi za složeniju aproksimaciju ciljne distribucije, a ti su pristupi nedavno objedinjeni u obećavajući radni okvir pod nazivom **normalizirajući tokovi** (engl. *normalizing flows*) ([Papamakarios et al., 2019](#)). Konkretnu usporedbu varijacijskih metoda i metoda uzorkovanja (konkretno, metode MCMC; v. bilješku ispod) možete naći u ([Salimans et al., 2015](#)). Predložene su i kombinacije varijacijskih metoda i metoda uzorkovanja; npr. vidi ([Wolf et al., 2016](#)).
- 5** Općenito, **Monte Carlo s Markovljevim lancem** (engl. *Markov chain Monte Carlo, MCMC*) naziv je za sve metode koje procjenjuju parametre složenih aposteriornih distribucija uzorkovanjem iz poznatih apriornih distribucija. “Monte Carlo” (prema poznatom kasinu u Monaku, u kojemu je James Bond bio rado viđen gost još od “Nikad ne reci nikad”) odnosi se na činjenicu da te metode koriste slučajno uzorkovanje. “Markovljev lanac” (prema ruskom matematičaru Andreju Markovu, po kojemu je Judea Pearl uveo naziv “Markovljev omotač”, a koji smo spomenuli prošli put) odnosi se na činjenicu da novi uzorak ovisi samo o prethodnom uzorku, a ne i o svim prethodnim uzorcima. O naziv MCMC više možete pročitati ovdje: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6874253/>
- 6** Algoritam uzorkovanja **Metropolis-Hastings** predložio je grčko-američki fizičar Nicholas Constantine Metropolis 1953. godine, radeći u Los Alamosu na poznatom projektu Manhattan, zajedno s John von Neumannom i poljsko-američkim matematičarom Stanisławom Ulamom, a 1970. godine algoritam je razradio kanadski statističar Wilfred Hastings. Usput, Metropolis je taj koji je za algoritme slučajnog uzorkovanja uveo naziv “Monte Carlo”, navodno kao internu šalu na Ulamov račun. Algoritam **Gibbsovog uzorkovanja** nazvan je tako u čast američkog fizičara Josiah Willard Gibbs, jednog od osnivača statističke fizike, a predložili su ga 1984. godine (mnogo godina nakon Gibbsove smrti) američki matematičari Stuart i Donald Geman, braća koja su vrlo poznata po svojim doprinosima strojnom učenju.
- 7** **Skrivene varijable** već smo spomenuli kada smo govorili o probabilističkim upitim nad PGM-ovima. Međutim, primijetite da skrivene varijable upita nisu isto kao i skrivene varijable modela. Naime, skrivene varijable upita su one varijable koje nisu opažene u trenutku kada provodimo upit. Međutim, to ne znači da te varijable nisu bile opažene onda kada smo učili model. S druge strane, skrivene varijable modela su uvijek skrivene. To su varijable koje ne možemo opažati u podatcima, jer ne postoje u podatcima, ali za koje prepostavljamo da inače postoje, bilo u konkretnom ili apstraktном

smislu. Budući da te varijable ne opažamo, o njihovim vrijednostima **zaključujemo** na temelju opaženih varijabli, i to pomoću modela koji u odnos dovodi opažene i skrivene varijable.

Inače, modele sa skrivenim varijablama također nazivamo **model latentnih varijabli** (engl. *latent variable models, LVM*). Modeli latentnih varijabli redovito se koriste u statistici i strojnom učenju, s primjenama u područjima kao što su psihologija, medicina i ekonomija. Npr., poznati peterofaktorski model ličnosti iz psihologije (engl. *Big Five*) dobiven je modelom latentnih varijabli (konkretno, faktorskom analizom) iz jezičnih podataka.

- [8] Primjer je preuzet iz ([Murphy, 2012](#)), poglavje 11.
- [9] Primijetite da kod EM-algoritma **zaključivanje** koristimo kao potproceduru algoritma učenja. Zato je bitno da zaključivanje bude učinkovito (najčešće je to onda aproksimativno zaključivanje).
- [10] Detaljnije u ([Murphy, 2012](#)), poglavje 11.3.

Literatura

- C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
- D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.
- E. V. Denardo. *Dynamic programming: models and applications*. Courier Corporation, 2012.
- C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.
- C. W. Fox and S. J. Roberts. A tutorial on variational bayesian inference. *Artificial intelligence review*, 38(2):85–95, 2012.
- D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
- K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.
- G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.
- T. Salimans, D. Kingma, and M. Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning*, pages 1218–1226, 2015.
- C. Wolf, M. Karl, and P. van der Smagt. Variational inference with Hamiltonian Monte Carlo. *arXiv preprint arXiv:1609.08203*, 2016.

19. Grupiranje

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v2.2

Do sada smo se na ovom kolegiju bavili nadziranim strojnim učenjem. Kod nadziranog strojnog učenja, primjeri za učenje su označeni, tj., to su parovi (\mathbf{x}, y) , i za svaki nam je primjer poznata ciljna klasa y (ako radimo klasifikaciju) ili ciljna vrijednost y (ako radimo regresiju). Informacija o ciljnoj klasi ili ciljnoj vrijednosti je vrlo jak signal koji imamo iz podataka.

Danas (i idući put) bavit ćemo se drugom stranom strojnog učenja: **nenadziranim strojnim učenjem** (engl. *unsupervised learning*). Kod nenadziranog strojnog učenja, nemamo označenih primjera. Očekivano, takav je problem teži, ali je i fascinantniji – kao naučiti nešto bez eksplicitnog signala koji bi nas vodio u pravom smjeru? Vidjet ćemo da postoje različiti pristupi nenadziranom učenju. Mi ćemo se usredotočiti na **grupiranje** (engl. *clustering*), kog kojega neoznačene primjere skupljamo u grupe, ovisno o tome koliko su primjeri međusobno slični. Ima raznih algoritama grupiranja: danas ćemo se usredotočiti na one najjednostavnije. Za početak, pričat ćemo općenito o nenadziranom strojnom učenju. Onda ćemo pogledati **algoritam K-sredina** i njegovo poopćenje, **algoritam K-medoida**. Na kraju, pričat ćemo o vrednovanju grupiranja, odnosno tzv. **provjeri grupiranja**.

1

1 Nenadzirano učenje

Osnovna motivacija za nenadzirano učenje jest što u mnogim slučajevima analize podataka jednostavno nemamo informaciju o tome koji primjer pripada kojoj klasi. To znači da umjesto skupa označenih primjera:

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$$

raspoložemo skupom **neoznačenih primjera** (engl. *unlabeled instances*):

2

$$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$$

Primjeri mogu biti neoznačeni iz dva razloga: (1) ne znamo ih označiti (ne znamo unaprijed koje klase postoje) ili (2) znamo koje klase postoje, ali označavanje je teško izvedivo (npr., preskupo je). Evo nekih zadataka u kojima bismo tipično koristili nenadzirano strojno učenje:

- Grupiranje klijenata prema ponašanju (segmentacija korisnika) – ne znamo unaprijed koji segmenti korisnika postoje, ali znamo da svi korisnici nisu isti i prepostavljam da postoji nekoliko vrsta prototipnih korisnika (korisnika koji se, uz neka manja odstupanja, u prosjeku slično ponašaju);
- Grupiranje novinskih članaka prema temama – ne znamo unaprijed koje se sve teme javljaju u novinskim tekstovima, ali znamo da razni novinski tekstovi obrađuju različite teme (pri čemu jedan novinski tekst može istovremeno obrađivati više tema), pa želimo grupirati novinske tekstove po temama;
- Grupiranje tekstova prema autorima – raspoložemo zbirkom tekstova anonimnih autora za koje prepostavljam da ih je pisalo nekoliko autora. Tekstove želimo grupirati tako da tekstovi istog autora završe u istoj grupi, a tekstovi različitih autora u različitim grupama;

3

- Analiza sličnosti rješenja laboratorijskih vježbi – zanima nas koja su rješenja laboratorijskih vježbi međusobno slična. Ovdje želimo dobiti grupe rješenja koja su međusobno slična, tako da svi oni koji su međusobno prepisivali ili su prepisivali od istog autora završe u istoj grupi i mogu biti zajednički sankcionirani (neugodan primjer, ali događa se);
- Grupiranje gena sa sličnom izražajnošću (funkcionalnošću) – ne znamo unaprijed koje su sve moguće funkcionalnosti, ali znamo da neki geni imaju sličnu funkcionalnost;
- Klasifikacija objekata na fotografiji (engl. *content-based image retrieval*) – ako ne postoji unaprijed definirane klase objekata, bolje nam je da rezultate pretraživanja slika jednostavno grupiramo po sličnosti, u nadi da će slike s istim objektima završiti u istim grupama;
- Klasifikacija poruka s Twittera prema iskazanoj emociji – označavanje emocija u porukama Twittera naporno je i skupo. Mnogo je lakše poruke grupirati na način da one poruke koje izražavaju iste ili slične emocije završe u istim grupama. Štoviše, možda niti ne znamo unaprijed koje sve emocije su izražene u skupu podataka kojim raspolažemo (postoje različite teorije i modeli emocija), pa je možda bolje pustiti da te emocije “isplivaju” same od sebe putem grupiranja;
- Otkrivanje napada korisnika na mreži (engl. *intrusion detection*) – ovo je primjer **otkrivanja vrijednosti koje odskaču** (engl. *outlier detection*). Zanima nas je li ponašanje korisnika na mreži bitno drugačije od ponašanja svih drugih korisnika. Ako je to slučaj, onda je to sumnjivo i moguće štetno ponašanje.

Kao što smo napomenuli u uvodu, nenadzirano učenje obuhvaća niz metoda, a svima njima je zajedničko to što rade na neoznačenih podatcima. Međutim, različite se metode koriste za različite zadatke. Četiri su osnovna zadatka nenadziranog učenja:

1. **Grupiranje** (engl. *clustering*) – nalaženje grupa sličnih primjera;
2. **Procjena gustoće** (engl. *density estimation*) – nalaženje gustoće vjerojatnosti $p(\mathbf{x})$ koja opisuje distribuciju neoznačenih podataka;
3. **Otkrivanje novih/stršećih vrijednosti** (engl. *novelty/outlier detection*) – nalaženje primjera koji po svojim značajkama bitno odskaču od većine ostalih primjera u skupu podataka;
4. **Smanjenje dimenzionalnosti** (engl. *dimensionality reduction*) – smanjenje broja značajki, odnosno preslikavanje podataka iz izvornog ulaznog prostora u prostor smanjenih dimenzija, s ciljem otkrivanja manjeg broja novih (tzv. latentnih) značajki odnosno dimenzija koje dovoljno dobro opisuju različitosti u podatcima.

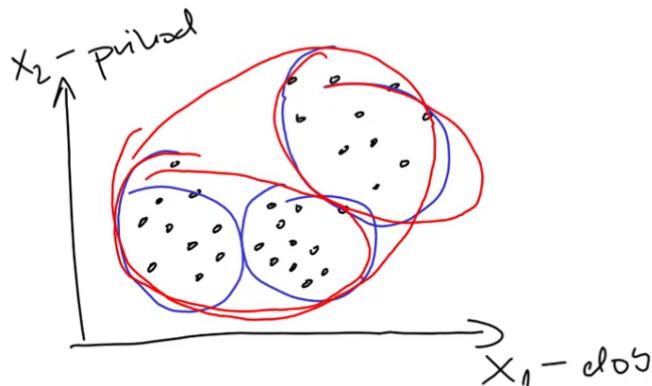
U nastavku ćemo se baviti isključivo grupiranjem podataka. Taj se oblik nenadziranog učenja u praksi daleko najčešće koristi.

2 Grupiranje

Grupiranje (engl. *clustering*) jest postupak razdjeljivanja primjera u grupe (engl. *clusters*), tako da **slični** primjeri (slični po nekom svojstvu) budu svrstani u istu grupu, a različiti primjeri u različite grupe. Svrha grupiranja jest nalaženje “prirodnih” (intrinzičnih) grupa u skupu neoznačenih podataka. Ili, pjesnički rečeno, svrha grupiranja jest “pustiti podatke da govore sami za sebe”.

► PRIMJER

Želimo grupirati korisnike prema prihodu i dobi. Dakle, ulazni prostor je dvodimenzijski, i svaki je korisnik prikazan dvodimenzijskim vektorom $\mathbf{x} = (x_1, x_2)$, gdje je značajka x_1 prihod a značajka x_2 je dob. Skup podataka neka je ovakav:



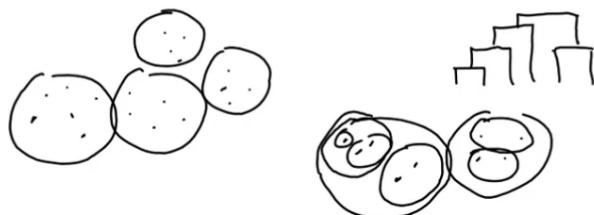
Neka je sličnost između primjera definirana kao inverz (euklidske) udaljenosti između vektora u dvodimenziskom ulaznom prostoru, tj. što su primjeri u ulaznom prostoru bliži jedan drugome, to ih smatramo sličnjima. (Općenito, sličnost možemo definirati i na neki drugi način, no ideju je najlakše ilustrirati ako koristimo euklidsku udaljenost odnosno bliskost.) S obzirom na sličnost između primjera, ovaj skup podataka možemo grupirati u tri grupe (označene plavom bojom). Međutim, primjere bismo isto tako mogli grupirati u dvije grupe (označene crvenom bojom), ako bismo primjere u donje dvije plave grupe smatrali dovoljno sličnim (dovoljno bliskima) da čine jednu grupu. Ovo ilustrira općenit problem kod grupiranja, koji će nas pratiti cijelo vrijeme: često postoji više "prirodnih grupiranja", odnosno primjere je moguće grupirati u različit broj grupa, ovisno o tome kada primjere smatramo "dovoljno sličnim" da bismo ih smjestili u istu grupu.

2.1 Vrste grupiranja

Postoje različite vrste grupiranja. Razmotrit ćemo dvije podjele. Prva podjela odnosi se na to generira li algoritam grupe primjera koje imaju nekakvu internu strukturu (podgrupe primjera) ili generira "plošne" grupe koje nemaju nikakvu internu strukturu. Tako razlikujemo:

- **Particijsko grupiranje** – grupe su plošne (engl. *flat*), tj. ne postoje podgrupe;
- **Hijerarhijsko grupiranje** – grupe imaju podgrupe, koje imaju svoje podpodgrupe, i tako dalje, rekurzivno. Drugim riječima, postupak rezultira hijerarhijom grupe.

Na donjoj slici lijevo prikazan je rezultat partijskog grupiranja, a na slici desno rezultat hijerarhijskog grupiranja:

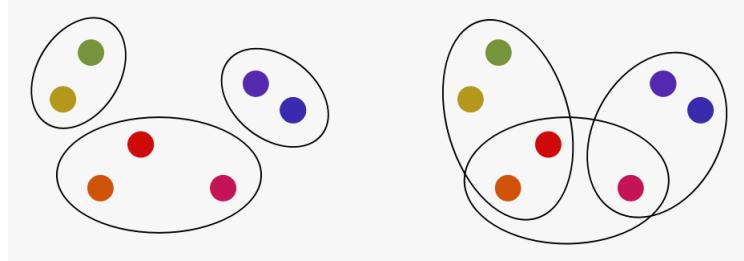


Hijerarhijsko grupiranje rezultira strukturu koje se može prikazati stablom, tzv. **dendrogramom**, gdje su u listovima tog stabla primjeri koje grupiramo. Više o tome idući put.

Neovisno o tome je li grupiranje partijsko ili hijerarhijsko, moguće je grupiranje ostvariti tako da jedan te isti primjer pripada uvijek samo jednoj (pod)grupi, ili dopustiti da primjer

istovremeno pripada u više (pod)grupa. U tom smislu razlikujemo:

- **Tvrdo grupiranje** (engl. *hard clustering*) – jedan primjer može pripadati jednoj i samo jednoj (pod)grupi;
- **Meko grupiranje** (engl. *soft clustering*) – jedan primjer može istovremeno pripadati u više grupe (npr. algoritam fuzzy k-means) ili može biti pridijeljen u više grupe s nekom vjerojatnošću, što modelira našu nesigurnost o njegovoj stvarnoj pripadnosti (npr. **probabilističko grupiranje** algoritmom GMM, o čemu ćemo pričati idući put). Na donjoj slici prikazana je razlika između tvrdog grupiranja (lijevo) i mekog grupiranja (desno):



2.2 Grupiranje kao predobrada za nadzirano učenje

Ako su primjeri označeni, onda možemo koristiti nadzirano učenje i ne treba nam grupiranje. Međutim, ponekad je korisno koristiti grupiranje kao predobradu podataka, dakle prije primjene nadziranog učenja. U tom slučaju na grupiranje možemo gledati kao na metodu smanjenja dimenzionalnosti. Tu imamo dvije mogućnosti. Prva je da grupiramo primjere, a druga da grupiramo značajke.

Grupiranje primjera svodi se na grupiranje redaka matrice dizajna. Na primjer:

$$\mathcal{D} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} & \dots & x_n^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & x_4^{(3)} & \dots & x_n^{(3)} \\ x_1^{(4)} & x_2^{(4)} & x_3^{(4)} & x_4^{(4)} & \dots & x_n^{(4)} \\ x_1^{(5)} & x_2^{(5)} & x_3^{(5)} & x_4^{(5)} & \dots & x_n^{(5)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ x_1^{(N)} & x_2^{(N)} & x_3^{(N)} & x_4^{(N)} & \dots & x_n^{(N)} \end{pmatrix}$$

$N \times n \rightarrow K \times n$

Rezultat će biti matrica dizajna s manje redaka (točno onoliko redaka koliko imamo grupe). To jest, u postupak ulazimo s matricom dizajna dimenzija $N \times n$, a dobivamo matricu dizajna dimenzija $K \times n$, gdje je K broj grupe i $K < N$. Svaku grupu sada možemo prikazati kao jedan primjer, npr., centroid (središnji vektor) svih primjera u toj grupi. Na taj smo način efektivno proveli **zaglađivanje primjera** (engl. *smoothing*). Možete se pitati zašto bismo ovo htjeli raditi, jer time očito smanjujemo broj primjera. No, neki algoritmi nadziranog učenja možda će bolje raditi ako im na ulaz dovedemo manji broj doista reprezentativnih primjera, nego ako ih obasipamo primjerima koji se razlikuju u šumu. Također, imajte na umu da ovime efektivno kombiniramo dvije induktivne pristrane: one algoritma grupiranja s onom algoritma nadziranog učenja. U nekim slučajevima moglo bi biti da takva kombinacija daje dobre rezultate.

Alternativno, umjesto da grupiramo primjere (retke matrice dizajna), možemo grupirati značajke (stupce matrice dizajna). Na primjer:

$$\mathcal{D} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} & \dots & x_n^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & x_4^{(3)} & \dots & x_n^{(3)} \\ x_1^{(4)} & x_2^{(4)} & x_3^{(4)} & x_4^{(4)} & \dots & x_n^{(4)} \\ x_1^{(5)} & x_2^{(5)} & x_3^{(5)} & x_4^{(5)} & \dots & x_n^{(5)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)} & x_2^{(N)} & x_3^{(N)} & x_4^{(N)} & \dots & x_n^{(N)} \end{pmatrix}$$

$N \times n \rightarrow N \times K$

Time matricu dizajna dimenzija $N \times n$ smanjujemo na matricu $N \times K$, gdje $K < n$. Ponovno možemo za svaki stupac izračunati centroid, čime više značajki svodimo na jednu reprezentativnu značajku. Ovdje, dakle, zadržavamo sve primjere iz izvornog skupa podataka, ali smanjujemo broj značajki, tako da značajke koje su međusobno vrlo slične zapravo svodimo na jednu reprezentativnu značajku, što opet moguće smanjuje šum. U nastavku pogledajmo napokon jedan konkretan algoritam grupiranja.

3 Algoritam K-sredina

Najjednostavniji i najpoznatiji algoritam grupiranja jest **algoritam K-sredina** (engl. *K-means algorithm*). Algoritmom se primjeri iz neoznačenog skupa primjera $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ grupiraju u K **tvrđih grupa**, gdje se parametar K (broj grupe) zadaje unaprijed. Primjetite da se ovdje radi o **particijskom grupiranju**, jer imamo tvrde grupe (svaki primjer pripada samo jednoj grupi).

Ideja algoritma jest da svaka grupa ima svoju srednju vrijednost (centroid) koja predstavlja grupu. Svaki primjer pripada grupi čiji mu je centroid najbliži (po euklidskoj udaljenosti). Postupak grupiranja je iterativan. Krenuvši od K slučajno odabranih sredina (centroida grupe), svi se primjeri svrstavaju u onu grupu čiji im je centroid najbliži. To može dovesti do pomaka centrioda, stoga se u idućem koraku, nakon stvrstavanja svih primjera u njima najbližu grupu, ponovno izračunavaju novi centriodi za svaku grupu. No, nakon što su se izračunali novi centriodi, to može dovesti do promjene u pripadanju primjera grupama, pa se primjeri ponovo svrstavaju u grupe tako da svaki primjer bude u grupi čiji im je centroid najbliži. To novo razvrstavanje primjera ponovo može dovesti do promjena centrioda, pa se ponovno izračunavaju centriodi za svaku grupu, i tako dalje. Postupak ponavlja ova dva koraka (pridjeljivanje primjera grupama i izračun centrioda) sve do konvergencije (dok nema promjene u pripadnosti primjera grupama, odnosno dok nema promjene u centroidima grupa).

3.1 Formalna definicija

Pogledajmo to sada formalno. Mnogi algoritmi grupiranja mogu se formalizirati (i izvesti) tako da se definira **funkcija pogreške** koju minimiziraju. Ta se funkcija u kontekstu algoritama grupiranja često naziva **kriterijska funkcija**. Označit ćemo je sa J . Za algoritam K-sredina, kriterijska je funkcija definirana ovako:

$$J = \sum_{k=1}^K \sum_{i=1}^N b_k^{(i)} \|\mathbf{x}^{(i)} - \boldsymbol{\mu}_k\|^2$$

Intuitivno, ova funkcija zbraja koliko primjeri unutar svake grupe odstupaju od centrioda dotične grupe (primjetite da imamo dvije sume, jedna ide po grupama, a druga po primjerima). Preciznije, svaka grupa predstavljena je svojim centrom, $\{\boldsymbol{\mu}_k\}_{k=1}^K$. Oznaka $\|\cdot\|$

je L_2 -norma (tj. euklidska norma), definirana kao $\|\mathbf{x} - \boldsymbol{\mu}\|^2 = (\mathbf{x} - \boldsymbol{\mu})^T(\mathbf{x} - \boldsymbol{\mu})$, pa je dakle $\|\mathbf{x}^{(i)} - \boldsymbol{\mu}\|^2$ kvadrat euklidske udaljenosti između primjera $\mathbf{x}^{(i)}$ i centroida $\boldsymbol{\mu}$. Vrijednost $b_j^{(i)}$ je binarna indikatorska varijabla koja indicira pripada li primjer $\mathbf{x}^{(i)}$ grupi k : ako $b_k^{(i)} = 1$, onda primjer $\mathbf{x}^{(i)}$ pripada grupi k , inače joj ne pripada. Prema tome, ukupna pogreška jednaka je zbroju, po svim primjerima, kvadrata euklidske udaljenosti primjera $\mathbf{x}^{(i)}$ od središta $\boldsymbol{\mu}_k$ grupe u koju je taj primjeri svrstan (za grupe u koje primjer $\mathbf{x}^{(i)}$ nije svrstan vrijedi $b_k^{(i)} = 0$ i tu nemamo doprinosa pogrešci).

Mi, naravno, želimo pronaći ono grupiranje koje minimizira pogrešku. Grupiranje je definirano dvama parametrima: indikatorskim varijablama $b_k^{(i)}$ (one definiraju kojoj grupi pripada koji primjer) i centroidima grupe $\boldsymbol{\mu}_k$ (oni definiraju gdje u prostoru primjera se grupe nalaze). Formalno, dakle, tražimo parametre takve da:

$$\underset{b_1, \dots, b_K; \boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K}{\operatorname{argmin}} J$$

Očito, pogreška će biti to veća što su primjeri dalje od centroida svoje grupe. To znači da, želimo li minimizirati pogrešku J , svaki primjer $\mathbf{x}^{(i)}$ trebamo svrstati u grupu čije je središte $\boldsymbol{\mu}_k$ tom primjeru najbliže, to jest:

$$b_k^{(i)} = \begin{cases} 1 & \text{ako } k = \underset{j}{\operatorname{argmin}} \|\mathbf{x}^{(i)} - \boldsymbol{\mu}_j\| \\ 0 & \text{inače} \end{cases}$$

Algoritam K-sredina radi tako da minimizira upravo kriterijsku funkciju J , ali to ne radi analitički jer ovaj optimizacijski problem nema rješenje u zatvorenoj formi. Naime, ako pokušamo derivirati funkciju J po oba parametra, nailazimo na problem jer su ti parametri međusobno ovisni: $b_k^{(i)}$ ovise o $\boldsymbol{\mu}_k$ i, obrnuto, $\boldsymbol{\mu}_k$ ovise o $b_k^{(i)}$. Umjesto toga, algoritam K-sredina optimizaciju provodi iterativno. Algoritam započinje sa slučajno odabranim srednjim vrijednostima $\boldsymbol{\mu}_k$. Zatim se u svakoj iteraciji temeljem izraza za $b_k^{(i)}$ za svaki primjer $\mathbf{x}^{(i)}$ izračunava vrijednost $b_k^{(i)}$, odnosno svaki se primjer pridjeljuje grupi čijem je centru najblizi. Nakon toga – budući da sad imamo fiksirane vrijednosti $b_k^{(i)}$ – možemo izravno minimizirati izraz za kriterijsku funkciju J . Konkretno, postavljenjem $\nabla_{\boldsymbol{\mu}_k} J = \mathbf{0}$ i rješavanjem po $\boldsymbol{\mu}_k$ dobivamo:

$$2 \sum_{i=1}^N b_k^{(i)} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_k) = \mathbf{0}$$

iz čega slijedi

$$\boldsymbol{\mu}_k = \frac{\sum_i b_k^{(i)} \mathbf{x}^{(i)}}{\sum_i b_k^{(i)}}$$

Vektor $\boldsymbol{\mu}_k$ jednak je dakle srednjoj vrijednosti vektora svih primjera koji su svrstani u grupu k . Budući da je ovime ostvarena promjena vektora $\boldsymbol{\mu}_k$ u odnosu na njegovu prethodnu vrijednost, sada treba opet primijeniti izraz za $b_k^{(i)}$ i ponovno izračunati koji primjeri pripadaju grupi k . Ova se dva koraka ponavljaju sve dok se ne dosegne stacionarno stanje, odnosno stanje u kojemu nema daljnjih promjena vrijednosti $\boldsymbol{\mu}_k$.

Kombinirajmo sada sve ovo u algoritam:

► **Algoritam K-sredina**

```

1:  inicijaliziraj centroide  $\mu_k$ ,  $k = 1, \dots, K$ 
2:  ponavljaj
3:    za svaki  $\mathbf{x}^{(i)} \in \mathcal{D}$ 
4:       $b_k^{(i)} \leftarrow \begin{cases} 1 & \text{ako } k = \operatorname{argmin}_j \|\mathbf{x}^{(i)} - \mu_j\| \\ 0 & \text{inače} \end{cases}$ 
5:    za svaki  $\mu_k$ ,  $k = 1, \dots, K$ 
6:       $\mu_k \leftarrow \sum_{i=1}^N b_k^{(i)} \mathbf{x}^{(i)} / \sum_{i=1}^N b_k^{(i)}$ 
7:  dok  $\mu_k$  ne konvergiraju

```

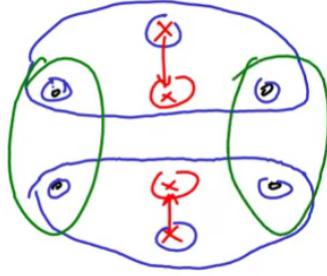
3.2 Svojstva algoritma

Razmotrimo računalnu složenost algoritma. Složenost izračuna euklidske udaljenosti je $\mathcal{O}(n)$, gdje je n broj značajki. U prvom koraku (pridjeljivanje primjera grupama) izračunavamo KN udaljenosti, pa je složenost prvog koraka $\mathcal{O}(nNK)$. U drugom koraku (izračun centroida), budući da je svaki primjer pridodan samo jednome centroidu (a za sve ostale centroide k vrijedi $b_k^{(i)} = 0$), to znači da ćemo efektivno iterirati samo jednom kroz cijeli skup primjera, pa je složenost $\mathcal{O}(nN)$. Ako još uvedemo T kao ukupan broj iteracija, onda je ukupna vremenska složenost algoritma $\mathcal{O}(TnNK)$. Složenost je, dakle, linearna u svim relevantnim parametrima, i to je vrlo dobro jer znači da će se algoritam dobro nositi i s velikim skupovima podataka i sa primjerima s mnogo značajki, a i sa zadatcima u kojima je potrebno grupirati u velik broj grupa. U praksi je broj iteracija T redovito mnogo manji od broja primjera N , pa broj iteracija nije ključan faktor pri razmatranju složenosti.

Osim složenosti, zanimat će nas je li ovaj algoritam deterministički, konvergira li i je li optimalan. Razmotrimo prvo determinističnost. Je li ovaj algoritam **deterministički**, u smislu da će dati uvijek iste izlaze za iste ulaze? Primjetite da unutar algoritma treba odabrat početna središta grupe. Očito, ako se taj odabir radi nekako stohastički, onda će i algoritam biti stohastički i time nedeterministički. No, čak i ako je odabir početnih središta deterministički, algoritam ima potencijalni dodatni izvor nedeterminističnosti, a to je razrješavanje izjednačenja udaljenosti dvaju primjera od centroida. Pri implementaciji treba voditi računa da se razrješavanje provodi na proizvoljan, ali konzistentan način (u suprotnom se može dogoditi da algoritam zaglavi u beskonačnoj petlji). Zaključujemo, dakle, da algoritam može biti deterministički, ako se početna središta određuju nekim determinističkim postupkom, a inače će biti stohastički.

Drugo pitanje jest konvergira li algoritam. Za početak, primjetimo da algoritam zapravo pretražuje prostor stanja. Svako stanje je jedno grupiranje (jedna particija primjera + raspored središta). Koliko različitih stanja postoji? Različitih stanja ima K^N (to je broj particija N primjera u K skupova). Dakle, konvergira li algoritam? Da, jer je broj konfiguracija konačan (konfiguracija = particija primjera + raspored središta), a optimizacijski je postupak definiran tako da se kriterijska funkcija J nužno smanjuje kroz iteracije. Iz ta dva svojstva slijedi da algoritam nikada ne posjeće istu konfiguraciju više puta, iz čega slijedi da mora konvergirati.

Konačno, treće pitanje je nalazi li algoritam, jednom kada konvergira, optimalnu particiju. Odgovor na ovo je negativan: algoritam je pohlepan i nalazi lokalno optimalno rješenje, koje ne mora biti globalno optimalno. Rezultat grupiranja, naime, ovisi o odabiru početnih središta. Na primjer:



Ovdje imamo četiri primjera (crne točke). Ako su početno odabrana dva centroida označena plavim križićima, algoritam će konvergirati do centroida označenih crvenim križićima, i grupirati četiri primjera u dvije grupe označene plavom. Međutim, to je suboptimalna particija. Bolje grupiranje bilo bi ono u dvije grupe označene zelenom bojom. To grupiranje bi imalo manji iznos J , jer bi primjeri bili bliže svojim centroidima.

Možemo, dakle, utvrditi da je algoritam K-sredina pohlepan i da pronalazi lokalno optimalno rješenje. Hoće li to rješenje biti i globalno optimalno, ovisi o izboru početnih sredina μ_k . To onda pitanje optimalnosti algoritma prebacuje na pitanje optimalnog odabira početnih sredina. Pogledajmo koje tu sve mogućnosti imamo.

3.3 Odabir početnih sredina

- **Nasumično odabrati K primjera** kao početne vrijednosti μ_k . Ovime se doduše izbjegava postavljanje centroida na mesta u prostoru primjera u kojemu uopće nema primjera (a to bi lako moglo dogoditi ako središta izaberemo posve nasumično), ali se ne rješava problem zaglavljivanja u lokalnom optimumu. Problem također predstavljaju **stršeći primjeri** (engl. *outliers*), koji lako mogu završiti izolirani svaki u svojoj grupama. Na prvi pogled možda se čini da je dobro da takvi primjeri završe u zasebnim grupama, ali to nije tako jer je broj grupe K ograničen i one se trebaju poklapati s “prirodnim” (većinskim) grupama koje postoje u podatcima;
- Izračunati srednju vrijednost (centroid) sviju primjera, μ , a zatim vektoru μ dodavati manje **slučajne vektore** i tako dobiti K vektora μ_k . Ovo rješava problem izoliranih primjera, jer će središta biti centrirana oko “centra mase” podataka (oko točaka u ulaznom prostoru gdje je gustoća vjerojatnosti primjera najveća), no ovo ne rješava problem zaglavljivanja u lokalnome optimumu;
- Izračunati prvu glavnu komponentu skupa primjera metodom **analize glavnih komponenti** (engl. *principal component analysis, PCA*) (prva komponenta definira smjer najveće varijance podataka), razdijeliti raspon na K jedakih intervala, čime se primjeri razdjeljuju u K grupe, a zatim uzeti srednje vrijednosti tih grupa kao početne vrijednosti μ_k . Ovo u načelu također ne rješava problem zaglavljivanja u lokalnom optimumu;
- Slučajno odabrati jedno početno središte μ_k , a zatim svako iduće središte odabrati tako da je što dalje od ostalih središta. Algoritam koji implementira ovakav pristup poznat je pod nazivom **k-means++**. Kod tog je algoritma vjerojatnost da primjer $\mathbf{x}^{(i)}$ bude odabran kao novo središte μ_{k+1} proporcionalna kvadratu udaljenosti tog primjera od njemu najbližeg, već odabranog središta μ_k :

$$P(\mu_{k+1} = \mathbf{x}^{(i)} | \mathcal{D}, \mu_1, \dots, \mu_k) = \frac{\min_k \|\mu_k - \mathbf{x}^{(i)}\|^2}{\sum_j \min_k \|\mu_k - \mathbf{x}^{(j)}\|^2}$$

Premda na ovaj način vrijednosti koje odskaču imaju veću vjerojatnost da budu odabранe za središte, njih je u pravilu manje, pa je ipak vjerojatniji odabir nekog od prosječnih

primjera, koji su brojniji. Pokazano je da ovaj način odabira početnih središta znatno smanjuje pogrešku grupiranja, a također ubrzava konvergenciju algoritma.

U slučajevima kada se početna središta određuju nedeterministički, npr., algoritam k-means++, običaj je algoritam K-sredina pokrenuti više puta i uzeti rezultat sa što manjom pogreškom grupiranja J .

4 Algoritam K-medoida

Algoritam K-sredina jednostavan je i učinkovit algoritam. Međutim, ima jedno veliko ograničenje: primjenjiv je samo u slučajevima kada je primjere moguće prikazati u **vektorskome prostoru**, pa je između njih moguće izračunati euklidsku udaljenost i srednju vrijednost (centroid) grupe. No, kao što smo već ustavili kada smo pričali o jezgrenim funkcijama, nekada primjere ne možemo prikazati u vektorskem prostoru. Npr., ako želimo grupirati riječi tako da ortografski slične riječi završe u istim grupama, nije skroz jasno kako bismo najbolje prikazali riječi kao vektore. Umjesto toga, mnogo je jednostavnije izračunati sličnost između riječi nekom funkcijom koja za par riječi vraća njihovu sličnost (npr., na temelju Levenshteinove udaljenosti). Slično je i ako želimo grupirati ljude na temelju poznanstva, tako da ljudi koji se međusobno poznaju završe u istim grupama. Mnogo je lakše za svaku osobu prikupiti informaciju o tome koliko dobro poznaje neku drugu osobu, nego osobe prikazati kao vektore tako da euklidska udaljenost između tih vektora odgovara upravo mjeri u kojoj se te osobe međusobno poznaju.

Zajedničko ovakvim situacijama jest da primjeri nisu elementi vektorskog prostora, pa između njih ne možemo izračunati euklidsku udaljenost niti možemo izračunati centroid skupa primjera. Umjesto toga, raspolažemo općenitom **mjerom sličnosti** (engl. *similarity measure*) ili njezinim komplementom, **mjerom različitosti** (engl. *dissimilarity measure*), definiranim između svih parova primjera. Mjera sličnosti može se izraziti **matricom sličnosti**.

► PRIMJER

Matrica sličnosti je simetrična kvadratna matrica koja definira sličnost između svih parova primjera u skupu primjera. Npr.:

$$S = \begin{pmatrix} 1 & 0.1 & 0.9 & 0.4 \\ 0.1 & 1 & 0.7 & 0.3 \\ 0.9 & 0.7 & 1 & 0.2 \\ 0.4 & 0.3 & 0.2 & 1 \end{pmatrix}$$

Osim što je simetrična, sličnost je također i refleksivna relacija, tj. svaki primjer je sličan samome sebi sa sličnošću 1, stoga su na dijagonali matrice sličnosti jedinice. Mjeru različitosti možemo dobiti jednostavno kao komplement mjerne sličnosti, $1 - S$.

Primijetite da je mjera sličnosti zapravo isto što i jezgrena funkcija, pa je matrica sličnosti zapravo isto što i **jezgrena matrica**.

Algoritam K-sredina ne može, dakle, raditi s primjerima koji nisu vektori. Međutim, princip tog algoritma – nalaženje prototipa grupe (centroida) i zatim pridjeljivanje primjera najsličnijoj (najbližoj) grupi – može se poopćiti na slučaj kada primjeri nisu vektori. Upravo to radi **algoritam K-medoida**. Taj algoritam je poopćenje algoritma K-sredina koji može raditi s općenitom mjerom različitosti između (moguće nevektorskih) primjera. Kod tog je algoritma kriterijska funkcija definirana pomoću općenite mjerne različitosti $\nu(\mathbf{x}, \mathbf{x}')$ između dvaju primjera:

$$\tilde{J} = \sum_{i=1}^N \sum_{k=1}^K b_k^{(i)} \nu(\mathbf{x}^{(i)}, \boldsymbol{\mu}_k)$$

Mjera različitosti ν (odnosno njoj komplementarna mjeru sličnosti) općenitija je od euklidske udaljenosti i od bilo koje druge mjerne udaljenosti budući da ne mora ispunjavati uvjete metrike (uključivo nejednakost trokuta).

Kod algoritma K-medoida prototipe grupe čine tzv. **medoidi**, odnosno reprezentativni primjeri iz skupa \mathcal{D} , a ne centroidi. Važno je napomenuti da medoid može biti samo neki od primjera iz skupa za učenje. To je drugačije od centroida, koji ne mora odgovarati niti jednom primjeru iz skupa za učenje, tj. to može biti neki nikad viđeni primjer. Razlog zašto medoid može biti samo jedan od primjera iz skupa za učenje je taj što primjeri nisu nužno vektorizirani, pa dakle ne možemo izračunati centroid, već primjere shvaćamo kao nedjeljive entitete, i jedino što o njima znamo jesu sličnosti/različitosti između primjera.

Tipična izvedba algoritma K-medoida jest **algoritam PAM** (engl. *partitioning around medoids*), čiji je pseudokod dan u nastavku.

► Algoritam PAM

- 1: **inicijaliziraj** medoide $\mathcal{M} = \{\boldsymbol{\mu}_k\}_{k=1}^K$ na odabrane $\mathbf{x}^{(i)}$
- 2: **ponavlja**
- 3: za svaki $\mathbf{x}^{(i)} \in \mathcal{D} \setminus \mathcal{M}$
- 4: $b_k^{(i)} \leftarrow \begin{cases} 1 & \text{ako } k = \operatorname{argmin}_j \nu(\mathbf{x}^{(i)}, \boldsymbol{\mu}_j) \\ 0 & \text{inače} \end{cases}$
- 5: za svaki $\boldsymbol{\mu}_k \in \mathcal{M}$
- 6: $\boldsymbol{\mu}_k \leftarrow \operatorname{argmin}_{\boldsymbol{\mu}_j \in \mathcal{D} \setminus \mathcal{M} \cup \{\boldsymbol{\mu}_k\}} \sum_i b_k^{(i)} \nu(\mathbf{x}^{(i)}, \boldsymbol{\mu}_j)$
- 7: dok $\boldsymbol{\mu}_k$ ne konvergiraju

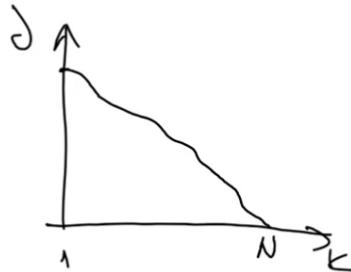
Kao i algoritam K-sredina, algoritam PAM također se izvršava s po dva koraka u svakoj iteraciji. U prvome koraku primjeri se svrstavaju u grupu za koju je vrijednost mjere različitosti najmanja. To iziskuje $\mathcal{O}(K(N - K))$ izračuna mjere različitosti ν . U drugom koraku prototipi grupe odabiru se tako da minimiziraju \tilde{J} . Za svaku od K grupe, svaki od $N - K$ primjera koji trenutno nisu odabrani kao medoidi razmatraju se kao kandidati za medoid umjesto trenutačnog medoida, izračunava se zbroj vrijednosti mjere ν između $N - K$ primjera i kandidata za medoid te se odabire onaj medoid za koji je ta vrijednost najmanja. To iziskuje $\mathcal{O}(K(N - K)^2)$ izračuna mjere različitosti ν . Posljedično, algoritam PAM ukupno iziskuje $\mathcal{O}(TK(N - K)^2)$ izračuna vrijednosti mjere ν , gdje je T ukupan broj iteracija. Vidimo dakle da sada imamo kvadratnu složenost u N , dok smo kod algoritma K-sredina imali linearnu složenost. Visoka vremenska složenost glavni je nedostatak algoritma PAM, pa su predložena različita poboljšanja, u koja međutim sada nećemo ulaziti.

Uvjereni ste li da je ovaj algoritam dobar? 8
Umjesto toga, pozabavite se jednom važnijom temom, a to je odabir broja grupe, odnosno problemom "provjere grupiranja".

5 Provjera grupe

Kod oba algoritma koje smo razmotrili – algoritma K-sredina i algoritma K-medoida – broj grupe, odnosno **hiperparametar** K , potrebno je odrediti unaprijed. To je slučaj kod mnogih algoritama grupiranja (iznimka su tzv. neparametarski algoritmi grupiranja, ali njih nećemo raditi). Odabir optimalnog broja grupe dio je većeg problema koji se naziva **provjera grupe** (engl. *cluster validation*): *koliko je dobro naše grupiranje?* Odabir broja grupe jedan je od glavnih problema kod grupiranja. Idealno, broj grupe odgovarat će broju "prirodnih grupa" u skupu podataka, no taj nam je najčešće nepoznat. Pitanje je onda: kako odrediti optimalan broj grupe K , kada ne znamo unaprijed koliko grupe postoji u podatcima?

Prva stvar koja bi nam možda mogla pasti na pamet jest jednostavno odabrati K koji minimizira pogrešku J . Međutim, to nije dobra ideja. Naime, vrijednost pogreške J u ovisnosti o broju grupe K općenito izgleda ovako:



tj. pogreška J monotono pada s brojem grupa K . Pogreška J biti će jednaka nuli tek onda kada se svaki primjer nalazi sam u svojoj grupi, tj. onda kada je broj grupa jednak broju primjera. No, takvo "grupiranje" potpuno je beskorisno. Problem odabira broja grupa zapravo je analogan **odabiru složenosti modela** kod nadziranog učenja: grupiranje s velikim brojem grupa odgovara složenom modelu. I kod nadziranog smo učenja imali situaciju da najsloženiji model daje najmanju pogrešku.

Dakle, minimizacija funkcije J nije opcija. Pogledajmo koje tehnike stvarno imamo na raspolaganju.

5.1 Tehnike provjere grupe

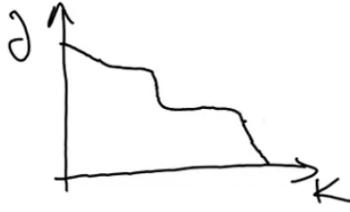
Ručna provjera kvalitete grupe. Inspiciramo primjere u pojedinim grupama i pokušamo dokučiti ima li takvo grupiranje smisla, pozivajući se na znanje o samom problemu koji rješavamo. Ovo je ograničeno upotrebljivo, jer ako primjera ima puno, teško da ćemo išta smisleno moći uočiti.

Smanjenje dimenzionalnosti plus vizualna provjera. Ovdje je ideja da primjere iz visokodimenzijskog ulaznog prostora (dimenzije n) preslikamo u dvodimenzijski prostor i da ih tamo vizualiziramo te očno utvrdimo optimalan broj grupa. Nakon toga primjere grupiramo u dočini broj grupa, ali u izvornom (n -dimenijskom prostoru). Za projiciranje primjera u dvodimenzijski prostor može se upotrijebiti bilo koja tehnika za smanjenje dimenzionalnosti; tipično se koristi **analiza glavnih komponenti** (PCA), **višedimenzijsko skaliranje** (engl. *multidimensional scaling, MDS*), **analiza korespondencije** (engl. *correspondence analysis, CA*) ili metoda **t-SNE** (ova posljednja trenutačno je najpopularnija).

9

Metoda koljena (engl. *elbow method*). Grafički prikažemo ovisnost kriterijske funkcije o parametru K i tražimo "koljeno" krivulje (mjesto gdje funkcija naprije naglo pada i zatim stagnira ili pada vrlo sporo). S porastom broja grupa K , vrijednost kriterijske funkcije će padati, međutim taj pad općenito neće biti ujednačen. Naime, za neke vrijednosti K algoritam će početi razdjeljivati prirodne grupe. Ako povećanjem broja grupa kriterijska funkcija brzo opadne i onda neko vrijeme stagnira, to je signal da smo upravo "uhvatili" neko prirodno grupiranje u podatcima. Naime, ako s porastom K vrijednost J naglo opada, to znači da s malim povećanjem broja grupa dobivamo znatno bolje grupiranje. U trenutku kada s povećanjem broja grupa K više ne dobivamo znatno bolje grupiranje, tj. kada J počne vrlo sporo opadati, to onda znači da novododane grupe ne hvataju baš neke prirodno grupirane primjere. Takvih koljena funkcije J može biti više, pogotovo ako grupe unutar sebe imaju podgrupe. Dakle, dobar odabir za broj grupa K jesu one vrijednosti koje odgovaraju točkama neposredno nakon koljena funkcije J , jer smo na tim mjestima upravo pogodili neki "prirodni" broj grupa. Npr., na donjoj slici imamo jedno takvo koljeno:

10



Ako je algoritam grupiranja nedeterministički, npr., algoritam K-sredina sa slučajno odabranim početnim središtimi, za svaki izbor vrijednosti K treba napraviti više mjerena za J , pa uzeti srednju vrijednost.

Analiza siluete. Slično kao metoda koljena, analiza siluete je grafička metoda. Metoda se provodi tako da se grafički prikaže "vrijednost siluete" za svaki primjer iz skupa \mathcal{D} , i zatim se na temelju toga očno odredi optimalan broj grupa. Vrijednost siluete za primjer $\mathbf{x}^{(i)}$ definirana je ovako:

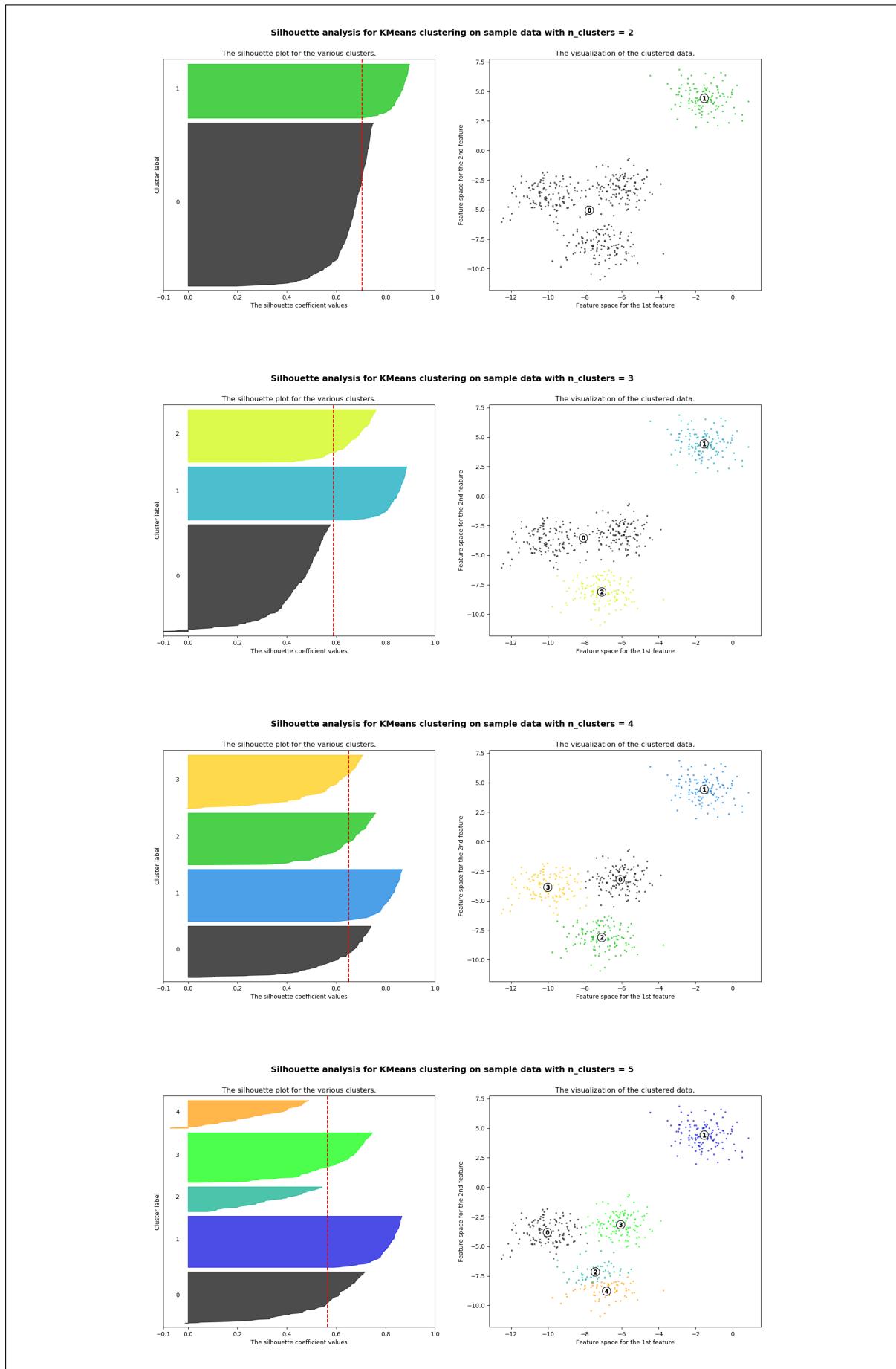
$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

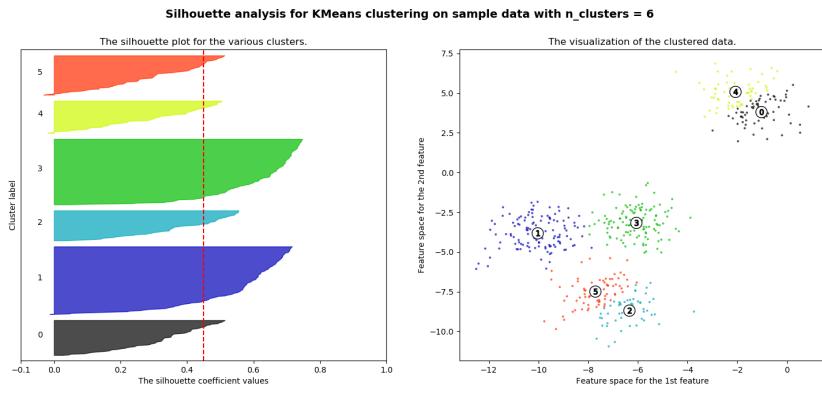
Vrijednost $a(i)$ je prosječna udaljenost primjera $\mathbf{x}^{(i)}$ do svih drugih primjera iz iste grupe u kojoj se nalazi primjer $\mathbf{x}^{(i)}$. Vrijednost $b(i)$ je također prosječna udaljenost primjera $\mathbf{x}^{(i)}$ do svih drugih primjera iz grupe, ali se sada razmatraju sve druge grupe u kojima se ne nalazi primjer $\mathbf{x}^{(i)}$, te se uzima grupa za koju je prosječna udaljenost najmanja. Dakle, $b(i)$ će biti najmanja prosječna udaljenost između primjera $\mathbf{x}^{(i)}$ i svih primjera najbliže susjedne grupe. Vrijednost siluete je u intervalu $[-1, +1]$. Ako je $s(i) = 1$, to znači da je primjer jako udaljen od primjera iz susjedne grupe, a vrlo blizu primjerima iz svoje grupe. Obrnuto, ako je $s(i) = -1$, to bi značilo da je primjer pogrešno grupiran jer je bliže primjerima iz druge grupe nego primjerima iz svoje grupe (to se ne može dogoditi kod algoritma K-sredina, ali se možda može dogoditi kod nekih drugih algoritama). Primjeri za koje $s(i) = 0$ nalaze se negdje na granici između dviju ili više grupa. Ideja analize siluete jest da izračunavamo $s(i)$ za sve primjere iz \mathcal{D} i grafički prikazujemo te vrijednosti. Također, izračunavamo prosječnu vrijednost siluete za sve primjere iz skupa podataka. Pogledajmo primjer.

► PRIMJER

U nastavku je prikazana analiza siluete za algoritam K-sredina, za različit broj grupa, $K \in \{2, 3, 4, 5, 6\}$. Na desnoj strani prikazan je rezultat grupiranja u dvodimenziskom ulaznom prostoru. Primjeru su, naravno, za svaki K jednakoraspoređeni u ulaznom prostoru, ali su moguće drugačije grupirani. Boja primjera odgovara oznaci grupe u koju je primjer dodijeljen. (Primijetite da oznaka grupe kojoj primjer pripada nije poznata unaprijed, već se dobiva grupiranjem; kod algoritma K-sredina to je jednostavno grupa čijemu je centroidu primjer najbliži). Na lijevoj strani prikazana je silueta, i to tako da su na x -osi prikazane vrijednosti siluete za svaki primjer. Primjeri su grupirani prema oznaci grupe u koju pripadaju, i to u padajućem redoslijedu prema vrijednosti siluete. Okomita crvena iscrtkana linija označava prosjek vrijednosti siluete na cijelom skupu primjera.

11





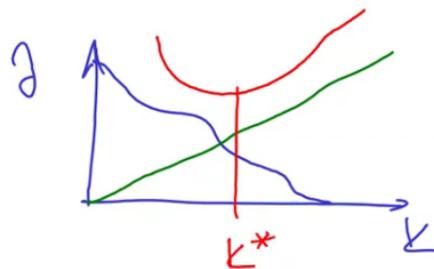
Lošim grupiranjima smatraju se ona grupiranja kod kojih su sve vrijednosti siluete za neku grupu manje od prosječne vrijednosti siluete za cijeli skup primjera. Ovdje su to grupiranja sa $K = 3$ i $K = 5$. Također, kod tih grupiranja postoji značajna varijanca u iznosima vrijednosti siluete. Npr., sa $K = 5$, kao što se vidi iz prikaza ulaznog prostora, narančasta grupa i tirkizna grupa preblizu su jedna drugoj i razdjeljivanje primjera u dvije grupe ovdje nije prirodno, što za posljedicu ima niske vrijednosti siluete za primjere koji su bliži drugoj grupi nego svojoj grupi. Za razliku od grupiranja sa $K = 3$ i $K = 5$, grupiranja sa $K = 2$, $K = 4$ i $K = 6$ su u redu. Zapravo, grupiranje sa $K = 4$ je možda najbolje jer su grupe približno jednakih veličina, što se lijepo vidi iz prikaza siluete (debljine silueta za svaku grupu su ujednačene).

U ovom konkretnom primjeru, budući da su primjeri u dvodimenzijском ulaznom prostoru, grupe smo mogli provjeriti i izravno u ulaznom prostoru, međutim imajte na umu da metodu analize siluete možemo primijeniti i za grupiranje u visokodimenzijском prostoru.

Minimizacija regularizirane funkcije pogreške. Ideja ovog postupka za odabir grupe slična je ideji **regularizacije**, koju smo kod nadziranog učenja koristili za sprječavanje prenaučenosti modela. Ideja je da proširimo kriterij grupiranja koji minimiziramo, tako da on kombinira izvornu kriterijsku funkciju i složenost modela te na neki način kažnjavamo modele s prevelikim brojem grupa. Općenit oblik tog kriterija je:

$$K^* = \operatorname{argmin}_K (J(K) + \lambda K)$$

gdje je $J(K)$ vrijednost kriterijske funkcije za model s K grupa, a hiperparametar λ je faktor regularizacije. Veće vrijednosti faktora λ favoriziraju rješenja s manjim brojem grupa, budući da će kazna za veći broj grupa biti veća ako je λ veći. Za $\lambda = 0$ povećanje broja grupa uopće se ne kažnjava i optimalan broj grupa tada je $K^* = N$. Ovdje, dakle, tražimo minimum zbroja dviju funkcija, jedne koja monotono opada (funkcija J) i druga koja raste linearno (λK). To izgleda ovako:



Što je λ veći, to je veći nagib druge funkcije, i minimum K^* će biti to manji, tj. odabrat ćemo manji broj grupa. Ideja je slična regularizaciji utoliko što funkciju pogreške kombiniramo s regularizacijskim izrazom koji ovisi o složenosti modela. Razlika u odnosu na

regularizaciju kod nadziranog učenja je u tome što se regularizacija tamo provodi u sklopu optimizacije, dok je ovdje provodimo nakon optimizacije (tj. nakon grupiranja).

Naravno, tu je sad problem prebačen s odabira optimalnog K na odabir optimalne λ . Kako bismo odabrali optimalnu λ ? (Unakrsna provjera ovog puta nije točan odgovor, jer nemamo označenih primjera, pa dakle niti nemamo ispitni skup primjera na kojima bismo mogli ispitati da li model dobro generalizira.) Ideja je da je odabir hiperparametra λ možda lakši problem nego odabir hiperparametra K . Odabir parametra λ može se temeljiti na našem iskustvu stečenome na grupiranju sličnih skupova podataka – skupova koji su možda imali drugačiji broj grupe, ali svi imaju približno istu λ , jer se odnose na isti ili sličan problem. Npr., zamislimo da želimo grupirati filmove tako da slični filmovi (po glumačkoj postavi, žanru, produkciji, sadržaju itd.) završe u istim grupama, i da to radimo na malom skupu podataka i zatim na velikom skupu podataka. Očekujemo da će broj grupe u ta dva slučaja biti različit, ali λ bi trebala biti više-manje ista, jer λ indirektno određuje što znači da su filmovi dovoljno slični da pripadaju u istu grupu, pa na taj način λ indirektno određuje broj grupe K u ovisnosti o veličini skupa primjera. Dakle, jednom određeni λ možemo koristiti za grupiranje na raznim skupovima podataka, dok god rješavamo približno sličan problem. Drugim riječima, na odabir hiperparametara λ možemo gledati kao na “meki odabir” broja grupe (odabir koji je prilagođen konkretnim podatcima).

12

Alternativa optimizaciji hiperparametra λ jest da koristimo neku teorijsku utemeljenu regulariziranu funkciju pogreške, poput **Akaikeova informacijskog kriterija (AIC)**. Konkretan oblik kriterija AIC za algoritam K-sredina je:

$$K^* = \operatorname{argmin}_K (J(K) + 2nK)$$

13

Ovdje koristimo $\lambda = 2n$, što znači da se za složenost modela uzima da je proporcionalna umnošku broja značajki n i broja grupe K .

Točnost na podskupu primjera. Ovaj se pristup temelji na usporedbi dobivenog grupiranja s grupiranjem koje smatramo točnim, tzv. **referentno grupiranje**. Naravno, mi ne znamo koje je točno grupiranje – kada bismo to znali, ne bismo ni trebali raditi grupiranje. No, referentno grupiranje možemo utvrditi na manjem uzorku primjera: za manji, slučajno odabran broj primjera odredit ćemo njihovu točnu grupu (ovdje pretpostavljamo da su nam grupe unaprijed poznate, ali nam nije poznata raspodjela primjera u grupe). Jednom kada smo to napravili, možemo grupirati zajedno označene i neoznačene primjere, a onda napraviti provjeru samo na označenim primjerima. Kao mjeru pogreške možemo koristiti bilo koju mjeru za točnost grupiranja: npr., **Randov indeks** (engl. *Rand index*), mjera **normalizirane uzajamne informacije** (engl. *normalized mutual information*, NMI) ili mjeru F_1 . O Randovom indeksu pričat ćemo u nastavku, a o ostalim mjerama u predavanju posvećenome vrednovanju modela.

14

15

5.2 Randov indeks

Pogledajmo sada **Randov indeks**. Randov indeks je mjera koja izračunava u kojoj mjeri dobiveno grupiranje odgovara referentnom grupiraju (grupiranju koje smatramo točnim). Randov indeks zapravo nije ništa drugo nego mjera **točnosti** izračunata na razini **parova primjera**. Što to znači? To znači da za svaki mogući par iz skupa primjera gledamo jesu li ta dva primjera završila u istoj grupi ili nisu. Ako su završili u istoj grupi, onda par primjera smatramo pozitivnim, inače ga smatramo negativnim. Ako su oba primjera iz para završila u istoj grupi, i ako su doista trebala završiti u istoj grupi, onda je taj par primjera **stvarno pozitivan** (engl. *true positive*, TP). Obrnuto, ako su primjeri iz para razdvojeni, tj. svaki je završio u drugoj grupi, a

16

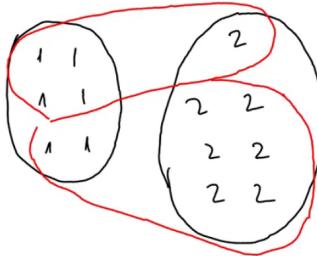
doista nisu trebala završiti u istoj grupi, onda je taj par primjera **stvarno negativan** (engl. *true negative*, TN). Randov indeks definiran je onda ovako:

$$R = \frac{a + b}{\binom{N}{2}}$$

gdje je a broj jednakoznačenih parova u istim grupama (tj. broj stvarno pozitivnih parova, TP), a b je broj različito označenih parova u različitim grupama (tj. broj stvarno negativnih parova, TN). Primijetite da u brojniku imamo $a + b = \text{TP} + \text{TN}$, što je ukupan broj točno grupiranih parova, dok je u nazivniku ukupan broj parova, kojih je $\binom{N}{2}$. Prema tome, Randov indeks je zapravo udio točno grupiranih parova mjera u ukupnom broju primjera, odnosno, kao što smo već rekli, to je točnost grupiranja izračunata na razini parova primjera.

► PRIMJER

Imamo $N = 13$ primjera. Referentno grupiranje sadrži dvije grupe – dakle, u stvarnosti primjeri zapravo dolaze iz dvije grupe. Primjere grupiramo u $K = 2$ grupe. Ovdje broj grupe K odgovara stvarnom broju grupa (broju grupa u referentnom grupiranju), no to ne mora uvijek biti tako (naprotiv, u stvarnosti ćemo rijetko pogoditi baš pravi broj grupa). Grupiranje je rezultiralo ovakvim dvjema grupama:



Crnom bojom označene su dvije referentne grupe (grupiranje koje smatramo točnim), a crvenom bojom dvije označene su grupe dobivene algoritmom grupiranja. Ovakav rezultat grupiranja možemo sažeto napisati ovako:

$$\{\{1, 1, 1, 1, 2\}, \{1, 1, 2, 2, 2, 2, 2, 2\}\}$$

gdje prvi i drugi skup odgovaraju prvoj odnosno drugoj dobivenoj grupi, a brojke označavaju iz koje je referentne grupe svaki od primjera. Vidimo da dobiveno grupiranje nije savršeno točno: u prvu grupu dobivenu algoritmom grupiranja ušuljao nam se jedan primjer iz druge referentne grupe, dok su u drugu grupu dobivenu algoritmom grupiranja upala dva primjera iz prve referentne grupe.

Na temelju ovih podataka lako možemo izračunati Randov indeks. Budući da imamo $N = 13$ primjera, to znači da je ukupan broj parova primjera jednak $\binom{N}{2} = \binom{13}{2} = 78$. U prvoj grupi imamo četiri primjera iz prve referentne grupe. Dakle, svi ti parovi primjera su ispravno grupirani. Ukupan broj tih točnih parova je $\binom{4}{2}$. Analogno računamo za ostale primjere u obje dobivene grupe.

Slično, vidimo da je naše grupiranje ispravno razdvojilo neke parove primjera. U prvoj našoj grupi su 4 primjera iz prve referentne grupe, a u drugoj grupi je 6 primjera iz druge referentne grupe. Dakle, $4 \cdot 6$ parova je ispravno razdvojeno. Nadalje, u prvoj grupi imamo 1 primjer iz druge referentne grupe, dok u drugoj grupi imamo 2 primjera iz prve referentne grupe, i ti su primjer također ispravno razdvojeni, što daje još $1 \cdot 2$ ispravno razdvojenih parova. Imamo, dakle:

$$\begin{aligned} a &= \binom{4}{2} + \binom{1}{2} + \binom{2}{2} + \binom{6}{2} = 6 + 1 + 0 + 15 = 22 \\ b &= 4 \cdot 6 + 1 \cdot 2 = 26 \\ R &= \frac{22 + 26}{78} = 0.62 \end{aligned}$$

Dobili smo Randov indeks od 0.62, odnosno 62%, što baš i ne zvuči tako dobro.

U ovom je primjeru odabrani broj grupe K i broj grupa u referentnom grupiranju bio identičan. No to općenito ne mora biti tako. Randov indeks funkcioniра i kada je broj grupe različit od

referentnog broja grupa, npr., kada u referentnom grupiranju postoje tri grupe, a mi grupiramo u samo dvije grupe.

Randov indeks je mjera točnosti, pa će biti u intervalu $[0, 1]$. Što je grupiranje točnije, tj. što se više podudara s referentnim grupiranjem, to će Randov indeks biti veći. Korisno je razmotriti kako se vrijednost Randovog indeksa mijenja u ovisnosti broju grupa K . Ako je broj grupa premalen, onda će grupe biti velike, pa ćemo u iste grupe smještati mnoge parove primjera koji ne bi trebali biti u istoj grupi, tj. imat ćemo mnogo lažno pozitivnih parova primjera. S druge strane, ako je broj grupa prevelik, onda će grupe biti malene, pa ćemo razdvojiti mnoge parove primjera koji bi trebali biti smješteni u istu grupu, tj. imat ćemo mnogo lažno negativnih parova primjera. U oba slučaja, ako je K prevelik ili premalen, Randov indeks bit će manji nego kada odaberemo optimalan broj grupa, tj. broj grupa koji najbolje odgovara referentnom grupiranju. Zaključujemo, dakle, da je Randov indeks unimodalna funkcija broja grupa K :



Randov indeks, dakle, uspoređuje grupiranje s referentnim grupiranjem. Možemo ga koristiti ako nam je poznato referentno grupiranje. To je korisno u situacijama kada, npr., imamo standardni skup podataka za koji je poznato koje su grupe u njemu, pa isprobavamo različite algoritme grupiranja, uključivo neki novi algoritam koji smo razvili. Međutim, kao što smo već napomenuli, u stvarnim primjenama grupiranja najčešće ipak ne znamo koji primjeri pripadaju u koju grupu, pa onda Randov indeks računamo na manjem, ručno označenom uzorku primjera.

U praksi se umjesto Randovog indeksa češće koristi **prilagođen Randov indeks** (engl. *adjusted Rand index*), koji mjeru korigira s obzirom na očekivanu sličnost između slučajno grupiranih parova primjera.

Sažetak

- Ako primjeri nisu označeni (ne znamo ih označiti ili je preskupo) moramo koristiti **nenadzirano strojno učenje**
- Tipičan zadatak je **grupiranje**, kojim se primjeri razdjeljuju u grupe prema udaljenosti/sličnosti
- Grupiranje može biti **tvrdo** ili **meko**, **particijsko** ili **hijerarhijsko**
- **Algoritam K-sredina** računa središta grupa i primjere razdjeljuje u grupe s najbližim središtimi, smanjujući postepeno funkciju pogreške. **Odabir početnih središta** utječe na grupiranje
- **Algoritam K-medoida (PAM)** je popravljanje na proizvoljnu mjeru različitosti (ne nužno udaljenost u vektorskom prostoru)
- Ključan problem grupiranja jest **odabir optimalnog broja grupa**, za što postoji nekoliko metoda
- **Randov-indeks** je mjera za **provjeru grupa** koja računa točnost grupiranja parova primjera u odnosu na referentno grupiranje

Bilješke

- 1 Ovo predavanje zasniva se na poglavljima 7.1, 7.3 i 7.8 iz (Alpaydin, 2020) te poglavlju 9.1 iz (Bishop, 2006), ali je značajno prošireno. Ipak, u odnosu na nadzirano strojno učenje, nenadziranom strojnom učenju, odnosno konkretno grupiranju, posvetit ćemo zapravo vrlo malo pažnje. Ako će vas zainteresirati algoritmi grupiranja, predlažem da produbite svoje znanje tako da krenete od nekih od sljedećih preglednih radova: (Xu and Wunsch, 2005; Berkhin, 2006; Xu and Tian, 2015).
- 2 Diskutabilno je bismo li **neoznačene primjere** željeli zvati "primjeri". Točnije bi bilo "instance". Međutim, u nastavku nećemo komplikirati, zvat ćemo ih i dalje "primjeri", ali imajte na umu da nemamo oznaku y , nego samo vektor \mathbf{x} .
- 3 Zadatak otkrivanja autora teksta naziva se **atribucija autorstva** (engl. *authorship attribution*). Nenadzirane metode, uključivo grupiranje, tipično se koriste da bi se otkrilo je li neki tekst (tipično se radi o povijesnim tekstovima), za koji se prethodno možda smatralo da je djelom jednog autora, zapravo napisalo više autora. U tu se svrhu tipično analiziraju stilometrijske značajke teksta te se zatim provodi grupiranje. Npr., u Savoy (2019) koristi se hijerarhijsko grupiranje za analizu autorstva Poslanica apostola Pavla. Standardni skup podataka za testiranje metoda atribucije autorstva su *Federalistički spisi* (engl. *The Federalist Papers*), nastali uoči ratifikacije ustava Sjedinjenih Američkih Država krajem 18. stoljeća. Za dvanaest spisa od njih 88 ne zna se puzdano je li autor Alexander Hamilton ili James Madison. To je motiviralo razvoj metoda za analizu autorstva, najprije ručnih, a kasnije računalnih metoda, npr., (Holmes and Forsyth, 1995), a sve to u okviru područja koje se danas naziva **računalna lingvistička forenzika**. Za dobar pregled metoda atribucije autorstva, pogledajte (Stamatatos, 2009).
- 4 Kao primjere znanstvenih radova koji grupiranje koriste za detekciju emocija iskazanih u tekstu, pogledati (Aragón et al., 2019; Mohammad and Kiritchenko, 2015; Chatzakou et al., 2013).
- 5 Algoritam je osmislio Stuart Lloyd 1957. godine, ali je objavljen tek tridesetak godina kasnije u (Lloyd, 1982). Naziv ovog rada ("Kvantizacija metodom najmanjih kvadrata za pulsno-kodnu manipulaciju") reflektira činjenicu da algoritam minimizira kvadratno odstupanje primjera od centroida grupe i da je izvorno bio namijenjen za pulsno-kodnu modulaciju.
- 6 Premda algoritam K-sredina linearno ovisi o svim relevantnim parametrima, problem u praksi ipak predstavlja visokodimensijski ulazni prostori, dakle velik n . To je pogotovo problem u nekim područjima primjene grupiranje, npr., kod obrade prirodnog jezika, gdje prostor može imati onoliko dimenzija koliko ima riječi u jeziku. U takvim se slučajevima mogu koristiti varijante algoritma K-sredina, npr., inačica sa skraćenim (engl. *truncated*) centroidima; v. <https://nlp.stanford.edu/IR-book/html/htmledition/k-means-1.html>. U drugim, pak, primjenama problem može predstavljati velik broj primjera N , koji ne stanu svi odjednom u memoriju. Tada se može koristiti **algoritam K-sredina s mini-grupama** (engl. *mini-batch K-means*) (Sculley, 2010). Popis različitih varijanti algoritma K-sredina, neke od kojih ciljaju smanjiti računalnu složenost, dok druge mijenjaju induktivne pretpostavke algoritma, možete naći na https://en.wikipedia.org/wiki/K-means_clustering#Variations.
- 7 **Analiza glavnih komponenti** (engl. *principal component analysis*, PCA) važna je tehnika za smanjenje dimenzionalnosti koja se koristi i u strojnom učenju i statistici. Mi se u ovom predmetu nažalost ne bavimo metodama za smanjenje dimenzionalnosti. Ako se slučajno dogodi da se nigdje na FER-u ne susretnete sa PCA, svakako preporučam da tu metodu savladate sami. U tu svrhu preporučam (Shlens, 2014) i (Smith, 2002).
- 8 Jedno poboljšanje algoritma PAM, razvijeno upravo s ciljem smanjenja vremenske složenosti, jest nedavno predložen **algoritam BanditPAM** (Tiwari et al., 2020). Zahvaljujem Domagoju Alagiću i Marijanu Smetku na ovoj informaciji.
- 9 **t-SNE** je kratica od **t-distribuirana stohastička ugradnja susjeda** (engl. *t-distributed stochastic neighbor embedding*). Metodu su izvorno predložili u (Hinton and Roweis, 2002), a nadogradili u (Maaten and Hinton, 2008). Pristupačno objašnjenje metode t-SNE možete naći na <https://towardsdatascience.com/t-sne-clearly-explained-d84c537f53a>. Metoda t-SNE je vrlo moćna, no ponekad generira neintuitivne vizualizacije u kojima je lako vidjeti više nego što doista pos-

toji u podatcima, pa, ako želite koristiti t-SNE za vizualizaciju podataka, svakako pročitajte ovo: <https://distill.pub/2016/misread-tsne/>. Pregled ostalih često korištenih tehnika za smanjenje dimenzionalnosti možete naći u (Van Der Maaten et al., 2009) i (Lee and Verleysen, 2010).

- [10] **Metoda koljena ili metoda laka.**
- [11] Primjer preuzet sa https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html.
- [12] Postoje algoritami grupiranja koji, umjesto broja grupe K , imaju neki hiperparametar, poput hiperparametra λ , koji indirektno određuje broj grupe, i to već pri samom grupiranju. Primjer je, npr., **algoritam propagacije afiniteta** (engl. *affinity propagation*) (Frey and Dueck, 2007) ili **algoritam Markovljevog grupiranja** (engl. *Markov Cluster Algorithm*) (Van Dongen, 2000). Ove algoritme ima smisla koristiti ako ne znamo unaprijed koji je optimalan broj grupe (a najčešće to ne znamo), a želimo provoditi grupiranje na skupovima podataka različitih veličina. Na primjer, ako želimo grupirati slične fotografije u rezultatima tražilice, onda ne možemo unaprijed znati koliko različitih grupa fotografija postoji za pojedini upit postavljen preko tražilice. Naprotiv, broj grupe bit će općenito različit za svaki upit.
- [13] **Akaikeov informacijski kriterij** osmislio je japanski statističar Hirotugu Akaike. Kriterij je zasnovan na teoriji informacije i izvorno je korišten kod modela **linearne regresije** za korekciju log-izglednosti uslijed prenaučenosti, no kasnije je našao općenitu primjenu u strojnem učenju za **odabir modela**. Zanimljivost Akaikeovog kriterija jest da je agnostičan na duboku podjelu u statistici između frekventističke i bayesovske statistike, i da zapravo sâm može biti korišten kao teorijski temelj statistike. Inače, pitanje **temelja statistike** je otvoreno (filozofsko ali i praktično) pitanje o prirodi i načinu statističkog zaključivanja, u kojem se, pored ostalog, debatira između frekventističke i bayesovske statistike. Više na https://en.wikipedia.org/wiki/Foundations_of_statistics, ili u (Savage, 1972) i (Efron, 2005).
- [14] U ovakovom kontekstu, treba razlikovati **vanjski (ekstrinzični) kriterij** i **unutarnji (intrinzični) kriterij** grupiranja: mjera koju koristimo za mjerjenje točnosti grupiranja predstavlja **vanjski kriterij**, dok kriterij koji algoritam interno koristi za samo grupiranje (npr., kriterijska funkcija kod algoritma K-sredina) predstavlja **unutarnji kriterij**. Pretpostavka je da je unutarnji kriterij, po kojem se provodi grupiranje, dobro uskladen s vanjskim kriterijem, koji mjerimo i koji nam je bitan. Ako to nije slučaj – ako unutarnji i vanjski kriterij grupiranja nisu dobro uskladjeni – onda nas ne treba čuditi ako algoritam grupiranja u smislu vanjskog kriterija daje suboptimalne rezultate.
- [15] Dobar pregled raznih mjera za provjeru grupe možete naći u (Amigó et al., 2009).
- [16] Randov indeks predložio je 1971. godine američki statističar William Rand sa MIT-a (Rand, 1971).
- [17] U idealnom slučaju, za neki odabrani algoritam grupiranja i neko referentno grupiranje, Randov indeks bit će u pravilu maksimalan ako je odabrani broj grupe K jednak pravom (“prirodnom”) broju grupe u podatcima. Međutim, ova tvrdnja dolazi s jednom važnom napomenom, a to je da K koji maksimizira Randov indeks ipak ne mora uvijek odgovarati “prirodnom” broju grupe u podatcima. To, naime, ipak ovisi i o algoritmu grupiranja. Konkretnije, svaki algoritam grupiranja ima svoju induktivnu pristranost, i moguće je da algoritam jednostavno nije u stanju naći grupiranje koje doista odgovara prirodnom grupiranju. Kao i uvijek u strojnem učenju, induktivna pristranost algoritma treba biti uskladjena s onime što imamo u podatcima, inače se ne trebamo čuditi da algoritam ne radi dobro. Dobar primjer za neusklađenost induktivne pristranosti algoritma grupiranja i podataka jest kada algoritam K-sredina koristimo za grupiranje primjera koji u ulazom prostoru čine grupe koje nisu sferične (npr., izdužene grupe, ili grupe koje se isprepliću i slično). Naime, time što koristi euklidsku udaljenost za pridjeljivanje primjera u najbližu grupu, algoritam K-sredina zapravo pretpostavlja da su grupe sferične. Ako one to nisu, algoritam K-sredina neće nam moći pronaći “prirodne” grupe. U takvim se situacijama može dogoditi da algoritam K-sredina daje bolje grupiranje (ostvaruje veći Randov indeks) s nekim brojem grupe K koji se ne podudara s pravim brojem grupa.
- [18] Korisno je primjetiti da Randov indeks možemo koristiti i onda kada ne znamo koji primjer pripada kojoj grupi, ali znamo da li dva primjera trebaju pripadati istoj grupi. Naime, budući da Randov indeks točnost izračunava nad parovima primjera, dovoljno je označiti **uzorak parova primjera**

tako da za svaki par označimo je li pozitivan (primjeri trebaju biti grupirani zajedno) ili negativan (primjeri trebaju biti razdvojeni). Iz takvog uzorka možemo onda izračunati sve što nam treba za Randov indeks: znamo ukupan broj parova u uzorku, a broj stvarno pozitivnih i stvarno negativnih parova dobit ćemo usporedbom rezultata grupiranja sa parovima iz našeg uzorka. Dakle, referentno grupiranje ne mora nužno doista grupirati primjere u grupe, nego može samo odrediti koji parovi primjera trebaju biti zajedno grupirani, a koji ne. Primijetite da je to mnogo lakši zadatak nego pridijeliti grupu svakom primjeru, jer ne pretpostavlja da znamo koje grupe postoje niti koliko ih je ukupno. Npr., ako grupiramo riječi nekog jezika u grupe značenjski sličnih riječi, onda možemo relativno dobro za neki par riječi odrediti znaće li te riječi isto ili ne, dok bi puno teži zadatak bio da unaprijed odredimo sva moguća značenja riječi nekog jezika i onda riječi razvrstavamo u takve grupe.

- 19 Za **prilagođen Randov indeks**, pogledati: https://en.wikipedia.org/wiki/Rand_index#Adjusted_Rand_index. Za razliku od Randovog indeksa, prilagođeni Randov indeks može poprimiti i negativne vrijednosti.

Literatura

- E. Alpaydin. *Introduction to machine learning*. MIT press, 2020.
- E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information retrieval*, 12(4):461–486, 2009.
- M. E. Aragón, A. P. López-Monroy, L. C. González-Gurrola, and M. Montes. Detecting depression in social media using fine-grained emotions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1481–1486, 2019.
- P. Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.
- C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
- D. Chatzakou, V. Koutsonikola, A. Vakali, and K. Kafetsios. Micro-blogging content analysis via emotionally-driven clustering. In *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, pages 375–380. IEEE, 2013.
- B. Efron. Bayesians, frequentists, and scientists. *Journal of the American Statistical Association*, 100(469):1–5, 2005.
- B. J. Frey and D. Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.
- G. E. Hinton and S. Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15:857–864, 2002.
- D. I. Holmes and R. S. Forsyth. The federalist revisited: New directions in authorship attribution. *Literary and Linguistic computing*, 10(2):111–127, 1995.
- J. A. Lee and M. Verleysen. Unsupervised dimensionality reduction: overview and recent advances. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.
- S. Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137, 1982.

- L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.
- S. M. Mohammad and S. Kiritchenko. Using hashtags to capture fine emotion categories from tweets. *Computational Intelligence*, 31(2):301–326, 2015.
- W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.
- L. J. Savage. *The foundations of statistics*. Courier Corporation, 1972.
- J. Savoy. Authorship of pauline epistles revisited. *Journal of the Association for Information Science and Technology*, 70(10):1089–1097, 2019.
- D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178, 2010.
- J. Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.
- L. I. Smith. A tutorial on principal components analysis. Technical report, 2002.
- E. Stamatatos. A survey of modern authorship attribution methods. *Journal of the American Society for information Science and Technology*, 60(3):538–556, 2009.
- M. Tiwari, M. J. Zhang, J. Mayclin, S. Thrun, C. Piech, and I. Shomorony. BanditPAM: Almost linear time k-medoids clustering via multi-armed bandits. In *NeurIPS*, 2020.
- L. Van Der Maaten, E. Postma, and J. Van den Herik. Dimensionality reduction: a comparative review. *J Mach Learn Res*, 10(66-71):13, 2009.
- S. M. Van Dongen. *Graph clustering by flow simulation*. PhD thesis, 2000.
- D. Xu and Y. Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.
- R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

20. Grupiranje II

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v2.3

Prošli smo puta počeli pričati o **nenadziranom strojnom učenju**, dakle učenju iz podataka koji nemaju oznaku y . Rekli smo da postoje razni pristupi nenadziranom strojnom učenju, ali da se najčešće koristi **grupiranje**, kod kojega primjere koji su međusobno slični želimo smjestiti u iste grupe, a oni koji su različiti u različite grupe. Spomenuli smo i da pristupa grupiranju ima raznih: grupiranje može biti **particijsko** ili **hijerarhijsko**, te može biti **tvrdo** ili **meko**. Zatim smo pričali o algoritmima K-sredina i K-medoida, koji su oba algoritmi za tvrdo partijsko grupiranje.

Danas nastavljamo s grupiranjem, no bavit ćemo se dvama algoritmima koji ne rade partijsko tvrdo grupiranje. Prvo ćemo razmotriti algoritam koji nazivamo **model Gaussove mješavine (GMM)**, koji radi **meko partijsko grupiranje**. Kao što ćemo vidjeti, taj je algoritam zapravo probabilističko poopćenje algoritma K-sredina, odnosno algoritam K-sredina možemo shvatiti kao poseban slučaj modela Gaussovih mješavina. Također ćemo vidjeti da je model Gaussove mješavine zapravo generativan model s latentnim varijablama, za čije će nam treniranje trebati nov algoritam, **algoritam maksimizacije očekivanja** ili **EM-algoritam**. Na kraju ćemo razmotriti jedan posve drugačiji, a vrlo jednostavan algoritam: **hijerarhijsko aglomerativno grupiranje**, koji služi za **tvrdo hijerarhijsko grupiranje**, dakle algoritam koji iz podataka inducira hijerarhiju grupa.

[1]

1 Model Gaussove mješavine

Prošli put bavili smo se algoritmima K-sredina i K-medoida. To su algoritmi tvrdog partijskog grupiranja. Međutim, u nekim problemima primjere ne želimo grupirati u tvrde grupe. Naime, ponekad primjeri legitimno mogu pripadati u više grupe, tj. može postojati preklapanje između grupe. Npr., ako radimo grupiranje ljudi u grupe po crtama ličnosti, gdje jedna grupa odgovara jednoj crti ličnosti, onda će svaka osoba imati kombinaciju više crta ličnosti, što znači da treba pripadati više grupe. Slično ako radimo grupiranje riječi po značenjima, gdje jedna grupa odgovara jednom značenju, onda jedna riječ može imati više značenja, pa može istovremeno pripadati u više grupe. Ovdje se dakle radi se tome da primjer pripada u više grupe istovremeno. Druga je mogućnost da svaki primjer ipak pripada uvijek u samo jednu grupu, ali da mi ne znamo koju, pa bismo željeli nekako modelirati tu neizvjesnost i to modeliramo tako da za svaku grupu definiramo vjerojatnost da joj primjer pripada. U oba slučaja – bilo da primjer pripada u više grupe istovremeno, ili da pripada samo jednoj, ali ne znamo točno kojoj – radi se o **mekom grupiranju**.

Konkretno, mi ćemo razmotriti **probabilističko grupiranje**, gdje svaki primjer svakoj grupi pripada s nekom vjerojatnošću. Alternativa bi bila **neizrazito grupiranje** (engl. *fuzzy clustering*), gdje svaki primjer svakoj grupi pripada s nekom mjerom pripadnosti. Vratit ćemo se kasnije na ovu razliku.

Najjednostavniji i najčešće korišten algoritam probabilističkog grupiranja je **model Gaussove mješavine** (engl. *Gaussian mixture model*, GMM), koji se također naziva **mješavina Gaussova** (engl. *mixture of Gaussians*, MoG). Taj model tipično treniramo optimizacijskim algoritmom koji se naziva **algoritam maksimizacije očekivanja** (engl. *expectation maximization algorithm*,

[2]

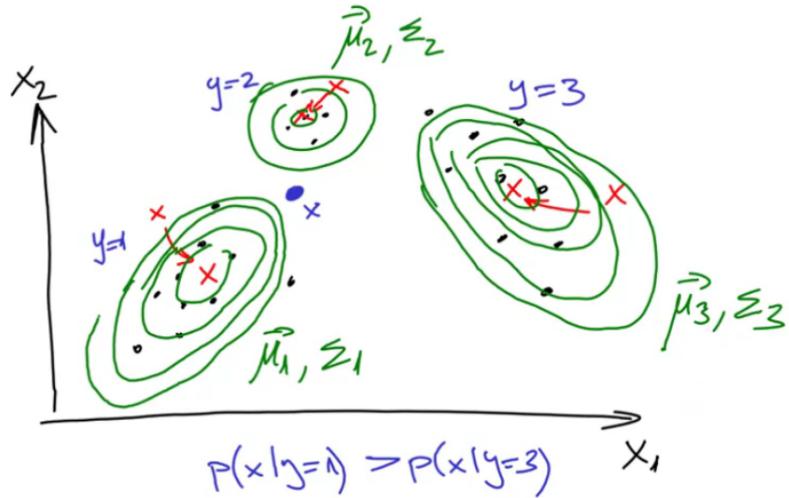
EM-algorithm), koji ćemo isto pogledati. Jednostavnosti radi, u nastavku ćemo umjesto o “modelu GMM učenom EM-algoritmom” jednostavno govoriti o “algoritmu GMM”.

Algoritam GMM može se shvatiti kao poopćenje algoritma K-sredina. Razmotrimo primjer.

► PRIMJER

Grupiramo dvodimenzionalne primjere u $K = 3$ grupe. Za razliku od algoritma K-sredina, kod algoritma GMM osim triju centroida želimo za svaku grupu procijeniti i njezinu **kovarijacijsku matricu**. Drugim riječima, svaka od triju grupa će biti jedna Gaussova gustoća vjerojatnosti $p(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$. Ta Gaussova gustoća vjerojatnosti definira vjerojatnost da primjer \mathbf{x} pripada dotičnoj grupi (preciznije, ta gustoća vjerojatnosti definira distribuciju vjerojatnosti primjera iz dotične grupe).

Dakle, ako imamo tri grupe, morat ćemo procijeniti parametre $\boldsymbol{\mu}_1, \Sigma_1, \boldsymbol{\mu}_2, \Sigma_2, \boldsymbol{\mu}_3$ i Σ_3 . Radimo iterativno, kao i kod algoritma K-sredina: kreuvši od tri slučajno odabrana centroida (premda i ovdje možemo na pametniji način inicijalizirati početna središta), polako ćemo ugađati centroide i kovarijacijske matrice svake od triju grupa, dok ne dođemo do stacionarnog stanja, tj. dok postupak ne konvergira. Početne kovarijacijske matrice možemo postaviti na jedinične matrice. Kod algoritma K-sredina primjere smo stvarstavali u grupu čijem su centroidu najbliži, i zatim smo ponovno izračunavali centroide. Slično radimo kod algoritma GMM: za svaki primjer izračunavamo koja je vjerojatnost da primjer pripada dotičnoj grupi, i zatim ponovno izračunavamo centroide kao težinski prosjek primjera (u ovisnosti o vjerojatnostima da primjer pripada grupi). To ponavljamo do konvergencije. Na kraju dobivamo mješavinu od tri Gaussove gustoće vjerojatnosti. Na primjer, ako je ulazni prostor dvodimenzionalni (imamo samo značajke x_1 i x_2), grupiranje bi moglo izgledati ovako:



Crvenom je označeno kretanje centroida kroz iteracije algoritma od početnog položaja do krajnjeg položaja centroida. Prema izokonturama ovih gustoća vidimo da postoji pozitivna korelacija između značajki x_1 i x_2 u grupi $y = 1$ (elipse izokontura su pozitivno nagnute) te negativna korelacija između značajki u grupi $y = 3$ (elipse izokontura su negativno nagnute). U grupi $y = 2$ nemamo korelacije između značajki (izokonture su kružnice). Za svaku od triju grupa y možemo zatim izračunati koja je vjerojatnost da je dotična grupa generirala primjer \mathbf{x} , tj. možemo izračunati **izglednost grupe**. U ovom slučaju, vrijedilo bi $p(\mathbf{x}|y = 1) > p(\mathbf{x}|y = 3)$ jer je primjer \mathbf{x} bliži grupi $y = 1$ nego grupi $y = 3$, tj. “više je u gustoći vjerojatnosti” prve grupe nego treće grupe. Također ćemo iz ovoga, primjenom Bayesovog pravila, lako moći izračunati i obrnuto (a to je ono što nas zapravo zanima): za svaku grupu možemo izračunati kolika je **vjerojatnost grupe** za dani primjer \mathbf{x} , tj. vjerojatnost $p(y|\mathbf{x})$.

Drugi pogled na probabilističko grupiranje modelom Gaussove mješavine jest taj da je naš zadatak zapravo procijeniti gustoću vjerojatnosti $p(\mathbf{x})$. Za tu gustoću vjerojatnosti pretpostavljamo da se interno sastoji od K komponenti – K Gaussovih gustoća vjerojatnosti. Nakon

što procijenimo parametre tako definirane gustoće vjerojatnosti $p(\mathbf{x})$, dobit ćemo zapravo meko grupiranje, jer ćemo za svaki primjer moći izračunati s kojom vjerojatnošću pripada kojoj od tih K komponenti.

Pogledajmo odmah kako izgleda konačan algoritam, a onda ćemo ga izvesti. Najprije, prisjetimo se kako izgleda algoritam K-sredina:

► Algoritam K-sredina

- 1: **inicijaliziraj** centroide $\boldsymbol{\mu}_k$, $k = 1, \dots, K$
- 2: **ponavlja**
- 3: za svaki $\mathbf{x}^{(i)} \in \mathcal{D}$
- 4: $b_k^{(i)} \leftarrow \begin{cases} 1 & \text{ako } k = \operatorname{argmin}_j \|\mathbf{x}^{(i)} - \boldsymbol{\mu}_j\| \\ 0 & \text{inače} \end{cases}$
- 5: za svaki $\boldsymbol{\mu}_k$, $k = 1, \dots, K$
- 6: $\boldsymbol{\mu}_k \leftarrow \sum_{i=1}^N b_k^{(i)} \mathbf{x}^{(i)} / \sum_{i=1}^N b_k^{(i)}$
- 7: **dok** $\boldsymbol{\mu}_k$ ne konvergiraju

Da je ovdje riječ o algoritmu tvrdog grupiranja vidi se po tome što je pripadanje primjera grupi modelirano binarnom indikatorskom varijablu $b_k^{(i)}$, tj. $b_k^{(i)} \in \{0, 1\}$. Drugim riječima, primjer $\mathbf{x}^{(i)}$ ili pripada ili ne pripada grupi k . Prisjetimo se: algoritam K-sredina u svakoj iteraciji obavlja dva koraka: u prvom koraku primjere dodijeljuje grupama s najbližim centroidima, a u drugom koraku izračunava nove centroide grupe.

Očekivano, ključna promjena kod algoritma GMM bit će ta da primjer grupi pripada s nekom vjerojatnošću. Tu ćemo vjerojatnost označiti sa $h_k^{(i)} \in [0, 1]$. Vjerojatnost $h_k^{(i)}$ naziva se **odgovornost** – vjerojatnost da primjer $\mathbf{x}^{(i)}$ pripada grupi k . Evo kako izgleda algoritam GMM:

► Algoritam GMM (model GMM + EM-algoritam)

- inicijaliziraj** parametre $\boldsymbol{\theta} = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$
ponavlja do konvergencije log-izglednosti ili parametara

E-korak:

Za svaki primjer $\mathbf{x}^{(i)} \in \mathcal{D}$ i svaku komponentu $k = 1, \dots, K$:

$$h_k^{(i)} \leftarrow \frac{p(\mathbf{x}^{(i)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \pi_k}{\sum_{j=1}^K p(\mathbf{x}^{(i)} | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \pi_j}$$

M-korak:

Za svaku komponentu $k = 1, \dots, K$:

$$\boldsymbol{\mu}_k \leftarrow \frac{\sum_i h_k^{(i)} \mathbf{x}^{(i)}}{\sum_i h_k^{(i)}}, \quad \boldsymbol{\Sigma}_k \leftarrow \frac{\sum_i h_k^{(i)} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_k)(\mathbf{x}^{(i)} - \boldsymbol{\mu}_k)^T}{\sum_i h_k^{(i)}}, \quad \pi_k \leftarrow \frac{1}{N} \sum_{i=1}^N h_k^{(i)}$$

Izračunaj trenutačnu vrijednost log-izglednosti: $\ln \mathcal{L}(\boldsymbol{\theta} | \mathcal{D}) = \sum_{i=1}^N \ln \sum_{k=1}^K \pi_k p(\mathbf{x}^{(i)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$

Primijetite najprije da algoritam K-sredina i algoritam GMM imaju istu strukturu: oba su algoritma iterativna i oba algoritma u svakoj iteraciji provode dva koraka. U prvoj koraku primjeri se dodijeljuju grupama, a u drugome koraku ažuriraju se parametri svake grupe. No, razlika je u tome što kod algoritma K-sredina primjeri tvrdo pripadaju grupama (binarna varijabla $b_k^{(i)}$), dok kod algoritma GMM primjeri grupama pripadaju “mekano”, s nekom vjerojatnošću (odgovornost $h_k^{(i)}$). Konkretno, u prvom koraku algoritma (tzv. E-korak), primjeri se dodijeljuju grupama na “mekan” način, tako da se izračunava odgovornost $h_k^{(i)}$, tj. za svaki se primjer $\mathbf{x}^{(i)}$ izračunava vjerojatnost da primjer pripada grupi k . U drugom koraku (tzv. M-korak) računaju

se novi parametri grupa, na temelju trenutačne (meke) dodijele primjera grupama. Najbolje se to može uočiti kod izračuna centroida grupe μ_k , gdje vidimo da se centroid za svaku grupu k izračunava tako da se vektori primjera zbrajaju težinski, u ovisnosti o tome kolika je vjerojatnost $h_k^{(i)}$ da primjer $\mathbf{x}^{(i)}$ pripada dotičnoj grupi k . Suprotno tomu, algoritam K-sredina jednostavno računa centroid grupu tako da uprosječi sve vektore primjera koji toj grupi pripadaju, bez ikakvog množenja primjera težinama.

U nastavku ćemo, korak po korak, izvesti algoritam GMM. Premda je osnovna ideja algoritma jednostavna, i zapravo već smo ju opisali, izvod algoritma iziskuje nešto matematičke artiljerije, pa budite strpljivi. Idemo redom. Pogledajmo najprije model.

1.1 Model miješane gustoće

Model Gaussove miješavine (GMM) zapravo je samo jedan specifičan slučaj **modela miješane gustoće** (engl. *mixture model*). Model miješane gustoće generativni je model, koji modelira gustoću $p(\mathbf{x})$ kao linearu kombinaciju K komponenti. Krećemo od pretpostavke da postoji zajednička vjerojatnost $p(\mathbf{x}, y)$, gdje je y oznaka grupe kojoj primjer pripada, $y = \{1, \dots, K\}$. Dakle, mi zapravo pretpostavljamo da primjeri \mathbf{x} pripadaju grupama y , pa načelno možemo govoriti o zajedničkoj vjerojatnosti $p(\mathbf{x}, y)$. Međutim, budući da mi ovdje radimo nenadzirano učenje, oznaka y nam nije dostupna. To znači da nam vjerojatnost $p(\mathbf{x}, y)$ nije poznata, već nam je poznata samo vjerojatnost $p(\mathbf{x})$. Formalno, ovu drugu možemo dobiti iz ove prve tako da zajedničku vjerojatnost marginaliziramo upravo po varijabli y , koristeći pravilo zbroja vjerojatnosti:

$$p(\mathbf{x}) = \sum_{k=1}^K p(\mathbf{x}, y=k)$$

Primijenimo li još pravilo umnoška vjerojatnosti, onda model miješane gustoće onda možemo napisati kao linearu kombinaciju od K **komponenti**:

$$p(\mathbf{x}) = \sum_{k=1}^K p(\mathbf{x}, y=k) = \sum_{k=1}^K \underbrace{P(y=k)}_{\pi_k} p(\mathbf{x}|y=k) = \sum_{k=1}^K \pi_k p(\mathbf{x}|\boldsymbol{\theta}_k)$$

gdje smo za apriornu vjerojatnost grupe, $p(y=k)$, uveli oznaku π_k , koju nazivamo **koeficijent mješavine**. Očito, $\sum_k \pi_k = 1$, jer su to zapravo vjerojatnosti kategoričke slučajne varijable. Vjerojatnosti $p(\mathbf{x}|\boldsymbol{\theta}_k)$, koje su zapravo izglednosti grupe, nazivamo **gustoće komponenti**.

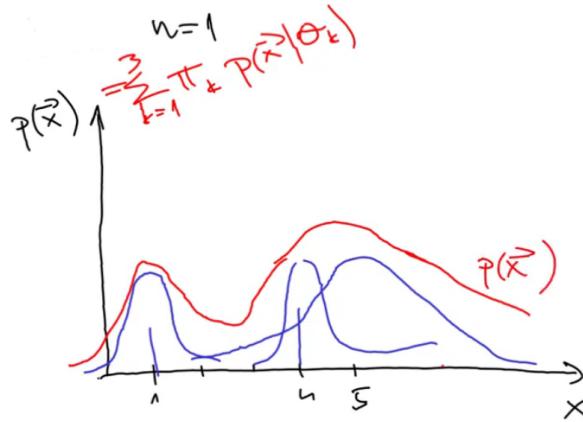
Ovo što smo napisali je općenit model miješane gustoće. Konkretno, kod GMM, gustoće komponenti su Gaussove gustoće vjerojatnosti sa srednjim vektorom $\boldsymbol{\mu}_k$ i kovarijacijskom matricom Σ_k , pa dakle imamo:

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \Sigma_k)$$

Pogledajmo primjer.

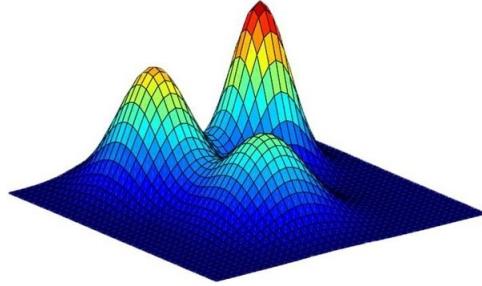
► PRIMJER

Razmotrimo kako bi izgledao Gaussov model miješane gustoće $p(x)$ za jednodimenzijski ulazni prostor i za $K = 3$ grupe. Neka $\mu_1 = 1, \mu_2 = 4, \mu_3 = 5, \sigma_1 = \sigma_2 = 1, \sigma_3 = 6, \pi_1 = \pi_2 = 0.2$ i $\pi_3 = 0.6$. Gustoća vjerojatnosti $p(\mathbf{x})$ onda izgleda (vrlo otprilike) ovako:



Gustoća vjerojatnosti $p(x)$ (ovdje označena crvenom) dobiva se zbrajanjem svih triju gustoća komponenti, pomnoženih s njima pridruženim koeficijentom mješavine.

U ovom jednostavnom primjeru zadržali smo se na jednodimenzijskome ulaznom prostoru, pa smo za gustoće komponenti koristili univarijatne Gaussove gustoće vjerojatnosti. No, naravno, u stvarnosti će primjeri \mathbf{x} imati više od jedne značajke, tj. ulazni će prostor biti višedimenzijski, i tada ćemo za gustoće komponenti koristiti multivarijatnu Gaussovou gustoću vjerojatnosti. Na primjer, za dvodimenzijski ulazni prostor koristili bismo **bivarijatnu Gaussovu gustoću vjerojatnosti**:



Ovdje imamo tri bivarijatne gustoće vjerojatnosti, dakle modeliramo $K = 3$ grupe.

GMM je **generativni model**, budući da marginalnu gustoću vjerojatnosti $p(\mathbf{x})$ definira preko zajedničke gustoće vjerojatnosti $p(\mathbf{x}, y)$. Kao i svaki generativni model, tako i GMM ima i svoju **generativnu priču**: skup podata je generiran tako da je prvo slučajno (po kategoričkoj distribuciji $P(y)$) odabrana jedna komponenta k , a onda je po njoj odgovarajućoj gustoći vjerojatnosti $p(\mathbf{x}|\boldsymbol{\theta}_k)$ uzorkovan primjer \mathbf{x} .

Međutim, u praksi, budući da mi želimo grupirati podatke, ono što nas zapravo zanima nije toliko sama gustoća $p(\mathbf{x})$, niti uzorkovanje primjera iz te gustoće, koliko **vjerojatnost da primjer pripada grupi** k , tj. vjerojatnost $P(y = k|\mathbf{x})$, za koju smo već rekli da se naziva **odgovornost**. Ako znamo tu vjerojatnost za svaki primjer, onda zapravo imamo meko grupiranje. Srećom, budući da se ovdje radi o generativnom modelu, tu vjerojatnost možemo lako izraziti iz vjerojatnosti koje smo već definirali, i to pomoću Bayesovog pravila. Evo kako:

$$\begin{aligned} P(y = k|\mathbf{x}^{(i)}) &= \frac{P(y = k)p(\mathbf{x}^{(i)}|y = k)}{p(\mathbf{x}^{(i)})} = \frac{P(y = k)p(\mathbf{x}^{(i)}|y = k)}{\sum_j P(y = j)p(\mathbf{x}^{(i)}|y = j)} \\ &= \frac{\pi_k p(\mathbf{x}^{(i)}|\boldsymbol{\theta}_k)}{\sum_j \pi_j p(\mathbf{x}^{(i)}|\boldsymbol{\theta}_j)} = h_k^{(i)} \end{aligned}$$

Vjerojatnost $P(y = k|\mathbf{x}^{(i)})$ je vjerojatnost da je grupa k odgovorna za nastanak (generiranje) primjera \mathbf{x} . I to je upravo ono što nas kod mekog grupiranja zanima!

Ovime smo definirali naš model. Sastoji se, dakle, od koeficijenata mješavine i od gustoće komponenata, kojih imamo onoliko koliko imamo grupe. Iz toga onda možemo izračunati odgovornosti grupe za svaki primjer. Pogledajmo sada koji su **parametri** modela koje moramo procijeniti. Parametri su:

$$\boldsymbol{\theta} = \{\pi_k, \underbrace{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k}_{\boldsymbol{\theta}_k}\}_{k=1}^K$$

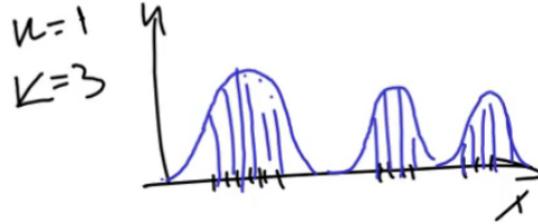
gdje su π_k koeficijenti mješavine a $\boldsymbol{\theta}_k$ su parametri gustoće komponenti (srednje vrijednosti i kovarijacijske matrice). Koliko je to ukupno parametara? Ukupno ih je $K \cdot \frac{n}{2}(n+1) + nK + K - 1$.

1.2 Log-izglednost modela miješane gustoće

Kako bismo odredili parametre GMM-a, odnosno kako bismo naučili model na temelju podataka? Prisjetimo se: raspolaćemo skupom **neoznačenih primjera** $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$. Kao i kod svih generativnih modela, učenje modela svodi se na statističku procjenu parametara iz podataka. Najlakše bi bilo da parametre probamo procijeniti sa MLE. Kada koristimo MLE za procjenu parametara, mi zapravo namještamo K komponenti tako da maksimiziramo izglednost parametara, odnosno, ekvivalentno, maksimiziramo vjerojatnost podataka pod modelom. To ilustrira sljedeći primjer.

► PRIMJER

Razmotrimo slučaj s jednodimenzijskim ulaznim prostorom i $K = 3$. MLE zapravo namješta tri Gaussove gustoće (određuje parametre srednje vrijednosti i kovarijacijske matrice) tako da primjeri "pokupe" što više gustoće vjerojatnosti:



Naime, uz tako namještene parametre izglednost parametara (tj. vjerojatnost primjera) biti će za zadani skup primjera najveća moguća.

Sada kada smo se toga prisjetili, primijenimo uobičajeni recept za izvođenje procjenitelja parametara probabilističkih modela, isti onu koju smo već bili primijenili za linearnu regresiju i logističku regresiju. Napišimo najprije funkciju log-izglednosti parametara za dani (u ovom slučaju neoznačen) skup primjera \mathcal{D} :

$$\begin{aligned} \ln \mathcal{L}(\boldsymbol{\theta} | \mathcal{D}) &= \ln \prod_{i=1}^N p(\mathbf{x}^{(i)}) \\ &= \ln \prod_{i=1}^N \sum_{k=1}^K \pi_k p(\mathbf{x}^{(i)} | \boldsymbol{\theta}_k) = \sum_{i=1}^N \ln \sum_{k=1}^K \pi_k p(\mathbf{x}^{(i)} | \boldsymbol{\theta}_k) \end{aligned}$$

Budući da tražimo parametre koji maksimiziraju izglednost, mogli bismo sada pokušati derivirati ovaj izraz po parametrima $\boldsymbol{\theta}$, odnosno izračunati $\nabla_{\boldsymbol{\theta}} \ln \mathcal{L}(\boldsymbol{\theta} | \mathcal{D}) = 0$, i na taj način pronaći maksimizator funkcije u zatvorenoj formi. Nažalost, to neće funkcioniрати. Maksimizacija ovog

izraza nema rješenje u zatvorenoj formi. Konkretno, problem je u logaritmu koji je “zapeo” između dviju sumacija, pa ne možemo derivirati zasebno po svakoj od komponenata, odnosno izglednost se **ne faktorizira po komponentama**.

Ako ne postoji rješenje u zatvorenoj formi, što nam je alternativa? Alternativa je iterativna optimizacija. Na primjer, **gradijentni uspon** (kao kod logističke regresije, gdje optimizacija također nije imala rješenje u zatvorenoj formi, premda doduše iz drugog razloga – nelinearnosti sigmoidne funkcije), **metode MCMC** (spomenuli smo ih u predavanju o probabilističkim grafičkim modelima, npr. Gibbsovo uzorkovanje) ili **algoritam maksimizacije očekivanja**. Mi ćemo u nastavku pogledati ovaj potonji. [3]

2 Algoritam maksimizacije očekivanja

Osnovna ideja algoritma maksimizacije očekivanja (engl. *expectation maximization algorithm*, **EM-algoritam**) jest da se generativni model proširi **skrivenim (latentnim) varijablama**. Nakon takvog proširenja, maksimizaciju log-izglednosti moći ćemo provesti iterativnim postupkom. Pogledajmo najprije što to znači da model proširujemo latentnim varijablama, a onda ćemo pokazati kako upotrijebiti EM-algoritam da bismo naučili takav model. [4]

2.1 Model miješane gustoće s latentnim varijablama

Prisjetite se da smo o skrivenim (latentnim) varijablama pričali u predavanju o probabilističkim grafički modelima. Rekli smo da su skrivene varijable one varijable čije vrijednosti ne opažamo u podatcima, ali u model ih uvodimo jer to može značajno pojednostaviti modeliranje, npr. onda kada skrivene varijable modeliraju neke zajedničke uzroke više opaženih varijabli. Ako skrivene varijable modeliraju neki apstraktni koncept, onda obično umjesto naziva skrivene varijable koristimo naziv **latentne varijable**. Tako će to biti u našem slučaju, gdje ćemo latentne varijable uvesti kako bismo modelirali vezu između primjera i grupe: latentne varijable će definirati koji primjer pripada kojoj grupi. Preciznije, svaki primjer $\mathbf{x}^{(i)}$ imat će pridruženu svoju latentnu varijablu $\mathbf{z}^{(i)}$ koja opisuje kojoj grupi taj primjer pripada. Ta varijabla je zapravo kategorička varijabla, i ima onoliko vrijednosti koliko ima grupe, tj. ima K vrijednosti. Kao što smo radili i do sada, tu kategoričku varijablu predstaviti ćemo kao K -dimenzijski vektor binarnih indikatorskih varijabli:

$$\mathbf{z}^{(i)} = (z_1^{(i)}, \dots, z_k^{(i)}, \dots, z_K^{(i)})$$

pri čemu koristimo kodiranje s jednom jedinicom (engl. *one-hot encoding*): ako primjer $\mathbf{x}^{(i)}$ pripada grupi k , sve komponente varijable $\mathbf{z}^{(i)}$ bit će na nuli, osim jedne, $z_k^{(i)}$, koja će biti postavljena na jedinicu.

Jednostavnosti radi, u nastavku ćemo umjesto $\mathbf{z}^{(i)}$ pisati \mathbf{z} , no imajte na umu da imamo po jednu varijablu \mathbf{z} za svaki primjer $\mathbf{x}^{(i)}$. Ideja je sada da te latentne varijable \mathbf{z} ugradimo u model GMM, tj. da ga proširimo s latentnim varijablama. Prethodno je model bio definiran tako da modelira gustoću vjerojatnosti $p(\mathbf{x})$, a model proširen s latentnim varijablama treba definirati zajedničku gustoću vjerojatnosti $p(\mathbf{x}, \mathbf{z})$. Pokušajmo izraziti tu zajedničku gustoću vjerojatnosti.

Krenimo od toga da uočimo da je vjerojatnost $P(\mathbf{z} = k)$ zapravo jednaka apriornoj vjerojatnosti grupe k , tj. vjerojatnosti da primjer pripada grupi k neovisno o tome o kojem se primjeru radi. Apriornu vjerojatnost grupe k već smo bili označili kao π_k . Budući da je \mathbf{z} kategorička varijabla prikazana kao binarni vektor, to onda znači da vjerojatnost da $\mathbf{z} = k$ možemo napisati kao:

$$P(\mathbf{z}) = \prod_{k=1}^K \pi_k^{z_k}$$

Ovakav zapis vjerojatnosti za kategoričke varijable smo već bili koristili. Na ovaj način zapravo dohvaćamo π_k za onaj k za koji $z_k = 1$. Na isti način možemo definirati izglednost grupe (vjerojatnost da grupa \mathbf{z} generira primjer \mathbf{x}):

$$p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \prod_{k=1}^K p(\mathbf{x}|\boldsymbol{\theta}_k)^{z_k}$$

Važno je da ovdje ne zaboravimo da mi naravno ne znamo kojoj grupi primjer pripada, i zato je varijabla \mathbf{z} skrivena (latentna). Međutim, mi pretpostavljamo da u stvarnosti svaki primjer pripada jednoj grupi, samo što mi ne znamo kojoj. Pimijetite, dakle, da mi pretpostavljamo da svaki primjer pripada *jednoj* grupi. To sad možda izgleda čudno, jer smo ranije rekli da radimo meko grupiranje i da je $h_k^{(i)}$ vjerojatnost, što kao da sugerira da pojedini primjer može pripadati u više grupa istovremeno, tj. svim onim grupama za koje $h_k^{(i)} > 0$. No, to nije tako. Naime, ovdje trebamo razlikovati dvije stvari, koje smo već bili spomenuli. Mi u modelu GMM pretpostavljamo da svaki primjer izvorno pripada jednoj i samo jednoj grupi (tj. da ga je generirala jedna komponenta). Međutim, mi ne znamo kojoj grupi primjer pripada, i to izražavamo pomoću vjerojatnosti $h_k^{(i)}$. Dakle, vjerojatnost pripadanja primjera grupi k nije posljedica toga što primjer istovremeno pripada u više grupa, nego toga što mi ne znamo kojoj točno grupi primjer pripada. Drugim riječima, vjerojatnost modelira naše neznanje, a ne razdijeljenu pripadnost primjera u više grupa. To je bitna razlika između mješavinskih modela i nekih drugih modela mekog grupiranja (npr., onih temeljenih na neizrazitoj logici).

Sada kada smo uveli latentne varijable koje opisuju vezu primjera i grupa, možemo napisati **zajedničku gustoću** $p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})$, koja nam govori koja je vjerojatnost generiranja primjera \mathbf{x} takvog da on bude generiran iz grupe kodirane sa \mathbf{z} . Jednostavno ćemo primijeniti pravilo umnoška i zatim uvrstiti gore definirane vjerojatnosti:

$$p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = P(\mathbf{z})p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \prod_{k=1}^K \pi_k^{z_k} \prod_{k=1}^K p(\mathbf{x}|\boldsymbol{\theta}_k)^{z_k} = \prod_{k=1}^K \pi_k^{z_k} p(\mathbf{x}|\boldsymbol{\theta}_k)^{z_k}$$

Dakle, ovdje smo najprije primijenili pravilo umnoška, a onda smo dva umnoška niza kombinirali u jedan umnožak niza (što možemo zbog komutativnosti množenja).

Time smo izveli **model miješane gustoće s latentnim varijablama**. Usporedimo sada izvorni model miješane gustoće (bez latentnih varijabli) i ovaj upravo izvedeni model (s latentnim varijablama). Model miješane gustoće bez latentnih varijabli je:

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k p(\mathbf{x}|\boldsymbol{\theta}_k)$$

dok je model miješane gustoće s latentnim varijablama:

$$p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = \prod_{k=1}^K \pi_k^{z_k} p(\mathbf{x}|\boldsymbol{\theta}_k)^{z_k}$$

Ako ćemo biti iskreni, moramo priznati da nismo napravili neku strašno pametnu stvar: model miješane gustoće s latentnim varijablama zapravo funkcioniра tako da, *ako znamo kojoj grupi primjer pripada* (to je grupa k za koju $z_k = 1$), onda model “dohvaća” i množi dotične π_k i $p(\mathbf{x}|\boldsymbol{\theta}_k)$, što nam daje zajedničku vjerojatnost primjera i grupe za taj primjer. Problem je, naravno, što ne znamo kojoj grupi primjer pripada. Sad se možda čini da nismo ništa napravili. No, ipak jesmo. Bitna razlika između ova dva modela jest što kod modela s latentnim varijablama imamo modeliranu vezu između primjera i grupe, i to je parametar koji možemo procijeniti u podatcima, budući da je to upravo informacija koja nas kod grupiranja zapravo

zanima. Kao posljedica toga, druga bitna razlika jest da kod latentnog modela umjesto zbroja vjerojatnosti po komponentama imamo umnožak vjerojatnosti po komponentama. Da je ta razlika ključna postaje jasno nakon što napišemo log-izglednost modela miješane gustoće s latentnim varijablama:

$$\begin{aligned}\ln \mathcal{L}(\boldsymbol{\theta} | \mathcal{D}, \mathbf{Z}) &= \ln p(\mathbf{X}, \mathbf{Z} | \boldsymbol{\theta}) = \ln \prod_{i=1}^N p(\mathbf{x}^{(i)}, \mathbf{z}^{(i)} | \boldsymbol{\theta}) \\ &= \ln \prod_{i=1}^N \prod_{k=1}^K \pi_k^{z_k^{(i)}} p(\mathbf{x} | \boldsymbol{\theta}_k)^{z_k^{(i)}} \\ &= \sum_{i=1}^N \sum_{k=1}^K z_k^{(i)} \left(\ln \pi_k + \ln p(\mathbf{x}^{(i)} | \boldsymbol{\theta}_k) \right)\end{aligned}$$

gdje \mathbf{Z} označava skup svih varijabli $\mathbf{z}^{(i)}$. Ovako definiranu log-izglednost nazivamo **potpuna log-izglednost** (engl. *complete log-likelihood*), dok log-izglednost prvog modela, onog bez latentnih varijabli, nazivamo **nepotpuna log-izglednost** (engl. *incomplete log-likelihood*). Prisjetite se da smo o tome smo već bili pričali u predavanju o probabilističkim grafičkim modelima, kada smo spomenuli učenje iz potpunih i nepotpunih podataka. Potpuni podatci su oni kod kojih opažamo sve što nam treba za učenje modela, dok su nepotpuni podatci oni kod kojih neke varijable ne opažamo (skrivene odnosno latentne varijable). Potpuna log-izglednost definirana je kao da su podatci iz kojih učimo potpuno, to jest kao da oznake svih primjera \mathbf{Z} možemo opažati.

Usporedimo sada nepotpunu log-izglednost i potpunu log-izglednost. Vidimo da se, za razliku od nepotpune log-izglednosti, potpuna log-izglednost lijepo faktorizira po komponentama. To znači da ćemo pri maksimizaciji moći derivirati log-izglednost svake komponente zasebno po njezinim parametrima, a to onda znači da ćemo imati rješenje u zatvorenoj formi. To je dobra vijest. Loša vijest jest da mi i dalje ne znamo koji primjeri pripadaju kojoj grupi, dakle ne znamo vrijednosti varijabli $\mathbf{z}^{(i)}$. Sve druge vrijednosti su nam poznate, ali te nisu. To onda znači da ipak ne možemo analitički provesti maksimizaciju log-izglednosti (nalaženje nultočke).

2.2 Algoritam maksimizacije očekivanja (napokon!)

I tu napokon dolazimo do **algoritma maksimizacije očekivanja**. Ideja je ova: budući da ne znamo vrijednosti za $\mathbf{z}^{(i)}$, ono što možemo napraviti jest izračunati **očekivanje** potpune log-izglednosti uz neke fiksirane vrijednosti za parametre π_k (koeficijenti mješavine) i $\boldsymbol{\theta}_k$ (parametri gustoće komponenti), a nakon toga ažurirati parametre π_k i $\boldsymbol{\theta}_k$ tako da maksimiziraju to očekivanje. I ta dva koraka onda možemo iterativno ponavljati: svaki puta izračunati očekivanje potpune log-izglednosti po latentnim varijablama, a onda odabratи parametre koji maksimiziraju to očekivanje. U ovom drugom koraku sve varijable će već biti fiksirane, pa će postojati rješenje u zatvorenoj formi.

Može se pokazati – a u to sada nećemo ulaziti – da maksimizacija očekivanja potpune log-izglednosti ujedno dovodi do povećanja nepotpune log-izglednosti, što je zapravo početni problem koji smo željeli riješiti. Treba, međutim, napomenuti da ovaj postupak ne mora nužno dovesti do globalnog maksimuma nepotpune log-izglednosti. To će, kao i kod algoritma K-sredina, ovisiti o odabiru početnih vrijednosti parametara.

Algoritam, dakle, ima **dva koraka**: izračun očekivanja potpune log-izglednosti i zatim njegovu maksimizaciju. Prvi korak (izračun očekivanja potpune log-izglednosti) naziva se **E-korak** (od “expectation”). Označimo sa $Q(\boldsymbol{\theta} | \boldsymbol{\theta}^{(t)})$ očekivanje potpune log-izglednosti uz fiksirane pa-

parametre $\boldsymbol{\theta}^{(t)}$ u iteraciji t . To očekivanje je jednako:

$$\begin{aligned}\mathcal{Q}(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) &= \mathbb{E}_{\mathbf{Z}|\mathcal{D},\boldsymbol{\theta}^{(t)}} \left[\ln \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}, \mathbf{Z}) \right] \\ &= \mathbb{E}_{\mathbf{Z}|\mathcal{D},\boldsymbol{\theta}^{(t)}} \left[\sum_{i=1}^N \sum_{k=1}^K z_k^{(i)} (\ln \pi_k + \ln p(\mathbf{x}^{(i)}|\boldsymbol{\theta}_k)) \right] \\ &= \sum_{i=1}^N \sum_{k=1}^K \mathbb{E}[z_k^{(i)}|\mathcal{D}, \boldsymbol{\theta}^{(t)}] (\ln \pi_k + \ln p(\mathbf{x}^{(i)}|\boldsymbol{\theta}_k))\end{aligned}$$

U ovom izrazu, jedino je latentna varijabla $z_k^{(i)}$ slučajna varijabla, dok su sve ostale vrijednosti fiksne. Dakle, očekivanje zapravo računamo po varijabli $z_k^{(i)}$ (pomnoženoj s nekom konstantom). Varijabla $z_k^{(i)}$ je binarna indikatorska varijabla, dakle Bernoullijeva varijabla, pa je njezino očekivanje zapravo jednak vjerojatnosti da ona bude jednaka 1. A to je upravo jednak odgovornosti $h_k^{(i)}$! Formalno:

$$\mathbb{E}[z_k^{(i)}|\mathcal{D}, \boldsymbol{\theta}^{(t)}] = \mathbb{E}[z_k^{(i)}|\mathbf{x}^{(i)}, \boldsymbol{\theta}^{(t)}] = P(z_k^{(i)} = 1|\mathbf{x}^{(i)}, \boldsymbol{\theta}^{(t)}) = \frac{p(\mathbf{x}^{(i)}|\boldsymbol{\theta}_k^{(t)})\pi_k^t}{\sum_{j=1}^K p(\mathbf{x}^{(i)}|\boldsymbol{\theta}_j^{(t)})\pi_j^t} = h_k^{(i)}$$

Dakle, očekivanje potpune log-izglednosti uz fiksirane parametre $\boldsymbol{\theta}^{(t)}$ u iteraciji t jednak je (samo uvrstimo $h_k^{(i)}$ umjesto $z_k^{(i)}$):

$$\begin{aligned}\mathcal{Q}(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) &= \sum_{i=1}^N \sum_{k=1}^K h_k^{(i)} (\ln \pi_k + \ln p(\mathbf{x}^{(i)}|\boldsymbol{\theta}_k)) \\ &= \sum_{i=1}^N \sum_{k=1}^K h_k^{(i)} \ln \pi_k + \sum_{i=1}^N \sum_{k=1}^K h_k^{(i)} \ln p(\mathbf{x}^{(i)}|\boldsymbol{\theta}_k)\end{aligned}$$

To je prvi korak algoritma maksimizacije očekivanja. U drugom koraku, koji nazivamo **M-korak** (od ‘‘maximization’’), želimo maksimizirati očekivanje potpune log-izglednosti po parametrima π_k i $\boldsymbol{\theta}_k$. Budući da u izrazu više nemamo latentnih varijabli, to sada možemo napraviti analitički, deriviranjem i nalaženjem nultočke:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) &= 0 \\ \nabla_{\pi_k} \left(\sum_{i=1}^N \sum_{k=1}^K h_k^{(i)} \ln \pi_k + \lambda \left(\sum_{\mathbf{k}} \pi_{\mathbf{k}} - 1 \right) \right) &= 0 \\ \nabla_{\boldsymbol{\theta}_k} \sum_{i=1}^N h_k^{(i)} \ln p(\mathbf{x}^{(i)}|\boldsymbol{\theta}_k) &= 0\end{aligned}$$

Vidimo da se maksimizacija može raditi odvojeno za svaki od dva pribrojnika. Ovdje ćemo sada preskočiti sam postupak maksimizacije. Za prvi pribrojnik radili bismo maksimizaciju metodom Lagrangeovih mnoštvenika, budući da postoji ograničenje da se koeficijenti mješavine, koji zapravo definiraju kategoričku varijablu, moraju zbrajati u jedinicu. Za drugi pribrojnik moramo najprije odabrati konkretnu gustoću vjerojatnosti (u našem slučaju to su Gaussove gustoće vjerojatnosti) pa tek onda provesti maksimizaciju. Krajni rezultat za komponente mješavina je:

$$\pi_k^{(t+1)} = \frac{1}{N} \sum_{i=1}^N h_k^{(i)}$$

a za parametre Gaussovi komponenata:

$$\boldsymbol{\mu}_k^{(t+1)} = \frac{\sum_i h_k^{(i)} \mathbf{x}^{(i)}}{\sum_i h_k^{(i)}}$$

$$\boldsymbol{\Sigma}_k^{(t+1)} = \frac{\sum_i h_k^{(i)} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_k^{(t+1)}) (\mathbf{x}^{(i)} - \boldsymbol{\mu}_k^{(t+1)})^T}{\sum_i h_k^{(i)}}$$

I to nas onda napokon dovodi do EM-algoritma za GMM, koji smo već bili pogledali:

► Algoritam GMM (model GMM + EM-algoritam)

inicijaliziraj parametre $\boldsymbol{\theta} = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$
ponavljaj do konvergencije log-izglednosti ili parametara

E-korak:

Za svaki primjer $\mathbf{x}^{(i)} \in \mathcal{D}$ i svaku komponentu $k = 1, \dots, K$:

$$h_k^{(i)} \leftarrow \frac{p(\mathbf{x}^{(i)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \pi_k}{\sum_{j=1}^K p(\mathbf{x}^{(i)} | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \pi_j}$$

M-korak:

Za svaku komponentu $k = 1, \dots, K$:

$$\boldsymbol{\mu}_k \leftarrow \frac{\sum_i h_k^{(i)} \mathbf{x}^{(i)}}{\sum_i h_k^{(i)}}, \quad \boldsymbol{\Sigma}_k \leftarrow \frac{\sum_i h_k^{(i)} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_k) (\mathbf{x}^{(i)} - \boldsymbol{\mu}_k)^T}{\sum_i h_k^{(i)}}, \quad \pi_k \leftarrow \frac{1}{N} \sum_{i=1}^N h_k^{(i)}$$

Izračunaj trenutačnu vrijednost log-izglednosti: $\ln \mathcal{L}(\boldsymbol{\theta} | \mathcal{D}) = \sum_{i=1}^N \ln \sum_{k=1}^K \pi_k p(\mathbf{x}^{(i)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$

Nepotpuna log-izglednost $\ln \mathcal{L}(\boldsymbol{\theta} | \mathcal{D})$ (isto kao i potpuna log-izglednost) rasti će kroz iteracije, sve do konvergencije. U tom trenutku zaustavljamo postupak grupiranja. Rezultat algoritma su naučeni parametri $\boldsymbol{\theta} = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$, tj. koeficijenti mješavine te srednji vektor i kovarijacijska matrica za svaku od K grupe. Iz toga lako, primjenom Bayesovog pravila, izračunamo odgovornosti $h_k^{(i)}$, koje opisuju vjerojatnost da primjer $\mathbf{x}^{(i)}$ pripada grupi (odnosno da ga je generirala grupa) k . I time smo zapravo ostvarili meko grupiranje primjera u K grupe.

Na ovom mjestu zgodno je napraviti usporedu GMM-a i **Gaussovog Bayesovog klasifikatora**, o kojem smo već bili pričali. Usporedba ima smisla, jer oba modela koriste Gaussov gustoću za modeliranje izglednosti te Bayesovo pravilo za izračun aposteriorne vjerojatnosti. Naravno, očita razlika između GMM i Gaussovog Bayesovog klasifikatora je u tome što je prvi nenađirani, a drugi nadzirani algoritam. Posljedično, kod ovog drugog imali smo označene primjere, i mogli smo u zatvorenoj formi izračunati MLE. Kod GMM-a nismo imali označene primjere i MLE nema rješenja u zatvorenoj formi, pa smo zato uveli latentne varijable i postupak optimizacije proveli smo iterativno. Ta se razlika onda svodi na to da kod Gaussovog Bayesovog klasifikatora kod procjene parametara Gaussove gustoće za svaku klasu znamo točno koji primjeri pripadaju toj klasi i samo njih koristimo kod procjene parametara, dok kod GMM-a to ne znamo, pa sve primjere moramo uzeti u obzir s nekom vjerojatnošću (što modeliramo odgovornošću $h_k^{(i)}$).

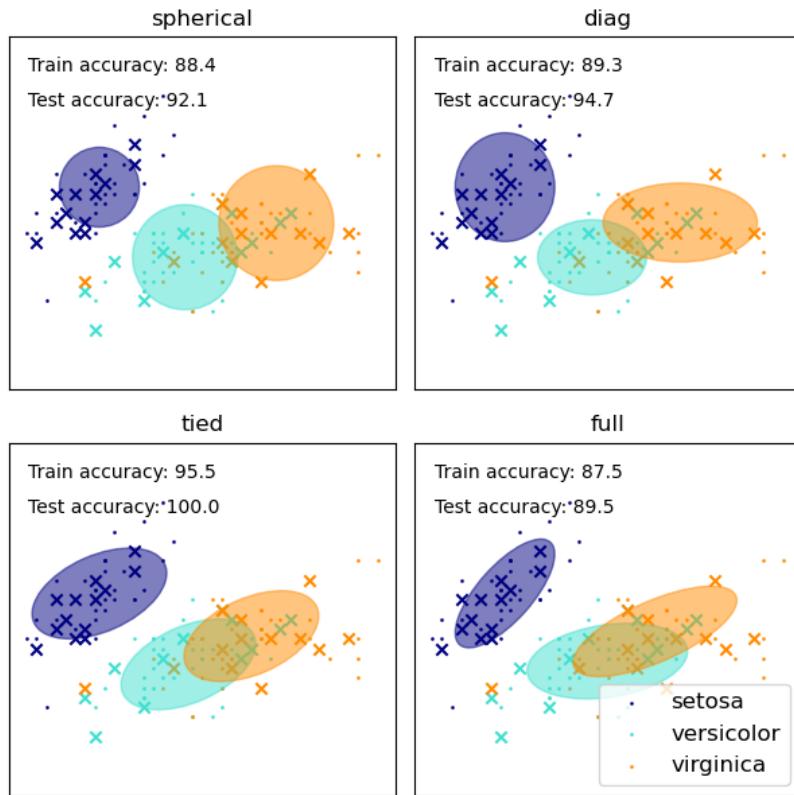
Osim ove razlike, postoji i jedna sličnost, a to je da oba modela koriste **kovarijacijsku matricu** kako bi modelirala korelacije između značajki. Isto kao i kod Gaussovog Bayesovog klasifikatora, uvođenjem prepostavki odnosno ograničenja nad kovarijacijskom matricom možemo definirati modele **manje složenosti** od modela s punom kovarijacijskom matricom. Pogledajmo primjer.

► PRIMJER

Razmotrimo grupiranje dvodimenzionalnih primjera iz poznatog skupa podataka Iris u $K = 3$ grupe. Koristimo algoritam GMM, pri čemu razmatramo četiri složenosti modela Gaussove mješavine. Te

[6]

su složenosti određene ograničenjima na kovarijacijske matrice Σ_k za svaku od K komponenti. Razmatramo punu, dijeljenu, dijagonalnu i izotropnu kovarijacijsku matricu. Rezultati grupiranja za ta četiri modela izgledaju ovako:



Na slici gore ljevo imamo grupiranje dobiveno modelom s izotropnom kovarijacijskom matricom, pa su izokonture Gaussove gustoće kružnice odnosno sferične su. Na slici gore desno imamo grupiranje dobiveno modelom s dijagonalnom kovarijacijskom matricom, pa su izokonture elipse poravnate s osima ulaznog prostora. Na slici dolje lijevo imamo dijeljenu (ali ne i dijagonalnu, budući da su elipse zakošene!) konvarijacijsku matricu, pa su sve tri gustoće jednakomjerno poravnate. Konačno, dolje desno imamo punu (i nedijeljenu) kovarijacijsku matricu, pa je svaka gustoća različita i epise nisu poravnate. Najsloženiji model je upravo ovaj posljednji, gdje imamo punu kovarijacijsku matricu, pa možemo modelirati (linearnu) zavisnost u šumu između značajki svih primjera iste grupe. Da je taj model najsloženiji vidi se i po tome što je točnost na skupu za učenje najveća. Međutim, taj model nije optimalan, jer točnost na ispitnome skupu nije maksimalna: maksimalnu točnost ostvaruje model s dijeljenim kovarijacijskim matricama. Gornja dva modela imaju znatno manju točnost i na skupu za učenje i na ispitnom skupu.

7

Uvođenjem različitih ograničenja na kovarijacijsku matricu možemo, dakle, upravljati složenošću modela. No, kako onda odrediti koji je model najbolji? Primijetite da ne možemo koristiti unakrsnu provjeru na ispitnom skupu, budući da se ovdje ne radi o modelu nadziranog učenja koji bismo mogli primijeniti za predikciju na nevidjenim primjerima. Problem odabira složenosti modela GMM zapravo je istovjetan problemu određivanja broja grupa, pa dakle možemo koristiti neke od metoda koje smo bili spomenuli prošli put, npr., AIC ili (tipičnije) BIC (engl. *Bayesian information criterion*). Alternativno, možemo označiti podskup primjera pa na tom podskupu provjeriti grupe, tako da izračunamo Randov indeks ili neku drugu mjeru točnosti grupiranja.

8

Krajnje pojednostavljenje modela jest da prepostavimo da sve komponente imaju **dijeljenu**

i izotropnu kovarijacijsku maticu, $\Sigma = \sigma^2 \mathbf{I}$. Ako pored toga odgovornosti $h_k^{(i)}$ zaokružimo na 0 ili 1, tj. napravimo tvrdo grupiranje, onda ćemo imati $h_k^{(i)} = b_k^{(i)}$ i algoritam GMM degenerira u algoritam K-sredina. Tada također vrijedi $\ln \mathcal{L}(\boldsymbol{\theta} | \mathcal{D}) \propto -J$, tj. log-izglednost je proporcionalna negativnoj pogrešci. Prisjetite se da smo sličnu vezu imali između log-izglednosti poopćenih linearnih modela i njihovih funkcija pogreški.

2.3 Napomene

Zaključimo naše izlaganje algoritma GMM s nekoliko praktičnih napomena:

- EM-algoritam nužno konvergira, ali može konverrirati u **lokalni optimum** funkcije log-izglednosti. To znači da će konačni rezultat vrlo ovisiti o inicijalizaciji parametara;
- Poznato je da algoritam GMM sporo konvergira. Kako bi se algoritam ponešto ubrzao, inicijalizacija središta $\boldsymbol{\mu}_k$ može se provesti algoritmom K-srednjih vrijednosti. Algoritam GMM onda će se moći fokusirati na procjenu parametara kovarijacijskih matrica te na fino ugađanje centroida;
- Kao što smo pokazali, možemo uvesti ograničenja na kovarijacijsku matricu Σ (npr., dijeljena, dijagonalna ili izotropna matrica). Takva ograničenja daje jednostavnije modele, koji su manje skloni prenaučenosti;
- Kao i kod svih algoritama grupiranja, broj grupe K je hiperparametar koji treba nekako unaprijed odrediti (metodama koje smo spominjali prošli put). U tu svrhu može se koristiti Akaikeov informacijski kriterij (AIC):

$$K^* = \operatorname{argmin}_K (-2 \ln \mathcal{L}(K) + 2q(K))$$

gdje je $q(K)$ broj parametara modela sa K grupa.

- EM-algoritam je općenit algoritam za **optimizaciju parametara modela s latentnih varijabla**. Ovdje smo ga primijenili na model GMM.

Algoritam GMM razlikuje se od algoritma K-sredina po tome što daje meko a ne tvrdo grupiranje. Međutim, oba su algoritma partijska. Pogledajmo sada jedan algoritam grupiranja koji nije partijski već hijerarhijski.

3 Hijerarhijsko grupiranje

Na kraju današnjeg predavanja razmotrit ćemo hijearhijski algoritam grupiranja, i to konkretno algoritam **hijerarhijskog aglomerativnog grupiranja** (engl. *hierarchical agglomerative clustering*, HAC). Radi se o jednom vrlo jednostavnom ali i vrlo dobrom i široko korištenom algoritmu grupiranja, pogotovo u dubinskoj analizi podataka. HAC, naravno, nije jedini algoritam za hijerarhijsko grupiranje, no mi nećemo razmatrati druge algoritme.

9

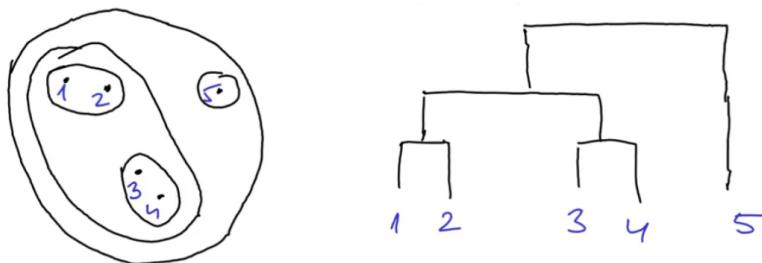
3.1 Dendrogram

Krenimo od toga da razmotrimo na koji se način može prikazati rezultat hijerarhijskog grupiranja. Za razliku od partijskog grupiranja, koje rezultira grupama koje odgovaraju skupovima primjera, hijerarhijsko grupiranje, pa tako i algoritam HAC, rezultira grupama koje se sastoje od podgrupa, odnosno **hijerarhijskom grupa**. Takva se hijerarhija može prikazati tzv. **dendrogramom**. Pogledajmo primjer.

10

► PRIMJER

Neka je pet primjera u dvodimenzijском ulaznom prostoru raspoređeno kao na slici dolje lijevo. Te primjere po euklidskoj udaljenosti (odnosno bliskosti) možemo grupirati tako da su primjeri 1 i 2 u jednoj grupi, primjeri 3 i 4 u drugoj grupi, a primjer 5 u trećoj grupi. Time smo primjere grupirali na najnižoj hijerarhijskoj razini. Na idućoj razini možemo grupirati grupe primjera: možemo spojiti grupe 1 i 2 u jednu nadgrupu te grupe 3 i 4 u drugu nadgrupu. Konačno, na najvišoj razini, ove dvije nadgrupe možemo spojiti u jednu nadnadgrupu. Takvom grupiranju onda odgovara dendrogram prikazan na slici dolje desno.



Dendrogram je zapravo stablasti prikaz hijerarhije. U listovima dendrograma su primjeri koje grupiramo, a grane povezuju primjere koji su u istoj grupi te grupe u nadgrupe. Na najnižoj razini (listovi) svaki primjer je u svojoj zasebnoj grupi. Kako se uspinjemo uz dendrogram, grupe od po jednog primjera najprije spajamo u grupe s po dva primjera, a zatim u grupe s po tri primjera, itd. Na najvišoj razini (korijen) svi primjeri čine jednu grupu. Vertikalna os dendrograma indicira na kojoj se udaljenosti događa spajanje grupa: što se više uspinjemo od listova prema korijenu dendrograma, to je veća udaljenost između grupa koje spajamo u jednu nadgrupu. Drugim riječima, visine grana indiciraju koliko su daleko grupe koje su spojene u nadgrupu: ako su grane dugacke, znači da su grupe dosta udaljene i da se spajaju na većoj udaljenosti. Ako su grane vrlo kratke, onda to znači da su grupe vrlo blizu i da se spajaju na maloj udaljenosti.

3.2 Povezivanje grupa

Hijerarhijsko se grupiranje općenito provodi na temelju neke **mjere udaljenosti** $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ između primjera ili na temelju **mjere sličnosti (ili različitosti)** $s : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ primjera. U tom je smislu hijerarhijsko grupiranje slično algoritmu K-medoida, o kojem smo pričali prošli put, a koji isto tako može koristiti općenitu mjeru sličnosti odnosno različitosti. Prisjetimo se da je mjeru sličnosti/različitosti općenitija od mjeru udaljenosti. Naime, mjeru udaljenosti mora zadovoljavati svojstva metrike, uključivo nejednakost trokuta, dok mjeru sličnosti/različitosti to ne mora. Tipično korištene mjeru udaljenosti su euklidska udaljenost (L_2), Manhattan udaljenost (L_1) i Mahalanobisova udaljenost, dok za mjeru različitosti postoji više mogućnosti, sve dok su zadovoljena sljedeća svojstva:

- (1) $s(\mathbf{x}, \mathbf{x}) = 1$,
- (2) $0 \leq s(\mathbf{x}^a, \mathbf{x}^b) \leq 1$,
- (3) $s(\mathbf{x}^a, \mathbf{x}^b) = s(\mathbf{x}^b, \mathbf{x}^a)$.

Prisjetimo se: to su ista ona svojstva koja smo imali za **jezgrene funkcije**, koje nisu ništa drugo nego funkcije sličnosti.

Hijerarhijsko grupiranje može biti aglomerativno ili divizivno. **Aglomerativno grupiranje** kreće od grupe koje sadrže svaka po samo jedan primjer, i zatim postepeno stapa najbliže grupe dok sve primjere ne stopi u jednu veliku grupu. To znači da se dendrogram gradi odozdo prema gore (od pojedinačnih primjera do zajedničke grupe). **Divizivno grupiranje** ide obrnuto: cijeli skup primjera dijeli se na grupe i podgrupe, tj. dendrogram se gradi odozgo prema dolje. Mi ćemo se u nastavku baviti isključivo aglomerativnim grupiranjem.

Kod aglomerativnog grupiranja, u svakom koraku treba stopiti dvije najbliže odnosno naj-sličnije grupe \mathcal{G}_i i \mathcal{G}_j . Ako te dvije grupe imaju svaka po jedan primjer, onda lako možemo izračunati udaljenost (odnosno sličnost) između takvih grupa jer se udaljenost (odnosno sličnost) svodi na udaljenost (odnosno sličnost) između dotični primjera. Međutim, postavlja se pitanje kako izračunati udaljenost (odnosno sličnost) između grupa koje sadrže više od jednog primjera? Računanje udaljenosti (odnosno sličnosti) između grupa naziva se **povezivanje** (engl. *linkage*), jer ćemo na temelju toga povezivati grupe u nadgrupe. Postoji više metoda povezivanja. U nastavku ćemo govoriti o metodama povezivanja na temelju udaljenosti, no razmatranja vrijede analogno i za mjeru sličnosti.

Prva mogućnost jest da udaljenost između dviju grupa definiramo kao **najmanju udaljenost** između pojedinačnih primjera u tim grupama. Formalno:

$$d_{min}(\mathcal{G}_i, \mathcal{G}_j) = \min_{\mathbf{x}^{(i)} \in \mathcal{G}_i, \mathbf{x}^{(j)} \in \mathcal{G}_j} d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

Takvo grupiranje nazivamo grupiranje **jednostrukе povezanosti** (engl. *single-linkage clustering*). Grupiranje nazivamo jednostrukim jer se dvije grupe povezuju na temelju udaljenosti samo jednog para primjera iz dviju grupa (svi ostali parovi primjera su više udaljeni). Alternativno, udaljenost između grupa možemo definirati kao **najveću udaljenost** između pojedinačnih primjera iz dviju grupa:

$$d_{max}(\mathcal{G}_i, \mathcal{G}_j) = \max_{\mathbf{x}^{(i)} \in \mathcal{G}_i, \mathbf{x}^{(j)} \in \mathcal{G}_j} d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

Takvo grupiranje nazivamo grupiranje **potpunim povezivanjem** (engl. *complete-linkage clustering*). Grupiranje nazivamo potpunim jer, ako smo primjere povezali na temelju udaljenosti dva najdalja primjera iz dviju grupa, onda su svi drugi parovi primjera iz dviju grupa još manje udaljeni, odnosno možemo reći da su grupe povezane preko svih parova primjera (potpuno). Nameće se pitanje: koje povezivanje odabratи u praksi? Ako su grupe kompaktne i prirodno dobro odvojene, onda ove dvije metode ne daju značajno različite rezultate. Međutim, ako to nije slučaj, razlike mogu biti značajne. Tada jednostruko povezivanje rezultira dugačkim, ulančanim grupama, dok potpuno povezivanje rezultira manjim, zbijenijim grupama.

Jednostruko i potpuno povezivanje predstavljaju dva krajnja slučaja izračuna udaljenosti između grupa i obje su metode dosta osjetljive na šum u podatcima. Kompromisno rješenje jest grupiranje na temelju **prosječne povezanosti** (engl. *average-linkage clustering*):

$$d_{avg}(\mathcal{G}_i, \mathcal{G}_j) = \frac{1}{N_i N_j} \sum_{\mathbf{x} \in \mathcal{G}_i} \sum_{\mathbf{x}' \in \mathcal{G}_j} d(\mathbf{x}, \mathbf{x}')$$

Konačno, četvrta mogućnost je povezivanje na temelju **centroida**:

$$d_{cent}(\mathcal{G}_i, \mathcal{G}_j) = \left\| \frac{1}{N_i} \sum_{\mathbf{x} \in \mathcal{G}_i} \mathbf{x} - \frac{1}{N_j} \sum_{\mathbf{x} \in \mathcal{G}_j} \mathbf{x} \right\|$$

Kod povezivanja na temelju centroida imamo ograničenje da ulazni prostor mora biti vektorski prostor. Primijetite da prosječno povezivanje i povezivanje centroida nisu identične mjerе udaljenosti.

Često je korištena i **Wardova metoda minimalne varijance**, koja udaljenost između grupa izračunava kao povećanje kvadrata udaljenosti između primjera i centroida prije i poslije stapa-nja dviju grupa. Ovo podsjeća na kriterijsku funkciju K-sredina. Ovdje nećemo u detalje.

12

Kako odabrati optimalnu metodu povezivanja? Različite metode unose sa sobom različite pretpostavke. Kako to inače biva, optimalna će biti ona metoda čije su pretpostavke najbolje usklađene sa stvarim stanjem na terenu, tj. s našim podatcima. Npr., potpuno povezivanje rezultirat će grupama kompaktnih rubova, dok Wardova metoda daje grupe koje mogu imati raspršene rubove. U praksi se odabir ipak nerijetko svodi na eksperimentiranje.

13

3.3 Hijerarhijsko aglomerativno grupiranje (HAC)

Složimo sada sve ovo u **algoritam hijerarhijskog aglomerativnog grupiranja** (engl. *hierarchical agglomerative clustering*, HAC). U nastavku je prikazan pseudokod.

► Algoritam hijerarhijskog aglomerativnog grupiranja (HAC)

- 1: **inicijaliziraj** K , $k \leftarrow N$, $\mathcal{G}_i \leftarrow \{\mathbf{x}^{(i)}\}$ za $i = 1, \dots, N$
- 2: **ponavlja**
- 3: $k \leftarrow k - 1$
- 4: $(\mathcal{G}_i, \mathcal{G}_j) \leftarrow \underset{\mathcal{G}_a, \mathcal{G}_b}{\operatorname{argmin}} d(\mathcal{G}_a, \mathcal{G}_b)$
- 5: $\mathcal{G}_i \leftarrow \mathcal{G}_i \cup \mathcal{G}_j$
- 6: **dok je** $k > K$

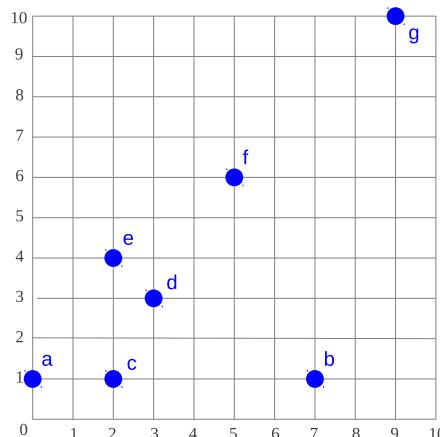
Hiperparametar K je unaprijed zadani broj grupa. Primijetite da ovaj pseudokod zapravo ne gradi eksplicitan dendrogram. No, to lako možemo ostvariti tako da umjesto unije skupova u koraku 5 kreiramo čvor dendrograma i u njegove listove stavimo grupe koje spajamo.

Za $K = 1$ dobivamo potpun dendrogram koji možemo naknadno **presijecati**. To znači da možemo odabrati neku udaljenost (odnosno razinu sličnosti, odnosno razinu različitosti) i na tom mjestu (na toj visini dendrograma) uzdužno presjeći grane dendrograma. To će nam dati partijsko grupiranje. Odabir mjesta presijecanja istovjetan je odabiru broja grupa.

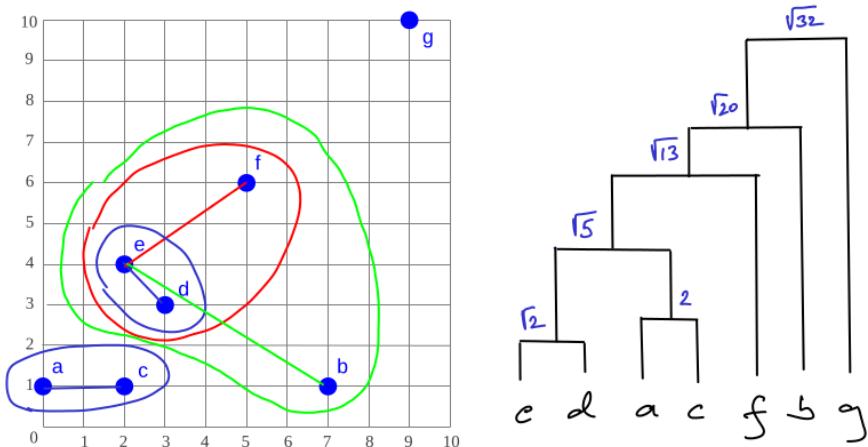
Pogledajmo jedan jednostavan primjer.

► PRIMJER

Algoritmom HAC grupiramo primjere prema euklidskoj udaljenosti u dvodimenzijskome prostoru. Skup se sastoji od sljedećih sedam primjera:

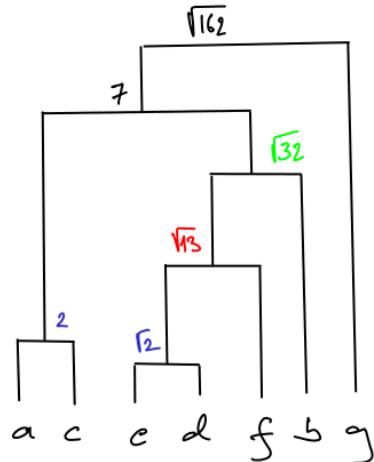


Razmotrimo najprije grupiranje s jednostrukom povezanošću, što znači da ćemo udaljenost između grupa računati kao minimalnu udaljenost između svih parova primjera iz tih dviju grupa. Prisjetite se da radimo aglomerativno grupiranje: dakle, krećemo s najvećim brojem grupa, a kako grupe spajamo u nadgrupe, tako smanjujemo broj grupa, sve dok sve grupe ne spjimo u jednu zajedničku grupu. Na početku grupiranja svaki primjer primjer trećiramo kao zasebnu grupu, dakle imamo $K = 7$ grupa s po jednim primjerom. Zatim spajamo one grupe koje su najbliže, računajući udaljenosti između grupa kao maksimalnu udaljenost između primjera u tim grupama. U prvom koraku to je jednostavno, budući da trebamo izračunati udaljenosti između grupa koje sadrže samo po jedan primjer, a to je isto kao da računamo udaljenosti između tih primjera. Dakle, u prvom koraku najbliže su grupe $\{d\}$ i $\{e\}$ (njihova udaljenost je udaljenost je $\sqrt{2}$), pa te dvije grupe spajamo u jednu grupu, $\{d, e\}$, i to spajanje se događa na udaljenosti $d_{min} = \sqrt{2}$. Time je ukupan broj grupa smanjen na $K = 6$. U drugom koraku, najbliže su grupe $\{a\}$ i $\{c\}$, pa njih spajamo u zajedničku grupu, $\{a, c\}$, i to na udaljenosti $d_{min} = 2$. Time je ukupan broj grupa smanjen na $K = 5$. U trećem koraku najbliže su upravo te dvije novostvorene grupe: naime, udaljenost između grupe $\{a, c\}$ i $\{d, e\}$ je minimalna udaljenost između 4 parova primjera između tih grupe (to su parovi (a, d) , (a, e) , (c, d) i (c, e)), i ta minimalna udaljenost iznosi $d_{min} = \sqrt{5}$ (za par (c, d)). Uvjerite se da je to doista najmanja udaljenost između svih parova grupa u ovom koraku, tj. da je udaljenost između svih drugih parova od 5 grupa koje u ovom koraku imamo veća od $\sqrt{5}$. Dakle, u trećem koraku na udaljenosti $d_{min} = \sqrt{5}$ spajamo grupe $\{a, c\}$ i $\{d, e\}$ te dobivamo nadgrupu $\{a, c, d, e\}$. Time je ukupan broj grupa smanjen na $K = 4$. U četvrtom koraku ispostavlja se da su najbliže grupe $\{f\}$ i grupa $\{a, c, d, e\}$, i to na udaljenosti $d_{min} = \sqrt{13}$. Spajanjem tih dviju grupe dobivamo grupu a, c, d, e, f te ukupno onda imamo $K = 3$ grupe. U iduća dva koraka još ćemo tu grupu spojiti s grupom $\{b\}$ i zatim s grupom $\{g\}$, te završiti s jednom velikom grupom, $K = 1$, čime se postupak grupiranja zaustavlja. Na slici dolje lijevo prikazan je rezultat grupiranja kao ugniježđene grupe (prikazan je predzadnji korak, prije stapanja s grupom koja sadrži primjer g).



Paralelno sa postupkom grupiranja možemo crtati dendrogram, pri čemu kod svakog spajanja grupa naznačavamo na kojoj je udaljenosti spajanje načinjeno. Dendrogram za ovo grupiranje prikazan je na slici gore desno. Budući da uvijek spajamo par grupa, dendrogram će uvijek biti binarno stablo.

Gornje grupiranje napravili smo s jednostrukim povezivanjem. Za vježbu, ponovite postupak grupiranja s potpunim povezivanjem. Postupak je sličan, samo što udaljenost između parova grupa treba računati kao maksimalnu udaljenost između parova primjera iz tih grupa, a ne kao minimalnu udaljenost. Grupiranje s potpunim povezivanjem daje ovakav dendrogram:



Ako sada iz hijerarhijskog grupiranja želimo dobiti partijsko grupiranje, onda ove dendrogram možemo presijecati na nekoj odabranoj udaljenosti ili na nekom odabranom broju grupa. Na primjer, ako gornji dendrogram presjećemo na udaljenosti između $\sqrt{13}$ i $\sqrt{32}$ (npr. na udaljenosti 5), onda ćemo dobiti $K = 4$ grupe, i to: $\{a, c\}$, $\{d, e, f\}$, $\{b\}$ i $\{g\}$. Dakle, grupe čija je točka u dendrogramu spajanja ispod točke presijecanja ostaju kao jedna grupa, dok se grupe čija je točka spajanja iznad točke presijecanja razlamaju u više grupe. Isti rezultat, naravno, dobili bismo da se odlučimo za $K = 4$ grupe. Kako odrediti optimalan broj grupa? Vrijede iste napomene kao i prošli put: ili unaprijed znamo koji bi broj grupa trebao biti, ili moramo upotrijebiti neku od metoda za nalaženje optimalnog broja grupa.

Primijetite da su primjeri ovdje bili definirani u vektorskome prostoru i da smo grupiranje radili na temelju euklidske udaljenosti. No, algoritam HAC može se primjeniti i za općenitiji slučaj gdje primjeri nisu u vektorskome prostoru i grupiranje provodimo na temelju neke općenite mjere sličnosti odnosno različitosti.

Što je s računalnom složenošću algoritma HAC? Prostorna složenost određena je matricom udaljenosti/sličnosti, koja sadržava udaljenosti/sličnosti između $\binom{N}{2}$ parova primjera. To je $\mathcal{O}(N^2)$. U praksi, kod vrlo velikog broja primjera, ova kvadratna prostorna složenost može već biti nepremostiv problem. Što se vremenske složenosti tiče, najprije ustanovima da je broj koraka algoritma $N - K$ (odnosno N ako $K = 1$, tj. ako gradimo kompletan dendrogram). Za pronalaženje najbližeg/najsličnijeg para grupa potrebno je $\mathcal{O}(N^2)$ izračuna. Dakle, ukupna je vremenska složenost $\mathcal{O}((N - K)N^2) \approx \mathcal{O}(N^3)$. Međutim, moguća je implementacija s prioritetsnom listom, čija je vremenska složenost $\mathcal{O}(N^2 \log N)$. Nadalje, specifično kod jednostrukog povezivanja, vremenska složenost je $\mathcal{O}(N^2)$.

Ovime završava naš kratak izlet u nenadzirano strojno učenje.

Sažetak

- Kod **probabilističkog grupiranja** primjeri pripadaju grupama s određenom vjerojatnošću
- Algoritam GMM je **poopćenje** algoritma K-srednjih vrijednosti gje su grupe modelirane Gaussovim gustoćama vjerojatnosti
- Probabilističko grupiranje možemo promatrati kao **optimizaciju log-izglednosti Gaussove mješavine**
- Procjenu parametara možemo napraviti **algoritmom maksimizacije očekivanja (EM-algoritam)**
- **Hijerarhijsko aglomerativno grupiranje (HAC)** postepeno stapa najbliže/najsličnije grupe i gradi **dendrogram**

- HAC može raditi s bilo kojom mjerom **sličnosti**, no glavni nedostatak je **kvadratna prostorna složenost**

Bilješke

- Ovo predavanje zasniva se na poglavljima 7.2 i 7.4 iz ([Alpaydin, 2020](#)) te poglavlju 9.2 iz ([Bishop, 2006](#)) (za temu GMM) te poglavlju 7.4 iz ([Alpaydin, 2020](#)) (za temu hijerarhijskog grupiranja).
- Algoritmima **neizrazitog grupiranja** (engl. *fuzzy clustering*) nećemo se baviti. Tipičan primjer je algoritam **fuzzy C-means (FCM)** ([Dunn, 1973](#)). Više možete pročitati ovdje: https://matteucci.faculty.polimi.it/Clustering/tutorial_html/cmeans.html.
- Za primjenu **Gibbsovog uzorkovanja** za učenje modela GMM, vidjeti ([Diebolt and Robert, 1994](#)). U ([Levine and Casella, 2001](#)) je predložen hibridni pristup, **Monte Carlo EM-algoritam**, kod kojega se u očekivanje izglednosti u E-koraku računa Monte Carlo simulacijom. Usporedbu ovih metoda za učenje modela GMM možete naći u ([Zaheer et al., 2015](#)).
- EM-algoritam** osmisili su 1977. godine američki statističari Arthur Dempster, Nan Laird i Donald Rubin ([Dempster et al., 1977](#)). EM-algoritam doista je jedan od važnijih i širokoprimenjivih algoritama, čemu u prilog govori i da je izvorni članak iz 1977. godine do sada citiran više od 62k puta. Pristupačan opis EM-algoritma, s primjenom na GMM i na HMM (ovo drugo mi nećemo raditi) možete naći u ([Bilmes et al., 1998](#)). Lijep pregled različitih pogleda na EM-algoritam te njegovih različitih varijanti možete naći u ([Roche, 2011](#)).
- Dokaz da maksimizacija očekivanja potpune log-izglednosti dovodi do povećanje nepotpune log-izglednosti može se naći u ([Wu, 1983](#)) ili na https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm#Proof_of_correctness
- Primjer je preuzet iz dokumentacije knjižnice scikit-learn, sa https://scikit-learn.org/stable/auto_examples/mixture/plot_gmm_covariances.html.
- Treba napomenuti da je ovaj primjer, preuzet sa web-stranice biblioteke scikit-learn, malo nerealan. Naime, sredine grupe μ_k izračunate su na temelju oznaka primjera (koje su ovdje bile na raspolaganju), dok su samo parametri π_k i Σ_k učeni iz podataka. Zbog toga je ovdje i točnost grupiranja (ovde vrlo netipično definirana kao podudaranje oznaka grupe i oznaka primjera) visoka. U stvarnosti ćemo rijetko vidjeti 100% točnost na ispitnome skupu. Također, u praksi nije tipično da ispitna točnost bude bolja od točnosti na skupu za učenje.
- BIC** (engl. *Bayesian information kriterion*) je kriterij za odabir modela sličan **Akaikeovom kriteriju** (o kojem smo pričali prošli put). O razlici između BIC i AIC možete pročitati ovdje: <https://stats.stackexchange.com/q/577/93766>. Za primjer uporabe BIC-a za odabir broja grupa modela GMM, v. <https://stats.stackexchange.com/q/368560/93766>.
- Dobar pregled **algoritama hijerarhijskog grupiranja** možete naći u ([Murtagh and Contreras, 2012](#)).
- Dendrogram** je složenica od grčkih riječi *dendron* (stablo) i *gramma* (crtež). Često ćete vidjeti da se (pogrešno) koristi naziv *dendogram* (bez "r").
- Grupiranje **jednostrukim povezivanjem** odnosno **potpunim povezivanjem** imaju lijepo tumačenje u **teoriji grafova**. Stapanje dviju grupa, \mathcal{G}_i i \mathcal{G}_j , odgovara uvođenju brida između odgovarajućih primjera u tim dvjema grupama. Kod jednostrukog povezivanja, to su dva najbliža primjera iz svake grupe. Budući da se bridovi uvijek uvode između primjera različitih grupa, a nikad između primjera iz iste grupe, rezultirajući graf je stablo (tj. nema ciklusa). Ako $K = 1$, algoritam HAC generira **minimalno razapinjuće stablo** (engl. *minimal spanning tree*) – stablo sa stazom između svaka dva brida kod kojega je ukupan težinski zbroj bridova minimalan. Suprotno, kod **potpunog povezivanja**, stapanje dviju grupa odgovara uvođenju bridova između svih parova primjera, pa algoritam HAC

rezultira **potpuno povezanim grafom**.

- [12] **Wardovu metodu minimalne varijance** predložio je 1963. godine Joe Ward (Ward Jr, 1963). Sažet opis postupka možete naći na https://en.wikipedia.org/wiki/Ward%27s_method. Lijep primjer primjene Wardove metode možete naći na <https://jbhender.github.io/Stats506/F18/GP/Group10.html>.
- [13] Ovdje možete naći dobar pregled različitih **metoda povezivanja** <https://stats.stackexchange.com/a/217742/93766> i opis toga kakvo grupiranje možete očekivati dobiti svakom od metoda.
- [14] Kod dendrograma imamo zgodnu mogućnost da na temelju visine grana dendrograma odlučimo o optimalnom mjestu **presjecanja dendrograma** (a time onda i optimalnom broju grupa): na mjestima gdje su sve grane razmjerno visoke možemo napraviti presjecanje jer je tamo grupiranje konzistentno. Naime, ako su sve grane dugačke, to onda znači da bi za bilo kakva daljnja stapanja trebalo stopiti poprilično udaljene (različite) grupe, kao i da bi za daljnja razdvajanja trebalo razdvojiti poprilično bliske (slične) grupe. Kvantifikaciju ove ideje pružaju **koefficijenti nekonzistentnosti**, koji se temelje na duljini grane i duljini grana poslije razdvajanja. Za detalje, pogledati <https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.inconsistent.html>
- [15] Grupiranje jednostrukim povezivanjem istovjetno je izračunavanju minimalnog razapinjućeg stabla. Vremenska složenost tog algoritma (**Prim-Jarníkov algoritam**) je $\mathcal{O}(N^2)$, gdje je N broj čvorova. V. https://en.wikipedia.org/wiki/Prim%27s_algorithm

Literatura

- E. Alpaydin. *Introduction to machine learning*. MIT press, 2020.
- J. A. Bilmes et al. A gentle tutorial of the EM algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.
- C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
- A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1): 1–22, 1977.
- J. Diebolt and C. P. Robert. Estimation of finite mixture distributions through bayesian sampling. *Journal of the Royal Statistical Society: Series B (Methodological)*, 56(2):363–375, 1994.
- J. C. Dunn. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. 1973.
- R. A. Levine and G. Casella. Implementations of the Monte Carlo EM algorithm. *Journal of Computational and Graphical Statistics*, 10(3):422–439, 2001.
- F. Murtagh and P. Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.
- A. Roche. EM algorithm and variants: An informal tutorial. *arXiv preprint arXiv:1105.1476*, 2011.
- J. H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.
- C. J. Wu. On the convergence properties of the EM algorithm. *The Annals of statistics*, pages 95–103, 1983.

M. Zaheer, M. Wick, S. Kottur, and J.-B. Tristan. Comparing Gibbs, EM and SEM for MAP inference in mixture models. 2015.

21. Vrednovanje modela

Strojno učenje 1, UNIZG FER, ak. god. 2022./2023.

Jan Šnajder, predavanja, v1.3

Do sada smo se na ovom kolegiju bavili algoritmima strojnog učenja: kako ti algoritmi rade, od čega su sastavljeni i koje induktivne pretpostavke su u njih ugrađene. Upoznali smo niz algoritama, dok je naravno još više onih koje nismo ni spomenuli. Činjenica da su nam na raspolaganju brojni algoritmi je izvrsna, ali ujedno predstavlja praktičan izazov: kako znati koji je algoritam najbolji za naš problem? Da bismo mogli odgovoriti na ovo pitanje, moramo nekako usporediti predikcije algoritama. Kriterij koji nam je pritom ključan je, naravno, točnost algoritma na neviđenim podatcima, odnosno pogreška generalizacije. A da bismo što bolje procijenili tu točnost odnosno pogrešku, moramo znati kako **ispravno vrednovati (evaluirati) algoritam**. Premda se na prvi pogled možda ne čini tako, vrednovanje algoritama strojnog učenja vrlo je sklizak teren i česti su slučajevi gdje ono nije ispravno odradeno, kako u industrijskoj praksi tako i u akademiji. Cilj ovog predavanja jest da vi nipošto ne budete među onima koji ne znaju vrednovati algoritme strojnog učenja!

Tri su stvari bitne za vrednovanje algoritama strojnog učenja: koju **mjeru vrednovanja** koristiti, kako pravedno (realistično) **procijeniti pogrešku modela** i kako napraviti **statističku analizu rezultata** kako bismo bili donekle sigurni da naš rezultat nije puka slučajnost. Danas ćemo se baviti prvim dvjema temama (mjerama vrednovanja i procjenom pogreške), dok statističku usporebu modela ostavljamo za iduće predavanje. Također, u ovom predavanju ograničit ćemo se na vrednovanje klasifikatora; vrednovanje regresije i algoritama grupiranja je važno, ali izvan dosega.

1 Osnovne mjere vrednovanja

Mjere vrednovanja (evaluacijska mjeru, evaluacijska metrika) je metoda koja na neki način kvantificira točnost (ili pogrešku) klasifikatora. Postoji mnogo mjeri vrednovanja, i zapravo je najveći izazov izabrati onu mjeru koja je prikladna za problem koji rješavamo.

Polazna točka kod vrednovanja klasifikatora jest da klasifikator primijenimo na skup primjera (tipično je to ispitni skup, ali možemo ga primijeniti i na cijeli skup podataka) tako da za svaki primjer dobijemo predikciju klasifikatora. Te označke nazivamo **predviđene označke** (engl. *predicted labels*). Naravno, da bismo mogli vrednovati klasifikator, za isti taj skup primjera moramo imati i točne označke svih primjera. Te označke nazivamo **stvarne označke** (engl. *true (actual) labels*). Ove označke možemo organizirati u vektore označaka. Tako ćemo tako dobiti **vektor predviđenih označaka**:

$$\mathbf{y}_{\text{pred}} = (y_{\text{pred}}^{(1)}, \dots, y_{\text{pred}}^{(i)}, \dots, y_{\text{pred}}^{(N)})^T$$

gdje je $y_{\text{pred}}^{(i)} = h(\mathbf{x}^{(i)})$ predviđena označka za i -ti primjer. Analogno, imamo i **vektor stvarnih označaka**:

$$\mathbf{y}_{\text{true}} = (y_{\text{true}}^{(1)}, \dots, y_{\text{true}}^{(i)}, \dots, y_{\text{true}}^{(N)})^T$$

gdje je $y_{\text{true}}^{(i)}$ stvarna označka za i -ti primjer. Ovi su vektori jednake duljine; pretpostavimo da je ta duljina jednaka N , broju primjera u skupu označenih primjera.

Sve mjere vrednovanja kojima ćemo se danas baviti temelje se na usporedbi vektora \mathbf{y}_{pred} i vektora \mathbf{y}_{true} . Očito, ako želimo izračunati običnu **točnost** (engl. *accuracy*) klasifikatora, možemo jednostavno pobrojati za koliko se primjera te oznake podudaraju i taj broj podijeliti s ukupnim brojem primjera:

$$Acc = \frac{1}{N} \sum_{i=1}^N \mathbf{1}\{\mathbf{y}_{pred}^{(i)} = \mathbf{y}_{true}^{(i)}\}$$

Pogrešku onda možemo izračunati jednostavno kao $1 - Acc$.

Ograničimo se u nastavku na problem binarne klasifikacije (višeklasnom klasifikacijom bavit će se kasnije). Za binarnu klasifikaciju oznake su $y \in \{0, 1\}$, pa su \mathbf{y}_{pred} i \mathbf{y}_{true} binarni vektori. U tom slučaju točnost možemo definirati i kao $Acc = \frac{1}{N} \mathbf{y}_{pred}^T \mathbf{y}_{true}$.

1.1 Matrica zabune

Mjera točnosti koju smo upravo definirali samo je jedna od mnogo mjera koje možemo izračunati na temelju usporedbe vektora \mathbf{y}_{pred} i \mathbf{y}_{true} . Mjera točnosti je intuitivna, ali, kao što ćemo uskoro vidjeti, nije uvijek prikladna. Koje još sve mjere vrednovanja možemo izračunati postaje jasnije ako iz vektora \mathbf{y}_{pred} i \mathbf{y}_{true} izgradimo **matricu zabune (kontingencije)** (engl. *confusion (contingency) matrix*). Matrica zabune je kvadratna matrica koja na sažet način prikazuje broj podudaranja i nepodudaranja predikcija i stvarnih oznaka. Za binarni klasifikator, matrica zabune je dimenzija 2×2 i općenito izgleda ovako:

$$\begin{array}{cc} y_{true} = 1 & y_{true} = 0 \\ y_{pred} = 1 & \left(\begin{array}{cc} TP & FP \\ FN & TN \end{array} \right) \\ y_{pred} = 0 & \end{array}$$

Ovdje retci odgovaraju predviđenim oznakama (*pred*), a stupci stvarnim oznakama (*true*). Razlikujemo četiri vrste slučajeva, koji odgovaraju elementima matrice:

- **stvarno pozitivni** (engl. *true positive*, *TP*): predikcija je 1 i stvarna oznaka je 1;
- **lažno pozitivni** (engl. *false positive*, *FP*): predikcija je 1 a stvarna oznaka je 0;
- **lažno negativni** (engl. *false negative*, *FN*): predikcija je 0 a stvarna oznaka je 1;
- **stvarno negativni** (engl. *true negative*, *TN*): predikcija je 0 i stvarna oznaka je 0.

Primijetite da, kako je uobičajeno u strojnom učenju, ovdje za “negativan” primjer koristimo oznaku $y = 0$. Za zadane konkretne vektore oznaka \mathbf{y}_{pred} i \mathbf{y}_{true} , matrica zabune bilježi ukupan broj primjera koji su stvarno pozitivni, lažno pozitivni, lažno negativni i stvarno negativni. Dakle, elementi matrice zabune su:

$$\begin{aligned} TP &= \sum_{i=1}^N \mathbf{1}\{\mathbf{y}_{pred}^{(i)} = \mathbf{y}_{true}^{(i)} = 1\} & FP &= \sum_{i=1}^N \mathbf{1}\{\mathbf{y}_{pred}^{(i)} = 1 \wedge \mathbf{y}_{true}^{(i)} = 0\} \\ FN &= \sum_{i=1}^N \mathbf{1}\{\mathbf{y}_{pred}^{(i)} = 0 \wedge \mathbf{y}_{true}^{(i)} = 1\} & TN &= \sum_{i=1}^N \mathbf{1}\{\mathbf{y}_{pred}^{(i)} = \mathbf{y}_{true}^{(i)} = 0\} \end{aligned}$$

Pogledajmo napokon jedan primjer.

► PRIMJER

Raspolažemo sa $N = 7$ primjera. Predikcije binarnog klasifikatora i stvarne oznake primjera su sljedeće:

$$\begin{aligned} \mathbf{y}_{pred} &= (1, 0, 1, 0, 0, 0, 1)^T \\ \mathbf{y}_{true} &= (1, 0, 0, 0, 1, 0, 0)^T \end{aligned}$$

Iz ovoga dobivamo sljedeću matricu zabune:

$$\begin{array}{cc} y_{true} = 1 & y_{true} = 0 \\ \begin{matrix} y_{pred} = 1 \\ y_{pred} = 0 \end{matrix} & \left(\begin{array}{cc} 1 & 2 \\ 1 & 3 \end{array} \right) \end{array}$$

Iz matrice zabune možemo vidjeti da je najviše stvarno negativnih primjera: od pet negativnih primjera, tri su stvarno negativna, a dva lažno negativna. Od dva primjera koji su pozitivni, jedan je stvarno pozitivan, a jedan je lažno negativan.

Primijetite da smo matricu zabune definirali tako da retci odgovaraju predviđenoj oznaci, a stupci stvarnoj oznaci, te smo prvo naveli vrijednost 1 a zatim vrijednost 0. Majte na umu da smo takvu organizaciju proizvoljno odabrali. U literaturi ćete naići na drugačije organizirane matrice zabune, pa je dobro uvijek prvo provjeriti što znaće pojedini elementi.

Pogledajmo još jedan primjer.

► PRIMJER

Testiramo klasifikator za dijagnostiku bolesti na $N = 150$ primjera. Usporedbom predviđenih i stvarnih oznaka dobili smo sljedeću matricu zabune:

$$\begin{array}{cc} y_{true} = 1 & y_{true} = 0 \\ \begin{matrix} y_{pred} = 1 \\ y_{pred} = 0 \end{matrix} & \left(\begin{array}{cc} 6 & 12 \\ 2 & 130 \end{array} \right) \end{array}$$

Iz ove matrice možemo iščitati da je ukupan broj primjera $TP + FP + FN + TN = 150$. Također, možemo iščitati da je ukupan broj pozitivnih primjera jednak $TP + FN = 8$, dok je ukupan broj negativnih primjera jednak $TN + FP = 142$. Brojevi na dijagonali matrice zabune odgovaraju točno klasificiranim primjerima, pa je ukupan broj točno klasificiranih primjera jednak tragu matrice (zbroju dijagonalnih elemenata). Ovdje je to $TP + TN = 136$.

1.2 Mjere vrednovanja nad matricom zabune

Sada kada znamo što je matrica zabune, nad njom možemo definirati niz mjera vrednovanja. Krenimo s očitim. **Točnost** (engl. *accuracy*) je udio točno klasificiranih primjera u skupu svih primjera:

$$Acc = \frac{TP + TN}{TP + FP + FN + TN}$$

U prethodnom primjeru, točnost je jednaka $Acc = \frac{136}{150} = 90.7\%$.

Već smo rekli da je točnost intuitivna mjeru vrednovanja, ali da nije uvijek prikladna. Što je točno problem s točnošću? Problem je ovaj: ako je skup podataka izrazito **neuravnotežen** (engl. *imbalanced*) – a to znači da ili imamo mnogo više pozitivnih primjera od negativnih primjera, ili obrnuto – onda je trivijalno ostvariti klasifikator koji ima visoku točnost (odnosno nisku pogrešku). Jednostavno, klasifikator definiramo tako da uvijek predviđa većinsku klasu. Npr., u gornjem primjeru, broj pozitivnih primjera je 8, a broj negativnih je 142. Ako napravimo klasifikator koji jednostavno uvijek predviđa negativnu oznaku ($y = 0$), točnost takvog modela bit će visokih $142/150 = 94.67\%$.

Zbog toga su osmišljene alternativne mjere vrednovanja. Zapravo, kad pogledamo konfuzijsku matricu, vidimo da ona nudi mnoge mogućnosti kako definirati mjere vrednovanja (što zbrajati i što s čime dijeliti). Tako se pored točnosti uobičajeno koriste sljedeće mjere:

- **Preciznost** (engl. *precision*):

$$P = \frac{TP}{TP + FP}$$

Preciznost definiramo kao udio stvarno pozitivnih primjera (TP) u skupu svih primjera koje je klasifikator označio pozitivno ($TP + FP$). Idealno, $P = 1$, tj. svi primjeri koje je klasifikator označio pozitivno doista i jesu pozitivni.

- **Odziv** (engl. *recall*):

$$R = TPR = \frac{TP}{TP + FN}$$

Odziv je udio stvarno pozitivnih primjera (TP) u skupu svih skupu svih pozitivnih primjera ($TP + FN$). Ova se mjera naziva "odziv" jer nam govori koliko se pozitivnih primjera "odazvalo" klasifikatoru. Idealno, $R = 1$, tj. sve pozitivne primjere klasifikator će označiti kao takve. Alternativni nazivi za odziv su **stopa stvarnih pozitiva** (engl. *true positive rate*, *TPR*) i **osjetljivost** (engl. *sensitivity*). 7

- **Ispadanje** (engl. *false-out*, *false positive rate*):

$$FPR = \frac{FP}{FP + TN}$$

Ispadanje (ili **stopa lažnog alarmu**) je udio lažno pozitivnih primjera (FP) u skupu svih negativnih primjera ($FP + TN$). Idealno, $FPR = 0$, tj. klasifikator niti jedan negativni primjer neće lažno proglašiti pozitivnim.

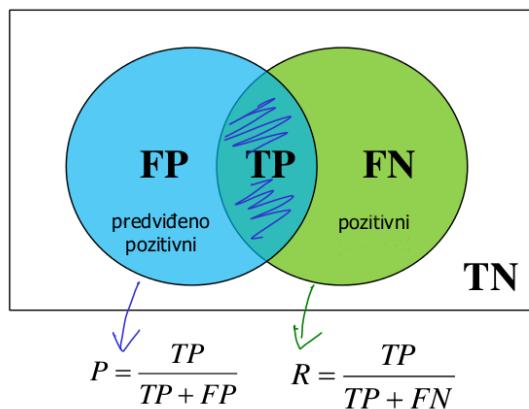
- **Specifičnost** (engl. *specificity*, *true negative rate*):

$$S = \frac{TN}{TN + FP}$$

Specifičnost je udio stvarno negativnih primjera (TN) u skupu svih negativnih primjera ($TN + FP$). Idealno, $S = 1$, tj. klasifikator će sve negativne primjere klasificirati kao takve.

Premda se sve navedene mjere koriste u praksi, pojedina područja obično koriste samo neke od njih. Na primjer, u pretraživanju informacija, obradi prirodnog jezika i računalnom vidu uglavnom se koriste preciznost i odziv, dok se kod primjena strojnog učenja u medicini i biostatistici uobičajeno koriste odziv (tada pod nazivom "osjetljivost") i specifičnost.

Mi ćemo se u nastavku koncentrirati na preciznost (P) i odziv (R). Te su dvije mjere međusobno komplementarne. To možemo lakše shvatiti ako ih vizualiziramo Vennovim dijagramom, ovako:



Područje unutar pravokutnika odgovara svim primjerima iz označenog skupa primjera. Zeleni krug odgovara primjerima koji su pozitivni (primjeri za koje $y_{true} = 1$), dok plavi krug odgovara primjerima koje je klasifikator označio kao pozitivne (primjeri za koje $y_{pred} = 1$). Presjek tih dvaju krugova odgovara stvarno pozitivnim primjerima (TP). Dio zelenog kruga koji je izvan presjeka odgovara primjerima koji jesu pozitivni, ali nisu klasificirani kao pozitivni, dakle to

su lažno negativni primjeri (FN). Analogno, dio plavog kruga koji je izvan presjeka odgovara primjerima koji su klasificirani kao pozitivni, ali oni to zapravo nisu, dakle to su lažno pozitivni primjeri (FP). Područje izvan oba kruga odgovara primjerima koji niti su pozitivni niti su klasificirani kao pozitivni, dakle negativni su i klasificirani su kao negativni, tj. to su stvarno negativni primjeri (TN). Mjere preciznosti P i odziva R možemo sada definirati kao omjere površina na sljedeći način. Preciznost nam govori koliko je primjera koje je klasifikator proglašio pozitivnima stvarno pozitivno. To je udio stvarno pozitivnih primjera u skupu pozitivno klasificiranih primjera, što odgovara omjeru površine presjeka krugova i površine plavog kruga. S druge strane, odziv nam govori koliko pozitivnih primjera je klasifikator stvarno proglašio pozitivnima. To je udio stvarno pozitivnih primjera u skupu pozitivnih primjera, što odgovara omjeru površine presjeka krugova i površine zelenog kruga. Idealna situacija jest ona kad je plavi krug savršeno preklopjen sa zelenim krugom: tada su svi pozitivno klasificirani primjeri stvarno pozitivni ($P = 1$) i, obrnuto, svi pozitivni primjeri su također klasificirani kao pozitivni ($R = 1$).

Vratimo se na naš prethodni primjer. Tamo je $P = \frac{6}{6+12} = 33.33\%$, a $R = \frac{6}{6+2} = 75\%$. To bi značilo da naš klasifikator nije baš precizan, ali možemo reći da ima relativno pristojan odziv. Obje mjere daju znatno manju vrijednost od mjere točnosti, koja je, zahvaljujući velikom broju negativnih primjera, iznosila 90.7%. Dakle, P i R nam daju realističniju procjenu točnosti klasifikatora (točnosti u širem smislu) od točnosti (u užem smislu).

Pogledajmo još dva primjera.

► PRIMJER

Od $N = 1000$ primjera, njih 100 je pozitivno. Klasifikator ispravno klasificira 90 pozitivnih i 650 negativnih primjera. Izračunajmo Acc , P i R .

Iz navedenog zaključujemo da je matrica zabune sljedeća:

$$\begin{array}{cc} y_{true} = 1 & y_{true} = 0 \\ \begin{matrix} y_{pred} = 1 \\ y_{pred} = 0 \end{matrix} & \left(\begin{array}{cc} 90 & 250 \\ 10 & 650 \end{array} \right) \end{array}$$

Iz ovoga slijedi:

$$\begin{aligned} Acc &= \frac{TP + TN}{TP + FP + FN + TN} = \frac{90 + 650}{1000} = 0.74 \\ P &= \frac{TP}{TP + FP} = \frac{90}{90 + 250} = 0.265 \\ R &= \frac{TP}{FP + FN} = \frac{90}{90 + 10} = 0.9 \end{aligned}$$

Preciznost je ovdje mnogo lošija od odziva jer postoji mnogo više lažno pozitivnih primjera nego lažno negativnih primjera. Drugim riječima, ovaj je klasifikator dobar u nalaženju pozitivnih primjera, ali loš u izbjegavanju lažno pozitivnih primjera.

► PRIMJER

Od $N = 1000$ primjera, njih 100 je pozitivno. Klasifikator ispravno klasificira 50 pozitivnih i 850 negativnih primjera. Izračunajmo Acc , P i R .

Iz navedenog možemo zaključiti da matrica zabune izgleda ovako:

$$\begin{array}{cc} y_{true} = 1 & y_{true} = 0 \\ \begin{matrix} y_{pred} = 1 \\ y_{pred} = 0 \end{matrix} & \left(\begin{array}{cc} 50 & 50 \\ 50 & 850 \end{array} \right) \end{array}$$

Iz ovoga slijedi:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} = \frac{50 + 850}{1000} = 0.9$$

$$P = \frac{TP}{TP + FP} = \frac{50}{50 + 50} = 0.5$$

$$R = \frac{TP}{FP + FN} = \frac{50}{50 + 50} = 0.5$$

Primijetimo da su preciznost i odziv ovdje jednaki, budući da je broj lažno pozitivnih primjera jednak broju lažno negativnih primjera.

Primijetite da su i preciznost i odziv definirane s brojem stvarno pozitivnih primjera (TP) u brojniku i da su neovisne o broju stvarno negativnih primjera (TN). Možemo reći da su obje mjere fokusirane na pozitivne primjere. Posljedično, ako klasifikator ne uspije baš niti jedan pozitivan primjer označiti kao pozitivan, onda u brojniku imamo $TP = 0$ te će i preciznost i odziv biti jednaki nuli. Poseban slučaj takvog slučaja jest klasifikator koji sve primjere označi kao negativne: u tom slučaju $TP = 0$ i $TP + FP = 0$, odziv je nula no preciznost je nedefinirana. Suprotan (i manje realan) slučaj bio bi klasifikator koji sve negativne i samo negativne primjere proglaši pozitivnima: tada $TP = 0$ i $TP + FN$, preciznost je nula no odziv je nedefiniran. U ovakvim slučajevima, kada su ili preciznost ili odziv nedefinirani, uobičajeno je (premda to nije pravilo) postaviti ih na nulu.

1.3 Mjera F_1

Preciznost i odziv daju nam različitu informaciju. U praksi su ove dvije mjere često izravno suprotstavljene: ako klasifikator oblikujemo tako da ima visok odziv, onda je tipično da ćemo to platiti s nešto nižom preciznošću, i obrnuto, klasifikator s visokom preciznošću obično će imati niži odziv. Zbog toga, kada izvještavamo o rezultatu klasifikatora, svakako trebamo navesti vrijednosti obje mjere. Međutim, često nam je potrebno točnost klasifikatora (u širem smislu) karakterizirati samo jednim brojem. Jedna mjera vrednovanja koja radi upravo to jest **mjera F_1** (engl. *F_1 score*). Mjera F_1 definirana je kao **harmonijska sredina preciznosti i odziva**. Konkretno:

$$F = \frac{2}{\frac{1}{P} + \frac{1}{R}} = \frac{2PR}{P + R}$$

Vrijednost mjere F_1 je iz intervala $[0, 1]$, gdje više znači bolje. U našem prvom primjeru, gdje $P = 0.33$ i $R = 0.75$, imamo $F = \frac{2}{\frac{1}{0.33} + \frac{1}{0.75}} = 0.458$. U drugom primjeru, gdje $P = R = 0.5$, imamo $F = \frac{2}{\frac{1}{0.5} + \frac{1}{0.5}} = 0.5$. Iz ovoga vidimo da, ako su preciznost i odziv različiti, mjera F_1 bit će bliža od te dvije manjoj vrijednosti. Ako su pak preciznost i odziv jednaki, onda će i mjera F_1 biti njima jednaka.

Ljubopitljivi među vama zapitat će se zašto baš harmonijska sredina. Što fali običnoj aritmetičkoj sredini? Odgovor je da, zato što su preciznost i odziv različite stvari, aritmetička sredina ovdje nema smisla. Naime, pogledamo li definicije preciznosti i odziva, vidimo da je u obje mjere brojnik identičan (TP), međutim nazivnik je različit. To zapravo znači da su skale ovih dviju mjer različite. Da bi skale bile iste, nazivnik mora biti isti, a to će biti ako P i R zamijenimo sa $1/P$ i $1/R$. Upravo to radi harmonijska sredina: harmonijska sredina je recipročna vrijednost aritmetičke sredine recipročnih vrijednosti. Dakle, koristimo harmonijsku sredinu kako bismo preciznost i odziv kombinirali na istoj skali. Obična aritmetička sredina ovdje jednostavno nema smisla jer ne možemo usrednjavati kruške i jabuke.

Zanimljivo svojstvo harmonijske sredine jest da je “stroža” od aritmetičke sredine. Pritom mislimo da će, za različite P i R , harmonijska sredina biti manja od aritmetičke. Npr., za $P = 0.1$ i $R = 0.8$, mjera F_1 iznosi 0.178, dok bi aritmetička sredina dala 0.45. U ovakvoj

8

9

situaciji točnost od 0.45 doima se previsoka. Kod vrednovanja klasifikatora bolje je biti što stroži, i to upravo ostvarujemo mjerom F_1 .

Rubni slučajevi mjere F_1 su oni kada je $P = 0$ ili $R = 0$ ili kada su P ili R nedefinirani. Obično se tada vrijednost mjere F_1 postavlja na nulu. Npr., za klasifikator koji sve primjere označi kao negativne P je nedefiniran a R je nula, pa se F_1 postavlja na nulu.

Pogledajmo još jedan primjer.

► PRIMJER

Od $N = 1000$ primjera, njih 100 je pozitivno. Klasifikator primjere klasificira potpuno nasumično, s jednakom vjerojatnošću za obje oznake. (Takav klasifikator nije od pretjerane koristi, ali može poslužiti za usporedbu s drugim klasifikatorima; o tome više kasnije.) Izračunajmo **očekivane** vrijednosti za Acc i mjeru F_1 .

Budući da klasifikator primjere označava nasumično s jednakom vjerojatnošću za obje oznake, očekivano je da će njih 500 označiti kao pozitivne a njih 500 kao negativne. To znači da $TP + FP = FN + TN = 500$. Također znamo da je $TP + FN = 100$ i $FP + TN = 900$. Iz toga možemo zaključiti da **očekivana matrica zabune** izgleda ovako:

$$\begin{array}{ll} y_{true} = 1 & y_{true} = 0 \\ \begin{matrix} y_{pred} = 1 \\ y_{pred} = 0 \end{matrix} & \left(\begin{array}{cc} 50 & 450 \\ 50 & 450 \end{array} \right) \end{array}$$

Iz ovoga sada možemo izračunati očekivane vrijednosti mjera vrednovanja:

$$\begin{aligned} Acc &= \frac{TP + TN}{TP + TN + FP + FN} = \frac{50 + 450}{1000} = 0.5 \\ P &= \frac{TP}{TP + FP} = \frac{50}{500} = 0.1 \\ R &= \frac{TP}{FP + FN} = \frac{50}{100} = 0.5 \\ F_1 &= \frac{2PR}{P + R} = \frac{2 \cdot 0.1 \cdot 0.5}{0.1 + 0.5} = 0.167 \end{aligned}$$

Očekivana točnost nasumičnog klasifikatora s jednakom vjerojatnošću za obje oznake je $Acc = 0.5$ i jednaka je očekivanom odzivu $R = 0.5$. (Uvjerite se da bi to bilo tako neovisno o udjelu pozitivnih primjera u skupu primjera). Međutim, očekivana preciznost je samo $P = 0.1$, pa je očekivana vrijednost mjeru F_1 jednaka mizernih 0.167, što je znatno niže i od točnosti i odziva.

Na kraju primijetimo još da mjeru F_1 , računajući harmonijsku sredinu preciznosti i odziva, zapravo daje jednaku važnost i preciznosti i odzivu. Međutim, u nekim situacijama to nije ono što želimo. O tome smo već pričali kada smo govorili o asimetričnim funkcijama gubitka. Na primjer, ako nam je važnije imati manje lažno negativnih primjera nego lažno pozitivnih primjera (npr., kod dijagnostike malignih bolesti), onda to znači da nam je važnije imati velik odziv nego veliku preciznost. U takvim se situacijama koristi poopćenje mjeru F_1 , mjeru F_β , koja može dati veći naglasak na preciznost ili odziv, ovisno o parametru β . Zapravo, mjeru F_1 je poseban slučaj mjeru F_β gdje je $\beta = 1$, što znači da preciznost i odziv imaju jednaku važnost.

Sada znamo kako računati uobičajene mjeru za vrednovanje binarnog klasifikatora. Pogledajmo kako to proširiti na slučaj višeklasne klasifikacije.

2 Višeklasna klasifikacija

Kod binarne klasifikacije ($K = 2$), matrica zabune dimenzija je 2×2 . Kod višeklasne klasifikacije ($K > 2$), matrica zabune dimenzija je $K \times K$. Na primjer, za klasifikaciju $N = 13$ primjera u

$K = 3$ klase, gdje za oznake vrijedi $\mathcal{Y} = \{1, 2, 3\}$, matrica zabune mogla bi izgledati ovako:

$$\begin{array}{ccc} y_{true} = 1 & y_{true} = 2 & y_{true} = 3 \\ y_{pred} = 1 & \begin{pmatrix} 1 & 1 & 0 \\ 2 & 2 & 3 \\ 0 & 0 & 4 \end{pmatrix} \\ y_{pred} = 2 & & \\ y_{pred} = 3 & & \end{array}$$

Mjera vrednovanja koju odmah možemo izračunati iz ove matrice jest točnost (Acc). Točnost smo definirali kao udio točno klasificiranih primjera u ukupnom broju primjera, pa ćemo ju izračunati tako da zbroj elemenata na dijagonali matrice (trag matrice) podijelimo s ukupnim brojem primjera (zbrojem svih elemenata u matrici). U ovom našem primjeru, dobivamo $Acc = \frac{1+2+4}{13} = 0.538$.

2.1 Konfuzijska matrica 2×2 iz matrice $K \times K$

Ovo je jednostavno, no već znamo da je problem s točnošću taj da će točnost biti visoka za trivijalne klasifikatore koji sve primjere klasificiraju u većinsku klasu. Upravo smo zato uveli alternativne mjere, poput preciznosti, odziva i mjere F_1 . No, problem s tim mjerama jest da njih možemo primijeniti samo na binarne klasifikatore, odnosno samo na matrice zabune dimenzija 2×2 , gdje postoji jedna klasa koju smatramo pozitivnom i jedna klasa koju smatramo negativnom. Naime, samo kod takvih matrica možemo govoriti o stvarno pozitivnim (TP), lažno pozitivnim (FP), lažno negativnim (FN) i stvarno pozitivnim (TP) primjerima. Kod matrice $K \times K$ to nema smisla, jer što je pozitivno a što negativno ovisi o tome o kojoj kiasi pričamo.

Međutim, ono što ima smisla jest fiksirati jednu klasu i nju tretirati kao pozitivnu, pa onda napraviti podjelu primjera na stvarno pozitivne, lažno pozitivne, lažno negativne i stvarno negativne u odnosu na tu odabranu klasu. Ilustrirajmo to na gornjem primjeru za klasu s oznakom $y = 2$. Za tu klasu, stvarno pozitivni primjeri su oni primjeri za koje $y_{pred} = 2$ i $y_{true} = 2$, a to u gornjoj matrici odgovara elementu [2,2]. Lažno pozitivni primjeri su oni za koje $y_{pred} = 2$ i $y_{true} \neq 2$, a to su elementi [2,1] i [2,3] u gornjoj matrici. Slično, lažno negativni primjeri su oni za koje $y_{true} = 2$ i $y_{pred} \neq 2$, a to su elementi [1,2] i [3,2] u gornjoj matrici. Konačno, stvarno negativni primjeri su svi oni za koje $y_{pred} \neq 2$ i $y_{true} \neq 2$, a to su elementi [1,1], [1,3], [3,1] i [3,3] u gornjoj matrici. Zbrajanjem odgovarajućih elemenata matrice dobivamo da je, za klasu $y = 2$, broj stvarno pozitivnih primjera jednak 2, broj lažno pozitivnih primjera jednak $2 + 3 = 5$, broj lažno negativnih primjera jednak $1 + 0 = 1$ te broj stvarno negativnih primjera jednak $1 + 0 + 0 + 4 = 5$. Iz ovoga vidimo da, ako fiksiramo jednu klasu i primjere iz te klase promatramo kao pozitivne primjere, a primjere iz svih ostalih kasa kao negativne, dobivamo zapravo matricu zabune 2×2 za dotičnu klasu. Konkretno, za klasu $y = 2$, dobivamo matricu:

$$\begin{array}{cc} y_{true} = 2 & y_{true} \neq 2 \\ y_{pred} = 2 & \begin{pmatrix} 2 & 5 \\ 1 & 5 \end{pmatrix} \\ y_{pred} \neq 2 & \end{array}$$

Na isti način možemo izvesti matrice zabune za klase $y = 1$ i $y = 3$.

Ideja je, dakle, da matricu zabune višeklasnog klasifikatora dimenzija $K \times K$ dekomponiramo u K matrica zabune binarnog klasifikatora dimenzija 2×2 . Formalno, za matricu zabune C dimenzija $K \times K$ i klasu s oznakom $y = j$ definiramo matricu zabune binarne klasifikacije:

$$\begin{array}{cc} y_{true} = j & y_{true} \neq j \\ y_{pred} = j & \begin{pmatrix} TP_j & FP_j \\ FN_j & TN_j \end{pmatrix} \\ y_{pred} \neq j & \end{array}$$

gdje

- $TP_j = C[j, j]$ (j -ti element dijagonale matrice zabune)

- $\text{FP}_j = \sum_{i:i \neq j} C[j, i]$ (zbroj elemenata j -tog retka izvan dijagonale)
- $\text{FN}_j = \sum_{i:i \neq j} C[i, j]$ (zbroj elemenata j -tog stupca izvan dijagonale)
- $\text{TN}_j = N - \text{TP}_j - \text{FP}_j - \text{FN}_j$ (zbroj elemenata izvan retka j i stupca j)

Ako ste se u svemu ovome pogubili, sljedeći primjer mogao bi pomoći.

► PRIMJER

Skup od $N = 15$ primjera klasificiramo u $K = 3$ klase. Oznake klase neka su $\mathcal{Y} = \{1, 2, 3\}$. Predikcije klasifikatora i stvarne označke primjera su sljedeće:

$$\begin{aligned}\mathbf{y}_{pred} &= (2, 2, 2, 2, 2, 2, 1, 3, 1, 2, 2, 1, 3, 2, 2)^T \\ \mathbf{y}_{true} &= (1, 2, 2, 2, 3, 1, 1, 1, 2, 2, 3, 3, 3, 2, 2)^T\end{aligned}$$

Konstruirajmo najprije višeklasnu matricu zabune dimenzija 3×3 :

$$\begin{array}{ccc} y_{true} = 1 & y_{true} = 2 & y_{true} = 3 \\ y_{pred} = 1 & \begin{pmatrix} 1 & & 1 \\ 2 & & 2 \\ 1 & & 0 \end{pmatrix} & \\ y_{pred} = 2 & & \\ y_{pred} = 3 & & \end{array}$$

Iz ovoga sada na gore opisani način izvodimo tri matrice zabune za binarnu klasifikaciju, po jednu za svaku klasu:

$$\begin{pmatrix} \text{TP}_j & \text{FP}_j \\ \text{FN}_j & \text{TN}_j \end{pmatrix} \Rightarrow \begin{pmatrix} y = 1 \\ 1 & 2 \\ 3 & 9 \end{pmatrix} \quad \begin{pmatrix} y = 2 \\ 6 & 4 \\ 1 & 4 \end{pmatrix} \quad \begin{pmatrix} y = 3 \\ 1 & 1 \\ 3 & 10 \end{pmatrix}$$

2.2 Mikro- i makro-uprosječivanje

Sada znamo kako matricu zabune dimenzija $K \times K$ dekomponirati u K matrica zabune dimenzije 2×2 . No, što dalje? Imamo K matrica zabune, a ne samo jednu, pa nije odmah jasno kako primijeniti ranije opisane mjeru vrednovanja koje smo definirali nad samo jednom matricom zabune dimenzija 2×2 . Tu imamo dvije mogućnosti.

- Prva mogućnost je da izračunamo željenu mjeru vrednovanja zasebno na svakoj od K matrica zabune, i zatim jednostavno izračunamo prosjek. To je takozvano **makro-uprosječivanje**. Definirajmo to formalno. Neka je m neka odabrana mjeru vrednovanja (npr., P , R ili F_1). Onda **makro-mjeru** m (ili makro- m), što označavamo sa m^M , definiramo kao:

$$m^M = \frac{1}{K} \sum_j m_j$$

gdje je K ukupan broj klasa, a m_j je mjeru izračunata na matrici zabune 2×2 za klasu $y = j$.

- Druga mogućnost je da najprije pozbrajamo svih K binarnih matrica zabuna te da zatim na takvoj združenoj matrici zabune izračunamo željenu mjeru vrednovanja. To je takozvano **mikro-uprosječivanje**. Dakle, združena matrica zabune je:

$$\begin{pmatrix} \sum_j \text{TP}_j & \sum_j \text{FP}_j \\ \sum_j \text{FN}_j & \sum_j \text{TN}_j \end{pmatrix}$$

i nad takvom matricom računamo neku odabranu mjeru vrednovanja m , kao da se radi o običnoj binarnoj klasifikaciji. To je onda **mikro-mjera** m (ili mikro- m), što označavamo sa m^μ .

► PRIMJER

Nastavimo s prethodnim primjerom. Matrice zabune 2×2 za svaku od triju klasa bile su:

$$\begin{array}{lll} y = 1 & y = 2 & y = 3 \\ \begin{pmatrix} 1 & 2 \\ 3 & 9 \end{pmatrix} & \begin{pmatrix} 6 & 4 \\ 1 & 4 \end{pmatrix} & \begin{pmatrix} 1 & 1 \\ 3 & 10 \end{pmatrix} \end{array}$$

Pogledajmo prvo makro-mjere, i to makro-preciznost, makro-odziv i makro- F_1 . Najprije računamo ove mjere za svaku klasu zasebno:

$$\begin{array}{lll} P_1 = 0.333 & R_1 = 0.25 & F_{1,1} = 0.286 \\ P_2 = 0.6 & R_2 = 0.857 & F_{1,2} = 0.706 \\ P_3 = 0.5 & R_3 = 0.25 & F_{1,3} = 0.333 \end{array}$$

a zatim uprosječujemo kroz klase:

$$P^M = 0.478 \quad R^M = 0.452 \quad F_1^M = 0.442$$

Za izračun mikro-mjera, najprije izračunavamo združenu matricu zabune zbrajanjem triju matrica zabune za svaku klasu:

$$\begin{pmatrix} 1 & 2 \\ 3 & 9 \end{pmatrix} + \begin{pmatrix} 6 & 4 \\ 1 & 4 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 3 & 10 \end{pmatrix} = \begin{pmatrix} 8 & 7 \\ 7 & 23 \end{pmatrix}$$

Iz te matrice sada izravno računamo mikro-mjere:

$$P^\mu = 0.533 \quad R^\mu = 0.533 \quad F_1^\mu = 0.533$$

Primijetimo još i da je točnost izračunata izravno na matrici $K \times K$ jednaka $Acc = \frac{8}{15} = 0.533$.

U prethodnom smo primjeru dobili $P^\mu = R^\mu = F_1^\mu = Acc$. Lako je pokazati da ova jednakost uvijek vrijedi. Zbog toga nema smisla govoriti posebno o mikro-preciznosti ili mikro-odzivu; umjesto toga, uobičajeno se govoriti ili o točnosti ili o mikro- F_1 (ovo drugo pogotovo ako se uspoređuje s makro- F_1).

Sažmimo što smo do sada opisali. Kod višeklasne klasifikacije u K klasa, matrica zabune dimenzija je $K \times K$. Nad tom matricom možemo izravno izračunati mjeru točnosti. Alternativa su mjere dobivene mikro- ili makro-uprosječivanjem, odnosno **makro-preciznost, makro-odziv, makro- F_1 i mikro- F_1** , gdje je ova potonja zapravo jednaka **točnosti**. Izbora, dakle, ne nedostaje, pa se postavlja pitanje koju mjeru vrednovanja koristiti. Sve se ove mjere koriste u praksi, i koju ćemo mjeru koristiti ovisi o tome na što želimo staviti naglasak. Ako nam je važno da naš klasifikator ne producira mnogo lažno pozitivnih primjera (npr., kod klasifikacije neželjene pošte), koristit ćemo mjeru preciznosti. Ako nam je pak važno da klasifikator ne producira mnogo lažno negativnih primjera, odnosno da identificira sve primjere neke klase (npr., kod dijagnostike malignih bolesti), onda ćemo koristiti mjeru odziva. Ako nam je oboje jednako važno, koristit ćemo mjeru F_1 . Nadalje, budući da makro-mjere pri uprosječivanju daju jednaku težinu svim klasama, makro-mjere ćemo koristiti upravo u situaciji kada sve klase želimo vrednovati jednakom neovisno o broju primjera koje sadrže. Suprotno tomu, kod izračuna mikro-mjere klase s većim brojem primjera imat će veći utjecaj na vrijednost mjeru, pa ćemo te mjeru koristiti ako klasifikaciju svakog primjera želimo vrednovati jednakom, neovisno o tome iz koje klase primjer dolazi. Općenito, zato što klasifikatori na manjim klasama ubičajeno rade lošije, možemo očekivati da će makro-mjere imati manju vrijednost od mikro-mjera, dakle u praksi je uobičajeno da vrijedi $F_1^M < F_1^\mu$. Konačno, primijetimo da, umjesto izračuna mikro- ili makro-prosjeka, čime se neminovno gubi dio informacije, ponekad ima više smisla izvijestiti o točnosti, preciznosti, odzivu ili mjeri F_1 svake klase pojedinačno.

13

14

15

3 Vrednovanje klasifikatora s pragom

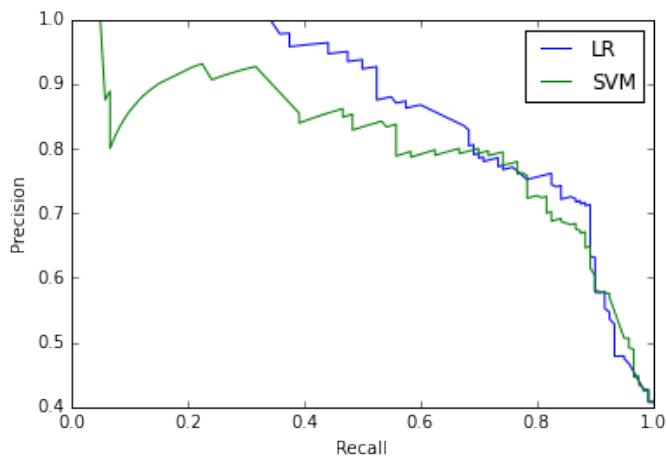
Do sada smo se bavili vrednovanjem klasifikatora koji na izlazu daje oznaku klase. Međutim, mnogi klasifikatori (npr., logistička regresija, naivan Bayesov klasifikator) na izlazu daju **vjerojatnost klasifikacije** primjera u neku klasu. Čak i onda kada to nije slučaj (npr., SVM), vjerojatnost klasifikacije može se dobiti nekom metodom kalibracije klasifikatora (npr., Plattova metoda, o kojoj smo pričali u 9. predavanju).

Standardno, klasifikacijski prag je 0.5, tj. primjer \mathbf{x} klasificira se u pozitivnu klasu $y = 1$ ako je $P(y = 1|\mathbf{x}) \geq 0.5$. Međutim, u nekim je situacijama možda bolje prag postaviti na vrijednost višu ili nižu od 0.5. Što ćemo dobiti ako prag postavimo vrijednost višu od 0.5, npr. na 0.8? U tom slučaju, samo primjere za koje je vrlo vjerojatno da su pozitivni bit će klasificirani u $y = 1$, u protivnom će biti klasificirani u klasu $y = 0$. Kakav je utjecaj toga na klasifikacijske pogreške? Ako klasifikator samo primjere za koje je vrlo siguran klasificira u klasu $y = 1$, onda će producirati manje lažno pozitivnih primjera, a to znači da će preciznost porasti. Suprotno, ako prag smanjimo na vrijednost manju od 0.5, klasifikator će u klasu $y = 1$ klasificirati i primjere s manjom vjerojatnošću pripadanja toj klasi, što znači da potencijalno smanjujemo broj lažno negativnih primjera, odnosno povećavamo odziv. Dakle, ugadajem klasifikacijskog praga izravno utječemo na preciznost i odziv klasifikatora.

16

3.1 Krivulja preciznost-odziv

Ako želimo vrednovati klasifikator s obzirom na sve moguće vrijednosti klasifikacijskog praga, možemo skicirati **krivulju preciznost-odziv** (engl. *precision-recall curve*, *PR curve*). Krivulju dobivamo tako da, za već trenirani model, vrijednost praga postepeno smanjujemo od 1 do 0 (u nekim fiksnim koracima) te za svaku vrijednost računamo odziv i preciznost klasifikatora. Zatim skiciramo dobivenu krivulju, s odzivom na x -osi i preciznošću na y -osi. Na primjer:



Na slici su prikazane dvije krivulje preciznost-odziv, jedna (plava) za model logističke regresije i druga (zelena) za model SVM. Obje su dobivene na skupu podataka Titanic. Kako smanjujemo klasifikacijski prag, tako odziv raste, ali preciznost pada, pa krivulja preciznost-odziv pada kako se pomičemo udesno (premda može biti kratkotrajnih porasta). Kod potpunog odziva ($R = 1$), preciznost mora pasti (nagradno pitanje: na koju vrijednost?), no želimo da se taj pad dogodi što kasnije, tj. na što višim razinama odziva. Drugim riječima, preferiramo krivulje koje su što bliže točki $(P = 1, R = 1)$. Tako je na gornjoj slici krivulja logističke regresije bolja je od krivulje SVM-a, pa možemo zaključiti da logistička regresija na ovom skupu podataka radi bolje od SVM-a (iznenadujuće, ali događa se).

Zamislite da imamo dva klasifikatora sa svojim krivuljama preciznost-odziv. Q: Kako bi izgledala krivulja prvog klasifikatora, ako je on definitivno bolji od drugog klasifikatora? A:

Bila bi uvijek iznad krivulje drugog klasifikatora. Naime, ako je krivulja klasifikatora h_1 uvijek iznad krivulje klasifikatora h_2 (znači P klasifikatora h_1 je veća od P klasifikatora h_2 za svaku razinu odziva R), onda stvarno nema razloga da ikada upotrijebimo klasifikator h_2 jer on uvijek ima lošiji odziv ili lošiju preciznost (ili oboje) od klasifikatora h_1 . Međutim, ako se krivulje preciznost-odziv križaju, onda to znači da niti jedan klasifikator nije bolji od drugoga. Na gornjoj slici imamo upravo takvu situaciju: preciznost logističke regresije nije na svim razinama odziva viša od preciznosti SVM-a (npr., za $R = 0.7$ i $R = 0.95$ preciznost SVM-a je nešto viša). U ovakvim slučajevima odabrat ćemo klasifikator koji je najbolji za željenu razinu odziva, ili možemo koristiti različite klasifikatore za različite razine odziva. Iz ovoga možemo zaključiti da krivulja preciznost-odziv zapravo definira parcijalni uređaj između klasifikatora po kriteriju preciznosti i odziva (neki su klasifikatori bolji od nekih drugih, dok neki nisu međusobno usporedivi).

Premda je krivulja preciznost-odziv vrlo informativna, nekad je točnost klasifikatora potrebno kvantificirati jednim brojem. To možemo načiniti tako da izračunamo **prosječnu preciznost** (engl. *average precision*). Prosječna preciznost zapravo je integral krivulje preciznost-odziv. Što je krivulja preciznost-odziv bliža točki ($P = 1, R = 1$), to će prosječna preciznost biti veća.

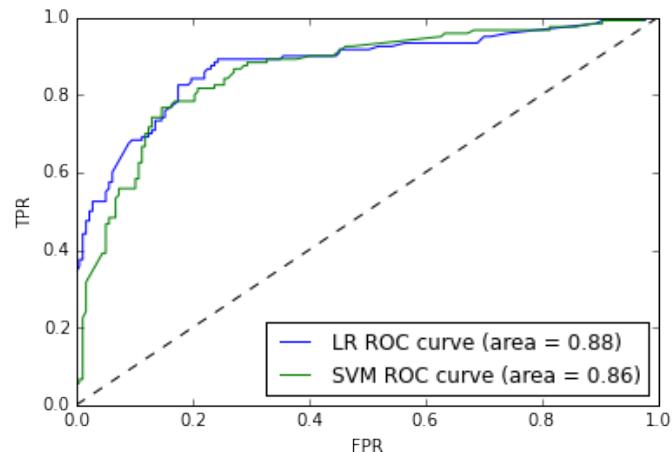
3.2 Krivulja ROC

Druga mjera vrednovanja koja uzima u obzir varijabilnost praga i koja se često koristi jest **površina pod krivuljom ROC** (engl. *area under ROC curve*), skraćeno **AUC**. No, pogledajmo najprije što je to krivulja ROC. Krivulja ROC (naziv dolazi od *receiver operating characteristics*, što nema previše veze s primjenom u strojnog učenju) je vrijednost **stope stvarnih pozitiva** (TPR), što je isto kao i **odziv**, kao funkcije **stope lažnog alarmu** (FPR), odnosno **ispadanja** (engl. *fall-out*). Prisjetimo se, FPR je:

$$FPR = \frac{FP}{FP + TN}$$

FPR mjeri koliki je udio primjera koje je klasifikator pogrešno proglašio pozitivnima. Idealno, $FPR = 0$. Mjera TPR, odnosno odziv, mjeri koliko je pozitivnih primjera klasifikator detektirao, i idealno je $TPR = 1$.

Kako smanjujemo prag, tako će FPR rasti, jer će klasifikator sve više primjera proglašavati pozitivnima, pa ćemo imati i sve više lažno pozitivnih primjera (FP). No, istovremeno će sa smanjivanjem praga rasti i odziv, jer će klasifikator detektirati sve više stvarno pozitivnih primjera (TP). Idealno bi bilo da odziv raste vrlo strmo, tako da FPR možemo zadržati na nekoj niskoj vrijednosti. Dakle, kako izgleda krivulja ROC? Ovako:



Ovo je krivulja ROC za logističku regresiju i SVM na istim podatcima kao i ranije (Titanic). Klasifikator je to bolji što njegova krivulja ROC prolazi bliže točki ($FPR = 0, TPR = 1$). I

ovdje vidimo da je logistička regresija na ovom skupu podataka bolja od SVM-a, osim kod nekih razina odziva (istih onih kao i za krivulju preciznost-odziv).

Kriva ROC zapravo izgleda vrlo slično krivulji preciznost-odziv (uzevši u obzir zrcaljenje), pa se možda pitate zašto nam trebaju i jedna i druga. Trebaju nam obje jer svaka ima svoje prednosti. Prednost krivulje preciznost-odziv je što je lako razumljiva: mnoga područja primjene strojnog učenja koriste preciznost i odziv kao standardne mjere vrednovanja. No, velika prednost krivulje ROC jest što ona za **nasumični klasifikator** (klasifikator koji primjere klasificira u nasumično odabранu klase) odgovara pravcu od $(0, 0)$ do $(1, 1)$. Ono što je posebno dobro jest da to vrijedi neovisno o tome je li broj pozitivnih i negativnih primjera uravnotežen. Zašto? (Meni to nije bilo odmah jasno, vama možda je.) Ako klasifikator slučajno odabere $k\%$ primjera i proglaši ih pozitivnima, onda će u tom skupu imati $k\%$ od ukupno pozitivnih primjera, dakle odziv (odnosno TPR) je $k\%$. Slično, u tom odabranom skupu bit će $k\%$ od ukupnog broja negativnih primjera, a te je primjere klasifikator proglašio pozitivnima, dakle FPR je također $k\%$. Drugim riječima, $\text{TRP} = \text{FPR}$, i imamo pravac, neovisno o distribuciji primjera u pozitivnu i negativnu klasu! Zašto nam je to korisno? Zato što je nasumični klasifikator dobra referentna točka za vrednovanje klasifikatora: ako naš klasifikator daje predikcije koje su tek neznatno bolje od nasumičnog klasifikatora (krivulja ROC je blizu pravca $\text{TRP} = \text{FPR}$) ili nedajbože radi lošije od nasumičnog klasifikatora (krivulja ROC je ispod tog pravca), onda je to jedan beskoristan klasifikator.

Mjera AUC jednostavno je definirana kao površina ispod krivulje ROC. AUC je u intervalu $[0, 1]$, što više, to bolje. Za nasumični klasifikator, $\text{AUC} = 0.5$.

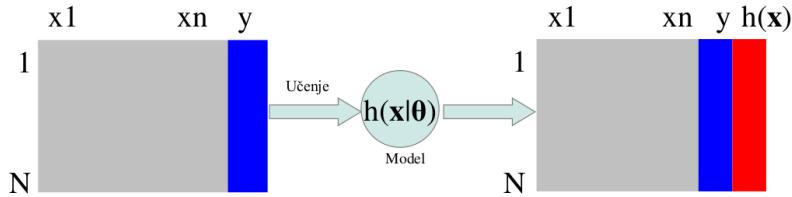
19

4 Procjena pogreške modela

Do sada smo govorili o tome kako izmjeriti točnost (odnosno pogrešku) klasifikatora koristeći razne mjere vrednovanja izračunate nad matricom zabune. Sve mjere vrednovanja izračunavaju se na skupu primjera koji imamo. U statističkome smislu, to je jedan **slučajan uzorak**. Dakle, naše mjere su funkcije slučajnog uzorka, što ih i samima čini slučajnim varijablama. Kada primjenimo mjeru na uzorak primjera, ono što dobivamo zapravo je empirijska **procjena** te mjeru. U tom smislu govorimo o **procjeni pogreške** (engl. *error estimation*) modela (ovdje “pogreška” može biti bilo koja mjeru vrednovanja, ali se uobičajeno govori baš o pogrešci).

Postavlja se pitanje kako napraviti **dobru** procjenu pogreške modela. Dobra procjena znači da je procjenitelj **nepristrand** (njegovo očekivanje jednak je pravoj vrijednosti populacije) i da je **konzistentan** (varijanca procjene se smanjuje kako veličina uzorka raste). To su statističke kvalitete procjenitelja koje bismo voljeli imati. Pored ovih kvaliteta, u strojnom učenju nam je bitno da je procjena pogreške **poštena**. Pod time mislimo da želimo mjeriti sposobnost generalizacije modela, dakle pogrešku modela na još neviđenim podatcima (uz standardnu pretpostavku da se ti podatci pokoravaju istoj distribuciji kao i podatci nad kojima smo učili model, tj. da je skup primjera za učenje reprezentativan uzorak problema koji rješavamo). U praksi to također znači da ćemo preferirati stvari postaviti tako da je naša procjena pogreške **pesimistična** prije nego optimistična. Naime, ako je procjena pogreške pesimistična, znamo da će u stvarnosti klasifikator raditi tako ili čak još bolje.

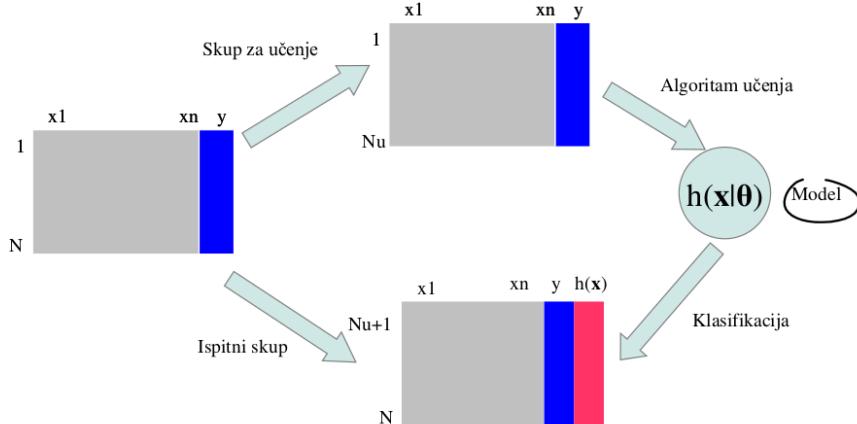
U statistici (i strojnom učenju) razvijen je niz postupaka za procjenu pogreške. Krenimo od loših (nepoštenih) procjena. Definitivno nepoštена procjena pogreške (odnosno točnosti, preciznosti, odziva, mjeru F_1 , AUC itd.) jest procjena na istom skupu primjera na kojem je klasifikator učen. Grafički to možemo prikazati ovako:



Ovdje smo dakle svih N primjera iskoristili za učenje modela, dobili smo naučeni model h , i zatim taj model primijenili na istih tih N primjera, usporedili oznaće \mathbf{y}_{pred} koje dobivamo modelom h sa stvarnim označama \mathbf{y}_{true} , izveli matricu zabune te na temelju nje izračunali neku mjeru vrednovanja. Ovo je naravno loše jer ne mjerimo pogrešku generalizacije već pogrešku učenja, koja je tipično manja od pogreške generalizacije te opada sa složenošću modela (to već znamo).

4.1 Metoda izdvajanja

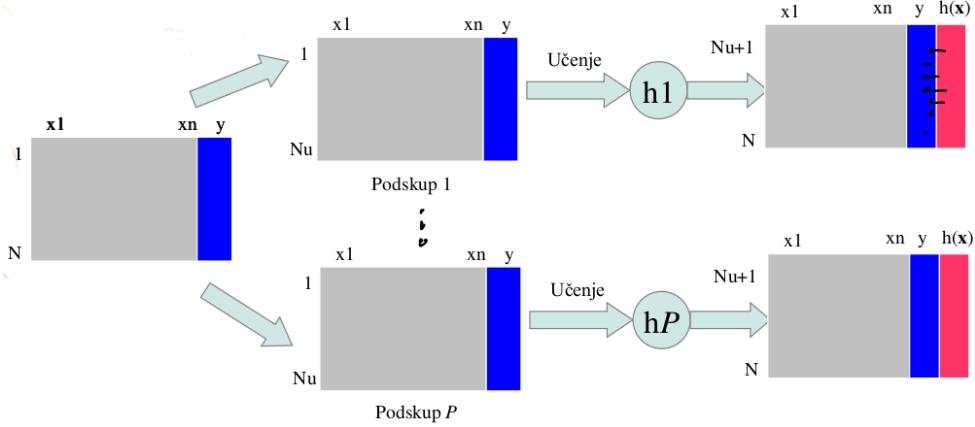
Alternativa je **metoda izdvajanja** (engl. *holdout method*), gdje skup primjera razdvajamo na **skup za učenje** (engl. *training set*) i **ispitni skup** (engl. *test set*):



Ovo je zapravo najjednostavnija varijanta nečega što smo već zvali **unakrsna provjera**. Prednost ove metode jest da doista procjenjujemo pogrešku generalizacije, a ne pogrešku učenja.

Međutim, nedostatak ovog pristupa je da doslovno bacamo primjere za učenje: budući da smo dio primjera morali ostaviti postrani za ispitivanje, imamo manje primjera za učenje, a to je uvijek loše jer gubimo vrijednu informaciju potrebnu za poboljšanje modela. To je pogotovo problematično ako je primjera i inače malo (npr., ako ukupno imamo 100-tinjak primjera). Drugi problem je loša točnost procjene: naša procjena pogreške temelji se na samo jednom uzorku, što znači da će procjena vrlo varirati ovisno o uzorku, odnosno o načinu kako smo primjere podijelili u dva skupa. Točnost procjene će doduše rasti što je ispitni skup veći, ali problem je što je skup primjera uvijek ograničen.

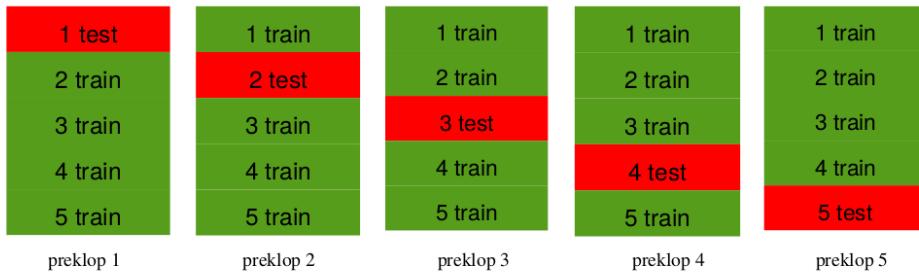
Oba nedostatka moguće je riješiti postupcima temeljenima na **ponovnom uzorkovanju** (engl. *resampling*). Ti postupci imaju dugu tradiciju u statistici. U strojnog učenju ti nam postupci omogućavaju da model učimo na većini raspoloživih primjera (pa dobivamo bolji model), a da istovremeno dobijemo i bolju procjenu pogreške. Najjednostavnija takva metoda je **ponovljeno izdvajanje** (engl. *repeated holdout*):



Ovdje jednostavno podjelu na skup za učenje i skup za ispitivanje radimo više (ukupno P) puta, svaki puta nanovo trenirano model na skupu za učenje i ispitujemo ga na skupu za ispitivanje. Procjena pogreške (ili koje već mijere) je srednja vrijednost pogreške na svim ispitnim skupovima.

4.2 k -struka unakrsna provjera

Ovo je lijepo, i čini se da rješava naš problem (sve primjere iskorištavamo, procjena je bolja jer je računamo kao srednju vrijednost više procjena), međutim problem je da nemamo nikakvu kontrolu koji su primjeri koliko puta upotrijebljeni. Metoda koja zadržava prednosti ponovljenog izdvajanja, a dodatno nam daje tu kontrolu, jest metoda **k -strukte unakrsne provjere** (engl. *k -folded cross validation*):



Skup primjera dijelimo u k disjunktnih podskupova (tj. particija ili "preklopa"). Zatim učimo klasifikator na $(k - 1)$ preklopu, a ispitujemo ga na k -tom preklopu, pa sve to ponavljamo ukupno k -puta, svaki puta koristeći drugi preklop kao ispitni skup. Procjenu pogreške (ili koje god mijere) izračunavamo kao srednju vrijednost pogreške na k preklopa. Tipično uzimamo $k = 10$ ili $k = 5$.

Ponekad se može dogoditi da je podjela skupa primjera na skup za učenje i skup za ispitivanje takva da ne zrcali pravu razdiobu primjera u skupu. To može rezultirati pretjerano pesimističnom procjenom. (Q: Zašto? A: Zato što to narušava našu pretpostavku da skup za učenje i skup za ispitivanje dolaze iz iste distribucije.) Rješenje u takvim situacijama jest da se skupovi **stratificiraju**, odnosno da se pobrinemo da razdioba klase bude sačuvana u oba skupa. To možemo jednostavno ostvariti na sljedeći način:

1. Skup primjera podijelimo u K podskupova, po jedan za svaku klasu;
2. Svaki takav podskup podijelimo u k preklopa;
3. Združimo K preklopa (po jedan od svake klase) u jedan preklop, i ponovimo to k puta.

Što su prednosti, a što nedostatci k -strukte unakrsne provjere? Prednost jest da je svaki primjer bio iskorišten za ispitivanje, i to točno jednom. Također, svaki je primjer bio iskorišten i za učenje (i to $k - 1$ puta). Prednosti su također da ju je jednostavno provesti (i implementirana je u mnogim standardnim alatima) te da nije računalno suviše zahtjevna (osim ako k nije prevelik). Naravno, prednost je i što dobivamo točniju procjenu pogreške jer su ispitni skupovi nepreklapajući. Nedostaci, međutim, su to što k klasifikatora nije međusobno nezavisno. Naime, svaka dva klasifikatora dijele $k - 2$ preklopa, tj. $(k - 2)/k$ skupa za učenje. To, pogotovo ako je k visok, dovodi do visoke varijance procjene pogreške (jer postoje korelacije između k vrijednosti pogrešaka).

Ekstremni slučaj k -strukte unakrsne provjere jest kada je broj preklopa jednak broju primjera, $k = N$. To znači da u svakoj od N iteracija unakrsne provjere klasifikator učimo na svim primjerima osim na jednom te klasifikator zatim ispitujemo na tom jednom primjeru. To se zove **unakrsna provjera "izdvoji jednog"** (engl. *Leave-One-Out Cross Validation, LOOCV*). Očita prednost metode LOOCV jest da iskorištavamo gotovo potpun skup primjera i dobivamo bolju procjenu pogreške. Nedostatak je da LOOCV može biti računalno vrlo zahtjevan, pogotovo za veliki N . Drugi problem je opet visoka varijanca procjene pogreške, budući da su klasifikatori učeni na skoro istim skupovima, pa dakle nisu više nezavisni. Treći problem jest da, kada ispitujemo na samo jednom primjeru, ne možemo izračunati sve one mjere koje smo uveli (jer su one definirane nad matricom zabune koju dobivamo iz skupa primjera), već samo pogrešku ili točnost na jednom primjeru. U načelu, LOOCV se prepruča koristiti kada je skup podataka srednje veličine.

Da sažmemo: za procjenu pogreške **najbolje je koristiti k -struku unakrsnu provjeru**. Ta metoda nije bez nedostataka, ali je među najboljima koje imamo. Ako je skup podataka vrlo velik, dovoljno je koristiti običnu unakrsnu provjeru (dakle, s $k = 1$). Ako je skup podataka srednje veličine, onda možemo koristiti LOOCV. Što je veliko, a što malo ovisi o problemu (za teži problem trebam nam više primjera za učenje, pa preferiramo manji k) i željenoj pouzdanosti procjene (željenom intervalu pouzdanosti procjene pogreške; o tome više drugi put).

4.3 Unakrsna provjera uz odabir modela

Do sada smo govorili o procjeni pogreške modela čije hiperparametre ne trebamo optimirati, dakle situaciji kada imamo jedan već odabrani modela i želimo vrednovati koliko on dobro generalizira. Međutim, stvarnost je rijetko tako idealna. U stvarnosti ćemo, pored treniranja i procjene pogreške modela trebati prije toga napraviti **odabir modela**, tj. optimizaciju hiperparametara iliti ugađanje složenosti modela. Do sada smo već usvojili ideju da, kod unakrsne provjere, skup primjera trebamo podijeliti na skup za učenje $\mathcal{D}_{\text{train}}$ i skup za ispitivanje $\mathcal{D}_{\text{test}}$, koji su međusobno disjunktni. No, ako trebamo napraviti odabir modela, onda unakrsnu provjeru moramo raditi nad tri međusobno disjunktna skupa:

1. $\mathcal{D}_{\text{train}} - \text{skup za učenje}$ (engl. *train(ing) set*), na kojem treniramo model;
2. $\mathcal{D}_{\text{val}} - \text{skup za provjeru (validaciju)}$ (engl. *validation set*), na kojem procjenjujemo pogrešku generalizacije modela kod optimizacije hiperparametara;
3. $\mathcal{D}_{\text{test}} - \text{ispitni skup}$ (engl. *test set*), na kojem procjenjujemo pogrešku generalizacije modela s optimiziranim hiperparametrima (dakle, modela optimalne složenosti).

pri čemu su ovi skupovi međusobno disjunktni, $\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{val}} = \mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \mathcal{D}_{\text{val}} \cap \mathcal{D}_{\text{test}} = \emptyset$.

Stvar funkcioniра ovako: iz neke familije modela odaberemo jedan model \mathcal{H} te ga treniramo na skupu $\mathcal{D}_{\text{train}}$. Nakon što smo naučili model, procjenjujemo njegovu pogrešku generalizacije na skupu \mathcal{D}_{val} . Zatim odaberemo drugi model iz familije modela, te ponavljamo to isto. Zatim odaberemo neki treći model, te ponavljamo isto. I tako dalje, ponavljamo treniranje na $\mathcal{D}_{\text{train}}$ i ispitivanje na \mathcal{D}_{val} sve dok ne pronađemo optimalan model \mathcal{H}^* (optimalne hiperparametre)

na skupu D_{val} . Nakon što smo pronašli optimalan model \mathcal{H}^* , taj model treniramo na skupu $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}}$. Koristimo oba skupa kako bismo iskoristili što više podataka. Na kraju procjenimo pogrešku generalizacije tako naučenog modela \mathcal{H}^* na ispitnome skupu $\mathcal{D}_{\text{test}}$, koji do sada uopće nismo koristili. Tako procjenjena ispitna pogreška je ono što objavljujemo. To je pogreška generalizacije modela optimalne složenosti na našem skupu primjera. Znamo da je model optimalne složenosti jer smo proveli optimizaciju hiperparametara. Također, znamo da je to pogreška generalizacije jer je dobivena na skupu primjera koji nije korišten niti za treniranje niti za odabir modela.

Zastanite ovdje na trenutak i zapitajte se razumijete li zašto smo uveli treći skup primjera (\mathcal{D}_{val}). Možda vam se čini da nam skup \mathcal{D}_{val} uopće ne treba jer smo optimizaciju hiperparametara mogli napraviti na ispitnom skupu $\mathcal{D}_{\text{test}}$, i na tom istom skupu onda u konačnici procjenjiti pogrešku optimalnog modela. No, razmislite što bi se dogodilo da za optimizaciju hiperparametara modela koristimo isti skup na kojem u konačnici procjenjujemo pogrešku generalizacije modela. Dobili bismo nepoštenu (pretjerano optimističnu) procjenu pogreške, jer bismo složenost modela namijestili upravo prema ispitnom skupu. Zbog toga skup primjera na kojem procjenjujemo pogrešku modela mora biti potpuno netaknut. To znači da ne smije biti upotrijebljen ni za kakve optimizacije: niti za optimizaciju parametara modela (treniranje modela) niti za optimizaciju hiperparametara modela (odabir modela optimalne složenosti). Također pogrešno bi bilo optimizaciju hiperparametara provesti na skupu za učenje $\mathcal{D}_{\text{train}}$. To je očito loša ideja jer na taj način odabir modela ne bismo radili prema pogrešci generalizacije nego pogrešci učenja, i sigurno bismo dobili presložen model. Dakle, kako god okrenemo, treba nam treći, validacijski skup, \mathcal{D}_{val} .

No, što ako želimo napraviti k -struku unakrsnu provjeru *zajedno* s odabirom modela? To je potpuno realističan scenarij: često trebamo napraviti i jedno i drugo. Tu se situacija onda dodatno komplificira. Pogledajmo to iduće.

4.4 Ugniježđena k -struka unakrsna provjera

Često želimo raditi i unakrsnu provjeru i odabir modela. To znači da trebamo prolaziti kroz k preklopa, ali isto tako moramo raditi na tri skupa podataka. U tom slučaju radit ćemo **ugniježđenu unakrsnu provjeru** (engl. *nested cross validation*). Ta metoda ima dvije ugniježđene petlje: **vanjsku petlju** za treniranje i ispitivanje modela (kao i ranije) te **unutarnju petlju** za odabir modela (za treniranje i provjeru). Postupak možemo opisati sljedećim pseudokodom:

► **Ugniježđena unakrsna provjera** $k \times l$

- 1: podijeli \mathcal{D} na vanjske preklope \mathcal{D}_i , $i = 1, \dots, k$
- 2: **za** $i = 1, \dots, k$ **radi:** *vanjska petlja*
 - 3: $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D} \setminus \mathcal{D}_i$, $\mathcal{D}_{\text{test}} \leftarrow \mathcal{D}_i$
 - 4: **za svaku** odabranu vrijednost hiperparametra α **radi:**
 - 5: podijeli $\mathcal{D}_{\text{train}}$ na unutarnje preklope \mathcal{D}_j , $j = 1, \dots, l$
 - 6: **za** $j = 1, \dots, l$ **radi:** *unutarnja petlja*
 - 7: $\mathcal{D}_{\text{train}'} \leftarrow \mathcal{D}_{\text{train}} \setminus \mathcal{D}_j$, $\mathcal{D}_{\text{val}} \leftarrow \mathcal{D}_j$
 - 8: treniraj model na $\mathcal{D}_{\text{train}'}$ i procjeni pogrešku na \mathcal{D}_{val}
 - 9: izračunaj prosjek mjere na l unutarnjih preklopa
 - 10: odaberi hiperparametar α koji minimizira procjenu pogreške
 - 11: nauči odabrani model na $\mathcal{D}_{\text{train}}$ i procjeni pogrešku na $\mathcal{D}_{\text{test}}$
 - 12: izračunaj prosjek mjere na k vanjskih preklopa

Pseudokod implementira dvije petlje: vanjska započinje na liniji 2 i odvija se k puta, a unutarnja petlja započinje na liniji 6 i odvija se l puta. To zovemo ugniježđenom unakrsnom

provjerom $k \times l$. Odabir vrijednosti za k i l je proizvoljan, a tipično se uzima 5×5 , 10×5 , 10×10 ili slično. Na početku (linija 1) skup podataka dijelimo na k vanjskih preklopa. U svakoj iteraciji vanjske petlje, $k - 1$ vanjskih preklopa uzimamo kao skup za učenje $\mathcal{D}_{\text{train}}$, a 1 preklop kao skup za testiranje $\mathcal{D}_{\text{test}}$. Na primjer, ako je $k = 5$, onda ćemo $4/5$ skupa podataka koristiti za treniranje, a $1/5$ za ispitivanje. Kroz iteracije ćemo se pomicati po preklopima: prvo ćemo model ispitivati na prvoj petini skupa podataka, zatim na drugoj petini, itd. U liniji 4 iteriramo kroz hiperparametre modela. Ako parametra nema mnogo, to se može ostvariti iscrpnim pretragom (npr., kroz rešetku $C \times \gamma$ kod SVM-a). Međutim, ako je parametra mnogo, ovdje nam treba neki pametniji, moguće heuristički način pretraživanja parametara. Nakon što smo odabrali jednu vrijednost hiperparametara α , u liniji 5 skup za učenje $\mathcal{D}_{\text{train}}$ dalje dijelimo na l unutarnjih preklopa. Od toga ćemo $l - 1$ preklop koristiti za učenje modela u unutarnjoj petlji, a 1 preklop za provjeru modela. Na primjer, ako je $l = 10$, onda ćemo uzeti 9 unutarnjih preklopa za učenje (skup $\mathcal{D}_{\text{train}'}$) te 1 unutarnji preklop za provjeru (skup \mathcal{D}_{val}). Primijetite da, budući da ovo sve radimo na $4/5$ skupa, zapravo ćemo model trenirati na $4/5 \cdot 9/10$ cjelokupnog skupa primjera, a provjeravati na $4/5 \cdot 1/10$ cjelokupnog skupa primjera. U liniji 6 iteriramo kroz sve unutarnje preklope, rotirajući skup za provjeru l puta. U liniji 8 model učimo na skupu za učenje $\mathcal{D}_{\text{train}'}$, koji je veličine $\frac{k-1}{k} \cdot \frac{l-1}{l}$, a ispitujemo na skupu za provjeru, koji je veličine $\frac{k-1}{k} \cdot \frac{1}{l}$. Nakon što smo to izvrtili l puta, imat ćemo l procjena mjere vrednovanja na \mathcal{D}_{val} . U liniji 9 zatim računamo prosjek te mjere. Time smo dobili procjenu pogreške generalizacije (ili koje god mjere) za model s hiperparametrima α . Sada sve ovo ponavljamo za neki drugi odabir vrijednosti hiperparametra α , dakle iteriramo opet od linije 4, sve dok ne ispitamo sve hiperparametre koje smo željeli ispitati. Nakon toga, u liniji 10, raspolažemo procjenama pogrešaka za sve modele koje smo isprobali, te odabiremo onaj model (one hiperparametre α) koji minimizira procjenu pogreške (ili maksimiziraj neku drugu mjeru vrednovanja). Time smo završili s odabirom modela. Sada taj model učimo na skupu $\mathcal{D}_{\text{train}}$ iz vanjske petlje (skupu koji smo imali prije podjele na unutarnje preklope) i njegovu pogrešku procijenjujemo na $\mathcal{D}_{\text{test}}$. Sve ovo ćemo ponoviti k puta, počevši od linije 1. Kada završimo, na liniji 12, imat ćemo k procjena pogreške za optimalne modele, gdje su optimalni modeli odabrani na temelju l procjena pogreške u unutarnjoj petlji. U konačnici nam ostaje samo izračunati srednju vrijednost k procjena pogrešaka u vanjskoj petlji.

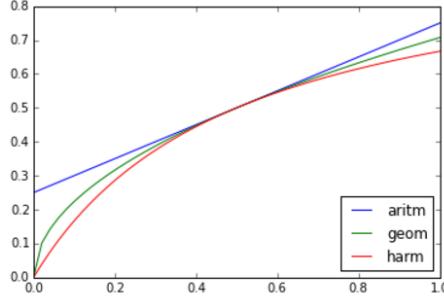
22

Sažetak

- **Matrica zabune** sažeto prikazuje broj ispravnih i neispravnih klasifikacija
- **Točnost** je najjednostavnija mjera vrednovanja klasifikatora, ali je zavaravajuća ako su klase neuravnotežene
- Pored točnosti, nad binarnom matricom zabune možemo definirati niz mjera vrednovanja, uključivo **preciznost** i **odziv**, koje bolje vrednuju klasifikator kod neuravnoteženih klasa
- **Mjera F1** kombinira preciznost i odziv pomoću njihove harmonijske sredine
- Višeklasne klasifikatore tipično vrednujemo pomoću **makro-preciznosti**, **makro-odziva**, **makro-F1** te **točnosti**
- Klasifikatore s pragom vrednujemo pomoću **krivulje preciznost-odziv** i **ROC krivulje**
- Procjenu generalizacijske pogreške/točnosti klasifikatora treba se napraviti na **odvojenom skupu podataka**
- **Unakrsna provjera uz odabir modela** skup podataka dijeli na tri podskupa: skup za učenje, skup za provjeru i skup za ispitivanje
- **k -struka unakrsna provjera** daje bolju procjenu pogreške od metode izdvajanja, a **k -struka ugniježđena unakrsna provjera** kombinira procjenu pogreške i odabir modela

Bilješke

- 1 Lijep pregled problematike vrednovanja algoritama strojnog učenja možete naći u ([Raschka, 2018](#)) (zahvaljujem Filipu Karlu Došiloviću na ovoj informaciji). Nešto starija, no mnogo detaljnija referenca je ([Japkowicz and Shah, 2011](#)). Očekivano, oba ova izvora idu u mnogo više detalja nego što ćemo mi obuhvatiti ovim predavanjem, pa se zainteresirani pojedinci za produbljivanje znanja upućuju na dotične izvore.
- 2 Vrednovanje grupiranja malo smo dotakli u pretprošlom predavanju, kada smo pričali o mjerama točnosti grupiranja u kontekstu odabira optimalnog broja grupa (Randov indeks, metoda siluete). Osnovna referenca za ocvu problematiku je ([Hubert and Arabie, 1985](#)). Dobar pregled evaluacijskih metrika za regresiju možete naći u [dokumentaciji za scikit-learn](#).
- 3 U nastavku koristim naziv **mjera vrednovanja**. U engleskom se tipično koristi **evaluacijska metrika** (engl. *evaluation metric*). Mjera i metrika nisu jedno te isto, ali nema potrebe da se time ovdje zamaramo.
- 4 Primjetite da točnost nije ništa drugo nego procjena očekivanja funkcije gubitka 0-1 izračunata na označenom skupu podataka (uzorku).
- 5 Ovdje postoji mogućnost terminološke konfuzije. **Stvarno pozitivni** (engl. *true positives*) primjeri su primjeri koji su pozitivni i označeni kao takvi. Međutim, **lažno negativni primjeri** (engl. *false negatives*) su zapravo također pozitivni primjeri. Ovdje treba biti pragmatičan i usvojiti sljedeće tumačenje: stvarno pozitivni primjeri su podskup pozitivnih primjera, a prilog "stvarno" znači "klasificirani kao pozitivni i stvarno su pozitivni".
- 6 **Neuravnoteženost skupa podataka** (engl. *dataset imbalance, class imbalance*) nije samo problem kod vrednovanja modela, već i kod njegovog treniranja. Standardni algoritmi strojnog učenja minimiziraju empirijsku pogrešku koja je aproksimacija gubitka 0-1, gdje se gubitak na primjeru tretira jednakom nevisno iz koje klase dolazi. Posljedično, modeli će tipično predviđati loše na klasama s manje primjera jer je tamo ukupan gubitak algoritma manji nego na većim klasama. Treniranje na neuravnoteženim skupovima podataka je realan problem koji se često pojavljuje u praksi. Na ovom kolegiju nažalost nemamo vremena baviti se njime. Zainteresirane upućujem na ([Chawla et al., 2004](#) i [Krawczyk, 2016](#)) te alat [imbalanced-learn](#) za [scikit-learn](#).
- 7 U kontekstu mjere vredovanja modela strojnog učenja, **odziv** (ili **odaziv**) ispravan je hrvatski prijevod engleskog termina *recall*. Međutim, u bespućima internetske zbiljnosti možda ćete nabasati na *opoziv*, što nije točan prijevod u kontekstu majera vrednovanja, premda jest jedan od mogućih prijevoda ove riječi (imenica *recall* u engleskom jeziku ima barem pet značenja; v. [ovdje](#)). Pogrešan prijevod možda je motiviran naslovom filma "Total Recall", koji je kod nas preveden kao "Totalni opoziv" (što je također moguće pogrešan prijevod, ako uzmete u obzir radnju filma).
- 8 **Mjera F_1** zapravo je poseban slučaj **F-mjere**, koju je 1992. godine predložio je van Rijsbergen ([van Rijsbergen, 1979](#)) u kontekstu pretraživanja informacija (engl. *information retrieval*). Mjera se od tamo preuzeta u druga područja, uključivo strojno učenje i obradu prirodnog jezika.
- 9 Od triju **pitagorejskih sredina** – aritmetičke, geometrijske i harmonijske – harmonijska sredina za različita opažanja daje najmanju, dok aritmetička sredina daje najveću srednju vrijednost. Drugim riječima, aritmetička sredina je gornja ograda geometrijske sredine, dok je geometrijska sredina gornja ograda harmonijske sredine (za dokaz, v. [ovdje](#)). To možemo pokazati i grafički. Razmotrimo sredinu dviju vrijednosti, pri čemu fiksirajmo jednu na 0.5. Ako sada mijenjamo drugu vrijednost u intervalu od 0 do 1, onda za aritmetičku, geometrijsku i harmonijsku sredinu dobivamo sljedeće:



Vidimo da, ako su obje vrijednosti jednake 0.5, onda su sve tri sredine također jednake 0.5. Međutim, čim su vrijednosti različite, harmonijska sredina manja je i od aritmetičke i od geometrijske sredine.

- [10] Premda ne postoji neko dogovorenog pravila za tretiranje rubnih slučajeva za mjere P , R i F_1 , uobičajeno je da se nedefinirana vrijednost postavlja na nulu. Tako to radi i [scikit-learn](#).
- [11] Mjera F_1 poseban je slučaj mjere F_β , definirane kao:

$$F_\beta = \frac{(1 + \beta^2)PR}{\beta^2 P + R}$$

Parametar β , gdje $\beta > 0$, određuje koliko je puta odziv važniji od preciznosti. Uobičajeno se koristi $\beta = 0.5$ (mjera $F_{0.5}$) ako se želi naglasiti preciznost, ili $\beta = 2$ (mjera F_2) ako se želi naglasiti odziv.

- [12] Zapravo, imamo tri mogućnosti. Pored mikro-prosjeka i makro-prosjeka postoji i treća mogućnost: izračun **težinskog prosjeka** mjera kroz sve klase, gdje su težine udjeli svake klase u skupu primjera. Formalno, za neku odabranu mjeru vrednovanja m , mjera **težinski- m** definirana je kao:

$$m^{avg} = \sum_j \frac{N_j}{N} m_j$$

gdje je N_j broj primjera u klasi $y = j$, tj. $N_j = \sum_{i=1}^N \mathbf{1}\{y^{(i)} = j\}$, a m_j je mjera izračunata na matrici zabune 2×2 za klasu s oznakom $y = j$ (kao što to radimo kod makro-mjera). Težinske mjere, npr., **težinski- F_1** (engl. *weighted F₁*), ima smisla koristiti kad god nam je važnost svake klase proporcionalna njezinoj veličini.

- [13] Pokažimo da vrijedi $P^\mu = R^\mu = F_1^\mu = Acc$. Pokažimo najprije $P^\mu = R^\mu$. Intuitivno, mjere mikro-preciznost i mikro-odziv daju jednakе vrijednosti zato što je u združenoj matrici zabune broj lažno pozitivnih primjera uvijek jednak broju lažno negativnih primjera (ono što je lažno pozitivno za jednu klasu lažno je negativno za drugu). Naime, za matricu zabune C dimenzija $K \times K$, broj lažno pozitivnih primjera i broj lažno negativnih primjera za klasu j jednak je:

$$FP_j = \sum_{i:i \neq j} C[j, i] \quad FN_j = \sum_{i:i \neq j} C[i, j]$$

U združenoj matrici zabune onda imamo zbroj lažno pozitivnih primjera kroz sve klase, $\sum_j FP_j$, i zbroj lažno negativnih primjera kroz sve klase, $\sum_j FN_j$, a za njih vrijedi:

$$\sum_j FP_j = \sum_j \sum_{i:i \neq j} C[j, i] = \sum_j \sum_{i:i \neq j} C[i, j] = \sum_j FN_j$$

Posljedično:

$$P^\mu = \frac{\sum_j TP_j}{\sum_j TP_j + \sum_j FP_j} = \frac{\sum_j TP_j}{\sum_j TP_j + \sum_j FN_j} = R^\mu$$

Mjera F_1^μ je harmonijska sredina mjeri P^μ i R^μ , pa, budući da $P^\mu = R^\mu$, to vrijedi $F_1^\mu = P^\mu = R^\mu$. Konačno, pokažimo da je mjera točnosti, definirana nad matricom zabune C dimenzija $K \times K$ kao

$$Acc = \sum_j \frac{C[j, j]}{N}$$

jednaka trima navedenim mjerama. Dovoljno je da pokažemo da vrijedi $Acc = P^\mu$. U združenoj matrici broj stvarno pozitivnih primjera odgovara ukupnomu broju točno klasificiranih primjera, budući da:

$$\sum_j \text{TP}_j = \sum_j C[j, j]$$

dok zbroj stvarno pozitivnih primjera i lažno pozitivnih primjera odgovara ukupnomu broju primjera, budući da:

$$\sum_j \text{FP}_j = \sum_j \sum_{i:i \neq j} C[i, j] = N - \sum_j \text{TP}_j$$

iz čega slijedi $N = \sum_j \text{TP}_j + \sum_j \text{FP}_j$. Uvrstimo li ovo u izraz za mjeru točnosti, dobivamo:

$$Acc = \frac{\sum_j C[j, j]}{N} = \frac{\sum_j \text{TP}_j}{\sum_j \text{TP}_j + \sum_j \text{FP}_j} = P^\mu$$

- 14** Na isti način kao što smo izračunali makro-preciznost i makro-odziv, mogli smo izračunati i **makro-točnost**, Acc^M . Također, kao što smo izračunali mikro-preciznost i mikro-odziv, mogli smo izračunati i **mikro-točnost**, Acc^μ . No, te dvije mjeru uvijek daju jednake vrijednosti. Naime:

$$\begin{aligned} Acc^M &= \frac{1}{K} \sum_j \frac{\text{TP}_j + \text{TN}_j}{\text{TP}_j + \text{FP}_j + \text{FN}_j + \text{TN}_j} = \frac{\sum_j (\text{TP}_j + \text{TN}_j)}{KN} \\ &= \frac{\sum_j (\text{TP}_j + \text{TN}_j)}{\sum_j (\text{TP}_j + \text{FP}_j + \text{FN}_j + \text{TN}_j)} = \frac{\sum_j \text{TP}_j + \sum_j \text{TN}_j}{\sum_j \text{TP}_j + \sum_j \text{FP}_j + \sum_j \text{FN}_j + \sum_j \text{TN}_j} = Acc^\mu \end{aligned}$$

Premda općenito daju različitu vrijednost od mjeru točnosti Acc izračunate na matrici $K \times K$, ove se mjeru u praksi rijetko koriste, budući da pate od istih problema kao i mjeru točnosti (visoka točnost trivijalnog klasifikatora na skupu primjera s neuravnoteženim klasama).

- 15** Klasifikatori će na stvarnim podatcima općenito biti lošiji na klasama s manjim primjerima, pa je **makro- F_1 tipično manji od mikro- F_1** ($F_1^M < F_1^\mu$). U tom smislu, ako vam netko prezentira samo vrijednost mjeru F_1^μ , a da za to ne ponudi adekvatno objašnjenje, zapitajte se da to nije možda zato što klasifikator radi dobro samo na klasama s velikim brojem primjera.

- 16** Ovdje prepostavljamo, što je uobičajena prepostavka, da za neki odabrani prag p , $p \in [0, 1]$, primjer klasificiramo u klasu $y = 1$ akko $P(y = 1|\mathbf{x}) \geq p$, a u klasu $y = 0$ akko $P(y = 1|\mathbf{x}) < p$. Drugim riječima, model binarnog klasifikatora definiramo kao $h(\mathbf{x}) = \mathbf{1}\{p(y|\mathbf{x}) \geq p\}$. (Primijetite da je prag p hiperparametar modela, budući da se ne optimira učenjem modela, ali se može optimirati post hoc). Međutim, u nekim primjenama, pogotovo kod donošenja odluka visokog rizika, bolje je klasifikator definirati tako da se suzdrži od klasifikacije ako ona nije dovoljno pouzdana, tj. ako je $P(y|\mathbf{x})$ previše blizu vrijednosti 0.5 (što je "previše blizu" ovisi, naravno, o konkretnom slučaju). Prepostavka u takvim slučajevima jest da je bolje da se klasifikator suzdrži od nepouzdane klasifikacije (i prepusti takve slučajeve drugim mehanizmima, na primjer ručnoj obradi stručnjaka) nego da napravi pogrešnu klasifikaciju. Takav se pristup naziva **klasifikacija s opcijom odbijanja** (engl. *classification with reject option*). Očekivano, za klasifikatore s opcijom odbijanja trebaju nam neke druge mjeru vrednovanja od ovih koje smo opisali, budući da sada treba vrednovati i koliko je klasifikator dobar u procjeni svoje vlastite pouzdanosti, odnosno koliko je dobro kalibriran, te treba uzeti u obzir kompromis između cijene pogrešne klasifikacije i cijene neprovodenja klasifikacije. Više o klasifikaciji s odbijanjem možete pročitati u ([Herbei and Wegkamp, 2006](#)), a o vrednovanju takvih klasifikatora u ([Condessa et al., 2017](#)).

- 17** Ovdje se zapravo radi o **višekriterijskoj optimizaciji**: nastojimo optimizirati vrijednost praga na temelju dvaju kriterija, preciznosti i odziva. Kombinacije preciznosti i odziva koje su bolje od nekih drugih kombinacija preciznosti i odziva, a međusobno nisu usporedive, čine **Paretovu frontu**.

- 18** Neobičan naziv "**ROC krivulja**" (engl. *receiver operating characteristics*) dolazi od primjene ove metode u Drugom svjetskom ratu za procjenu uspješnosti analize radarskih signala.

- 19** Ako je binarni klasifikator lošiji od nasumičnog, njegove predikcije uvijek možemo obrnuti i tako dobiti bolji klasifikator. Ipak, takve situacije su neobične u praksi. Ako je točnost klasifikatora lošija

od nasumičnog odabira, to upućuje ili na tehničku pogrešku (npr., u optimizaciji) ili da ne postoji jasan signal koji bi klasifikator mogao naučiti, pa je klasifikator naučio šum. Ni u kojem od ta dva slučaja obrtanje oznaka ne rješava stvarni problem.

- [20] Široko korištene metode **ponovnog uzorkovanja** (engl. *resampling*) u statistici su **jackknife**, **bootstrap** i **permutacijski test**. Posljednje dvije neizostavne su u *toolboxu* onih koji se žele baviti znanosti o podatcima. Više možete pronaći u ([Davison and Hinkley, 1997](#)) i ([Good, 2006](#)).
- [21] Možemo, međutim, pomoću LOOCV dobiti predikcije za svaki primjer pojedinačno pa sve to objediniti i nad time izračunati koju god mjeru vrednovanja želimo, no takva je procjena lošija jer nije dobivena kao srednja vrijednost procjena nad više uzoraka.
- [22] Pažljivi čitatelj primjetit će da nam ugniježđena unakrsna provjera $k \times l$ nam procjenu prosječne pogreške modela, ali nam ne daje jednoznačan odgovor na pitanje koji je model zapravo najbolji, jer optimalni modeli mogu biti različiti u svakoj iteraciji vanjske petlje. Također ne dobivamo odgovor na pitanje koji je naučeni model (hipoteza) najbolji, jer se i naučeni modeli se mogu razlikovati u svakoj iteraciji vanjske petlje. Ovo je u redu, jer unakrsna provjera služi za nepristranu i poštenu procjenu pogreške generalizacije algoritma strojnog učenja, a ne jednog specifičnog modela. U konačnici, međutim, nama treba jedan naučen model, kako bismo ga isporučili odnosno ugradili u neki drugi sustav. Do njega tipično dolazimo tako da odabiremo onaj model (one hiperparametre) koji su najčešće davali optimalan model (ili interpoliramo između vrijednosti hiperparametara koji su davali najbolje modele) u unutarnjoj petlji. Takav model onda učimo na kompletном skupu označenih primjera \mathcal{D} , i njega isporučujemo. Za taj model više nemamo procjenu pogreške, jer više nemamo izdvojenih podataka na kojima bismo ga ispitali, ali očekujemo da će pogreška biti jednaka ili manja od one koje smo dobili ugniježđenom unakrsnom provjerom. Zašto? Zato što smo model učili na kompletnom skupu \mathcal{D} , a ne samo na skupu za učenje. U tom smislu, procjena pogreške dobivena ugniježđenom unakrsnom provjerom na skupu \mathcal{D} je pesimistična procjena stvarne pogreške optimalnog modela naučenog na cijelom skupu \mathcal{D} .

Literatura

- N. V. Chawla, N. Japkowicz, and A. Kotcz. Special issue on learning from imbalanced data sets. *ACM SIGKDD explorations newsletter*, 6(1):1–6, 2004.
- F. Condessa, J. Bioucas-Dias, and J. Kovačević. Performance measures for classification systems with rejection. *Pattern Recognition*, 63:437–450, 2017.
- A. C. Davison and D. V. Hinkley. *Bootstrap methods and their application*. Number 1. Cambridge university press, 1997.
- P. I. Good. *Resampling methods*. Springer, 2006.
- R. Herbei and M. H. Wegkamp. Classification with reject option. *The Canadian Journal of Statistics/La Revue Canadienne de Statistique*, pages 709–721, 2006.
- L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.
- N. Japkowicz and M. Shah. *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.
- B. Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, 2016.
- S. Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*, 2018.
- C. van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979.