Uvod v odkrivanje znanj iz podatkov

(zapiski predavatelja, samo za interno uporabo)

Blaž Zupan

31. december 2019

Kazalo

Poglavje 1

Odkrivanje skupin

Odkrivanje skupin, ali razvrščanje podatkov v skupine, je ena od osnovnih tehnik podatkovne analitike. Na podlagi podatkov o uporabnikih marketinški oddelek lahko zanima segmentacija uporabnikov, oziroma kateri so njihovi tipični uporabniki in koliko takih skupin sploh imajo. Poznavanje skupin in njihovih lastnosti lahko privede do izdelave boljših ciljnih akcij. Na področju medicine nas lahko zanimajo tipične skupine bolnikov, da bi lahko njim prilagodili in izboljšali bolnišnični sistem. Raziskovalce vesolja lahko zanima, kakšne skupine galaksij (poleg teh, ki so jih že odkrili) poznamo in predvsem, kakšne so njihove karakteristike. V množici dokumentov nas mnogokrat lahko razveseli, če lahko z algoritmi odkrijemo skupine med seboj podobnih enot in uporabnikom na ta način olajšamo brskanje po gradivu. Fotografu lahko pomagamo, če mu s pomočjo računalnika uredimo slikovno gradivo, ki ga je v zadnjem desetletju nakopičil na zunanjem disku. Iz podatkov o odnosih šolskimi otroci šolske sociologe mnogokrat zanima oblikovanje skupin, pa tudi odkrivanje osamelcev, ki jim je potrebno pomagati pri socializaciji. Različnih uporab tehnik odkrivanja skupin je mnogo in vsekakor preveč, da bi jih tu vse našteli, zato raje pričnemo s primerom.

1.1 Primer podatkov

V razredu 8.A je šestnajst učencev, ki jih je šolski sistem ravno izmučil z republiškim eksternim preverjanjem znanja. Uspeh, ki so ga dosegli, je podan v tabeli **??**. Številke so percentili; Branka je na primer bila dobra pri angleščini, kjer je bila boljša od 95 odstotkov vseh učencev, ki so pisali test. Ni pa ji ravno šla fizika, kjer je bilo slabših od nje le 11 odstotkov.

Zanima nas, ali so v razredu kakšne značilne skupine učencev in če so, kakšne so njihove karakteristike oziroma kako se razlikujejo med sabo. Zanima nas tudi, kateri učenci so si med sabo podobni. Spoznanja na tem področju bodo šoli pomagala pri oblikovanju skupin, kjer bi na primer skupini, ki je dobra v naravoslovnih predmetih slaba pa pri slovenščini posebej poiskali kakšno zanimivo knjigo za domače branje, te, ki pa jim gre družboslovje, navdušili za novinarski krožek.

Tabela 1.1: Rezultati zunanjega preverjanja za 16 učencev razreda 8.A. Rezultati posameznih predmetov so podani v percentilih z ozirom na republiške rezultate.

ime	slo	ang	zgo	geo	mat	bio	fiz	kem	tel
Albert	22	81	32	39	21	37	46	36	99
Branka	91	95	65	96	89	39	11	22	29
Cene	51	89	21	39	100	59	100	89	27
Dea	9	80	18	34	61	100	90	92	8
Edo	93	99	39	100	12	47	17	12	63
Franci	49	83	17	33	92	30	98	91	73
Helena	91	99	97	89	49	96	81	94	69
Ivan	12	69	32	14	34	12	33	48	96
Jana	91	80	20	10	82	93	87	91	22
Leon	39	100	19	29	99	31	77	79	23
Metka	20	91	10	15	71	99	78	93	12
Nika	90	60	45	34	45	20	15	5	100
Polona	100	98	97	89	32	72	22	13	37
Rajko	14	4	15	27	61	42	51	52	39
Stane	9	22	8	7	100	11	92	96	29
Zala	85	90	100	99	45	38	92	67	21

1.2 Nekaj matematičnih oznak

Učencem iz tabele **??** bomo rekli učni primeri in jih bomo označili z $x^{(i)}$, kjer je i indeks primera. Ker se bomo na teh primerih skušali naučiti, kakšne skupine poznamo v razredu 8.A, bomo tem primerom rekli učni primeri. Množico učnih primerov označimo z $\mathcal{U}$, njeno moč oziroma število primerov pa označimo z m, torej $m = |\mathcal{U}|$. Naj bo $C \subset \mathcal{U}$ skupina primerov, $\mathcal{C} = \{C_j\}$ pa razvrstitev, to je množica skupin, ki smo jo odkrili v podatkih. Tu se bomo ukvarjali samo s postopki, ki odkrivajo neprazne skupine, $|C_i| > 0$. Zanimali nas bodo različni tipi razvrstitev. Možna razvrstitev je lahko takšna, kjer si skupine med seboj ne delijo nobenega primera,

$$C_i \cap C_j = \emptyset;\ C_i, C_j \in \mathcal{C}$$

in kjer je unija vseh skupin enaka učni množici,

$$\bigcup_{C_i \in \mathcal{C}} C_i = \mathcal{U}$$

Pri taki razvrstitvi torej vsak primer pripada natanko eni skupini. Tako razvrstitev imenujemo razbitje.

Zanimale nas bodo tudi hierarhije skupin, kjer velja $C_i \cap C_j \in \{C_i, C_j, \emptyset\}$. Presek dveh skupin je lahko samo prazna množica, ali pa v celoti ena od skupin. Pri nepraznem preseku

to pomeni, da je ena od skupin v celoti vsebovana v drugi. Tako lastnost bodo imele skupine pri hierarhičnem razvrščanju.

Vsak primer v tabeli **??** je opisan z n atributi (številskimi spremenljivkami). Primer x zapišemo kot:

$$x = (x_1, x_2, \ldots, x_n)^T$$

Tu smo zaradi poenostavitve namenoma izpustili indeks primera. Primer $x^{(i)}$ smo tako označili s simbolom x. Ker je vsak primer vektor, bi temu primerno morali izpisati tudi oznako, torej kot $\mathbf{x}$, a bomo tudi poenostavili in primer označili kar z x.

Primer smo zapisali kot stolpični vektor, v tabeli s primeri pa je zapisan kot vrstični vektor, ki ga zato, da dobimo stolpec, transponiramo. Pravzaprav je to, ali je primer zapisan kot stolpec ali vrstica, za sedaj precej nepomembno, bo pa notacija pomembna takrat, kadar bomo o naboru primerov razmišljali kot o matriki. Primeri v naši tabeli so poimenovani (imena učencev), atributi pa imajo prav tako ime. Lahko bi sicer za naš primer Albert pisali $x_{slo}^{(Albert)} = 22$, a bomo v nadaljevanju zaradi splošnosti raje uporabljali numerične indekse.

1.3 Mere različnosti

V postopku odkrivanja skupin primerov bi želeli poiskati množice primerov, kjer so si ti med seboj čim bolj podobni. Oziroma enakovredno, vsebujejo primere, kjer so si ti med sabo čimmanj različni. Tu nam bo torej prav prišla mera različnosti. Najbolj enostavno je pričeti s parom primerov in za ta zapisati enačbo, s katero ocenimo, kako različna sta. Vzemimo primera a in b in predpostavimo, da sta ta opisana z atributi, ki lahko zavzamejo zvezne vrednosti:

$$a = (a_1, a_2, \ldots, a_n)$$

$$b = (b_1, b_2, \ldots, b_n)$$

Tu bi lahko primera označil z $x^{(a)}$ in $x^{(b)}$, a bi potem enačbe, ki sledijo, dobile nepotrebno prtljago.

Če si predstavljamo, da je domenski atributni prostor Evklidski, potem je primerna razdalja oziroma mera različnosti med primeri kar evklidska razdalja:

$$d(a, b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

Kako bi najbolj enostavno izmerili razdalje med primeri za $n = 2$ in za primere, ki jih v ta namen izrišemo na milimetrski papir? Z drugimi besedami, kako bi izmerili razdalje med primeri v razsevnem diagramu? Z ravnilom. Zato razdalji, ki je opisana z zgornjo enačbo, rečemo Evklidska.

Še enostavnejša je Manhattanska razdalja (od kod in zakaj to ime?):

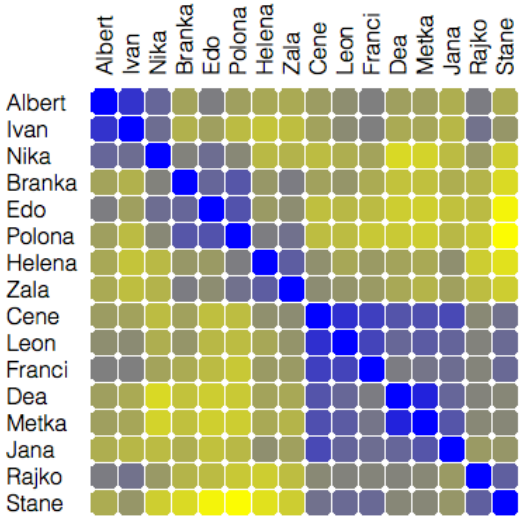
$$d(a,b) = \sum_{i=1}^{n} | a_i - b_i |$$

Obe, Manhattanska in Evklidska razdalja pa sta posebni primer razdalje Minkowskega:

$$d(a,b) = (\sum_{i=1}^{n} |a_i - b_i|^r)^{\frac{1}{r}}$$

O merah različnosti bomo še govorili. Premisliti bo na primer še potrebno, kako merimo razdalje pri različnih tipih atributov. Na primer, če ti zavzemajo diskretne vrednosti. Ali pa kako merimo razdalje med besedili in nizi znakov, med dokumenti, med stvarmi v priporočilnih sistemih, med slikami, med glasbenimi posnetki in podobno. A za naš primer so zgornje definicije dovolj.

Z izbrano mero različnosti lahko izmerimo razdalje med vsemi pari primerov. Te potem lahko vizualiziramo s toplotno karto (angl. *heatmap*), kot to prikazuje slika **??**. Če "močne" skupine obstajajo, in če smo primere na toplotni karti pravilno uredili (kako?), bi se na karti morali prikazati vzorci, ki nakazujejo skupine (kakšni?).



Slika 1.1: Vizualizacija podobnosti oziroma različnosti med primeri s tabele **??**.

1.4 Ocenjevanje razdalj med skupinami

Tudi med skupinami primerov lahko merimo razdaljo. A tu naletimo na problem, saj so sedaj skupine podane z množico primerov. Med primeri znamo meriti razdalje, med skupinami

pa (še) ne. Razdaljo bomo merili med dvemi skupinami primerov, C_u in C_v. Predpostavimo najprej, da si skupine ne delijo primerov, torej $C_u \cap C_v = \emptyset$. Uvedimo nekaj možnih razdalj:

- Razdalja med skupinama C_u in C_v je razdalja med njunima najbližjima primeroma $a \in C_u$ in $b \in C_v$ (angl. *single linkage*):

$$d_{\min}(C_u, C_v) = \min_{a,b}\{d(a,b) \mid a \in C_u, b \in C_v\}$$

- Razdalja med skupinama C_u in C_v je razdalja med njunima najbolj oddaljenima primeroma $a \in C_u$ in $b \in C_v$ (angl. *complete linkage*):

$$d_{\max}(C_u, C_v) = \max_{a,b}\{d(a,b) \mid a \in C_u, b \in C_v\}$$

- Razdalja med skupinama C_u in C_v je povprečna razdalja med vsemi pari primerov iz teh dveh skupin (angl. *average linkage*), torej:

$$d(C_u, C_v) = \sum_{a \in C_u} \sum_{b \in C_v} \frac{d(a,b)}{|C_a||C_b|}$$

- Wardova razdalja, ki privzame, da je za vsako skupino C znan njen centroid R oziroma točka v središču skupine. Naj bo R_u centroid skupine C_u, R_v centroid skupine C_v, R_{uv} pa centroid združene skupine $C_{uv} = C_u \cup C_v$. Potem je Wardova razdalja določena kot:

$$d_w(C_u, C_v) = \sum_{x \in C_{uv}} d(x, R_{uv})^2 - \left(\sum_{x \in C_u} d(x, R_u)^2 + \sum_{x \in C_v} d(a, R_v)^2\right)$$

1.5 Hierarhično razvrščanje v skupine

Postopek, ki iz podatkov razvije hierarhijo skupin, kjer je najnižje v hierarhiji vsak primer predstavljen s svojo skupino, najvišje v hierarhiji pa tvori skupino kar celotna množica učnih primerov, se imenuje hierarhično razvrščanje v skupine.

1.5.1 Algoritem

Hierarhično razvrščanje v skupine najlažje opišemo kar s psevdokodo:

- vsak primer naj tvori svojo skupino, $C_i = \{x^{(i)}\}$, $x^{(i)} \in \mathcal{U}$
- ponavljaj, dokler ne ostane ena sama skupina
  - poišči najbližji si skupini C_a in C_b, $d(C_a, C_b) = \min_{u,v} d(C_u, C_v)$
  - združi skupini C_a in C_b v skupino $C_{ab} = C_a \cup C_b$

- nadomesti skupini C_a in C_b s skupino C_{ab}
- izmeri razdaljo med novo skupino C_{ab} in ostalimi skupinami

Prostorska kompleksnost tega algoritma je $O(m^2)$, kjer je m število primerov oziroma velikost učne množice. Časovna kompleksnost pa je $O(m^3)$ saj se algoritem izvede v m korakih, v katerih mora osvežiti in uporabiti matriko razdalj velikosti m^2. Z malo truda se da postopek izračunavanja novih razdalj spisati tako, da ima manjšo zahtevnost in je potem zahtevnost celotnega algoritma $O(m^2 \log(m))$. Tudi to ni malo. Predstavljajte si, da morate algoritem pognati na vseh uporabnikih nekega večjega mobilnega operaterja. Ali pa na vseh uporabnikih družabnega omrežja. Kam shranite matriko razdalj med pari primerov? Je opisani postopek tudi časovno zahteven?

1.5.2 Dendrogram

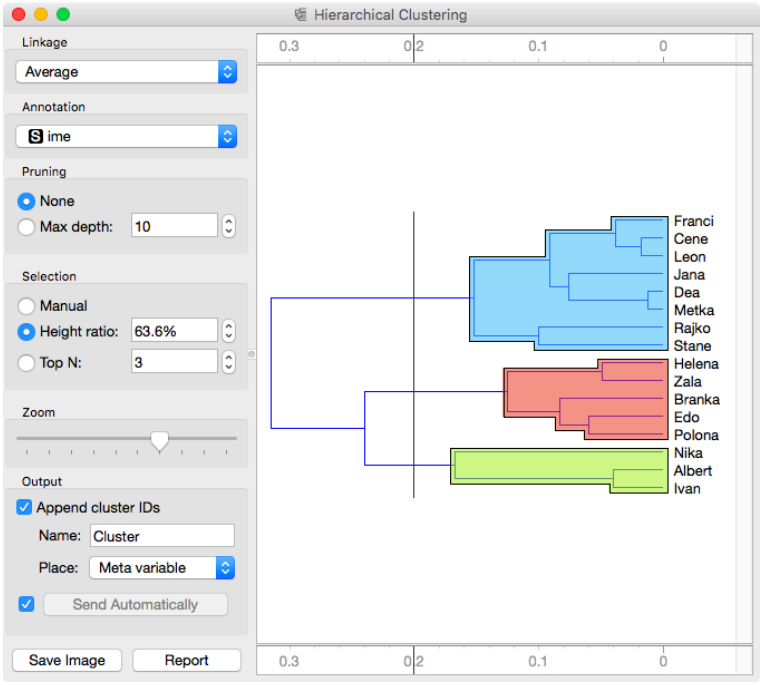
Postopek združevanja v skupine in rezultat tega postopka — hierarhijo skupin, lahko ponazorimo v drevesnem izrisu ali dendrogramu (gr. *dendron* pomeni drevo, *gramma* pa risba). Dendrogram hierarhičnega razvrščanja v skupine podatkov s tabele **??** prikazuje slika **??**. Dendrogram je izrisan od leve proti desni. Stičišča skupin so od desnega roba odmaknjena skladno z razdaljo med skupinami. Pred izvedbo postopka razvrščanja v skupine orodja podatke tipično normalizirajo, tako da so po normalizaciji povprečne vrednosti atributov enake nič in njihove standardne deviacije enake 1.

1.5.3 Razlaga skupin

Dendrogram s slike **??** nakazuje na tri skupine, ki smo jih v odkrili v podatkih. Zanima nas, kaj je značilnega za posamezne skupine. Preprosta tehnika, ki jo lahko uporabimo je, da za vsak skupino primerjamo povprečno vrednost posameznega atributa s povprečno vrednostjo tega atributa v celotni učni množici. Naj bo Albert v skupini, ki jo označimo z vrstnim številom 1, Branka v skupini 2, Cene pa v skupini 3. Povprečne vrednosti učne množice prikazuje tabela **??**.

Tabela 1.2: Povprečne vrednosti percentilov pri posameznih predmetih v učni množici in odkritih skupinah.

	slo	ang	zgo	geo	mat	bio	fiz	kem	tel
$\mathcal{U}$	54	78	40	47	62	52	62	61	47
skupina 1	41	70	36	29	33	23	31	30	98
skupina 2	92	96	80	95	45	58	45	42	44
skupina 3	35	69	16	24	83	58	84	85	29



Slika 1.2: Primer dendrograma, kot ga izriše ustrezna komponenta v programu Orange. Uporabnik se je tu odločil, da dendrogram odreže v točki, ki primere razdeli v tri skupine. Primerna razdelitev v tem primeru bi lahko bila ta, ki uporabi dve skupine. Zakaj?

Zanima nas pravzaprav odstopanje od povprečnih vrednosti atributov učne množice. Tu bi se sicer morali poigrati s statistiko, in ugotoviti, katera odstopanja so značilna in kako daleč so od takih, ki bi jih dobili iz naključne razvrstitve. Zaradi enostavnosti, ki pa bo prav tako čisto primerna za naš primer, se tu raje zatečemo k veliko preprostejši rešitvi. Za vsako skupino označimo odstopanja navzdol za več kot 10 odstotnih točk z "−", in podobna odstopanja navzgor s "+".

Tabela 1.3: Odstopanja (±10 točk) od povprečnih vrednosti v učni množici.

	slo	ang	zgo	geo	mat	bio	fiz	kem	tel
skupina 1	−			−	−	−	−	−	+
skupina 2	+	+	+	+	−		−	−	
skupina 3	−		−	−	+		+	+	−

Rezultat prikazuje tabela **??**. Albert, Ivan in Nika so kot kaže športniki. Njihovo udejstvovanje s športom jim vzame veliko časa, morda bi jim bilo potrebno pomagati pri naravoslovnih predmetih, kjer jim malce škripa. Branka, Edo in ostali iz skupine 2 imajo rajši družboslovne predmete. Skupina s Cenetom, Deo in Metko pa zanima naravoslovje. Ne bi bilo narobe, če bi se ti vpisali v kakšen športni krožek.

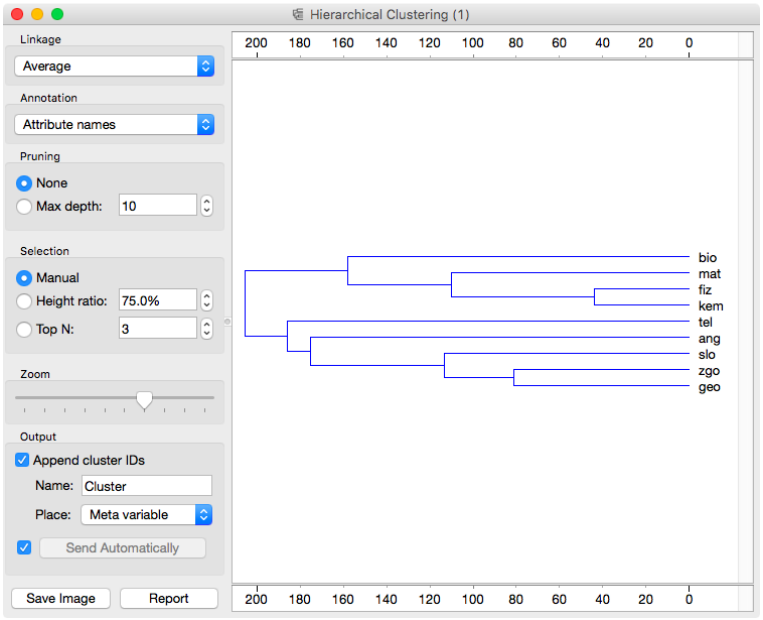
1.5.4 Stvari pa lahko tudi obrnemo

Namesto skupin učencev nas bi lahko zanimale tudi skupine predmetov. Kako? Tabelo podatkov enostavno zasukamo oziroma transponiramo. Vse ostalo ostane isto. Rezultat je podan na sliki **??**.

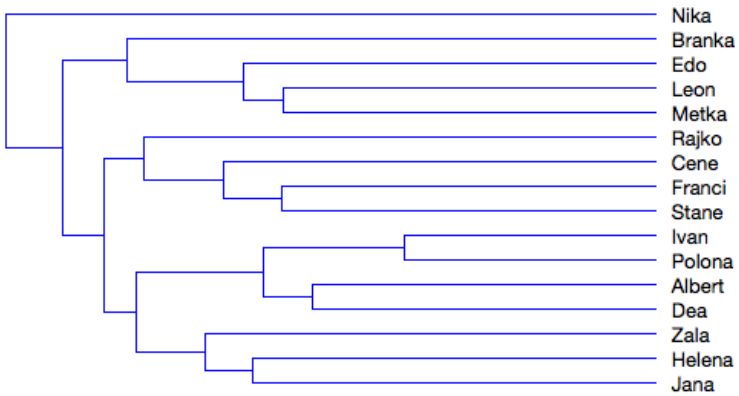
1.5.5 Kaj pa, ko ni skupin?

Hierarhično razvrščanje v skupine bo vedno našlo skupine. Postopek bo vedno razvil dendrogram, ne glede na to, ali skupine dejansko obstajajo v podatkih ali ne. Pravzaprav, kaj pa so sploh skupine? Kako lahko sploh ocenimo, ali te dejansko obstajajo? Tole se v tem trenutku precej težka vprašanja in se bomo z njimi še ukvarjali. A vseeno, za okus, naredimo naslednji poskus. Naše podatke o ocenah učencev 8.A razreda uničimo tako, da naključno premešamo števila v vsaki od kolon, za vsako kolono posebej. Dendrogram na takih podatkih kaže slika **??**. Razdalje med posameznimi skupinami so tu večje, stičišča povezav v dendrogramu so se premaknila proti levi strani. Stukture, iz katere bi bile jasno razvidne skupine, ki bi se potem združevale pri opazno večjih razdaljah kot se združujejo primeri znotraj posameznih skupin, ni več.

Če imamo prave podatke, te seveda ne premešamo in potem izrisujemo "pokvarjene" dendrograme (ali pač? a o pomenu postopkov, ki temeljijo na ponaključenih podatkih, kdaj



Slika 1.3: Dendrogram skupin predmetov. Telovadba je nekakšen osamelec, samosvoj predmet precej različen od ostalih. Kako je to vidno iz dendrograma?



Slika 1.4: Dendrogram na "ponaključenih" podatkih. Struktura, ki bi nakazovala jasne skupine v podatkih, je izginila.

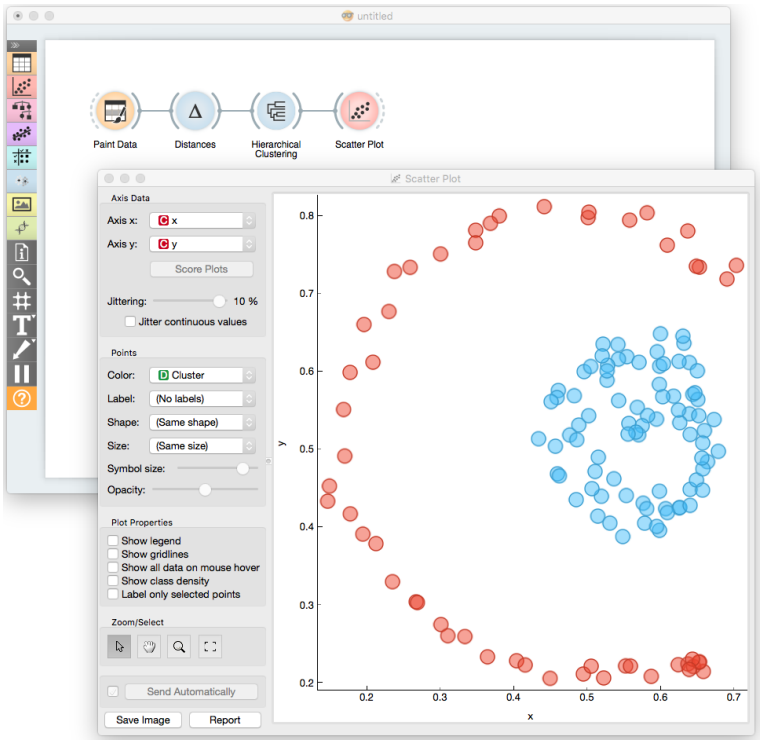
drugič). A je vseeno dobro vedeti, kakšni dendrogrami so taki, kjer skupin pač ni. Torej, če dobimo kaj podobnega, kot je dendrogram na sliki **??**, je zaključek tak, da iz danih podatkov pri dani meri razdalje in pri dani metodi za ocenjevanje razdalj med skupinami postopek hierarhičnega razvrščanja v skupine ni našel značilnih skupin. Prejšnji stavek je precej dolg. Krajši bi recimo lahko samo povedal, da v podatkih ni skupin. A taka ugotovitev ne bi bila nujno resnična. Iz naših poskusov lahko samo ugotovimo, da če morda skupino so, jih mi pač nismo uspeli najti. Lahko pa, seveda, da jih sploh ni.

1.6 O izboru mer razdalje

Metoda hierarhičnega gručenja ima pravzaprav dva parametra: prvi je metoda merjenja razdalj med primeri, drugi pa način določanja razdalj med skupinami. Ko imamo podatke predstavljene atributno, torej v tabeli, kjer je vsak primer predstavljen z vektorjem atributnih vrednosti, je evklidska razdalja čisto primerna mera. Če atributi izvirajo na primer iz tekstovnih podatkov in je teh veliko, se izkaže, da je pomembna ne toliko velikost posameznega vektorja kot njegova smer. Prava razdalja za take podatke je tako imenovana kosinusna razdalja. O uporabi te več v naslednjih poglavjih. Na področju biomedicine, kjer so atributi istega tipa in je bolj kot njihova vrednost pomemben vrstni red (tipa: kateri atribut je zavzel največjo vrednost, drugo največjo, ipd.) je mnogokrat uporabljana razdalja Spearmanova korelacija, ki jo izračunamo na rangi. V tem odstavku nam ni namen formalno uvesti vse te različne mere, ampak samo poudariti, da je izbor mere razdalje med primeri odvisen od problemske domene in jo določimo skladno z vedenjem oziroma intuicijo o tem, kaj res najbolj primerno določa razdaljo med primeri.

Malce drugače je z izborom pristopov merjenja razdalje med skupinami. Kot smo že zapisali, ti pristopi (npr. *single linkage*, *complete linkage*) agregirajo pare razdalj med primeri v dveh različnih skupinah. V splošnem, če ne poznamo podatkov ali pa njihovih projekcij v nižje dimenzije, bo primerna izbira povprečne razdalje (angl. *average linkage*), mnogokrat pa presenetljivo dobro deluje Wardova razdalja. Po drugi strani je pristop z minimalno razdaljo (angl. *single linkage*) v splošnem neuporaben, razen za res posebne primere (slika **??**).

Recepta, katere mere razdalj uporabiti, v zgodnjih dveh odstavkih nismo zapisali. Ker tega ne znamo. Namreč, vse je odvisno od podatkov. Ker na področju razvrščanja v skupine nimam prav dobrih mer o tem, kako dober je rezultat razvrščanja, se avtomatizacija iskanja najbolj ustreznih parametrov ne obnese najbolje. Z drugimi besedami: v splošnem ne znamo spisati enačbe, s katero bi numerično ocenili kvaliteto nastalih skupin. Ker te enačbe ni, ne moremo strojno primerjati rešitve med sabo. Ostane nam, da se zanesemo na intuicijo in na pregled rešitve s strani domenskega eksperta. Gručenje, ki je uporabniku koristno (za karkoli že), je potem tisto, za katero se na koncu odločimo.



Slika 1.5: Primer dvodimenzionalnih podatkov in njihove razvrstitve, kjer je ustrezna metoda minimalne razdalje za merjenje podobnosti med skupinami, in kjer ostale metode merjenja razdalj odpovedo (za merjenje razdalj med primeri smo uporabili Evklidsko razdaljo).

Poglavje 2

Metoda voditeljev

Velika prednost metode hierarhičnega gručenja, ki smo jo spoznali v prejšnjem poglavju, je odkrivanje strukture skupin v podatkih, ki jih lahko enostavno ponazorimo v vizualizaciji imenovani dendrogram. Na podlagi te vizualizacije lahko potem (intuitivno) presojamo o kvaliteti gručenja ter se odločamo o tem, na koliko skupin bomo razbili podatke. Zanimiva je tudi uporaba te tehnike pri interaktivni analizi podatkov v orodjih, ki nam take vizualizacije ponujajo in kjer lahko gradimo delokroge (angl. *workflows*), s katerimi lahko izbrane skupine nadalje opišemo in interpretiramo.

A pri tem naletimo na največjo pomankljivost hierarhičnega gručenja: metoda deluje samo na relativno majhnih množicah podatkov. Če imamo podatkov veliko, recimo nekaj deset tisoč ali pa celo milijon, hierarhično gručenje zaradi časovne in prostorske kompleksnosti odpove, dendrogram pa postane prevelik in neuporaben.

Rešitev, ki dobro deluje na velikih množicah podatkov je postopek, ki ga imenujemo metoda voditeljev (angl. *K-means*). Seveda zastonj kosila tudi tu ni. V svoji osnovni inačici metoda voditeljev predpostavlja, da je uporabnik vnaprej določil število skupin, na katero želi razbiti učno množico primerov. Algoritem uporablja t.im. voditelje oziroma primere, ki določajo te skupine oziroma so središča skupin. Postopek iskanja optimalnega razbitja je potem naslednji:

- prični s K naključno izbranimi voditelji $\mathcal{V} = \{v^{(i)};\ i \in 1 \ldots K\}$
- **ponavljaj**
  - določi razvrstitev C tako, da vsak primer prirediš najbližjemu voditelju
  - novi voditelji naj bodo centroidi $R(C_i)$ skupin $C_i \in C$, $v^{(i)} \leftarrow R(C_i)$
- **dokler** se lega voditeljev spreminja

Voditelji (angl. *centroids*) so navadno kar geometrijska središča primerov skupine:

$$R(C_i) = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

Izračun teh je za zvezne atribute preprost. Ker moramo v vsaki iteraciji izračunati razdaljo do K voditeljev, je časovna kompleksnost tega algoritma le linearno odvisna od števila primerov, to je $O(I*k*m)$, kjer je I število iteracij. Algoritem tipično hitro konvergira h končni rešitvi. Za srednje velike probleme (npr. nekaj 1000 primerov) je lahko potrebnih manj kot 100 iteracij.

Potek optimizacije za podatke, ki so bili opisani z dvema zveznima atributoma in so vključevali 780 primerov, je prikazan na sliki **??**. Za prikazano optimizacijo je zanimivo tudi število primerov, ki so pri posameznih iteracijah spremenili svojega voditelja:

```
Iteration: 1, changes: 54
Iteration: 2, changes: 21
Iteration: 3, changes: 18
Iteration: 4, changes: 18
Iteration: 5, changes: 18
Iteration: 6, changes: 29
Iteration: 7, changes: 63
Iteration: 8, changes: 131
Iteration: 9, changes: 64
Iteration: 10, changes: 6
Iteration: 11, changes: 0
```

Optimizacija se skoraj ulovi v lokalni minimum, potem pa se v osmi iteraciji pobere in poišče rešitev, ki se nam v tem primeru zdi smiselna.

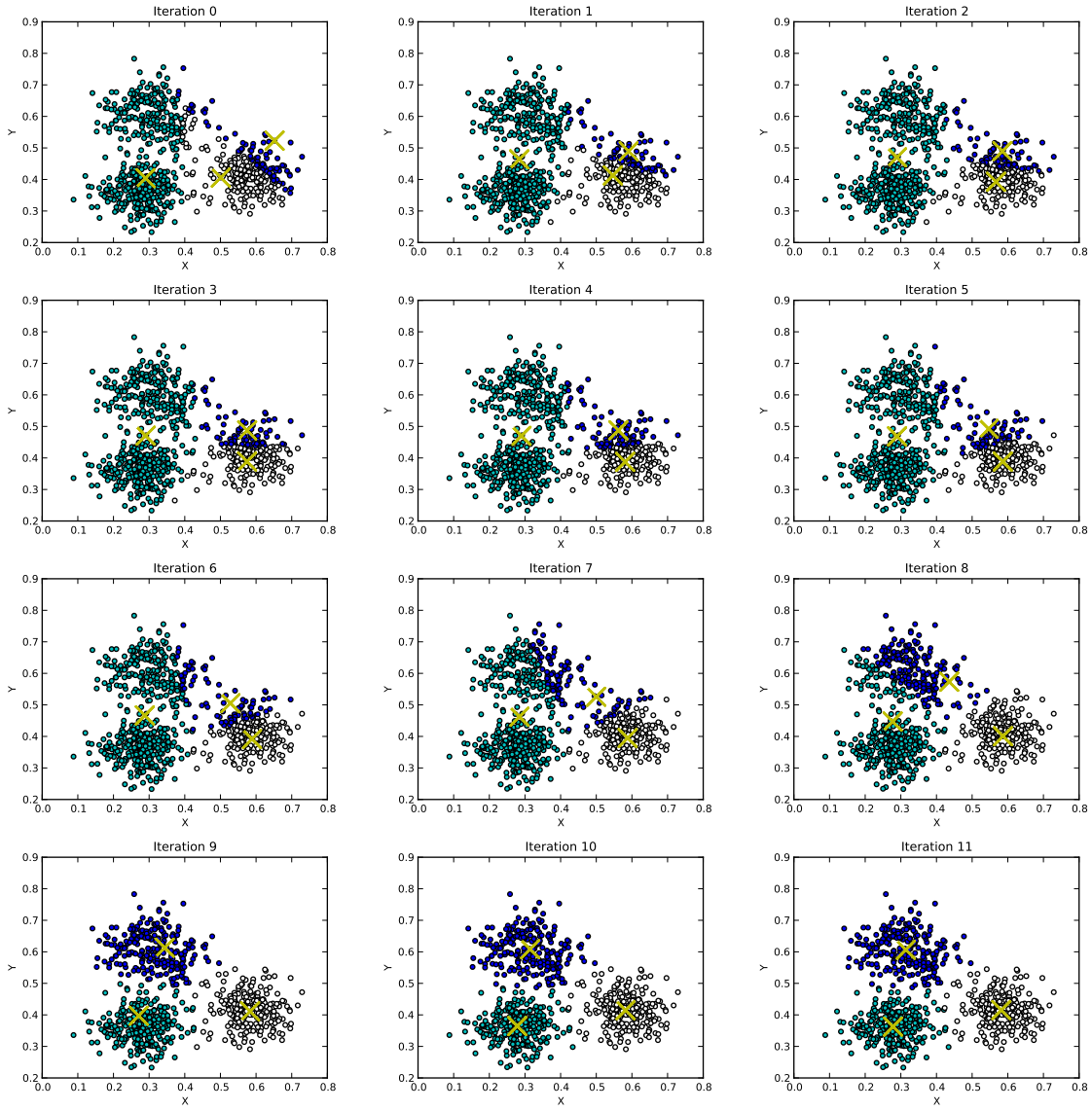
Tehnika voditeljev pa nas, za izbrane podatke in parametre metode, lahko tudi preseneti in vrne nepričakovane rezultate. Primer take razvrstitve je prikazan na sliki **??**.

Rezultati metode voditeljev so lahko zelo občutljivi na izbor primernih začetnih pogojev in mero razdalje med primeri (pri zgornjih primerih smo uporabljali evklidsko razdaljo).

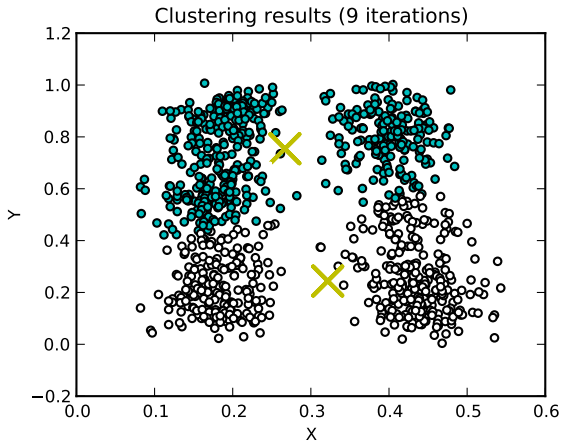
2.0.1 Začetni izbor voditeljev

Razbitje, ki ga vrne metoda voditeljev, je lahko zelo odvisna od začetnega izbora voditeljev. Od te bo tudi odvisno število iteracij, ki so potrebne, da postopek privedejo v stabilno stanje, v katerem so izpolnjeni ustavitveni pogoji. Zato je smiseln študij tehnik, ki bi voditelje izbrala čim bolje. Znanih je nekaj standardnih prijemov:

Naključni izbor voditeljev. Za tega smo sicer že povedali, da lahko privede v neoptimalno razbitje. A se lahko temu izognemo, če postopek ponovimo večkrat in med poiskanimi razbitji izberemo najboljše. Kako to storimo? Katero razbitje je najboljše?



Slika 2.1: Potek optimizacije razbitja za 780 primerov v evklidski ravnini. Lega voditeljev je označena s križcem. Primeri, ki pripadajo istemu voditelju, so označeni z isto barvo.



Slika 2.2: Na pogled nepravilno razbitje, ki ga je predlagala metoda voditeljev.

Izbor razpršenih voditeljev. Poiščemo primer, ki je najbolj oddaljen od drugih. Naj bo to naš prvi voditelj. Potem poiščemo primer, ki je njemu najbolj oddaljen. To je drugi voditelj. Naslednje voditelje poiščemo tako, da so ti najbolj oddaljeni od voditeljev, ki smo jih že določili.

Uporaba hierarhičnega razvrščanja. Na podmnožici točk s hierarhičnim razvrščanjem poiščemo K skupin in njihova središča uporabimo kot začetne voditelje. Zakaj ne uporabimo celotnega nabora primerov?

2.0.2 Ustavitveni kriterij

V osnovnem algoritmu za iskanje razbitij po metodi voditeljev zahtevamo, da se postopek ustavi šele takrat, ko se razbitje več ne spreminja. Takrat noben primer ne zamenja svojega centroida. V splošnem temu pogoju ne moremo vedno zadostiti, saj se nam lahko zgodi, da pride do oscilacij in se postopek na omenjeni način nikoli ne zaustavi. A so ti primeri v praksi zelo redki. Pri velikih množicah primerov, in v izogib nestabilnosti, lahko namesto popolne stabilnosti rešitve postopka zahtevamo, da se ta izteče, ko v iteraciji samo nekaj (na primer 10) primerov ali manj zamenja skupino.

2.0.3 Ocenjevanje kvalitete razbitij

Glavna pomanjkljivost metode voditeljev, tudi z ozirom na tehniko hierarhičnega razvrščanja v skupine, je potreba po izboru števila skupin K. Uporabnik le redkokdaj ve, na koliko skupin bi bilo potrebno rezultate razbiti. Pravzaprav je prav “pravo” število skupin ena od ključnih informacij, ki bi jih radi na podlagi podatkov ocenili.

Tu se zatečemo k razmisleku o cilju razbitja podatkov na skupine. V splošnem bi želeli poiskati taka razbitja, kjer so si primeri znotraj skupine čim bolj podobni, ter primeri iz različnih skupin čim bolj različni. Če razmišljamo samo o prvem, potem želimo, da metoda voditeljev poišče tako razbitje, kjer bo vsota oddaljenosti od pripadajočih voditeljev oziroma spodnji izraz čim manjši:

$$\sum_{i=1}^{K} \sum_{x \in Ci} d(v^{(i)}, x)$$

Predpostavimo, da se naši primeri nahajajo v evklidski ravnini in da imamo opravka s podatki z enim samim atributom. Razdaljo med točkami bomo merili z evklidsko razdaljo. Zgornji izraz se prevede v vsoto kvadratnih napak (angl. *sum of squared errors, SSE*) oziroma odstopanj od centroidov:

$$SSE = \sum_{i=1}^{K} \sum_{x \in Ci} (v^{(i)} - x)^2$$

Iščemo take centroide C_k, kjer je ta napaka najmanjša:

$$\frac{\partial}{\partial v^{(k)}} = \frac{\partial}{\partial v^{(k)}} \sum_{i=1}^{K} \sum_{x \in C_i} (v^{(i)} - x)^2 \tag{2.1}$$

$$= \sum_{i=1}^{K} \sum_{x \in C_i} \frac{\partial}{\partial v^{(k)}} (v^{(i)} - x)^2 \tag{2.2}$$

$$= \sum_{x \in C_k} 2(v^{(k)} - x) \tag{2.3}$$

$$\sum_{x \in C_k} 2(v^{(k)} - x) = 0 \;\Rightarrow\; |C_k|\, v^{(k)} = \sum_{x \in C_k} x \;\Rightarrow\; v^{(k)} = \frac{1}{|C_k|} \sum_{x \in C_k} x \tag{2.4}$$

kjer je $|C_k|$ število primerov v skupini C_k. Najboljši centroidi, ki minimizirajo vsoto kvadratnih napak so ravno centri (središnjice) skupin. Če bi vzeli namesto evklidske razdalje Manhattansko, bi s pomočjo podobnega izvajanja kot zgoraj dobili, da so najbolj primerni centri mediane točk v skupini.

Pri zgornjem izvajanju smo se osredotočili le na zmanjševanje oddaljenosti točk v skupini od njihovih centroidov. Pravzaprav pa nas zanima podobnost primerov znotraj skupine oziroma t.im. kohezija, ter oddaljenost primerov med različnimi skupinami, oziroma t.im. ločljivost. Mera kvalitete razbitja, ki uspešno združuje tako kohezijo kot ločljivost je silhuetni koeficient, oziroma silhueta razbitja. Izračunamo ga s sledečim postopkom:

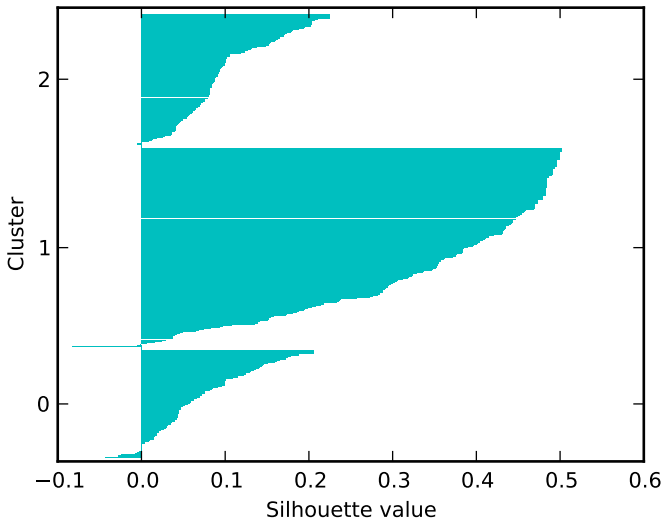
- Naj bo a_i povprečna razdalja primera $x^{(i)}$ do vseh ostalih primerov v njegovi skupini.
- Za primer $x^{(i)}$ in neko skupino C_j; $x_i \notin C_j$, ki je različna od te, ki vsebuje x_i, izračunaj povprečno razdaljo med x_i in primeri v tej skupini. Poišči skupino C_j, kjer je ta razdalja najmanjša. Imenujmo to razdaljo b_i.

- Za primer x_i je njegova silhueta enaka $s_i = (b_i - a_i)/\max(a_i, b_i)$.

- Silhueta razbitja je enaka povprečni silhueti primerov v učni množici:

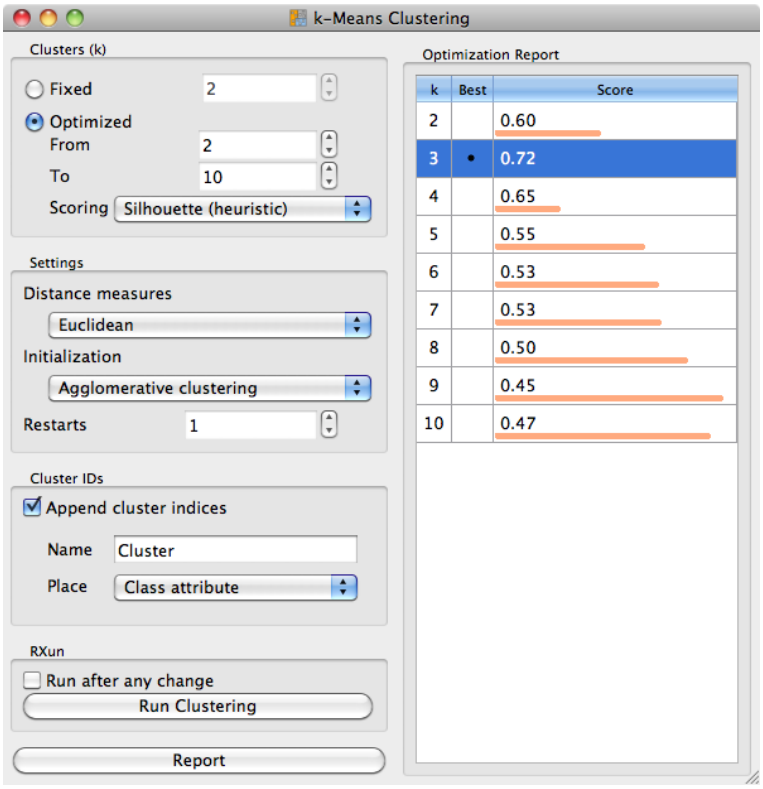
$$s = \frac{1}{|\mathcal{U}|} \sum_{i=1}^{|\mathcal{U}|} s_i$$

Možne vrednosti silhuetnega koeficienta v teoriji ležijo na intervalu med -1 do 1, v praksi pa pričakujemo, da bo za dani primer razdalja do primerov v lastni skupini veliko manjša od razdalj do primerov v najbližji tuji skupini, torej $a_i < b_i$ in zato $s_i > 0$. V primeru, ko so te razlike velike, je vrednost silhuete 1. Silhuete primerov lahko izrišemo za vsako skupino. Uredimo jih od največjega do najmanjšega (za primer, kjer smo razvrstili podatke nabora "voting" v tri skupine, glej sliko **??**). Kateri primeri so tisti, ki imajo kratko silhueto?



Slika 2.3: Silhuete primerov iz podatkovnega nabora "voting" pri razvrstitvi v tri skupine.

Kvaliteto razbitja za dano število skupin K lahko torej ocenimo s povprečno silhueto primerov. To nam omogoča, da ocenimo, kako primerne so različne vrednosti K (glej primer take študije s slike **??**). Hevristika sicer ne deluje idealno, a nam lahko, z malce previdnosti, pomaga, da izberemo primerno število skupin za naše podatke. Previdnost je potrebna predvsem tam, kjer so razlike med silhuetami majhne. V takih primerih nam pomaga razmišljanje o tem, kaj posamezne skupine sploh sestavlja in kakšne so skupne značilnosti primerov v skupinah.



Slika 2.4: Silhuetna študija razvrstitve v različno število skupin za podatke s slike **??**.

2.1 Kombiniranje razvrstitev in razvrščanje s konsenzom

"Več ljudi več ve", ali modrost množic. Tudi na področju razvrščanja je lahko tako. Različni algoritmi, njihovi parametri ali pa rahlo različne učne množice iz iste problemske domene nam lahko pomagajo razkriti skupine, ki so lahko rahlo, ali pa tudi precej različne med sabo. Pri takem nizu razvrstitev bi bilo pametno te skupine nekako zliti in poiskati stabilni del razvrstitev. Če bi na primer dva primera pri večini razvrstitev bila del istih skupin bi to bil dober pokazatelj, da bi bilo dobro ta dva primera tudi sicer razvrstiti skupaj. Po drugi strani pa bi lahko za dva druga primera, ki ju skoraj nikoli nismo našli v skupnih skupinah, rekli, da pač ne sodita skupaj.

Na zgornji ideji temelji algoritem razvrščanja s konsenzom (Monti in sod., 2003), ki ga opišemo s spodnjim algoritmom.

input:
učna množica primerov D
algoritem razvrščanja Cluster in izbrano število skupin K
število vzorčenj H

$M \Leftarrow \emptyset$ množica povezovalnih matrik
for $h = 1, 2, \ldots, H$ **do**
  $D^{(h)} \leftarrow \text{Resample}(D)$ izberemo vzorec iz učne množice
  $M^{(h)} \leftarrow \text{Cluster}(D^{(h)}, K)$ razvrstimo primere iz $D^{(h)}$ v K skupin
  $M \leftarrow M \cup M^{(h)}$
end
$\mathcal{M} \leftarrow \text{ComputeConsensusMatrix}(M)$
$C \leftarrow$ razbitje D v K skupin na podlagi $\mathcal{M}$

Oznaka $M^{(h)}$ označuje povezovalno matriko:

$$M^{(h)}(i,j) = \begin{cases} 1 & = X_i \in C \wedge X_j \in C \\ 0 & = \text{otherwise} \end{cases}$$

Iz množice povezovalnih matrik izračunamo matriko konsenzov:

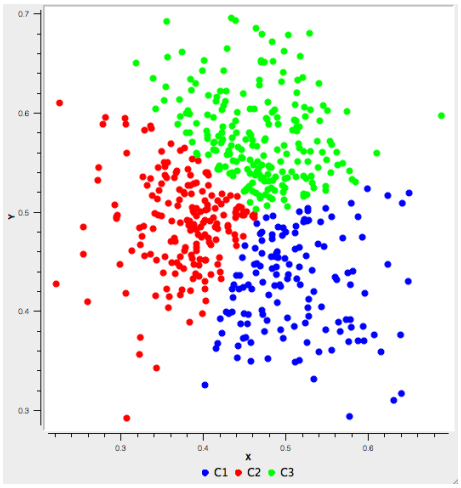
$$\mathcal{M}(i,j) = \frac{\sum_h M^{(h)}(i,j)}{\sum_h I^{(h)}(i,j)}$$

Kjer je I matrika indikatorjev, ki ima za par i in j vrednost 1, če sta oba primera X_i in X_j prisotna v vzorcu $D^{(h)}$, sicer pa vrednost 0. Elementi povezovalne matrike so realna števila na intervalu od 0 do 1. Elementi ustrezajo podobnostim primerov, matrika pa podobnostni matriki. Matrika $\mathbf{1} - \mathcal{M}$ je zato matrika razdalj med primeri, ki jo lahko uporabimo pri hie-

rarhičnem razvrščanju v primere ter na ta način pridobimo možne razvrstitve, ki predstavljajo konsenz delnih razvrstitev na vzorcih iz učne množice.

2.1.1 Permutacijski test

Včasih nam tudi kvantitativna ocena, kot je silhueta, neposredno ne pove prav dosti o tem, ali so v podatkih zares skupine ali pa je algoritem našel neko razbitje, ki pa bi bilo podobne kvalitete tudi, če bi bili podatki naključni oziroma, bolje, taki, kjer bi sicer ohranili porazdelitve vrednosti posameznih atributov a izničili interakcije med atributi. V nadaljevanju bomo zaradi poenostavitve take podatke imenovali naključni podatki.



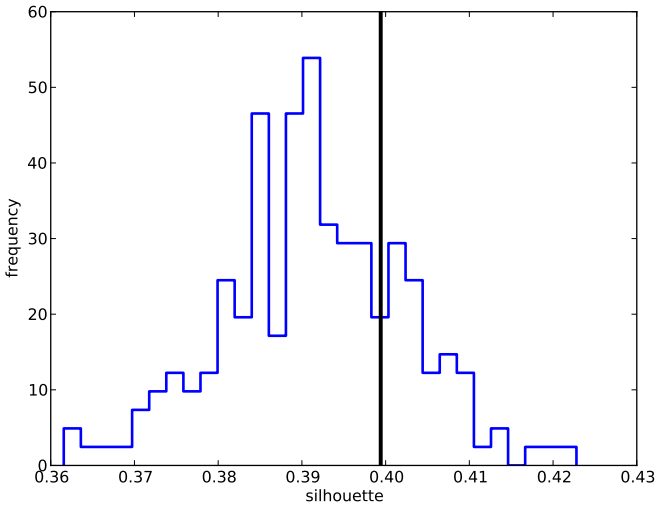
Slika 2.5: Rezultat razbitja s tehniko voditeljev na dvoatributni domeni izgleda lepo strukturiran, vprašanje pa je, ali smo res odkrili tri značilne skupine. Silhueta tega razbitja je 0.50 in je bila le rahlo različna od silhuete za ostala razbitja na dve do deset skupin, kjer je bila silhueta od 0.44 do 0.49.

Naša hipoteza, ki jo preizkušamo in jo v statistiki imenujemo ničelna hipoteza, je (H_0): silhueta s ima vrednost, kot bi jo dobili, če bi bili podatki naključni.

Zgornjo hipotezo je enostavno računsko preveriti. Generirajmo naključne podatke. To storimo tako, da v naših podatkih za vsak atribut naključno premešamo njihove vrednosti po primerih. Ker so podatki podani v tabeli, je to isto, kot če vrednosti vsake kolone med sabo premešamo. (Je to res potrebno narediti za vse kolone oziroma atribute?) Na teh podatkih uporabimo tehniko razvrščanja in izmerimo silhueto. To ponovimo, na primer vsaj 100-krat ali pa bolje 1000-krat. Na ta način dobimo porazdelitev vrednosti naše statistike s, kot bi jo dobili, če bi ta izhajala iz naključnih podatkov. Izmerimo sedaj to isto statistiko na nepremešanih, originalnih podatkih in jo primerjamo z ničelno porazdelitvijo.

Zgornji postopek smo uporabili na podatkih, ki so bili podobni tem iz slike **??**. Za oceno ničelne porazdelitve smo izračunali 200 silhuet, vsakič na podatkih s premešanimi vrednostmi

atributov. Ugotovimo, da je kar 23% vseh vrednosti iz ničelne distribucije večje od naše opazovane statistike na originalnih podatkih. Če bi trdili, da je naša ničelna hipoteza H_0 napačna, bi na podlagi naših eksperimentov verjetnost napačne zavrnitve te hipoteze bila $p = 0.23$. V statistiki je to veliko prevelik prag za zavrnitev ničelne hipoteze. Tipično je sprejemljiva meja tega praga enaka $\alpha = 0.01$, včasih celo $\alpha = 0.05$.

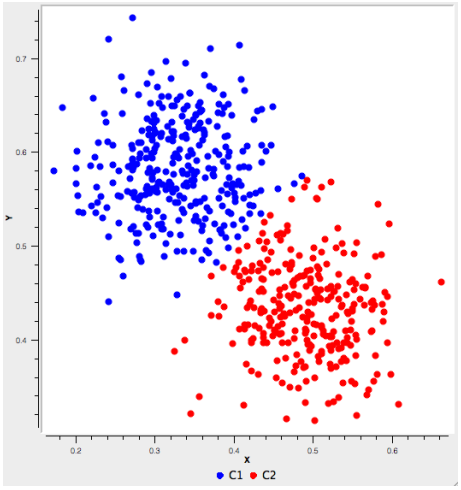


Slika 2.6: Porazdelitev silhuet na permutiranih podatkih in projekcija (ravna črta) silhuete na originalnih podatkih kaže na to, da, vsaj za izbrano število skupin, za te podatke nismo našli smiselnega razbitja.

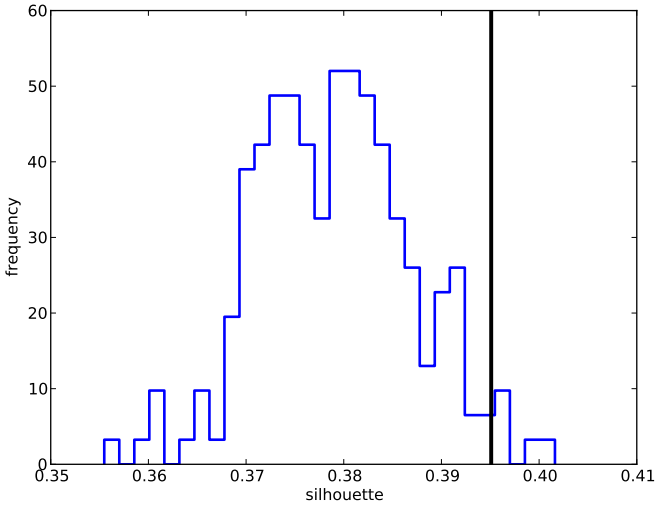
Za konec si poglejmo še malce drugačne podatke (slika **??**). Tu so rezultati smiselni oziroma tako vsaj zgledajo. Podatki dejansko nakazujejo, da bi lahko imeli dve skupini. To preverimo s permutacijskim testom (slika **??**). Ničelno hipotezo zavrnemo z verjetnostjo, da smo se pri tem zmotili $p = 0.025$.

2.2 Predobdelava podatkov

Do tu smo predpostavili, da imajo atributi v naši podatkovni množici isto, zvezno, zalogo vrednosti. Taka lastnost podatkov je med temi, ki jih uporabljamo pri reševanju praktičnih problemov, le redka. Navadno z atributi zapišemo različne lastnosti primerov, ki nastopajo v različnih merskih enotah. Na primer dolžina, teža, višina, ocena v neki ocenjevalni lestvici, pa rezultati na različnih testih, in podobno. Mere razdalje, kot je Evklidska, ne ločijo med različnimi merskimi lestvicami, zato je potrebno podatke spremeniti tako, da bodo vrednosti atributov med seboj primerljive. To storimo z normalizacijo. In sicer, za vsak atribut j



Slika 2.7: Rezultat razbitja je tu morda smiseln. Silhueta za $K = 2$ je 0.68, za ostale vrednosti K pa pod 0.56. Rezultate bi bilo dobro preveriti s permutacijskim testom.



Slika 2.8: Porazdelitev silhuet na permutiranih podatkih in projekcija (ravna črta) silhuete na originalnih podatkih s slike **??**.

določimo najprej njegovo povprečno vrednost:

$$\mu_j = E[X_j] = \frac{1}{N}\sum_{i=1}^{N} X_{ij}$$

kjer je X_{ij} vrednost j-tega atributa za i-ti primer. Podobno določimo tudi standardni odklon atributa:

$$\sigma_j = \sqrt{E[(X_j - \mu_j)^2]} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_{ij} - \mu_j)^2}$$

Podatke zdaj normaliziramo tako, da so nove vrednosti v tabeli podatkov Z_{ij} enake:

$$Z_{ij} = \frac{X_{ij} - \mu_j}{\sigma_j}$$

Poglavje 3

Razvrščanje besedil

V prejšnjih dveh poglavjih so predlagane tehnike razvrščanja predpostavljale, da lahko razdaljo, ali pa podobnost med primeri enostavno izračunamo. Vsi naši primeri so bili opisani atributno in razdalja je bila lepo določena z, na primer, razdaljo v evklidskem prostoru. Podatke smo morali predobdelati in vsakega od atributov normalizirati, drugih, bolj kompleksnih priprav podatkov pa postopki razvrščanja niso zahtevali.

Drugače je s tekstovnimi dokumenti, ali besedili. Vzemimo na primer, da moramo v skupine urediti naslednje tri novice:

> Na odru ljubljanske Drame se drevi ustavlja prva slovenska godba na pihala, ki že več kot 15 let preigrava skoraj izključno ulični džez New Orleansa. Cerkljanski Kar Češ Brass Band, ki neworleanški džez interpretira na svojevrsten in svež način, so v gledališče povabili v sklopu cikla Drama Akustika.

> Tretji oktobrski konec tedna na Ravnah na Koroškem zdaj že tradicionalno poteka Festival slovenskega jazza, katerega spremljajoči dogodki se na Koroškem vrstijo že od septembra. Tridnevni festivalski vrhunec je v Kulturnem centru Ravne odprl Big Band RTV Slovenija z vsestranskim glasbenikom Boštjanom Gombačem.

> Murskosoboški policisti so 33-letniku z območja Murske Sobote zasegli posušeno konopljo, sadike in gotovino, za katero sumijo, da jo je dobil s preprodajo. Pri tem so v hiši osumljenega odkrili poseben prostor za gojenje konoplje pod umetno svetlobo. V tem prostoru so našli in zasegli tudi 20 sadik konoplje, visokih do 90 centimetrov, in dober kilogram posušene oz. delno posušene konoplje.

Nam, ljudem, je ta razvrstitev enostavna. Zadnja, tretja novica, sodi v črno kroniko, prvi dve pa med kulturne novice.

Programsko razvrščanje novic s tehnikami iz prejšnjega poglavja pa ni trivialno. Očitno bomo morali določiti nove mere, ki nam bodo pomagale oceniti, kako različna so si med sabo besedila. Pristopi k razvrščanju dokumentov, ki jih predlagamo v tem poglavju, temeljijo na preoblikovanju besedil v atributno obliko, kjer je moč take mere določiti, da bi nato za

razvrščanje uporabili tehnike, kot sta hierarhično razvrščanje v skupine in metodo voditeljev, ki jih že poznamo.

3.1 Elementi predstavitve besedilnih dokumentov

3.1.1 *k*-terke znakov

"Murskoboški" lahko predstavimo z dvojkami "mu", "ur", "rs", ..., torej s pari znakov, ki si sledijo v zaporedju. Za vsak par lahko izračunamo frekvenco pojavitve v besedilu oziroma relativno frekvenco, ki je enaka ocenjeni verjetnosti pojavitve *k*-terke.

Število *k*-terk je veliko že za pare ($n = 2$), za večje vrednosti k pa jih je mnogo ali ogromno. V praksi se uporabljajo *k*-terke do $n = 5$. Zanimivo pa je, da je že porazdelitev dvojk lahko karakteristična za posamezne jezike in lahko na podlagi teh razpoznamo, v katerem jeziku je napisano določeno besedilo[1]. Za bolj zahtevne naloge je potrebno seveda uporabiti daljše *k*-terke.

Ta predstavitev je načelno enostavna, a jo precej oteži uporaba posebnih znakov, ki jih moramo ali upoštevati ali pa primerno predobdelati besedilo.

3.1.2 Besede

Dokumente lahko predstavimo z vrečo besed (angl. *bag of words*), to je s skupino besed in številom njihovih pojavitev v dokumentu. Pri tem lahko besedilo najprej predobdelamo:

- odstranimo manj pomembne besede (angl. *stop-words*), ki so za slovenščino na primer "in", "ali", "ter", ipd.
- vse besede nadomestimo z njihovimi koreni ali pa lemami. V računalništvu je lematizacija algoritmični postopek določevanja leme določeni besedi. Postopek ni enostaven in je tipično odvisen od jezika, v katerem je zapisano besedilo. Za angleščino je na primer znan Porterjev iskalnik korenov besed (angl. *Porter stemmer*), ki je sestavljen iz ročno izdelanih pravil[2].

Predstavitev z vrečo besed zanemarja dejstvo, da je pomen besed mnogokrat odvisen od konteksta. Ista beseda ima lahko različne pomene, ali pa imajo isti pomen lahko različne besede. Prav tako nam taka predstavitev ne bo mogla razrešiti problema podpomenk in drugačnih povezav med različnimi izrazi.

Število pojavitev besed v dokumentu je seveda odvisno od dolžine dokumenta. Da ta vpliv izničimo, namesto števila pojavitev besed uporabljamo relativno frekvenco, to je verjetnost, da bi pri naključnem izboru besede iz dokumenta izbrali določeno besedo.

[1] Glej `www.lingua-systems.com/language-identifier/lid-library/identify-language.html`

[2] Implementacija Porterjevega korenjenja v različnih programskih jezikih je dostopna na `http://tartarus.org/martin/PorterStemmer`

3.1.3 Fraze

Fraze so lahko *k*-terke besed, ki se v besedilu nahajajo neposredno druga ob drugi ali pa v bližnji okolici. Okolico lahko določa, na primer, okno zaporedja petih besed. Težava pri tej predstavitvi je, da je lahko možnih kombinacij že za pare besed ogromno. Pred leti je Google objavil svojo zbirko fraz [3], ki jih je odkril iz besedilnih dokumentov. Vsebovala je na primer 314.843.401 par besed, 977.069.902 trojki, 1,313,818,354 četvorčkov in 1,176,470,663 fraz s petimi besedami.

Uporaba fraz je smiselna ob souporabi fraznega korpusa, to je, baze že znanih oziroma izbranih fraz za določeno problemsko področje. Predstavitev z vrečo besed je načelno ustrezno moč dopolniti s frazami iz besedila, tako da vsaka fraza tvori nov atribut.

3.1.4 Besedne vrste

Koristi nam lahko tudi prepoznavanje besednih vrst (angl. *part-of-speech*), kot so imena ljudi, krajev, organizacij. Za prepoznavanje besednih vrst bomo morali uporabiti algoritme, ki znajo besedilo analizirati veliko globlje in pri tem uporabljajo lingvistično znanje ali pa vnaprej pripravljeni korpus. Seveda bo taka analiza tudi odvisna od uporabljenega jezika. Pri katerih besedah v spodnjem besedilu bi določitev besednih vrst lahko bila koristna pri analizi?

```
We walk down the dirty old street.
The mailbox stood on the walk.
We heard cries echoing in the night.
The babied cries all night.
```

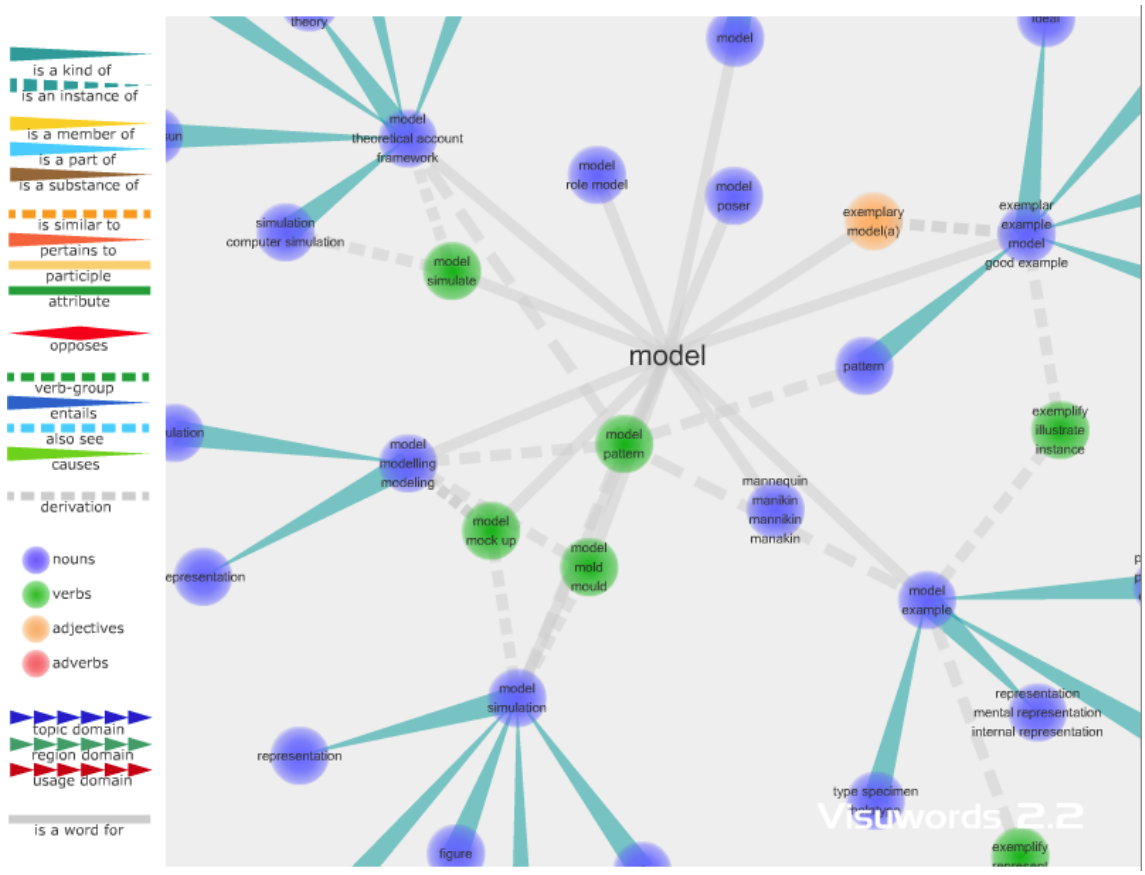
3.1.5 Taksonomije

V splošnem se taksonomija nanaša na stopenjsko razvrstitev stvari. Pomaga nam pri ugotavljanju povezanosti med stvarmi, pri bolj podrobnih taksonomijah pa iz njih zvemo še za vrsto relacije. Primer take taksonomije za angleške besede je WordNet [4]. Odlična vizualizacija povezav med besedami, kjer je prikazan tudi pomen povezav, je dostopna na spletni strani Visuwords [5] (slika **??**). Enostavnejša oblika zapisa take taksonomije je slovar sopomenk ali tezaver [6].

3.1.6 Slovnična analiza

K računalniškem razumevanju besedila bi seveda zelo pripomogla slovnična analiza. Uporabe tovrstnega globljega razumevanja besedila pri razvrščanju dokumentov je predmet velikega

[3] Glej `http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html`.
[4] WordNet je prosto dostopen na naslovu `http://wordnetweb.princeton.edu`.
[5] `http://www.visuwords.com/`
[6] Glej npr. `http://www.tezaver.si/`.

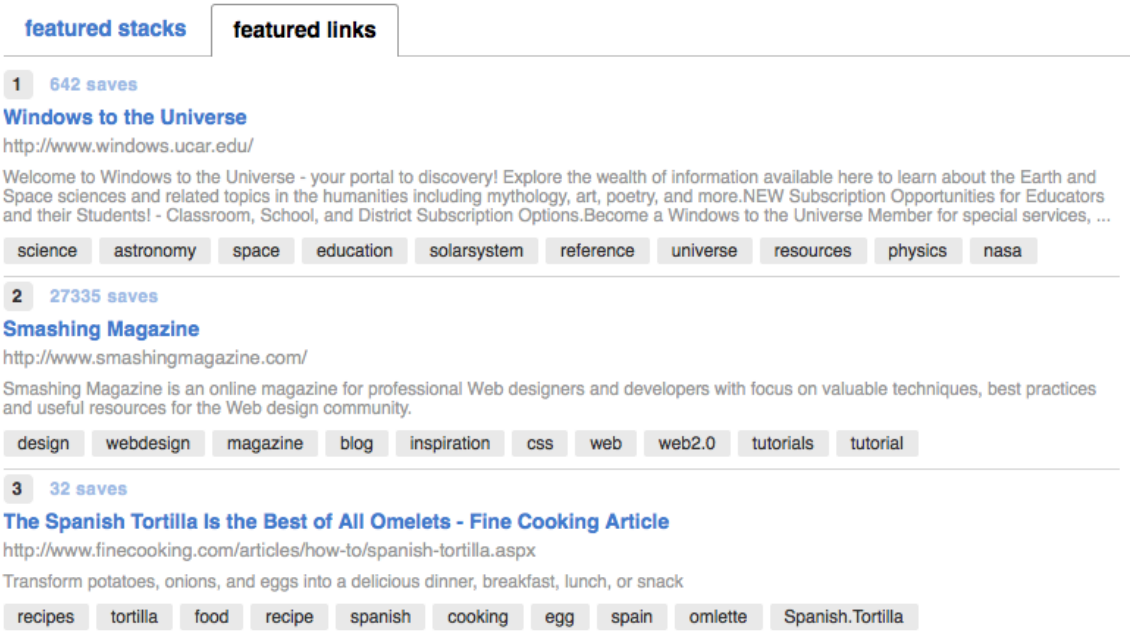


Slika 3.1: Del taksonomije okoli angleške besede “model”, kot jo prikaže spletna aplikacija Visuwords.

števila trenutnih raziskav. Prav možno je, da bodo v prihodnosti te, bolj kompleksne jezikovne tehnologije nadomestile plitvejše, statistične tehnike ki uporabljajo štetje parov črk ali preštevanje besed. A je prav enostavnost slednjih in točnost, ki jo omogočajo že ti izjemno preprosti pristopi ena od ovir za prevlado bolj kompleksnih tehnologij.

3.1.7 Uporaba pripadajočih oznak

Mnogo dokumentov je danes označenih. Naj bodo to spletne strani (npr. `del.icio.us`), dokumenti v raznih zbirkah, zapisi v programu EverNotes, seznami bibliografskih enot v skladiščih povzetkov CiteULike[7] ali PubMed[8]. Oznake so lahko bile prosto izbrane, ali pa vzete iz posebej za to določenega slovarja ali pa ontologije. Dokumente lahko označujejo vsi, ki imajo do njih dostop, ali pa samo pooblaščeni uporabniki ali kuratorji. V vsakem primeru nam oznake lahko služijo kot osnova za predstavitev vsebine dokumentov in zamenjujejo ali pa dopolnjujejo elemente, kot smo jih predstavili zgoraj.



Slika 3.2: Oznake spletnih strani, kot so jih uporabniki določili v družabnem okolju del.icio.us.

3.2 Ocenjevanje podobnosti med dokumenti

V tem besedilu se bomo omejili na najpreprostejšo obliko predstavitve, kjer besedilni dokument predstavimo z vektorjem enot in njihovo frekvenco ali relativno frekvenco. Nestruktu-

[7] `www.citeulike.org`
[8] `http://www.ncbi.nlm.nih.gov/pubmed/`

riran dokument torej predstavimo z atributnim jezikom. Ker je atributov lahko zelo veliko, in ker vsi dokumenti nimajo nujno določene vse atribute, je matrika z dokumenti v vrsticah in vrednosti atributov v kolonah zelo prazna. Namesto te predstavitve lahko zato uporabimo slovar, z elementi kot ključi in njihovimi frekvencami kot vrednostmi.

3.2.1 Transformacija tf-idf

Pogostost elementa (besede, terke) v dokumentu je enostavno število pojavitev tega elementa v dokumentu. Če bi uporabljali to število, bi prihajalo do večjih razlik med daljšimi in krajšimi dokumenti. Zato pogostost pojavitve normaliziramo in namesto njih uporabljamo relativne frekvence, oziroma iz dokumenta ocenimo verjetnosti pojavitve določenega elementa. To označimo z $\text{tf}(t, d)$ (angl. *term frequency*).

Načelno imamo raje elemente, ki se pojavljajo v manj dokumentih in so zaradi tega bolj specifični. Na osnovi elementov, ki se pojavijo v večini dokumentov, bi te zelo težko razlikovali. Zato uvedemo pojem inverzne frekvence v dokumentih (angl. *inverse document frequency*), ki jo uporabljamo kot splošno mero za pomembnost določenega elementa:

$$\text{idf}(t) = \log \frac{|D|}{|\{d : t \in d\}|}$$

kjer je $|D|$ število dokumentov v množici dokumentov D in $|\{d : t \in d\}|$ število dokumentov, ki vsebujejo element t (torej, kjer je $\text{tf}(t, d) \neq 0$). Če elementa ni v korpusu, bi zgornje vodilo k deljenju z 0. Zato lahko zgornji izraz prilagodimo in zapišemo $1 + |\{d : t \in d\}|$.

Utež elementa t v danem dokumentu d je potem zmnožek njegove frekvence v tem dokumentu in splošne pomembnosti tega elementa:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t)$$

3.2.2 Kosinusna podobnost

Čeprav bi za merjenje razdalj med vektorji, ki predstavljajo dokumente, lahko uporabljali tudi Evklidsko razdaljo (glej prejšnje poglavje), se ta na področju obravnave besedilnih dokumentov ne uporablja. Razlog je preprost. Imejmo dva vektorja, X in Y in predpostavimo, da sta zelo različnih velikosti, a da kažeta v isto smer. Evklidska razdalja med njima je lahko večja kot med vektorjema, ki bi bila kratka, a kazala v popolnoma različne smeri. Na področju besedilnih dokumentov smer vektorja ustreza specifičnemu profilu dokumenta v smislu vsebovanosti posameznih elementov. Pomembna je smer, kamor kaže vektor, in manj (ali pa čisto nič) njegova dolžina. Razliko med smerjo dveh vektorjev lahko merimo s kotom med vektorji, ta pa je proporcionalna kosinusu kotov. Za dva vektorja $\mathbf{a}$ in $\mathbf{b}$ velja:

$$\mathbf{a} \cdot \mathbf{b} = ||\mathbf{a}|| \, ||\mathbf{b}|| \cos \theta$$

Podobnost med primeroma X in Y lahko zato zapišemo kot:

$$\text{sim}(X, Y) = \cos(\theta) = \frac{X \cdot Y}{||X||\ ||Y||} = \frac{\sum_{i=1}^{m} X_i \times Y_i}{\sqrt{\sum_{i=1}^{m} (X_i)^2} \times \sqrt{\sum_{i=1}^{m} (Y_i)^2}}$$

3.2.3 Podobnost po Jaccardu

Kadar imamo opraviti z oznakami (torej, ne z besedami iz besedila, pri katerih poleg vsebnosti opazujemo tudi število pojavitev) je poleg kosinusne razdalje smiselno opazovati tudi podobnost po Jaccardu:

$$J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

Tu so dokumenti predstavljeni z množico oznak, kjer je $X \cup Y$ množica oznak uporabljena pri vsaj enem od obeh dokumentov, $X \cap Y$ pa množica oznak, ki so skupne obema dokumentoma.

Poglavje 4

Linearna regresija

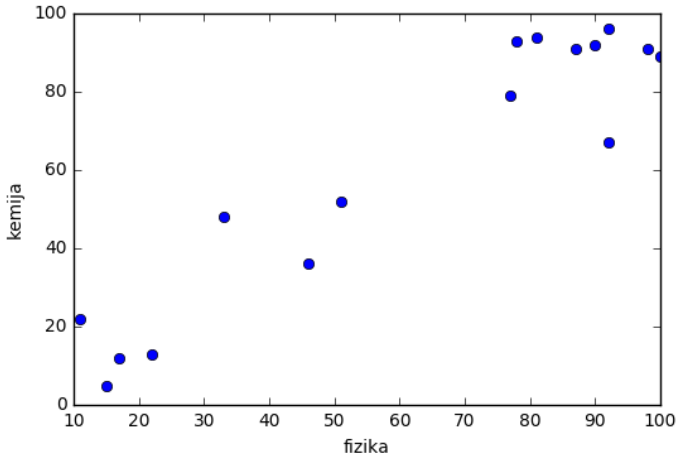
Vinkove rezultate iz kemije so založili. Enostavno, komisija je izgubila izpitne pole. Rešitev: Vinko bo kemijo pisal še enkrat. Ampak, ne more, je ravno odšel na trening odbojke v Rio. Ravnatelj pravi, da je tako ali tako vseeno, ker da eksterci ne štejejo. Bi pa razredničarka silno želela imeti to oceno, da lahko Vinka uvrsti v primerno učno skupino. V zbornici je učiteljica fizike rekla, da so ocene iz kemije tako ali tako podobne tem iz fizike (slika **??**). Zamrmrala je še nekaj okoli regresijskih modelov in premic, potem pa odšla na mete Galilejeve krogle iz strehe. Tisti dan je niso več videli.

Tabela 4.1: Rezultati zunanjega preverjanja za 16 učencev razreda 8.A iz fizike in kemije.

ime	fiz	kem
Albert	46	36
Branka	11	22
Cene	100	89
Dea	90	92
Edo	17	12
Franci	98	91
Helena	81	94
Ivan	33	48
Jana	87	91
Leon	77	79
Metka	78	93
Nika	15	5
Polona	22	13
Rajko	51	52
Stane	92	96
Zala	92	67

Vinko je fiziko pisal 70. Razredničarka je lepo prosila za pomoč učiteljico matematike, ki

je bila tisti dan posebej razpoložena. Začnimo z grafom, pravi. Tistim z ocenami iz fizike in kemije (slika **??**).



Slika 4.1: Ocene fizike in kemije na razsevnem diagramu.

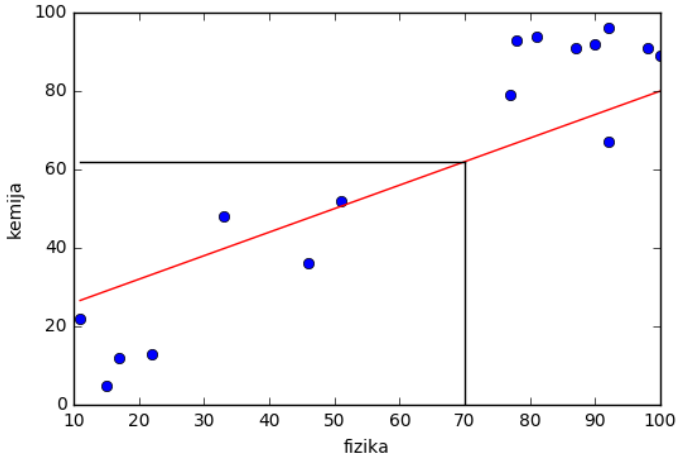
4.1 Linearni model ene spremenljivke in kriterijska funkcija

Učiteljica fizike ima kot kaže prav, ocene kemije in fizike so med sabo nekako povezane. Tudi učiteljica matematike ima prav: če se le da skušamo podatke najprej predstaviti v kakšnem grafu. Od tu dalje bomo poskušali sami. Morda začnemo s premico, ki ji bomo rekli kar model. Model zato, ker lahko za vsako oceno pri fiziki na podlagi modela napovemo oceno pri kemiji. Da bo vse skupaj zgledalo bolj učeno in primerno za univerzitetni študij, označimo oceno pri fiziki s spremenljivko x, oceno pri kemiji, ki jo računamo na podlagi ocene iz fizike in jo v našem modelu modeliramo pa z y. Ocena kemije je funkcija ocene iz fizike, zato lahko zapišemo:

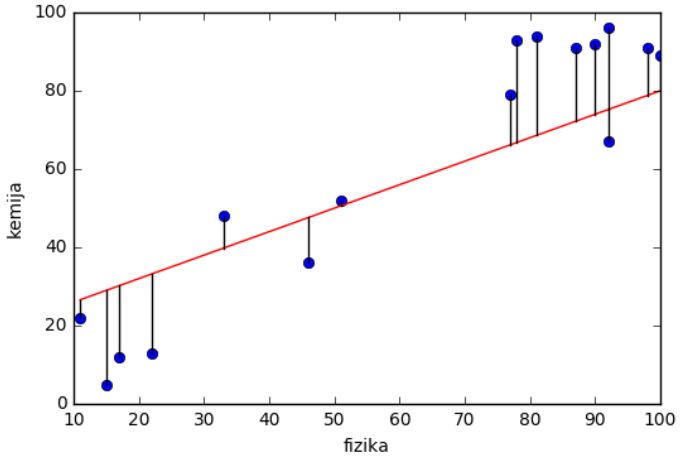
$$y = f(x) \tag{4.1}$$

Rekli smo, da bomo zadeve poenostavili in da bo naš model kar premica. Nekaj takega, kot kaže slika **??**. Na njej bi iz ocene fizike 70 ocenili, da bo ocena kemije znašala 62. Ampak, ali je premica, ki jo kaže slika, res tista "prava"? Obstaja kakšna boljša premica, kakšen boljši model? Kako pa sploh ocenimo kvaliteto modela?

Za vsako točko v grafu, vsak znan podatek, torej za vsak par vrednosti $(x^{(i)}, y^{(i)})$ lahko izračunamo napako, ki jo dobimo, ko vrednost odvisne spremenljivke y napovemo z modelom. Napoved označimo z $\hat{y}$ in za primer i izračunamo napako napovedi $\epsilon^{(i)} = \hat{y}^{(i)} - y^{(i)} = f(x^{(i)}) - y^{(i)}$. Vse napake za naš dani model in primere v učni množici smo označili na sliki **??**.



Slika 4.2: Linearni model, ki iz ocene fizike izračuna oceno za kemijo. Za Vinkovo oceno iz fizike 70 predvidi, da je ocena pri kemiji enaka 62. Pravilno?



Slika 4.3: Napake linearnega modela na učni množici.

Napake $\epsilon^{(i)}$ so lahko pozitivne ali negativne. Pravzaprav nas predznak ne zanima, zanima nas samo velikost napake. In če že, nas ne zanimajo tiste majhne napake, ampak nas motijo predvsem tiste večje, recimo napake pri Zali in Albertu. Radi bi, da bi naš model bil tak, da bi napake, ki ga naredi pri napovedi primerov iz učne množice bile čim manjše. Razmišljanje iz tega odstavka lahko kvantificiramo, oziroma izrazimo numerično s funkcijo:

$$J = \frac{1}{2m}\sum_{i=1}^{m}\epsilon^{(i)2} = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{2m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)})^2 \tag{4.2}$$

Funkcijo J imenujemo *kriterijska funkcija*, ker z njo izrazimo našo preferenco, kakšen model bi radi imeli. Enačba za J je povprečna kvadrirana napaka. Povprečna zato, da lahko njeno vrednost primerjamo nad različno velikimi učnimi množicami. Kvadrirana zato, ker nam je vseeno, ali so napake pozitivne ali negativne in zato, da izpostavimo večje napake. Tisto dvojko v $\frac{1}{2m}$ smo tu dodali kar tako (ne škodi), kasneje pa se izkaže, da se nam pri kakšni operaciji okrajša in nam pride prav pri poenostavitvi rezultatov.

Radi bi, da bi bila vrednost kriterijske funkcije J čim manjša. Najbolje nič. A to vsaj pri naši učni množici z modelom - premico, ki mu od tu dalj lahko kar rečemo linearni model, ne bo šlo (če misliš pa, da gre, poskusi).

Premico lahko zapišemo z enačbo:

$$y = f(x) = ax + b \tag{4.3}$$

S tem, ko poznamo model, naša kriterijska funkcija postane funkcija dveh parametrov, parametra a in parametra b:

$$J(a, b) = \frac{1}{2m}\sum_{i=1}^{m}((ax^{(i)} + b) - y^{(i)})^2 \tag{4.4}$$

Da povzamemo: oceno iz kemije bomo za Vinka ocenili tako, da bomo zgradili model, ki iz znanih ocen fizike in kemije pridobi model. Model bo tak, da bo za učno množico, torej za učence, kjer poznamo ocene iz obeh predmetov, iz ocene fizike izračunal oceno kemije. Želeli bi tak model, ki je na učni množici čimbolj natančen. Natančnost ocenimo s kriterijsko funkcijo J, katere vrednost izračunamo iz učnih podatkov in modela. Kriterijska funkcija je funkcija parametrov modela. Ker gradimo linearni model in ker je naš model premica v ravnini, sta parametra dva, a in b. Želeli bi torej pridobiti taka parametra, pri katerih je vrednost kriterijske funkcije najmanjša. Iskanju takih parametrov pravimo optimizacija.

4.2 Razmislek o optimizaciji

Tu se bomo delali, da o optimizaciji še nimamo pojma. (Čeprav to prav gotovo ni res, saj smo pri matematiki in nekaterih ostalih predmetih veliko slišali o njej. A nič hudega, da zadevo

ponovimo). Recimo da imamo funkcijo $J(a)$, torej funkcijo parametra a, in želimo poiskati vrednost parametra a^*, kjer je vrednost funkcije najmanjša. Dodatno recimo, da vemo, da je funkcija konveksna. Za konveksne funkcije velja, da so zvezne in da za vsak interval njene domene (vrednosti parametra a, za naš primer) velja, da vrednost funkcije v srednji točki intervala ne presega aritmetičnega povprečja vrednosti funkcije v skrajnih točkah intervala. Po domače, konveksna funkcija enega parametra ima obliko črke U. Recimo tudi, da funkcije $J(a)$ nimamo zapisane analitično, a da lahko izvemo oziroma povprašamo po njeni vrednost za dano vrednost parametra.

Naše iskanje minimuma funkcije $J(a)$ lahko začnemo v neki točki. Recimo, v točki nič. V tej točki nas pravzaprav raje kot njena vrednost zanima njen odvod. Če je odvod enak nič (ali pa zelo blizu ničle), potem vemo, da smo našli pravo vrednost parametra a. Če je odvod pozitiven, vemo da smo s parametrom a desno od optimalne vrednosti a^*, in da je $a^* < a$. Če je odvod funkcije $J(a)$ negativen, bo vrednost $a*$ morala biti večja od trenutne vrednosti a. Odvoda funkcije J, torej $dJ(a)/da$ nimamo zapisanega v analitični obliki. Odvod pri izbrani vrednosti parametra a ni nič drugega kot naklon funkcije v tej točki, to pa lahko približno izračunamo tako, da pogledamo, kakšne so vrednosti te funkcije malce stran od točke a, na levo in na desno:

$$\left.\frac{\Delta J(a)}{\Delta a}\right|_a = \frac{J(a+\delta) - J(a-\delta)}{2\delta} \tag{4.5}$$

Ko vemo za odvod $J(a)$, lahko našo trenutno rešitev, torej začetno točko a, premaknemo v nasprotni smeri odvoda. Še enkrat, v nasprotni zato, ker iščemo minimum. Torej

$$a \leftarrow a - \alpha \left.\frac{dJ(a)}{da}\right|_a \tag{4.6}$$

Vrednost α nam določa, za koliko se premaknemo. Tipično lahko izberemo neko majhno vrednost, recimo $\alpha = 0.1$.

Pythonovska implementacija iskanja optimalne vrednosti bi lahko torej bila nekako takšna:

```
def derivative(f, a, delta=1e-3):
    return (f(a+delta) - f(a-delta)) / (2*delta)

a = 8
for _ in range(10):
    a = a - 0.2 * derivative(J, a)
    print("%.2f" % a)
```

Tokrat smo se odločili, da naše iskanje optimalne vrednosti a^* pričnemo pri $a = 8$ in da je $\alpha = 0.2$. Za funkcijo

```
def J(a):
    return (a-5)**2+3
```

Dobimo naslednji izpis:

```
6.80
6.08
```

```
5.65
5.39
5.23
5.14
5.08
5.05
5.03
5.02
```

Kar je kar fino, saj smo že v desetih korakih prišli precej blizu prave vrednosti a^*. Pravi vrednosti bi se lahko, zaradi konveksnosti naše kriterijske funkcije, s primernim številom iteracij in s primerno vrednostjo α poljubno približali.

Pri zgornji optimizaciji smo predpostavili, da odvoda funkcije ne poznamo. Če bi imeli na voljo funkcijo, ki bi nam neposredno izračunala odvod, torej tako, da bi ga izračunala brez klica osnovne funkcije, bi nam bilo še enostavneje in odvoda potem ne bi rabili ocenjevati. Za dano funkcijo $J(a)$ v zgornjem primeru bi tako lahko izračunali njen analitični odvod, ter tega uporabili v zapisu funkcije `derivative`. Bi znal ustrezno spremeniti kodo? Dobiš podobne rezultate, kot jih dobimo s približkom za odvod?

4.3 Iskanje parametrov univariatne linearne regresije

Tako učeno pravimo modelu - premici, s katerim bomo napovedovali ocene kemije iz ocen fizike. Univariatne zato, ker je to linearna regresija nad enim samim atributom x (atribute imenujemo tudi variate). Linearna zato, ker je povezava med atributi in vrednostjo modela linearna ($ax+b$). Regresija zato, ker je izhod modela zvezna vrednost (drugačni od regresijskih modelov bodo klasifikacijski, ki še pridejo na vrsto).

Pri linearni regresiji iščemo take parametre regresije, torej parametre modela, ki nam dajo minimalno vrednost kriterijske funkcije J. V prejšnjem razdelku smo si pogledali trik, kako tako vrednost poiščemo v primeru, ko je J funkcija enega samega parametra. A pri univariatni analizi je ta funkcija odvisna od dveh parametrov, torej $J = J(a, b)$. Zato bomo iskanje optimalne vrednosti parametrov a^* in b^* pričeli pri neki vrednosti teh parametrov, potem pa te vrednosti popravljali, pač glede na odvod po vsakem od parametrov. Lahko torej zapišemo:

$$a \leftarrow a - \alpha \frac{dJ(a,b)}{da}\Big|_{a,b} \tag{4.7}$$

$$b \leftarrow b - \alpha \frac{dJ(a,b)}{db}\Big|_{a,b} \tag{4.8}$$

Pravzaprav kakšne večje spremembe glede na stvari iz prejšnjega razdelka ni, dela pa je le malce več, ker moramo izračunati odvod po enem in drugem parametru. Odvod bi lahko izračunali tudi numerično, tako, kot smo to počeli zgoraj, a ker kriterijsko funkcijo

poznamo, ne bo odveč, da odvod izračunamo analitično. Upoštevamo, da je odvod vsote enak vsoti odvodov. Spomnimo, odvajamo torej kriterijsko funkcijo iz enačbe **??**, njena odvoda po parametrih a in b sta:

$$\frac{\partial J(a,b)}{\partial a} = \frac{1}{m}\sum_{i=1}^{m}((ax^{(i)}+b)-y^{(i)})x^{(i)} \tag{4.9}$$

$$\frac{\partial J(a,b)}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}((ax^{(i)}+b)-y^{(i)}) \tag{4.10}$$

Pri odvajanju smo izgubili konstanto 2 v imenovalcu normalizacijskega člena $\frac{1}{2m}$. Omenimo samo, da bi lahko odvoda zložili v vektor in s tem dobili nekaj, čemur pravimo gradient. Ne se ustrašiti, gradient je čisto preprosta zadeva: funkcijo z več parametri smo vsakič odvajali po posameznem parametru, in s tem dobili vektor, ki ga imenujemo gradient funkcije.

Rezultat zgleda zanimivo enostaven, a je bila, navkljub seštevanju, taka tudi naša kriterijska funkcija. Izpišimo sedaj korak osveževanja vrednosti parametrov a in b, ki jih uporabljamo pri iskanju našega linearnega modela:

$$a \leftarrow a - \frac{\alpha}{m}\sum_{i=1}^{m}((ax^{(i)}+b)-y^{(i)})x^{(i)} \tag{4.11}$$

$$b \leftarrow b - \frac{\alpha}{m}\sum_{i=1}^{m}((ax^{(i)}+b)-y^{(i)}) \tag{4.12}$$

Pythonovska koda za ta osvežitveni korak je preprosta in predvideva, da so podatki shranjeni v seznamih (vektorjih) `x` in `y`:

```
def update(a, b, alpha=0.0001):
    cons = alpha / len(x)
    a = a - cons * sum(((a*xi+b)-yi)*xi for xi, yi in zip(x, y))
    b = b - cons * sum(((a*xi+b)-yi) for xi, yi in zip(x, y))
    return a, b
```

Če našo optimizacijo pričnemo v točki $(0,0)$ je konvergenca relativno hitra in se do rešitve prebijemo že po nekaj iteracijah:

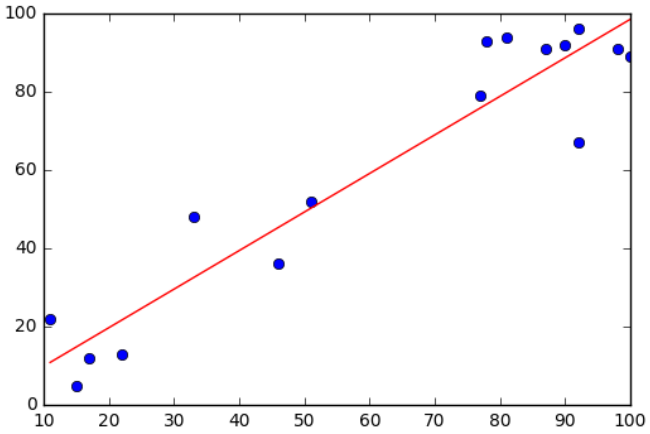
```
a, b = 0, 0
for _ in range(10):
    a, b = update(a, b)
    print("%.3f %.3f" % (a, b))
```

Zgornja koda namreč izpiše:

```
0.479 0.003
0.726 0.005
0.852 0.006
0.918 0.006
0.951 0.006
```

```
0.968 0.006
0.977 0.007
0.982 0.007
0.984 0.007
0.985 0.007
```

Optimalni vrednosti parametrov a in b sta torej (približno) $a = 0.985$ in $b = 0.007$, naš model s podatki iz učne množice pa je prikazan na sliki **??**.



Slika 4.4: Ocene fizike in kemije na razsevnem diagramu.

4.4 Multivariatna linearna regresija

Približek ocene za kemijo izračunan iz ocene za fiziko je sicer čisto v redu, a našo razredničarko in matematičarko mučijo ostale ocene. Namreč, zgubila se je ocena za kemijo, vse ostale ocene pa imamo. Ne samo za fiziko, ampak za slovenščino, matematiko, in ostale. Bi lahko zgradili linearni model, ki bi upošteval tudi ostale ocene? Torej, upošteval prav vse ostale atribute, ki so nam na razpolago. In zgradil linearni model.

Ker imamo sedaj več atributov (variat) gre za multivariatno linearno regresijo. Najprej nekaj sicer nam že dobro poznane notacije. Naj bo x vektor atributnih vrednosti. Privzemimo, da so vsi atributi zvezni in da imamo n različnih atributov, to je $x_1, x_2, \ldots x_n$. Označimo odvisno spremenljivko z y. Naš cilj je na podlagi učnih primerov, to je parov atributnih vektorjev in vrednosti odvisne spremenljivke (x, y) zgraditi model:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n \tag{4.13}$$

Tokrat smo namesto parametrov a in b parametre modela označili s črko θ, ter parametrom modela dodali tudi indeks. Parameter θ_0 ustreza prejšnjemu parametru b, θ_1 pa parametru

a. Da bo zapis bolj enostaven, določimo tudi x_0 ki naj vedno, to je, za vse primere, zavzame vrednost 1. Potem lahko zapišemo:

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i \tag{4.14}$$

oziroma v vektorski obliki

$$h(x) = \theta^T x \tag{4.15}$$

Parametre modela θ želimo izbrati tako, da bo napaka naše napovedi čim manjša. Napaka napovedi za i-ti primer, kjer je $\hat{y}^{(i)}$ napovedana vrednost, $y^{(i)}$ pa prava vrednost odvisne spremenljivke, je

$$\epsilon^{(i)} = \hat{y}^{(i)} - y^{(i)} = h_\theta(x^{(i)}) - y^{(i)} \tag{4.16}$$

Model smo tokrat označili s simbolom h, ker model predstavlja hipotezo nad našimi podatki. Napake v negativno in pozitivno smer obravnavamo enako, zato je kot prej tudi tu najbolje, da napako kar kvadriramo. Zanimajo nas taki parametri, pri katerih je kvadrat napake na učni množici minimalen, oziroma kjer se model čimbolj prilega učnim podatkom:

$$\min_\theta \sum_{i=0}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{4.17}$$

Na podlagi zgornjega kriterija zapišimo kriterijsko funkcijo. Kriterij pomnožimo z $\frac{1}{2}$, da nam bo lažje kasneje pri izpeljavi odvodov, ter z $1/m$ da vrednost kriterijske funkcije ne bo odvisna od števila primerov:

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{4.18}$$

Naš cilj je sedaj poiskati take vrednosti parameterov, pri katerih bo kriterijska funkcija $J(\theta)$ minimalna, oziroma take vrednosti, pri katerem se bo glede na našo izbrano kriterijsko funkcijo naš model čim bolje prilegal dani učni množici.

4.5 Metoda najhitrejšega spusta

Iskanje optimalnih parametrov začnimo v neki točki, na primer kar v $\theta = \vec{0}$. Potem spreminjamo θ tako, da se $J(\theta)$ zmanšuje, to je, v nasprotni smeri parcialnega odvoda kriterijske funkcije. Enako, kot smo počeli v prejšnjem poglavju z parametroma a in b, le da tokrat ta korak zapišemo splošno za parameter θ_i:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta) \tag{4.19}$$

Konstanto α imenujemo stopnja učenja, njena vrednost pa bo narekovala, kako hitro bomo hiteli proti cilju. Če bo vrednost α prevelika, je možno, da bomo preleteli cilj in se odstrelili v neskončnost. Če bo vrednost premajhna, bo konvergenca počasna.

Izpeljimo sedaj vrednost parcialnih odvodov oziroma vrednost gradienta, ko parcialne odvode zložimo v vektor. Pri tem zaenkrat privzemimo, da bomo vrednosti parametra θ_i prilagodili enemu samemu primeru (x, y):

$$\begin{aligned}
\frac{\partial}{\partial\theta_i}J(\theta) &= \frac{\partial}{\partial\theta_i}\frac{1}{2m}(h_\theta(x)-y)^2 && (4.20)\\
&= 2\frac{1}{2m}(h_\theta(x)-y)\frac{\partial}{\partial\theta_i}(h_\theta(x)-y) && (4.21)\\
&= \frac{1}{m}(h_\theta(x)-y)\frac{\partial}{\partial\theta_i}(\theta_0x_0+\ldots+\theta_ix_i+\ldots\theta_nx_n-y) && (4.22)\\
&= \frac{1}{m}(h_\theta(x)-y)x_i && (4.23)
\end{aligned}$$

Vsakokratni popravek parametra θ_i bo tako:

$$\theta_i \leftarrow \theta_i - \frac{\alpha}{m}(h_\theta(x)-y)x_i \tag{4.24}$$

oziroma za vse primere:

$$\theta_i \leftarrow \theta_i - \frac{\alpha}{m}\sum_{j=1}^{m}(h_\theta(x^{(j)})-y^{(j)})x_i^{(j)} \tag{4.25}$$

Pri linearni regresiji oziroma zgoraj opisanemu pristopu najmanjših kvadratov je $J(\theta)$ kvadratna funkcija z enim samim minimumom, zato se nam o tem, da bi se optimizacija zaustavila v nekem lokalnem minimumu ni potrebno bati. Lahko pa, kot smo že zapisali zgoraj, pri velikih vrednostih α minimum zgrešimo in se pričnemo vse bolj oddaljevati od njega. Pomaga seveda zmanjšanje α na vrednost, pri kateri je optimizacija stabilna in skonvergira k pravi vrednosti parametrov θ.

Poglavje 5

Regularizacija

Pri vpeljavi linearne regresije v prejšnjem poglavju je bil cilj gradnja modela, ki se čimbolj prilega učni množici. Pa je to res pravi kriterij za določanje parametrov modela? Bo model, ki se najbolje prileže učni množici res tisti, ki bo dobro napovedoval nove primere? V tem poglavju pokažemo, da temu ni tako, nakažemo, kakšna bi bila lahko rešitev problema in potem tudi povemo, da je ta žal odvisna od parametra, ki ga moramo nekako oceniti iz podatkov. S tem smo sklenili začarani krog (in morda nismo rešili prav dosti), a odprli skrinjico problemov, ki je tipična za strojno učenje in znanost v podatkih.

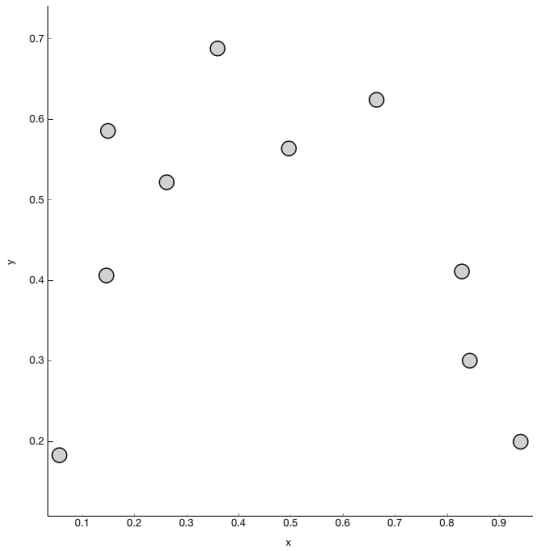
5.1 Polinomska regresija

Na sliki **??** so prikazani podatki, kjer linearni model očitno ni pravi in je odvisnost med atributom in razredom nelinearna. Linearni model z enim samim atributom, oziroma tako imenovani univariatni linearni model, očitno ne bo kos našemu problemu.

Model z eno samo spremenljivko zapišemo kot $y = \theta_0 + \theta_1 x$. Kaj pa, če podatke spremenimo tako, da število atributov povečamo. S stolpci, ki so potence našega osnovnega atributa. Torej, namesto, da vhodni podatki vsebujejo samo stolpec z atributom x, dodamo še stolpce potenc te spremenljivke, torej na primer stolpce z vrednostmi x^2 in x^3. Za trenutek uvedimo nova imena spremenljivk, oziroma nova imena atributov, in poimenujmo $a = x$, $b = x^2$ in $c = x^3$. S takimi spremenljivkami lahko zapišemo novi linearni model $y = \theta_0 + \theta_1 a + \theta_2 b + \theta_3 c$. S takimi modeli smo tudi že imeli opravka (v prejšnjem poglavju), zato nam niso novi. Še vedno gre za linearni model: torej model, kjer so a, b in c neke številke (vrednosti atributov za posamezni primer), ki so pomnožene s parametri modela $\theta_0 \dots \theta_3$, katerih vrednost iščemo.

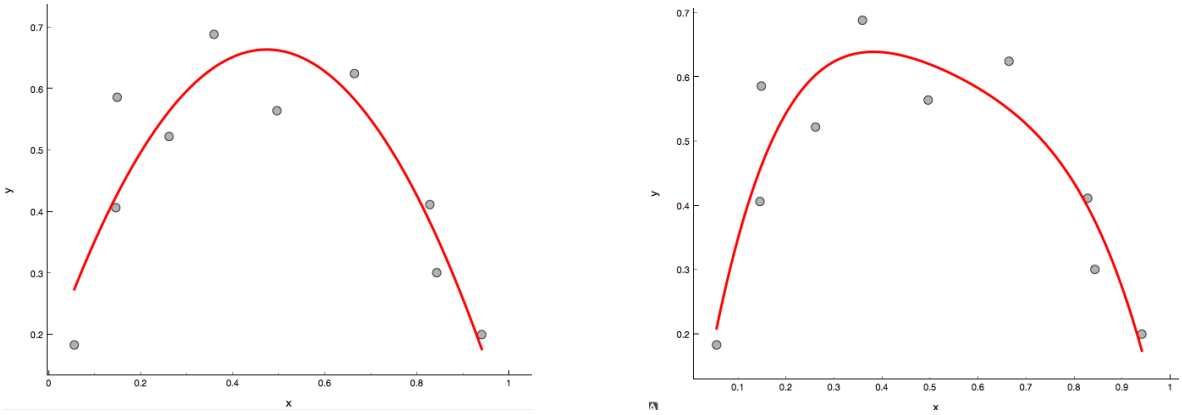
Sedaj pa preoblikujmo naš model tako, da spet vstavimo potence atributa x. Tokrat se zavedajmo, da potenčni izrazi označujejo vrednosti atributov, torej nekih konstant iz naše tabele podatkov. Prav zato je model še vedno linearen:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$



Slika 5.1: Linearni model $y = \theta_0 + \theta_1 x$ se ne ujema dobro s podatki, saj je napaka velika, odvisnost med atributom x in razredom y pa očitno ni linearna.

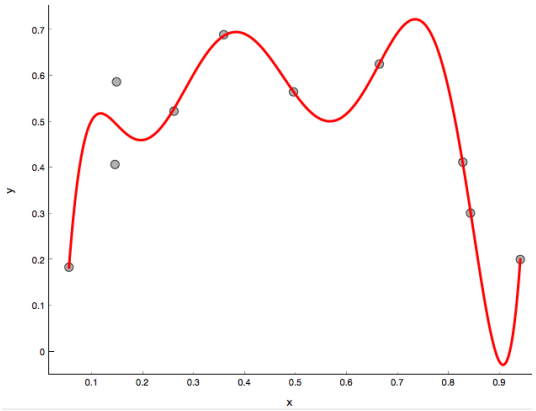
Pravimo, da smo prostor atributov razširili s potenčnimi vrednostmi. Ker bomo nad tem prostorom gradili linearni regresijski model, temu postopku razširitve prostora atributov in gradnje linearnega modela pravimo polinomska regresija. In rezultat? Na sliki **??** je prikazan rezultat pri polinomski razširitvi stopnje 2 (uporabili smo atributa x in x^2) ter stopnje štiri (uporaba atributov x, x^2, x^3 in x^4).



Slika 5.2: Polinomska regresija druge in četrte stopnje.

Z dviganjem stopnje polinoma se naš model čedalje bolj prilega učnim podatkom, napovedna napaka oziroma naša kriterijska funkcija pa je čedalje manjša. S polinomsko razširitvijo sedmega reda smo dosegli skoraj popolno prileganje (slika **??**). Vrednost naše kriterijske funkcije gre z dvigovanjem stopnje polinoma proti nič in je dejansko enaka nič takrat, ko

je stopnja polinoma oziroma razširitve atributnega prostora enaka $m - 1$, kjer je m število primerov.



	name	coef
1	1	-1.7244949
2	x	59.4661520
3	x^2	-603.2469062
4	x^3	2996.3288934
5	x^4	-7936.7980519
6	x^5	11418.3588803
7	x^6	-8400.1096545
8	x^7	2470.3488969

Slika 5.3: Polinomska regresija sedme stopnje in koeficienti modela.

Navidezno smo problem iskanja modela, ki bi minimiziral kriterijsko fukcijo, z uvedbo polinomske regresije rešili. Pa smo res? Spomnimo se, da je naš cilj gradnje modelov napovedovanje, torej ne samo modeliranje na osnovi učne množice, ampak uporaba modela za napovedovanje primerov, ki jih pri učenju nismo videli in ki so novi. Uporabo polinomske regresije moramo zato preskusiti na tovrstnih primerih, ki jim pravimo tudi testna množica. Pred tem pa se moramo dogovoriti, kako bomo merili uspešnost modela oziroma njegovo napovedno točnost.

5.2 Napovedna točnost

Imejmo testno množico $\mathcal{T}$ s k primeri, $k = |\mathcal{T}|$, za katere poznamo njihov atributni opis $x^{(i)}$ in vemo njihov pravi razred $y^{(i)}$. Oceniti želimo napovedno točnost regresijskega modela, ki za i-ti primer napove oziroma oceni vrednost njegovega razreda kot $\hat{y}^{(i)}$.

Pri prvi meri za napovedno točnost, ki jo bomo tu uvedli, sledimo dosedanji logiki, da nas predznak napake ne zanima (torej kvadriramo) in da bi radi izračunali povprečno vrednost take, kvadrirane napake. Ker bi želeli, da se napaka izrazi v istih enotah kot vrednosti razreda v učni množici, povprečno kvadrirano napako na koncu še korenimo. Dobimo koren povprečne kvadrirane napake (angl. *Root Mean Squared Error*)):

$$\mathrm{RMSE}(\mathcal{T}) = \sqrt{\frac{\sum_{i=1}^{k} y^{(i)} - \hat{y}^{(i)}}{k}}$$

Mera RMSE je sicer čisto v redu, a je njena vrednost odvisna od domene odvisne spremenljivke. Na primer, če je RMSE = 42.0 s tem pravzaprav nismo povedali ničesar, saj ne

poznamo razpon odvisne spremenljvike. Če smo s tem ocenili težo tovornjakov v tonah, je napaka ogromna, če pa smo njegovo težo ocenjevali v kilogramih, je skoraj zanemarljiva. Bolje bi bilo imeti oceno napake, kjer je ta neodvisna od domene odvisne spremenljivke. Recimo, nekaj, čemur statistiki pravijo delež razložene variance:

$$R^2(\mathcal{T}) = 1 - \frac{\sum_k (y^{(i)} - \hat{y}^{(i)})^2}{\sum_k (y^{(i)} - \bar{y})^2}$$

V imenovalcu ulomka je vsota kvadratov napak, ki bi jo naredili, če bi model napovedoval s povprečno vrednostjo odvisne spremenljivke učne množice. To bi bil prav slab model in pričakujemo, da bo njegova napaka večja, ali pa morda veliko večja od te, ki bi jo naredil boljši model in katerega kvadrat napake izračunamo v števcu ulomka. Za zelo dober model gre člen z ulomkom proti vrednosti 0, ocena R^2 pa zato proti vrednosti 1. Slabši modeli pa bodo imeli napako le malo manjšo od napake napovedi s povprečno vrednostjo.

Ocena R^2 bo takrat šla proti 0. Delež razložene variance ima zato pričakovano vrednost med 0 in 1. Opozorimo le, da čeprav vemo, da bodo visoke vrednosti R^2, torej take blizu 1.0 zaželene, so lahko uporabni tudi modeli z vrednosti R^2 blizu nič. Na primer, z modelom, ki ima na testni množici pri napovedi tečaja neke valute $R^2 = 0.1$ bi obogateli, model z isto točnostjo za napoved jutrišnje povprečne temperature pa bi lahko vrgli v koš.

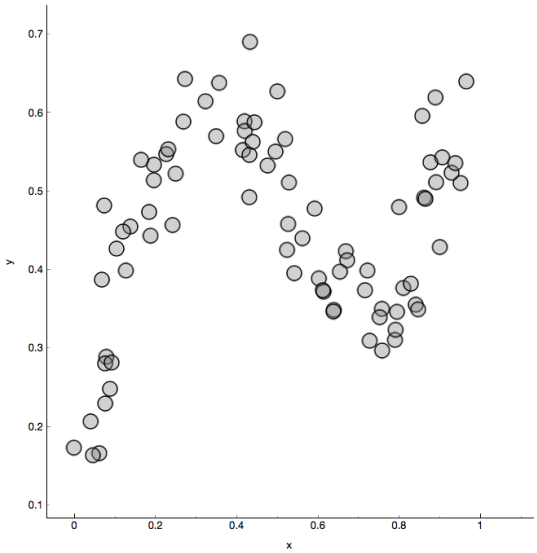
5.3 Ocenjevanje napovedne točnosti

Naredimo eksperiment: učno množico, ki jo prikazuje slika **??**, naključno razdelimo na polovico, kjer uporabimo polovico naključno izbranih primerov za učenje linearnega modela, na drugi polovici modela pa testiramo napovedno uspešnost. Za ocenjevanje točnosti smo uporabili R^2, in to merili na učni in testni množici.

Rezultate kaže tabela **??**. S stopnjo polinomske razširitve atributnega prostora napovedna točnost modela na učni množici monotono narašča. Na množici testnih primerov pa je slika drugačna: s kompleksnostjo modela napovedna točnost nekaj časa narašča, potem pa strmo pade. Pravimo, da se je model preveč prilegel učni množici, postal zato prekompleksen in ni zajel glavnih vzorcev, ki so prisotni tudi v novih primerih.

5.4 Regularizacija linearne regresije

Ko višamo stopnjo polinomske razširitve atributnega prostora smo opazili (slika **??**), da parametri modela podivjajo, oziroma postane njihova vrednost zelo visoka. Kar ni nič čudnega, saj se model zelo prilagodi učnim podatkom, funkcija odvisnosti med x in y je zelo razgibana, njeni odvodi pa zelo visoki. Prileganje učni množici se torej izrazi v absolutno velikih vrednostih parametrov modela (absolutno zato, ker so te vrednosti lahko visoke a negativne ali visoke in pozitivne). Želeli bi si manj "divje" modele, torej take, kjer bi bili parametri modela



Slika 5.4: Množica vseh primerov, ki jih naključno porazdelimo med učno in tesno množico za ocenjevanje gradnjo in ocenjevanje kvalitete modela polinomske regresije.

Tabela 5.1: Delež razložene variance na učni in testni množici (podatki s slike **??**) za polinomsko regresijo stopnje k.

k	učna	testna
0	0.000	0.002
1	0.031	0.037
2	0.161	0.151
3	0.806	0.688
4	0.822	0.687
5	0.832	0.715
6	0.841	0.716
7	0.841	0.714
8	0.857	0.365
9	0.863	0.118

po absolutni meri manjši. To željo enostavno izrazimo kot dodatek h kriterijiski funkciji za linearno regresijo:

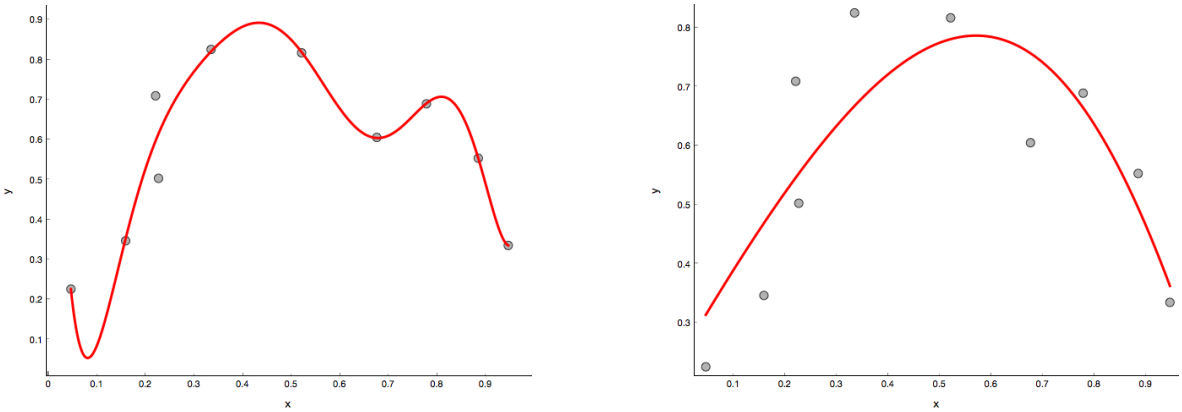
$$J = \frac{1}{2m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)})^2 + \eta\sum_{j=1}^{n}\theta_j^2 \tag{5.1}$$

Prvi člen v zgornji enačbi je cenovna funkcija za linearno regresijo, kjer želimo, da se model čimbolj prilega učni množici. Drugi člen pa pravi, da naj bo kvadratna vrednost parametrov takega modela čim manjša. Uporabili smo kvadratno vrednost parametrov, saj nam je vseeno, ali je njihova vrednost pozitivna ali negativna. Pravimo, da smo cenovno funkcijo regularizirali, stopnjo regularizacije pa uravnavamo z vrednostjo η, katere tipične vrednosti so nizke, recimo 0.1 ali pa 0.01 ali pa celo 0.0001.

Za učenje modela linearne regresije, torej, za določanje vrednosti parametrov θ_i modela iz podatkov rabimo še izračunati gradient. Izračun je enostaven, saj je enak, kot pri neregularizirani linearni regresiji plus odvod člena za regularizacijo (spodnja enačba velja za $i = 1 \ldots n$, za $i = 0$ pa ostane odvod enak kot prej):

$$\frac{\partial}{\partial\theta_i}J(\theta) = \frac{1}{m}(h_\theta(x) - y)x_i + 2\eta\theta_i \tag{5.2}$$

Ostane le še preskus: ali se, tudi z regularizacijo, lahko preveč prilagodimo manjšemu naboru podatkov v učni množici. Seveda je vse odvisno od stopnje regularizacije η, a v splošnem (pri primerni vrednosti parametra η) je odgovor ne. Regularizacija nam pomaga pri preprečevanju prevelikega prileganja. Naj sliki **??** je tako primer podatkov in polinomske regresije brez regularizacije in z regularizacijo.



Slika 5.5: Model polinomske regresije stopnje osem brez (levo) in z regularizacijo ($\eta = 0.01$, desno).

Poglavje 6

Logistična regresija

Se bo osnovnošolec Vinko uvrstil v drugi krog tekmovanja iz kemije? Se bo po srednji šoli vpisal na Kemijsko fakulteto? Ga bo zaposlilo podjetje, ki se ukvarja z biofarmacijo? Bo njegov košarkaški klub Olimpija premagal Crveno zvezdo to soboto? Se bo poročil s sosedovo Niko? Bo njegova ideja o tekočini, ki vsakih 35 minut spremeni barvo, uspela na Kickstarterju?

Morda smo s kakšnim od zgornjih vprašanj zašli, a na vsaj eno od naštetih vemo, da je nanj možno dovolj dobro odgovoriti. So pa to vprašanja, ki niso regresijska, oziroma je napoved kategoričnega tipa "bo" ali "ne bo". Še bolj natančno: za vsako od zgornjih vprašanj bi pravzaprav radi ocenili verjetnost, ali se bo nek dogodek zgodil. Recimo, zanima nas, s kakšno verjetnostjo se bo Vinko poročil z Niko. In pa, s kakšno verjetnostjo bo njegov kicksterski projekt uspel.

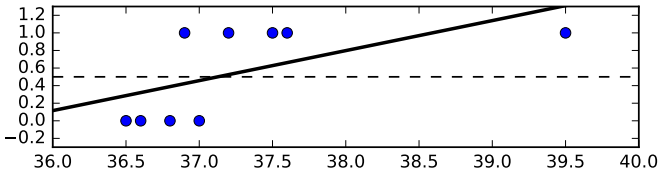
Seveda bomo tudi tokrat napovedne modele, sedaj klasifikacijske, gradili iz podatkov. Za silo bi take podatke in s tem problem spremenili v regresijskega tako, da bi odgovore označili z vrednostjo 0 (dogodek se ni zgodil) ali 1 (dogodek se je zgodil), potem pa to vrednost obravnavali kot zvezno vrednost odvisne spremenljivke oziroma razreda y. Primer take obravnave nam kaže tabela **??** s podatki o obiskovalcih ordinacije, ki jim je zdravnik izmeril telesno temperaturo in potem (seveda na podlagi dodatnih preiskav, ki pa jih tu ne podajamo) določil zdravstveno stanje. Naš cilj je zgraditi model, ki zdravstveno stanje oceni (samo) na podlagi telesne temperature.

Podatke predstavimo grafično (slika **??**) in si potem nanje drznemo vpeti linearno funkcijo $y = h(x)$. Najprej opazimo, da je zaloga vrednosti funkcije $h(x)$ neprimerna, saj gre ta od $-\infty$ do ∞. Na intervalu telesnih temperatur okoli točke 37° ima funkcija sicer vrednost med 0 in 1. Pojavi pa se vprašanje, kako vrednost te linearne funkcije sploh interpretirati, oziroma kako jo pretvoriti v verjetnost. Spomnimo se, da vsaka verjetnostna funkcija vrača vrednosti med 0 in 1, naša linearna funkcija pa je omejena na ta interval samo v določenem območju vrednosti vhodnega atributa. Dodaten problem je še z osamelci, oziroma obiskovalci s skrajnimi vrednostnimi atributov. Če bi podatkom dodali še en primer bolnika z zelo visoko temperaturo, bi se naša funkcija precej spremenila in pomaknila v desno. Potrebujemo nek

Tabela 6.1: Telesna temperatura in stanje pregledovanca, ki smo ga zapisali v numerični obliki kot razred y.

temperatura	stanje	y
36,5	zdrav	0
36,6	zdrav	0
36,8	zdrav	0
36,9	bolan	1
37,0	zdrav	0
37,2	bolan	1
37,5	bolan	1
37,6	bolan	1
39,5	bolan	1

predpis, ki bi vrednosti take funkcije pretvoril v verjetnosti in jih morda blizu $y = 0$ in $y = 1$ malce zmehčal.



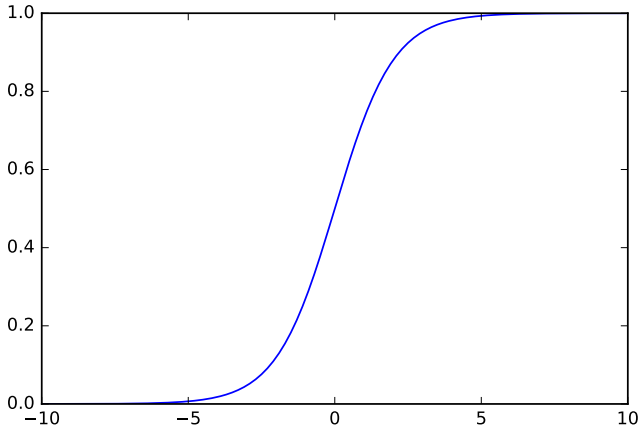
Slika 6.1: Poskus klasifikacije z linearno regresijo.

6.1 Logistična funkcija

Funkcija, ki jo bomo uporabili in ki neko realno število pretvori v število na intervalu [0, 1], je logistična funkcija (slika **??**):

$$g(z) = \frac{1}{1 + e^{-z}} \tag{6.1}$$

Logistična funkcija je zvezna, monotona, z $z \to -\infty$ konvergira proti 0 in z $z \to \infty$ proti 1.



Slika 6.2: Logistična funkcija.

Je odvedljiva pri vseh vrednosti njenega parametra. Njen odvod je:

$$\begin{aligned}
\frac{dg(z)}{dz} &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\
&= \frac{1}{(1+e^{-z})^2}e^{-z} \\
&= \frac{1}{1+e^{-z}}\frac{e^{-z}}{1+e^{-z}} \\
&= \frac{1}{1+e^{-z}}\frac{1+e^{-z}-1}{1+e^{-z}} \\
&= \frac{1}{1+e^{-z}}\Big(1-\frac{1}{1+e^{-z}}\Big) \\
&= g(z)[1-g(z)]
\end{aligned} \tag{6.2}$$

6.2 Logistična regresija

Zapišimo sedaj naš klasifikacijski model. V osnovi bo to linearni model, torej utežena vsota vrednosti atributov. A tokrat jo bomo transformirali z logistično funkcijo, da bo vrednost modela izražala pripadnost (verjetnost) ciljnemu razredu:

$$\begin{aligned}
h_\theta(x) &= g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots \theta_n x_n) \\
&= g(\theta^T x) \\
&= \frac{1}{1+e^{-\theta^T x}}
\end{aligned} \tag{6.3}$$

Model zaradi uporabe logistične funkcije imenujemo *logistična regresija*.

6.3 Računanje verjetnosti razreda

V nadaljevanju bomo torej logistično funkcijo uporabili tako, da nam bo naš model $h_\theta(x)$, ki je v osnovi linearni model, vračal vrednosti med 0 in 1. Še naprej bomo predpostavljali, da je naša razredna spremenljivka dvovrednostna, a privzeli, da je njena ciljna vrednost razred z oznako $y = 1$ in da zanj model $h_\theta(x)$ vrača verjetnost tega razreda. K oznaki modela smo namenoma pripisali oznako za vektor parametrov θ in na ta način poudarili, da ti parametri tudi polno opredelijo naš model. Zapišemo torej lahko:

$$P(y = 1|x; \theta) = h_\theta(x) \tag{6.4}$$

$$P(y = 0|x; \theta) = 1 - h_\theta(x) \tag{6.5}$$

Izraz za $P(y = 1|x; \theta)$ nam torej podaja verjetnost, da je razred primera, ki je opisan z vektorjem atributnih vrednosti x, enak 1. Oziroma podaja verjetnost, da neodvisna spremenljivka zavzame vrednost 1 pri opisu primera x in parametrizaciji modela s parametri θ. En sam izraz, ki združi zgornji enačbi v eno in ki podaja verjetnostno porazdelitev za spremenljivko y je:

$$p(y|x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y} \tag{6.6}$$

Pravilnost zgornjega izraza preveri tako, da za vrednost spremenljivke y vstaviš enkrat $y = 1$ in drugič $y = 0$.

6.4 Verjetje

Izraz za $p(y|x; \theta)$ v enačbi **??** torej podaja verjetnost za določeno vrednost neodvisne spremenljivke in določen vektor atributnih vrednosti pri modelu, ki je podan z θ. Sedaj pa si predstavljamo, da vrednosti elementov vektorja θ spreminjamo. Prav gotovo se bo na ta način tudi spreminjala verjetnost za dani razred pri izbranem primeru; enkrat bo ta verjetnost višja, drugič nižja.

Zamrznimo sedaj parametre modela θ in izračunajmo verjetnosti $L(\theta)$ pravih razredov $\vec{y}$ za primere v učni množici, ki jo opišemo z matriko atributnih vrednosti X. Predpostavljamo, da so primeri iz učne množice neodvisni in zato verjetnost za vrednosti spremenljivke y za celotno učno množico lahko zapišemo kot produkt verjetnosti za vrednost y za posamezne

primere:

$$\begin{aligned} L(\theta) &= p(\vec{y}|X;\theta) \\ &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m} h_\theta(x^{(i)})^{y^{(i)}}(1-h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned} \tag{6.7}$$

Kot smo označili v zgornji funkciji L, je ta verjetnost odvisna od parametrov modela θ. Kakšen model bi želeli imeti? Tak, pri katerem je verjetnost $L(\theta)$ največja. Torej tak model, kjer bo verjetnost, da bo model napovedal take vrednosti razredov, kot so v učni množici, največja. $L(\theta)$ igra vlogo kriterijske funkcije, ki pa jo tokrat maksimiziramo.

Funkcijo L imenujemo *verjetje*. Iščemo torej parametre θ, kjer je njena vrednost največja. Tudi tokrat bomo tako vrednost parametrov iskali z gradientnimi pristopi, za te pa sedaj že vemo, da moramo znati izračunati odvode po posameznih parametrih, oziroma, da moramo znati izračunati gradient kriterijske funkcije. Ker je odvajanje funkcij z mnogo produkti, torej funkcij, kot je naša $L(\theta)$, precej mučno, se vprašajmo, ali obstaja kakšna preslikava funkcije L, za katero bi bilo lažje izračunati odvod in pri kateri bi vrednosti θ, ki to funkcijo maksimizirajo prav tako maksimizirale tudi funkcijo $L(\theta)$. Taka preslikava je logaritemska funkcija, ki je na intervalu od 0 do ∞ monotona in bo torej vektor θ, ki maksimizira $\log(L(\theta))$ tak, da maksimizira tudi $L(\theta)$. Logaritem $L(\theta)$ označimo kot $l(\theta)$ in ga izračunamo kot:

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} \left[y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)})) \right] \end{aligned} \tag{6.8}$$

Namesto kriterijske funkcije $L(\theta)$ bomo torej uporabili funkcijo $l(\theta)$, ki ji pravimo tudi *logaritem verjetja* (angl. *log likelihood*).

6.5 Gradient logaritma verjetja

Iščemo torej tak θ, ki maksimizira logaritem verjetja $l(\theta)$. Ker bomo uporabili gradientno metodo in ker je θ vektor $[\theta_0\theta_1 \ldots \theta_n]$, moramo izračunati parcialne odvode naše kriterijske

funkcije:

$$\begin{aligned}
\frac{\partial}{\partial\theta_j}l(\theta) &= \sum_{i=1}^{m}\frac{\partial}{\partial\theta_j}\Big[y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\Big] \\
&= \sum_{i=1}^{m}\Big[y^{(i)}\frac{1}{g(\theta^T x^{(i)})} - (1-y^{(i)})\frac{1}{1-g(\theta^T x^{(i)})}\Big]\frac{\partial}{\partial\theta_j}g(\theta^T x^{(i)}) \\
&= \sum_{i=1}^{m}\Big[\frac{y^{(i)}}{g(\theta^T x^{(i)})} - \frac{(1-y^{(i)})}{1-g(\theta^T x^{(i)})}\Big]g(\theta^T x^{(i)})(1-g(\theta^T x^{(i)}))\frac{\partial}{\partial\theta_j}\theta^T x^{(i)} \\
&= \sum_{i=1}^{m}\Big[\frac{y^{(i)}-g(\theta^T x^{(i)})}{g(\theta^T x^{(i)})(1-g(\theta^T x^{(i)}))}\Big]g(\theta^T x^{(i)})(1-g(\theta^T x^{(i)}))x_j^{(i)} \\
&= \sum_{i=1}^{m}(y^{(i)}-g(\theta^T x^{(i)}))x_j^{(i)} \\
&= \sum_{i=1}^{m}(y^{(i)}-h_\theta(x^{(i)}))x_j^{(i)}
\end{aligned}$$

Zelo enostavno! Smo tak rezultat oziroma tako enačbo videli že kje prej? Seveda! Pri linearni regresiji. Parcialni odvodi so identični tem za linearno regresijo. Seveda z majhno razliko. Tokrat naša funkcija h_θ uporablja logistično funkcijo (en. **??**), pri linearni regresiji pa je bila h_θ samo utežena vsota atributnih vrednosti.

6.6 Optimizacija parametrov modela

Postopek iskanja parametrov modela je identičen temu pri linearni regresiji. Seveda, z majhno razliko, da je tokratni model $h_\theta(x)$ model logistične regresije. Vseeno zapišimo korak za osveževanje vrednosti parametra θ_j:

$$\theta_j \leftarrow \theta_j + \alpha\sum_{i=1}^{m}\Big(y^{(i)} - h_\theta(x^{(i)})\Big)x_j^{(i)} \tag{6.9}$$

Tudi tokrat je α stopnja učenja, katerega vrednost je pri normaliziranih podatkih tipično majhna (npr. 0.001).

Tukaj velja opozorilo. Pristop z gradientnim sestopom je počasen in je, že za srednje velike podatke, potrebno izvesti mnogo iteracij popravljanja vrednosti parametrov θ. Namesto te tehnike tipično uporabljamo optimizacijske tehnike, ki imajo hitrejšo konvergenco. Ena od teh je postopek L-BFGS, ki pa ga tipično uporabimo iz za to dostopne knjižnice. Na vhodu postopek L-BFGS potrebuje cenovno funkcijo (torej $l(\theta)$) in pa njen gradient, ki smo ga izračunali zgoraj (en. **??**).

Poglavje 7

Klasifikacijska drevesa in gozdovi

Drevesa in gozdovi za klasifikacijo so zelo podobna tem za regresijo. Edina večja razlika je le v načinu ocenjevanja pomembnosti atributov. V tem poglavju se zato najprej seznanimo s temi metodami in jih potem uporabimo pri gradnji dreves in gozdov.

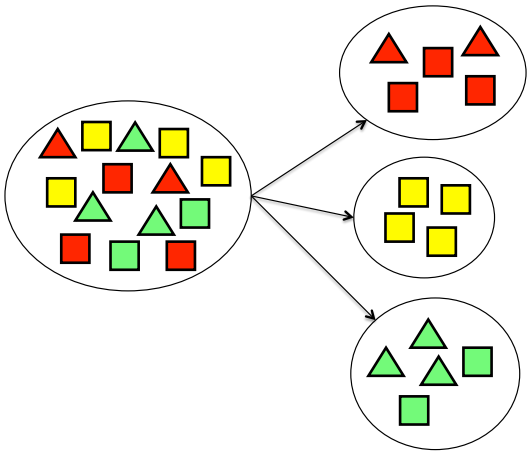
7.1 Ocenjevanje pomembnosti atributov

Pri prav vseh praktičnih problemih uvrščanja v skupine nas zelo zanimajo atributi, ki so kar najbolj povezani z znanimi uvrstitvami primerov oziroma, kot bi temu rekli v strojnem učenju, z razredi primerov. Razkritje takih atributov nam lahko dosti pove o problemu, ki ga raziskujemo, odkrije, katere lastnosti opazovanih objektov moramo spremljati še v prihodnje oziroma so tiste, ki igrajo pomembno vlogo pri računalniški podpori odločanja. Velika večina tehnik ocenjevanj pomembnosti atributov obravnava en sam atribut ločeno od ostalih, to je, zanemari kakršnekoli medsebojne povezanosti atributov. Peščica tehnik, med katerimi bomo tu omenili eno samo, najbolj znano, pa zna razkriti pomembnost atributa tudi z ozirom na kontekst, to je, skladno z vrednostmi ostalih atributov.

Začnimo pri tehnikah, ki atribute obravnavajo vsakega zase in kontekst ostalih atributov zanemarijo. Za začetek poglejmo množico trikotnikov in kvadratov s slike **??**. Želeli bi oceniti, kako dobro lahko na podlagi barve lika napovemo njegovo obliko.

7.1.1 Informacijski prispevek

Če bi bila povezanost med barvami in oblikami likov popolna, bi bile v posameznih podmnožicah na desni strani slike **??** vsi liki enakih oblik. Pa niso. Pravimo, da množice niso čiste. Podmnožica rumenih likov je čista, čistočo ostalih pa bi radi zmerili. Obliko likov, naš razred, obravnavajmo kot naključno spremenljivko C, njene vrednosti pa označimo z c_i. Naša spremenljivka ima dve različni vrednosti, "trikotnik" in "kvadrat". Ocenimo še verjetnosti, da med liki iz naše celotne množice likov ($N = 14$) izberemo, na primer, trikotnik. Za oceno



Slika 7.1: Množica različno obarvanih kvadratov in trikotnikov (levo) in ista množica razdeljena na tri podmnožice glede na barvo likov. Vseh likov skupaj je $N = 14$.

uporabimo kar relativno frekvenco:

$$P_{\triangle} = \frac{N_{\triangle}}{N} = 5/14 = 0.357$$

Podobno izračunamo, da je $P_{\square} = 0.643$. Nečistočo naše množice likov lahko ocenimo z entropijo H diskretne naključne spremenljive C, ki je, izmerjena v bitih:

$$H(C) = \sum_{i=1}^{n} p(c_i)\, I(c_i) = -\sum_{i=1}^{n} p(c_i) \log_2 p(c_i)$$

oziroma za naš primer:

$$H(C) = -0.357 \times log_2 0.357 - 0.643 \times log_2 0.643 = 0.940$$

Ravno tako, kot za osnovno množico, lahko ocenimo nečistočo podmnožic, ki jih dobimo, ko smo like razdelili glede na njihovo barvo. Tako dobimo:

$$H(C|\text{color} = \text{red}) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.971$$

$$H(C|\text{color} = \text{yellow}) = -\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4} = 0.0$$

$$H(C|\text{color} = \text{green}) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971$$

Oceniti moramo še verjetnost, da lik pripada eni od podmnožic. Z drugimi besedami, kakšna je verjetnost, da bo naključno izbran lik iz naše osnovne množice na primer rdeč.

Enostavno,

$$P_{\text{red}} = \frac{N_{\text{red}}}{N} = \frac{5}{14} = 0.357$$

Podobno ocenimo še $P_{\text{yellow}} = 0.286$ in $P_{\text{green}} = 0.357$. Pričakovana nečistoča podmnožic, ko našo množico likov razdelimo skladno z njihovo barvno, ocenimo kot vsoto nečistoč podmnožic, ki jo utežimo z verjetnostjo, da bo lik pripadal določeni množici. Dobljeni uteženi vsoti pravimo residualna entropija:

$$H_{res} = H(C|X) = \sum_i p(x_i)H(C|x_i)$$

in za naš primer znaša:

$$H_{res} = 0.357 \times 0.971 + 0.286 \times 0.0 + 0.357 \times 0.971 = 0.694$$

Pričakovana entropija, ko smo like razdelili po barvi, je torej manjša od entropije osnovne množice. Njuni razliki pravimo informacijski prispevek (angl. *information gain*) in z njim ocenimo, koliko informacije nam prispeva poznavanje vrednosti posameznega atributa:

$$\text{IG}(X) = H(C) - H(C|X) = 0.940 - 0.694 = 0.246$$

7.1.2 Relativni informacijski prispevek

Pri atributih, ki imajo veliko zalogo vrednosti, se nam prav lahko zgodi, da nam ti primere iz osnovne množice razdelijo v zelo majhne podmnožice, morda celo take s po enim samim elementom. Entropija oziroma nečistoča slednjih je nič, a takih podmnožic si vsekakor ne želimo. Pravzaprav lahko informacijski prispevek favorizira atribute z veliko zalogo vrednosti. To "prednost" lahko uravnotežimo s tem, da se vprašamo, koliko informacije potrebujemo, da zvemo vrednost posameznega atributa. V našem primeru imamo med štirinajstimi liki pet rdečih, štiri rumene in pet zelenih, zato:

$$H(X) = -\frac{5}{14}\log_2\frac{5}{14} - \frac{4}{14}\log_2\frac{4}{14} - \frac{5}{14}\log_2\frac{5}{14} = 1.58$$

Mero, kjer vrednost zgornjega izraza uporabimo za uravnoteženje informacijskega prispevka, imenujemo relativni informacijski prispevek (angl. *information gain ratio*):

$$\text{IGR}(X) = \frac{\text{IG}(X)}{H(X)}$$

V našem primeru je relativni informacijski prispevek barve likov enak $0.246/1.58 = 0.156$.

Predlagano uravnoteženje ni edini način, da izenačimo ocene za atribute, ki imajo (zelo) različno število vrednosti. Morda bolj naravna, a računsko zahtevnejša je tehnika, ki uporablja permutacijski test in jo omenjamo v nadaljevanju.

7.1.3 Mera nečistoče po Giniju

Prikladna mera nečistoče je tudi ginijev indeks:

$$\text{Gini}(C) = \sum_i p(C = c_i) \times (1 - p(C = C_i)) = 1 - \sum_i [p(C = c_i)]^2$$

Ta ocena enaka nič za čiste množice, torej množice, kjer vsi primeri pripadajo istemu razredu, in višje vrednosti pri množicah primerov, ki pripadajo različnim razredom. Nečistoča po Giniju pove, kako pogosto bi bil za naključno izbran primer napačno napovedan razred, ki bi ga uteženo naključno napovedali iz dane porazdelitve razredne spremenljivke.

7.1.4 Permutacijski test

Pri merah, kot je večina zgornjih, včasih le težko ločimo med informativnimi in neinformativnimi atributi. Pri kakšni vrednosti mere postaviti mejo? Še težje se odločimo, če uporabljamo različne mere, kjer bi za vsako morali poznati njene statistične lastnosti. Je vrednost ocene ReliefF, ki je enaka 0.42, dovolj visoka, da razglasimo, da je atribut informativen. Kaj pa 0.12? Pa 0.06? Dobro bi bilo vedeti, ali so vrednosti teh ocen pozitivne zaradi resničnih zakonitosti v podatkih ali pa smo jih dobili po naključju. Še posebej bodo ta vprašanja zanimiva za probleme, kjer je primerov malo, atributov pa veliko.

Ocene informativnosti atributov, ki jih dobimo za vsakega od atributov, lahko primerjamo z njihovimi ničelnimi porazdelitvami, ki bi jih dobili, če bi informativnosti ocenjevali na naključno pridobljenih podatkih. Za slednje bo za nekontekstne mere dovolj premešati vrednosti razredne spremenljivke, za ReliefF ali podobne pa bo potrebno premešati celotno tabelo podatkov tako, da bomo premešali vrednosti v vsaki koloni (po atributu, torej) in na ta način "pokvarili" morebitni kontekst.

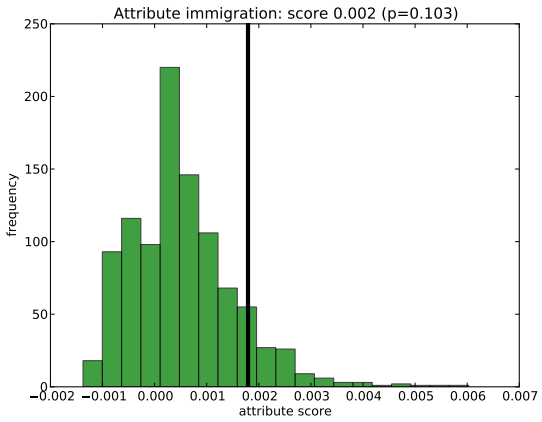
Za naključno premešanje razredne spremenljivke nam lahko služi spodnji razred v Pythonu. Ta si ob inicializaciji zapomni tudi izvorno uvrstitev razredov, ki jo lahko uporabi za vzpostavitev začetnega stanja. Razred je implementiran tako, da podatke ne podvaja, zapomni pa si samo vektor razredov.

```
import random

class PermuteClass:
    """Permute the class column of the data, can also restore to original."""
    def __init__(self, data):
        self.c_vals_backup = [d.get_class() for d in data]
        self.c_vals = [d.getclass() for d in data]
    def __call__(self, data):
        random.shuffle(self.c_vals)
        for d, c in izip(data, self.c_vals):
            d.set_class(c)
    def restore(self, data):
        for d, c in izip(data, self.c_vals_backup):
            d.set_class(c)
```

Razred za premešanje vrednosti razredne spremenljivke uporablja spodnja koda, ki za vsak atribut v podatkih zgradi ničelno porazdelitev atributnih ocen.

```
import Orange.data

epochs = 1000
alpha = 0.05
data = Orange.data.Table("voting.tab")

# define the scorer and score original data
scorer = Orange.feature.scoring.Gini()
scores = {a: scorer(a, data) for a in data.domain.attributes}

# compute null distribution, one for each attribute
null = {a:[] for a in data.domain.attributes}
permuter = PermuteClass(data)
for i in range(epochs):
    permuter(data)
    ns = [(a, scorer(a, data)) for a in data.domain.attributes]
    for a, s in ns:
        null[a].append(s)

# analyze
ps = {a: sum(1 for x in null[a] if x>scores[a])/float(epochs)
      for a in data.domain.attributes}
trash = sum(1 for p in ps.values() if p > alpha)
print "Uninformative features: %d (%d%%)" % \
    (trash, trash*100/len(data.domain.attributes))
```

Rezultat te, permutacijske analize lahko ponazorimo tudi grafično, kot to prikazuje slika **??**.



Slika 7.2: Rezultat permutacijske analize. Prikazana je porazdelitev informativnosti atributa, kot bi jo opazili, če bi bil razred primera naključen. Informativnost atributa na dejanskih, nenaključnih podatkih prikazuje vertikalna črta. Kar 10,3% zmerjenih informativnost na naključnih podatkih je od te večja, zato tega atributa pri nadaljnji analizi ne bomo upoštevali.

Risanje grafov, kot je ta s slike **??**, je tipično zanimivo in informativno. Nikoli ne smemo zaupati samo golim številkam. Grafične ilustracije raznih analiz nam vedno pomagajo globlje

razumeti problemsko domeno, čeprav vsi grafi niso ravno tisti, ki jih bomo vključili v naše končno poročilo. Da bo risanje takih grafov enostavneje, kodo za zgornji graf podajamo spodaj.

```
import matplotlib.pyplot as plt

# plot score vs. null distribution for k-th worst attribute
k = 1
att, p = sorted(ps.items(), key=itemgetter(1), reverse=True)[k]
plt.close() # clear the drawing canvas
plt.hist(null[att], 20, facecolor="green", alpha=0.75)
plt.axvline(x=scores[att], color="k", linewidth=4)
plt.xlabel("attribute score")
plt.ylabel("frequency")
plt.title("Attribute %s: score %5.3f (p=%5.3f)" % (att.name, scores[att], p))
plt.savefig("att-score-null.pdf")
```

7.2 Klasifikacijska drevesa

Klasifikacijska drevesa so ena najbolj osnovnih in enostavnih orodij za gradnjo napovednih modelov pri problemih z diskretnim razredom. Ker ste ta algoritem že dobro spoznali pri uvodnih predavanjih iz umetne inteligence, tu le na kratko. Osnovna ideja algoritma je razbitje začetne množice podatkov na čim bolj (razredno) čiste podmnožice. Za razbitje uporabimo en sam atribut, razbitje pa izvedemo na podlagi njegovih vrednosti. Tako smo na sliki **??** za razbitje množice likov uporabili informacijo o njihovi barvi. Ker tipično podatki vsebujejo več atributov, izberemo tistega, ki vodi k najbolj čistim podmnožicam, to je tistega, katerega informativnost je za dano množico primerov največja. V osnovnem algoritmu postopek ponavljamo na dobljenih podmnožicah vse dokler te niso čiste, ali pa dokler nismo uporabili že vseh atributov.

Rekurzivni algoritem zgradi drevo s korenom drevesa, v katerem smo imeli vse učne primere. Korenu sledijo vozlišča – njegovi neposredni nasledniki, kjer smo primere iz osnovne množice razdelili skladno z vrednostmi najbolj informativnega atributa na podmnožice. Postopek smo ponovili vse do listov drevesa. Listom pripadajo primeri, ki izpolnjujejo pogoje oziroma imajo vrednosti atributov podane tako, kot je to zapisano na poti od lista do korena drevesa. Za vsak list lahko iz pripadajočih primerov ocenimo, kateri razred je najbolj pogost. Pravimo, da list uvršča v ta razred. Za potrebe uvrščanja se zato, glede na atributne vrednosti danega primera, sprehodimo od korena navzdol vse do lista, ki nam določa ustrezno uvrstitev.

Zgoraj smo opisali osnovni algoritem, tako imenovan algoritem ID3 [] za gradnjo dreves. V izboljšanih verzijah algoritma pa lahko uporabimo tudi nekaj dodatnih trikov:

Obravnava zveznih atributov. Te v danem vozlišču nadomestimo z binarnim atributom tako, da poiščemo mejo t_X, kjer skupina primerov razpade na skupino, kjer je $X < t_X$ in kjer je $X >= t_X$. Mejo t_X izberemo tako, da vrednosti tega atributa, ki jih uporabljajo primeri, za katere iščemo razbitje, uredimo in preiščemo vse točke na sredini med dvemi

zaporednimi vrednostmi. Izberemo tako mejno vrednost, kjer je informativnost (ocena) dobljenega binarnega atributa najvišja. Postopek imenujemo tudi kategorizacija oz. diskretizacija atributa, in je lahko primeren za predobdelavo celotne množice učnih primerov pri situacijah, kjer uporabljene metode ne znajo neposredno uporabiti zveznih atributov.

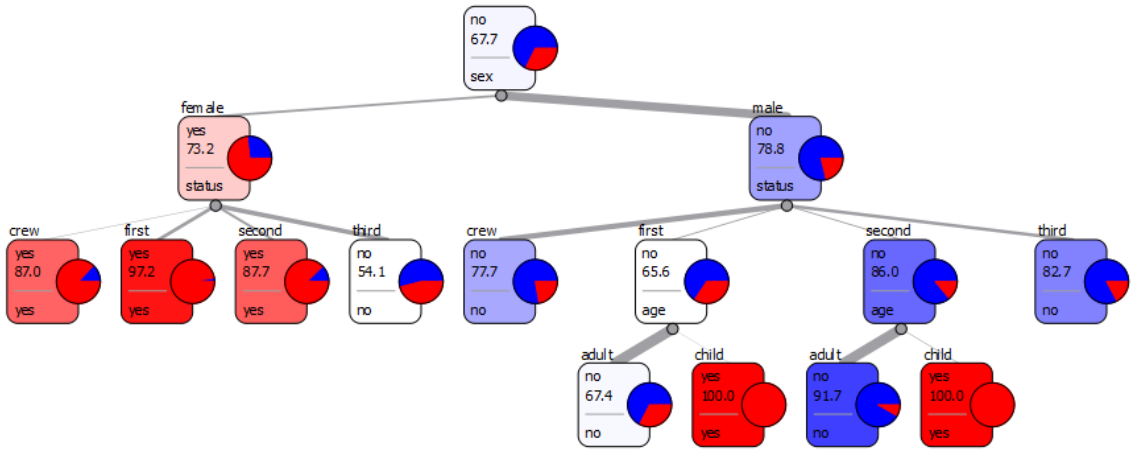
Obravnava neznanih vrednosti atributov pri gradnji drevesa. Te lahko preprosto zanemarimo oziroma jih ne upoštevamo pri ocenjevanju verjetnosti. Lahko pa jih upoštevamo tako, da ocenimo verjetnosti posameznih vrednosti in primere z neznanimi vrednostmi ustrezno "razpršimo" po podmnožicah.

Neznane vrednosti atributov in uvrščanje. Pri uvrščanju primerov, pri katerih nekateri atributi nimajo danih vrednosti (pravimo, da so te vrednosti neznane), lahko naletimo na vozlišče, ki zahteva poznavanje enega od teh atributov. Uvrstitev poiščemo tako, da se spustimo po vseh njegovih vejah in v vsaki poiščemo ustrezni razred, ki ga utežimo z verjetnostjo vrednosti atributa v danem vozlišču. Slednjo seveda dobimo pri učenju drevesa in si jo moramo v vozlišču, za namene uvrščanja, zapomniti.

Rezanje dreves. Drevesa tipično vodijo k drobljenju oziroma fragmentaciji problemske domene. Na razred namesto iz celotne množice primerov sklepamo iz veliko manjših podmnožic, ki pa s stališča statistike niso nujno ustrezne (bi bilo prav o pravičnosti igralne kocke razmišljati že po nekaj metih?). Bolje bi zato bilo zgraditi drevesa tako, da ta ne vodijo k premajhnim množicam v listih. Postopek se imenuje rezanje dreves. Lahko ga izvajamo vnaprej (na primer, postavimo mejo za najmanjše število primerov v listih), ali pa vzvratno tako, da zgradimo drevo in potem rekurzivno odstranjujemo nezanesljiva spodnja vozlišča. Eden od znanih postopkov za slednje imenujemo rezanje z zmanjševanjem napake (angl. *minimal error prunning*) in temelji na spremenjenem ocenjevanju verjetnosti tako, da pri njej upoštevamo še nekaj navideznih primerov, po enega iz vsakega razreda (t.im. ocena po Laplace-u []) ali pa m primerov, ki so porazdeljeni skladno z porazdelitvijo razredov v učni množici (t.im. m-ocena verjetnosti []).

Združevanje vrednosti atributov. Nekateri diskretni atributi uporabljajo lahko (zelo) veliko število vrednosti in bi njihova uporaba pri gradnji dreves lahko vodila v takojšnjo veliko razpršitev primerov. Zato raje te vrednosti uredimo v podmnožice, ki jih poiščemo tako, da ima tako dobljeni atribut največjo informativnost. Zakaj za ocenjevanje slednje osnovna mera z informacijskim prispevkom ni primerna?

Tehnika gradnje in uporabe klasifikacijskih dreves ima vrsto prednosti. Je sorazmerno hitra, enostavna, zna obravnavati kakršnekoli podatke, ki jih lahko zapišemo v atributni obliki, deluje tudi na zelo velikih množicah podatkov. Uvrščanje je hitro. Model je odprt, ter ga je moč enostavno vizualizirati (slika **??**) in ob njem razpravljati o možnih vzorcih v podatkih, ki jih je algoritem odkril.



Slika 7.3: Klasifikacijsko drevo zgrajeno iz podatkov o potnikih na ladji Titanic. Razred govori o preživetju nesreče. Debelina povezav med vozlišči drevesa ustreza številu primerov na teh povezavah. Barva vozlišč govori o verjetnosti prevladujočega razreda.

Klasifikacijska drevesa pa imajo tudi pomanjkljivosti. O problemu fragmentacije, ki omejuje splošnost dreves in povzroči, da drevo sklepa o razredu na premajhnem vzorcu primerov, smo že govorili (kje? kdaj?). Tudi predstavljeni algoritem ne odkrije optimalnega drevesa, saj ne implementira vračanja in s tem kompleksnejše optimizacije. Drevesa so lahko izjemno velika in se lahko hitro preveč prilagodijo učnim podatkom. V splošnem drevo lahko modelira kompleksnejše povezave med primeri, na primer tudi take iz tabele **??**, a njih odkritje je pri uporabi univariatnih mer prepuščeno zgolj precej neverjetnemu naključju. Seveda pa lahko gradnjo dreves usmerjamo tudi s kontekstnimi merami kot je ReliefF, a s tem močno upočasnimo postopek gradnje.

Drevesa so prav tako lahko občutljiva na zelo majhne spremembe v učni množici. Prav tako se lahko zgodi, da sta dva ali več atributov med seboj zelo primerljiva po informacijski vsebini, in da majhne spremembe vplivajo na to, kateri atribut bomo izbrali. To situacijo lahko opazimo v korenu ali pa kakšnem drugem od notranjih vozlišč. V obeh primerih so lahko spremembe v strukturi drevesa izjemno velike, še posebej, če gre za vozlišče blizu vrha drevesa. Ta občutljivost je seveda nezaželena. Domenskim ekspertom bi želeli pokazati model, ki je stabilen, in se ne spreminja ob najmanjši spremembi učne množice. Izkorišča pa jo tehnika, o kateri govorimo spodaj.

7.3 Naključni gozd

Naključni gozd (angl. *random forest*) v uvrščanju sestavlja skupina klasifikacijskih dreves, ki pri klasifikaciji primera v razred glasuje []. Naključni gozd napove razred, kamor primer uvršča večina klasifikacijskih dreves, ki gozd sestavlja. Najbolje bi bilo, da gozd sestavljajo

drevesa, ki so med seboj različna in kjer vsako od dreves odkrije kakšen svoj, zanimiv koncept, ki se je skrival v podatkih. Podatke ti modeli osvetlijo iz različnih strani in potem skupaj o razredu glasuje. Več dreves več ve.

Taka drevesa dobimo v gozdu tako, da postopek malo "ponaključimo". In sicer:

1. Pričnemo z učno množico D, ki vsebuje N primerov in M atributov.

2. Izberemo število m, ki nam pove, med koliko naključno izbranih atributov iz primerov v vozliščih drevesa bomo izbrali najboljšega. Število m naj bo veliko manjše od števila atributov, na primer $m = \sqrt{M}$. Izberemo tudi število dreves K v gozdu, na primer $K = 100$, ki ga želimo zgraditi. Nastavimo števec dreves, $i \leftarrow 1$.

3. Med N primeri učne množice D naključno z vračanjem izberemo novo množico primerov D_i enake velikosti. To vrsto vzorčenja imenujemo metoda stremena (angl. *bootstrap*). Seveda bodo v našem vzorcu posamezni primeri iz osnovne učne množice podvojeni.

4. Na množici primerov D_i zgradimo klasifikacijsko drevo T_i, a tako, da v vsakem vozlišču naključno izberemo m atributov, ki so uporabljeni v primerih vozlišča, in med njimi za delitev teh primerov izberemo najbolj informativnega upoštevaje razred primerov.

5. Če $i == K$ smo končali, sicer $i \leftarrow i + 1$ in nazaj na korak 3.

Tehnika naključnega gozda je danes ena od najbolj zanesljivih tehnik uvrščanja. Njene napovedne točnosti so tipično med najboljšimi. Tehnika podeduje vse prednosti klasifikacijskih dreves, zgubi pa možnost interpretacije modela, saj tega sedaj sestavlja vrsta modelov katerih vpliv na odločitve ni jasen in je odvisen od konteksta. Pristop v fazi gradnje modela ni med najhitrejšimi, ga pa je enostavno povzporediti.

Brieman [] je ob tem, ko je predlagal algoritem za gradnjo naključnega gozda, predlagal tudi metodo, ki oceni pomembnost, ki jo v gozdu imajo posamezni atributi pri pravilnem uvrščanju. Vsakemu vzorcu primerov D_i, na katerem smo zgradi drevo T_i, določimo njegov komplement $D'_i = D - D_i$. Množica D'_i je torej sestavljena iz primerov osnovne učne množice D, ki jih ni v vzorcu D_i (angl. *out-of-bag set*). Naj bo C_i število primerov iz množice D_i, za katere je drevo T_i pravilno napovedalo razred. Sedaj za vsak atribut X premešajmo njegove vrednosti v D_i in na taki, spremenjeni množici, izmerimo število pravilnih napovedi $C_{i,X}$. Pomembnost (angl. *raw importance*) atributa določimo kot:

$$\mathrm{RI}(X) = \frac{1}{K} \sum_{i=1}^{K} C_i - C_{i,X}$$

Poglavje 8

Priporočilni sistemi

Priporočilni sistemi so danes v računalništvu in informatiki prisotni praktično povsod. Še najbolj očitno se z njimi srečamo pri obisku raznih spletnih strani, kot so Facebook, eBay, Amazon, IMDB, Netflix in podobnih. Te strani si vneto beležijo podatke o naših obiskih, ogledu izbranih podstrani, kupljenih izdelkih, ki jih strani ponujajo, ali pa odzivih na ponujene informacije (so nam te všeč ali ne?). Samo na podlagih naših podatkov bi te strani ne bile preveč "pametne". A ker se tam zbirajo podatki tudi o vseh ostalih obiskovalcih, je lahko na ta način zbranih informacij hitro dovolj. "Inteligenco" te strani kažejo v vsebinah, ki so prilagojene nam, uporabnikom. Strani nam svetujejo, kaj vse bi se nam splačalo pogledati, katere izdelke ali druge strani bi lahko bile zanimive in kaj nasploh lahko še počnemo glede na naše, največkrat implicitno izražene interese.

Jedro takih spletnih strani, programov in sistemov so priporočilni sistemi. Ti lahko na podlagi zbranih informacij o uporabnikih sklepajo o uporabnikovih preferencah, željah in interesih. Za posameznega uporabnika lahko recimo na podlagi njemu podobnih uporabnikov napovedo, kaj bi uporabnika lahko (še) zanimalo in mu ponudijo ogled stvari, ki jih ta do sedaj še ni pogledal, pa bi ga morda lahko navdušile. Uporabnost in privlačnost teh priporočilnih storitev ter posledično njihova finančna uspešnost je lahko zelo odvisna od kvalitete uporabljenih priporočilnih sistemov oziroma temu, kako dobro ti modelirajo dejanske uporabnikove interese.

Priporočilni sistemi so lahko uporabni v trženju, analitiki in priporočanju v socialnih omrežjih, biomedicini, humanistiki, astronomiji, kemiji – pravzaprav jih lahko uporabimo prav na vseh področjih, kjer se zbirajo podatki o uporabnikih in stvareh, ki jih neka spletna stran ali programskih sistem, ali pa dejanska fizična trgovina ponuja. V tem poglavju se bomo osredotočili na priporočilne sisteme, ki delujejo na preferencah, ki so jih uporabniki eksplicitno ali implicitno podali o množici stvari. Ko govorimo o "stvareh" mislimo na izdelke, spletne strani, filme, glasbo, kakršnekoli koncepte, torej na karkoli, o čemer so se uporabniki preko neke storitve ali spletne strani izrekli. Stvari so uporabniku lahko bile všeč ali ne, lahko jih je številčno ocenil, morda samo označil zanimive izdelke. Lahko pa si je sistem samo zapomnil,

katere spletne strani ali storitve je uporabnik izbral, katero glasbo je poslušal ali kateri film si je ogledal. Vrsto priporočilnih sistemov, ki temelji na takem preferenčnem znanju in pri podajanju priporočil uporablja uporabniške profile vseh ostalih uporabnikov v sistemu imenujemo skupinsko filtriranje (angl. *collaborative filtering*).

Čeprav bomo v tem poglavju govorili samo o skupinskem filtriranju, naj tu omenimo, da obstajajo tudi drugačni pristopi k priporočanju. Na spletu recimo lahko najdemo sezname najboljših izdelkov določenega tipa, ki jih je priporočil nek urednik ali pisatelj bloga. Spletne strani, kot so na primer domača enaa.com, nam na vhodni strani predlaga izdelke, ki so najbrž izbrani med najbolj dostopanimi ali pa so bili v preteklem obdobju s strani vseh uporabnikov najbolje ocenjeni. Tu gre za enostavno združevanje ocen uporabnikov, ali pa uporabo enostavnih postopkov štetja dostopov uporabnikov do izdelkov.

Uporabimo lahko tudi dodatna znanja o uporabnikih in stvareh. Uporabnike lahko opišemo s starostjo, spolom, naštejmo interesna področja, ki jih zanimajo, opišemo, kaj delajo v službi, kakšni so njihovi hobiji, ali pa celo s kom se družijo. Stvarem lahko določimo njihov namen, jih opremimo s ključnimi besedami iz opisa, razvrstimo v razne skupine. Na podlagi teh podatkov lahko priporočilni sistemi sklepajo o nam podobnih uporabnikih ali pa stvareh, ki so podobne tem, ki smo jih že pozitivno ocenili. Priporočilne sisteme, ki temeljijo na takem dodatnem poznavanju uporabnikov in stvari (angl. *knowledge-based recommendation systems*) je morda, zaradi potrebnih dodatnih podatkov, je tipično težje vzpostaviti od sistemov skupinskega filtriranja, njihova točnost pa je presenetljivo lahko celo slabša od točnosti slednjih. Uporaba metod, ki uporabljajo dodatna znanja o uporabnikih in stvareh, je lahko ključna pri hladnem zagonu sistema (angl. *cold start*), ko uporabnik storitve še ni uporabljal in zanj še nimamo nimamo podatkov o preferencah oziroma všečnosti stvari, na katerih bi temeljila priporočila skupinskega filtriranja.

V nadaljevanju spregovorimo torej samo o tehnikah skupinskega filtriranja. Priporočilnim sistemom, ki uporabljajo dodatna znanja, se izognemo, bo pa radovedni bralec lahko sam razpoznal, da bi bilo moč te tehnike enostavno razviti in pri tem uporabiti zelo podobne koncepte kot pri skupinskem filtriranju. Oba pristopa namreč temeljita na ocenjevanju podobnosti med uporabniki in med stvarmi, razlikujeta se le v načinu, kako te podobnosti izračunamo in v podatkih, ki jih za to uporabimo.

8.1 Podatki

Naš cilj je razviti priporočilni sistem za skupino uporabnikov $\mathcal{U}$, ki lahko izbirajo med množico stvari $\mathcal{I}$, kjer bomo imeli N različnih uporabnikov ($|\mathcal{U}| = N$) in M različnih stvari ($\mathcal{I} = M$). Preferenčno znanje $\mathcal{R}$ bomo zapisali kot množico trojk (u, i, r) za uporabnika $u \in \mathcal{U}$, stvar $i \in \mathcal{I}$ in preferenco r, ki bo podala oceno oziroma zaželenosti stvari i s strani uporabnika u.

Preference merimo v neki izbrani merski lestvici. Na primer, ocene zapišemo s celimi števili od 1 do 5, ali pa od 0 do 10, ali pa z realnimi števili od 0.0 do 1.0 ali pa od -1.0 do 1.0.

Včasih pa imamo na voljo samo oznake, ali je uporabnik za določeno stvar izrazil zanimanje. Slednji primer je težji, saj nimamo urejenih ocen oziroma imamo samo označene pozitivne preference. Predpostavka, da so vse stvari, ki jih uporabnik ni označil, zanj nezanimive, pa je lahko napačna. V tem poglavju se bomo predvsem ukvarjali s podatki, kjer imamo na voljo numerične ocene, tu in tam pa bomo namignili, kako bi lahko obravnavali podatki, ki vsebujejo samo seznam zanimivih stvari, za katere uporabnik ni podal preferenčne ocene v neki numerični lestvici.

Priporočilne sisteme lahko razvijamo za manjše skupine uporabnikov in stvari, opravka pa imamo lahko tudi z večjimi množicami, kjer je lahko število uporabnikov in stvari tudi po več tisoč, sto tisoč ali pa celo nekaj milijonov. Celotno preferenčno znanje, torej ocene za vse kombinacije uporabnikov in stvari tako ne bo nikoli dostopno, opravka pa bomo imeli samo z manjšim vzorcem preferenčnega znanja, ki ga označimo s $\tau' \subset \mathcal{R}$. Ker bomo na podlagi tega vzorca zgradili naš priporočilni sistem, ga tu imenujmo učna množica. Predpostavili bomo, da obstaja največ ena ocena za vsako kombinacijo (u, i), torej kombinacijo uporabnik-stvar. Ker se bomo še največkrat spraševali o tem, ali naša učna množica vsebuje oceno za dan par (u, i), uvedimo množico parov τ, za katere imamo dano preferenčno znanje:

$$\tau = \{(u, i); \exists r : (u, i, r) \in \tau'\}$$

Preferenčno oceno uporabnika u za stvar i zapišimo z $r_{u,i}$. Množica ocen izbranega uporabnika u tvori uporabniški profil $\mathbf{r}_u$. Podobno lahko vse ocene uporabnikov za izbrano stvar i zberemo v stvarni profil oziroma profil stvari $\mathbf{r}_i$.

Primer s seznamom podanega preferenčnega znanja podaja tabela **??**, kjer so

$$\begin{aligned} \mathcal{U} &= \{\text{Mojca, Miha, Sandra, Jana, Tina, Nik, Janez, Cene}\} \\ \mathcal{I} &= \{\text{olive, vampi, testenine, blitva, ocvirki, krofi}\} \end{aligned}$$

Namesto podajanja preferenčnega znanja oziroma naše učne množice s trojkami (uporabnik, stvar, ocena) bi lahko, konceptualno, to znanje zapisali v obliki matrike (tabela **??**). "Konceptualno" namreč zato, ker bi za večjo skupino uporabnikov in stvari take matrike postale izjemno velike in ker jih kot take, v resnih računalniških implementacijah, nikoli eksplicitno ne uporabimo, ampak namesto njih podatke zapišemo v slovarje.

8.2 Skupinsko filtriranje na podlagi podobnosti

Skupinsko filtriranje (angl. *collaborative filtering*) je danes ena od osnovnih, morda glavnih tehnik za izdelavo praktičnih priporočilnih sistemov, ki temeljijo na preferenčnem znanju množic uporabnikov. Za danega uporabnika v teh sistemih predpostavimo, da je ta že ocenil določene stvari. Priporočilo, katera stvar ga bi zanimala med temi, za katere še ni podal

Tabela 8.1: Primer preferenčnega znanja podanega kot množica trojk (uporabnik, stvar, ocena).

uporabnik	izdelek	ocena
Mojca	olive	5.0
Mojca	vampi	1.2
Mojca	testenine	4.1
Mojca	blitva	5.0
Miha	ocvirki	4.3
Miha	vampi	4.9
Miha	blitva	1.2
Sandra	ocvirki	2.3
Sandra	olive	5.0
Sandra	vampi	0.1
Sandra	blitva	5.0
Jana	ocvirki	4.0
Jana	vampi	3.0
Jana	krofi	4.5
Tina	olive	5.0
Tina	krofi	3.5
Tina	testenine	2.1
Tina	blitva	4.6
Nik	ocvirki	2.0
Nik	olive	3.8
Nik	vampi	0.3
Janez	olive	0.3
Janez	vampi	4.8
Janez	testenine	2.5
Cene	ocvirki	3.9
Cene	krofi	4.8
Cene	blitva	3.0

Tabela 8.2: Primer preferenčnega znanja podanega z matriko

	olive	vampi	testenine	blitva	ocvirki	krofi
Mojca	5.0	1.2	4.1	5.0		
Miha	4.3	4.9		1.2		
Sandra	5.0	0.1		5.0	2.3	
Jana		3.0			3.0	4.5
Tina	5.0		2.1	4.6		3.5
Nik	3.8				2.0	
Janez	0.3	4.8	2.5			
Cene				3.0	3.9	4.8

preference, pridobimo tako, da za vse take stvari izračunamo pripadajočo oceno (preferenco) ter uporabniku priporočimo stvar, ki je bila na ta način najbolje ocenjena. Priporočila za danega uporabnika razvijemo ob predpostavki, da je v celotni množici uporabnikov nekaj takih, ki so izbranemu uporabniku podobni. Stvari, ki so všeč njim, bodo najbrž všeč tudi izbranemu uporabniku.

Najbolj neposreden način, ki se ga pri napovedovanju ocen lahko poslužimo je, da na podlagi že danih ocen izračunamo, kako podobni so med seboj uporabniki ter ocene ostalih uporabnikov za posamezno stvar utežimo s podobnostjo ciljnemu uporabniku.

8.2.1 Podobnost med uporabniki

Za danega uporabnika u nas zanima, kako podobni so mu ostali uporabniki. Določiti moramo mero podobnosti med uporabniškimi profili. Spomnimo se: uporabniški profil je množica vseh ocen stvari, ki jih je uporabnik ocenil oziroma za katere imamo podatke v učni množici. Nekaj kandidatov za mere podobnosti poznamo že iz prejšnjih poglavij, zato tu le na hitro omenimo, da je primerna in mnogokrat uporabljana mera kosinusne podobnosti:

$$s_{\mathrm{C}}(u, u') = \frac{\mathbf{r}_u \mathbf{r}_{u'}}{|\mathbf{r}_u|\,|\mathbf{r}_{u'}|}$$

Pri kosinusni podobnosti v skalarnem produktu manjkajoče podatke izpustimo. Ko računamo razdaljo med Mojco in Mihom, na primer, bomo upoštevali samo ocene za olive, vampe in blitvo. Njuna profila za te tri stvari bosta zato $(5.0, 1.2, 5.0)$ in $(4.3, 4.9, 1.2)$. Skalarni produkt teh dveh vektorjev znaša 33.38. Pri računanju dolžine pa upoštevamo celoten Mojčin in Mihov profil. Dolžini njunih profilov sta tako 8.26 in 6.63, torej je njuna kosinusna podobnost enaka 0.61. Podobnost med Mojco in Tino (vektorja, ki ju primerjamo, sta $(5.0, 4.1, 5.0)$ in $(5.0, 2.1, 4.6)$) je večja in znaša 0.86. Opazimo tudi lahko, da bo recimo podobnost med Janezom in Cenetom enaka 0.0, saj ne obstaja izdelek, ki bi ga ta dva uporabnika oba ocenila.

V primeru, da preferenčnih ocen nimamo in uporabniške profile tvori samo seznam stvari, ki so uporabniku všeč, lahko oceno podobnosti med uporabniki izračunamo z mero po Jaccardu:

$$s_{\mathrm{J}}(u, u') = \frac{\|\mathbf{r}_u \cap \mathbf{r}_{u'}\|}{\|\mathbf{r}_u \cup \mathbf{r}_{u'}\|}$$

kjer smo, na primer, z $\|\mathbf{r}_u \cap \mathbf{r}_{u'}\|$ označili število stvari, ki so všeč tako uporabniku u kot uporabniku u'.

8.2.2 Priporočila na podlagi uporabniških profilov in podobnosti med uporabniki

Za danega uporabnika u želimo z uporabo učnih podatkov oceniti, kakšna je njegova preferenčna ocena za stvar i. To označimo z $\hat{r}_{ui}$, saj gre za oceno oziroma približek. Preferenčno

oceno lahko na ta način pridobimo tudi za že obstoječo oceno iz učne množice, a bo ta verjetno različna od ocene r_{ui}, ki jo je podal uporabnik in bomo pri njeni napovedi storili določeno napako.

Izhodišče za izračun preference $\hat{r}_{ui}$, torej ocene koristnosti stvari i za uporabnika u, so podobnosti med uporabniki. Ideja pa je enostavna: pogledamo, kdo so vsi uporabniki, ki so ocenili stvar i. Njihove ocene utežimo s podobnostjo s ciljnim uporabnikom u ter dobljeno vsoto normiramo z vsoto uteži:

$$\hat{r}_{ui} = \frac{\sum\limits_{u':(u',i)\in\tau,\ u'\neq u} s(u,u') \times r_{u'i}}{\sum\limits_{u':(u',i)\in\tau,\ u'\neq u} s(u,u')} \tag{8.1}$$

8.2.3 Priporočila na podlagi profilov stvari in podobnosti med stvarmi

Tudi stvari imajo svoje profile. Če so uporabniški profili vrstični vektorji v preferenčni matriki, katere primer je prikazan v tabeli **??**, so profili stvari stolpci te matrike, oziroma stolpčni vektorji. Za stvar i njen profil zapišemo z $\mathbf{r}_i$. Podobno kot za dva uporabnika lahko podobnost med stvarmi merimo, na primer s kosinusno razdaljo:

$$s_{\mathrm{c}}(i,i') = \frac{\mathbf{r}_i \mathbf{r}_{i'}}{|\mathbf{r}_i|\ |\mathbf{r}_{i'}|}$$

Sedaj pa nazaj k priporočanju. Želeli bi torej oceniti preferenco $\hat{r}_{ui}$, ki jo ima za stvar i uporabnik u. Predpostavljamo, da je uporabnik že ocenil nekaj drugih stvari (vsako od teh bomo označili z i'). Oceno za stvar i lahko izračunamo iz teh ocen, a pri tem močneje upoštevamo tiste stvari, ki so podobne stvari i, ki jo ocenjujemo:

$$\hat{r}_{ui} = \frac{\sum\limits_{i':(u,i')\in\tau,\ i'\neq i} s(i,i') \times r_{ui'}}{\sum\limits_{i':(u,i')\in\tau,\ i'\neq i} s(i,i')} \tag{8.2}$$

Ta izračun je na moč podoben izračunu iz prejšnjega razdelka, le da tu utežimo ocene našega ciljnega uporabnika, v prejšnjem razdelku pa smo utežili ocene ostalih uporabnikov.

8.2.4 Priporočanje na podlagi podobnosti uporabnikov in stvari

Priporočilno tehniko iz prejšnjih dveh podpoglavij lahko združimo. Iz učne množice lahko vzamemo poljubno oceno $r_{u'i'}$ ter jo utežimo glede na podobnost s ciljnim uporabnikom u in ciljno stvarjo i, za katero oceno $\hat{r}_{ui}$ iščemo. Utežena vsota vseh teh ocen je naš približek za

$\hat{r}_{ui}$:

$$\hat{r}_{ui} = \frac{\sum_{(u',i')\in\tau,\ (u',i')\neq(u,i)} s(u,u') \times s(i,i') \times r_{u'i'}}{\sum_{(u',i')\in\tau,\ (u',i')\neq(u,i)} s(u,u') \times s(i,i')} \tag{8.3}$$

Na prvi pogled se nam taka rešitev lahko zazdi še najboljša, oziroma bolj primerna od teh, ki smo jih omenjali v prejšnjih dveh podpoglavjih. A njena očitna slabost je časovna kompleksnost, saj moramo poznati praktično vse pare podobnosti, torej med ciljnim in vsemi ostalimi uporabniki in ciljno in vsemi ostalimi stvarmi. V prejšnjih dveh podpoglavjih smo namreč računali na to, da so ciljno stvar i ocenili samo nekateri uporabniki in je bilo potrebno izračunati podobnosti samo za te. Oziroma, računali smo, da je uporabnik u ocenil samo manjši nabor stvari in da je samo te potrebno primerjati s ciljno stvarjo i.

8.2.5 Priporočanje kar tako

Vrednost $\hat{r}_{ui}$ lahko ocenimo kar s povprečno oceno stvari i vseh (ostalih) uporabnikov, ki so to stvar ocenili:

$$\hat{r}_{ui} = \frac{\sum_{(u',i)\in\tau,\ u'\neq u} r_{u'i}}{|\{(u',i) \in \tau,\ i' \neq i\}|} \tag{8.4}$$

Ta ocena bo neodvisna od uporabnika, nam pa lahko služi za primerjavo z ostalimi načini izračuna $\hat{r}_{ui}$, kot smo jih podali zgoraj. To oceno lahko morda še dodatno popravimo tako, da upoštevamo povprečno oceno uporabnika:

$$\hat{r}_{ui} = \frac{1}{2}\frac{\sum_{(u',i)\in\tau,\ u'\neq u} r_{u'i}}{|\{(u',i) \in \tau,\ i' \neq i\}|} + \frac{1}{2}\frac{\sum_{(u,i')\in\tau,\ i'\neq i} r_{ui'}}{|\{(u,i') \in \tau,\ i' \neq i\}|} \tag{8.5}$$

8.3 Pristranost

Uporabniki smo v naših ocenah pristrani: nekateri uporabniki bolj vedre narave bodo stvari ocenjevali z višjimi ocenami, nekateri uporabniki pa bodo le stežka dajali najvišje ocene. Problemu pristranosti uporabnikov se lahko vsaj deloma izognemo tako, da vsem ocenam iz učne množice odštejemo povprečno oceno uporabnika. Se pravi r_{ui} nadomestimo z $r_{ui} - \overline{\mathbf{r}_u}$. Tako popravljene ocene potem uporabimo pri učenju. Pri napovedovanju pa ne smemo pozabiti, da smo povprečne ocene uporabnika pred tem odšteli, in zato za par (u,i) napovemo preferenčno oceno z $\hat{r}_{ui} + \overline{\mathbf{r}_u}$.

8.4 Ocenjevanje kvalitete priporočilnih sistemov

Tudi tu lahko uporabimo prečno preverjanje, ki ga tokrat izvedemo kar nad množico trojk (u, i, r_{ui}). Torej tako, kot če bi vzorec iz tabele **??** vzeli za učenje, na preostalem delu pa preverili naše izračunane preferenčne ocene. Za ocenjevanje kvalitete priporočanja lahko uporabimo iste mere, kot smo jih uporabili pri regresiji, torej na primer RMSE ali pa R^2.

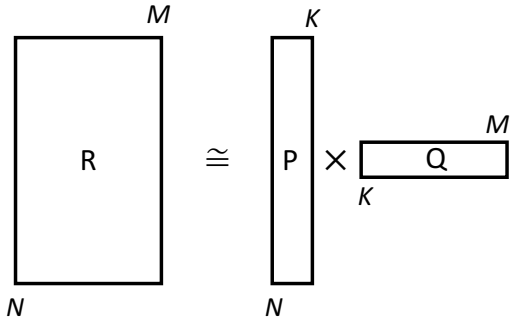
8.5 Priporočanje z matričnim razcepom

V tabeli **??** smo učne podatke predstavili kot matriko R, katere elementi so ocene r_{ui} uporabnika u za stvar i. Matrika R je redka, za njo pa mora priporočilni sistem znati oceniti vrednost manjkajočih elementov. Ker je uporabnikov N, stvari pa M, je matrika R oblike $M \times N$.Tipično sta dimenziji M in N visoki: uporabnikov je lahko nekaj tisoč, deset tisoč ali celo milijon, stvari pa ravno tako.

Že iz poglavja o gručenju vemo, da lahko stvari in uporabnike razvrstimo v skupine in za te potem poiščemo tipične stvari in tipične uporabnike. Na primer, pri izposoji filmov bi lahko zaznali, da obstajajo uporabniki storitve, ki radi gledajo večinoma akcijske filme, pa uporabniki, ki gledajo samo filme, ki so narejeni v Hollywoodu ter uporabniki, ki jim to ne pade na misel in si sposojajo samo umetniške filme iz manjših produkcijskih hiš. Uporabnike bi zato lahko predstavili s kratkimi profili, ki bi nam poročali o zaželenosti posameznih filmskih zvrsti. Pravimo, da bi na ta način uporabnike opisali v latentnem prostoru filmov, torej v prostoru filmskih kategorij, ki jih moramo iz podatkov šele razbrati. Po drugi strani pa bi tudi filme lahko opisali s profili v latentnem prostoru skupin uporabnikov, torej skupin, ki jih prav tako moramo šele pridobiti. Ta opis, predstavitev podatkov v latentnem, navideznem prostoru skupin stvari in skupin uporabnikov lahko pridobimo z enotnim postopkom, ki mu pravimo matrični razcep.

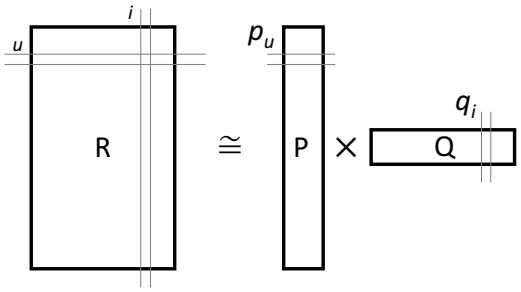
V pristopu z matričnim razcepom matriko $\mathrm{R}_{N\times M}$ predstavimo z matrikami $\mathrm{P}_{N\times K}$ in $\mathrm{Q}_{K\times M}$ tako, da velja $\mathrm{R} \approx \mathrm{PQ}$ in da je $K << M, N$ (slika **??**). Torej, tako, da je produkt matrik P in Q po elementih čim bolj podoben prvotni matriki R oziroma elementom, ki so tam definirani. Konstanta K podaja število latentnih dimenzij, in je pri obliki razcepa, kot smo ga določili tu, enaka tako za uporabnike kot za stvari. Pravimo ji tudi stopnja razcepa. Ker je K veliko manjši od dimenzij M in N, v skrajnem primeru pa enak 1, pomeni, da s produktom PQ prav gotovo ne bomo mogli verno predstaviti matriko R, torej tako, da bi vsi elementi produkta bili enaki elementom vhodne matrike R (torej, $\mathrm{PQ} \equiv \mathrm{R}$).

Matrika P je matrika uporabniških profilov v latentnem prostoru filmov, torej prostoru tipičnih zvrsti filmov (teh nimamo, a jih pridobimo z razcepom matrike na produkt dveh matrik). Latentni profil uporabnika u označimo z vektorjem $\mathbf{p}_u$. Podobno je matrika Q matrika profilov stvari v prostoru latentnih uporabnikov, profil stvari i v tem prostoru pa zapišimo z vektorjem $\mathbf{q}_i$. Produkt latentnih profilov ustreza približku vrednosti ocene stvari i uporabnika



Slika 8.1: Razcep matrike R na manjši matriki P in Q.

u, torej $\hat{r}_{ui}$ (slika **??**).



Slika 8.2: Predstavitev uporabnika u z latetnim profilom $\mathbf{p}_u$ in stvari i z latentnim profilom $\mathbf{q}_i$. Skalarni produkt latentnih profilov nam da približek za oceno stvari i uporabnika u, torej $\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$.

Razcep matrike R na matriki P in Q bomo poiskali tako, da bosta matriki P in Q polni, torej, da bodo vse vrednosti teh dveh matrik določene. Prav zato bomo lahko s produktom latentnih profilov lahko poiskali vrednost $\hat{r}_{ui}$ za vsako kombinacijo uporabnika u in stvari i. Ostane nam torej le še, da izračunamo razcep, oziroma določimo matriki P in Q.

8.5.1 Matrični razcep

Z matrikami P in Q lahko izračunamo približek ocene za stvar i uporabnika u, ki je:

$$\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i = \sum_{k=1}^{K} p_{uk} q_{ki} \tag{8.6}$$

Za ocene iz učne množice r_{ui} bi želeli, da je ta približek čim boljši, oziroma da je napaka, ki jo s tem približkom naredimo, čim manjša. Izračunajmo kvadrat napake za oceno stvari i uporabnika u:

$$e_{ui}^2 = (r_{ui} - \hat{r}_{ui})^2 \tag{8.7}$$

Želeli bi poiskati taki matriki P in Q, kjer so te napake čim manjše. Ker bomo pri numeričnem iskanju teh matrik v vsaki iteraciji obravnavali natančno enega med učnimi primeri (torej r_{ui} za določen par $(u, i) \in \tau$), potrebujemo izračunati gradient napake za vsakega od parametrov modela. Naš model sta matriki P in Q, vrednosti parametrov modela pa vsi njuni elementi p_{uk} in q_{ki}. V enačbo za napako (enačba **??**) vstavimo izraz iz enačbe **??**:

$$e_{ui}^2 = \frac{1}{2}\left(r_{ui} - \sum_{k=1}^{K} p_{uk}q_{ki}\right)^2. \tag{8.8}$$

Enačbo smo pomnožili z $\frac{1}{2}$ tako, da se nam dvojka kasneje pri odvajajnju pokrajša. Odvod enačbe je seledeč:

$$\begin{aligned} \frac{\partial e_{ui}^2}{\partial p_{uk}} &= -e_{ui}q_{ki} \\ \frac{\partial e_{ui}^2}{\partial q_{ki}} &= -e_{ui}p_{uk} \end{aligned} \tag{8.9}$$

Uporabimo pristop gradientnega sestopa. Matriki P in Q tokrat na začetku nastavimo na majhne naključne vrednosti. Začetna nastavitev vseh njunih elementov na 0 nas namreč ne bi pripeljala nikamor (zakaj?). Nato iteriramo po vseh elementih (trojkah) v učni množici in za vsak primer ustrezno popravimo odgovarjajoče elemente (profile) P in Q, tako da za vsak $k = 1 \ldots K$ popravimo vrednosti:

$$\begin{aligned} p_{uk} &\leftarrow p_{uk} + \alpha e_{ui}q_{ki} \\ q_{ki} &\leftarrow q_{ki} + \alpha e_{ui}p_{uk} \end{aligned} \tag{8.10}$$

Tudi tokrat, kot pri linearni in logistični regresiji, je α stopnja učenja. Zgornjo enačbo lahko zapišemo tudi vektorsko:

$$\begin{aligned} \mathbf{p}_u &\leftarrow \mathbf{p}_u + \alpha e_{ui}\mathbf{q}_i \\ \mathbf{q}_i &\leftarrow \mathbf{q}_i + \alpha e_{ui}\mathbf{p}_u \end{aligned} \tag{8.11}$$

Postopek iskanja razcepa matrike R na matriki P in Q, kot smo ga opisali zgoraj, imenujemo tudi inkrementalna simultana matrična faktorizacija, algoritem pa s tem dobi akronim ISMF. Gre pravzaprav za stohastični gradientni spust. Stohastični za to, ker optimizacijo vsakokrat vršimo na enem primeru iz učne množice in ne optimiziramo vrednosti P in Q za celotno učno množico hkrati.

8.5.2 Regularizacija

Zgoraj opisan matrični razcep se lahko preveč prilagodi učnim podatkom, še posebej pri večjih vrednostih K. Tudi tu se preveliko prileganje odrazi v velikih vrednostih parametrov modela, torej velikih vrednostih elementov matrik P in Q. Preveliko prileganje preprečimo tako, da v našo ciljno funkcijo vključimo kaznovanje, ki izhaja iz velikosti parametrov:

$$e'^{\,2}_{ui} = \frac{1}{2}(e^2_{ui} + \eta \mathbf{p}_u^T \mathbf{p}_u + \eta \mathbf{q}_i^T \mathbf{q}_i) \tag{8.12}$$

Gradient, ki ga uporabimo pri metodi sestopa, dobimo s parcialnim odvajanjem zgornje kriterijske funkcije:

$$\begin{aligned} \frac{\partial e'^{\,2}_{ui}}{\partial p_{uk}} &= -e_{ui} q_{ki} + \eta p_{uk} \\ \frac{\partial e'^{\,2}_{ui}}{\partial q_{ki}} &= -e_{ui} p_{uk} + \eta q_{ki} \end{aligned} \tag{8.13}$$

Upoštevaje izračunani gradient je popravek vrednosti elementov matrik P in Q v postopku gradientnega sestopa zapisan vektorsko:

$$\begin{aligned} \mathbf{p}_u &\leftarrow \mathbf{p}_u + \alpha(e_{ui}\mathbf{q}_i - \eta \mathbf{p}_u) \\ \mathbf{q}_i &\leftarrow \mathbf{q}_i + \alpha(e_{ui}\mathbf{p}_u - \eta \mathbf{q}_i) \end{aligned} \tag{8.14}$$

Pri regularizaciji nastopi problem ocene primerne stopnje regularizacije η. Kot pri regresiji in klasifikaciji lahko tudi tu primerno stopnjo regularizacije ocenimo s prečnim preverjanjem in uporabimo η, pri katerem dobimo najmanjšo napako.

8.5.3 Algoritem

Elemente algoritma za matrični razcep smo pravzaprav popisali že zgoraj, a ne bo škodilo, da opišemo celotni postopek še enkrat:

Vhod: učna množica τ', stopnja učenja α, stopnja regularizacije η, stopnja razcepa K
Izhod: matriki P in Q

ponastavi P in Q z naključnimi majhnimi števili $(-0.01 \ldots 0.01)$
ponavljaj:
  za vsak (u, i, r_{ui}) **v** učni množici τ':
    izračunaj e_{ui}
    izračunaj gradient e'_{ui} (enačba **??**)
    za vsak k **v** $1 \ldots K$:

osveži p_{uk} in q_{ki} (enačba **??**)
končaj ob konvergenci

V zgornjem algoritmu se nam je zapisala čudna, zadnja vrstica, saj ustavitvenega kriterija nismo določili. Tudi sicer ga je težko določiti. Lahko bi postopek ustavili takrat, kadar napaka RMSE za celotno učno množico pade pod določeno mejo, ali pa kadar izvedemo več kot določeno število iteracij zgornjega algoritma (zunanja zanka), ali pa kadar se po nekaj urah naveličamo čakati, da se optimizacijskih postopek ustavi. Boljši postopek od zgornjih je, da učno množico razcepimo na dve, novo učno in validacijsko, se na novi učni naučimo, na validacijski pa preverjamo napovedi. Z učenjem prekinemo, ko se nam po na primer dveh iteracijah zunanje zanke napaka na validacijski množici ne zmanjša.

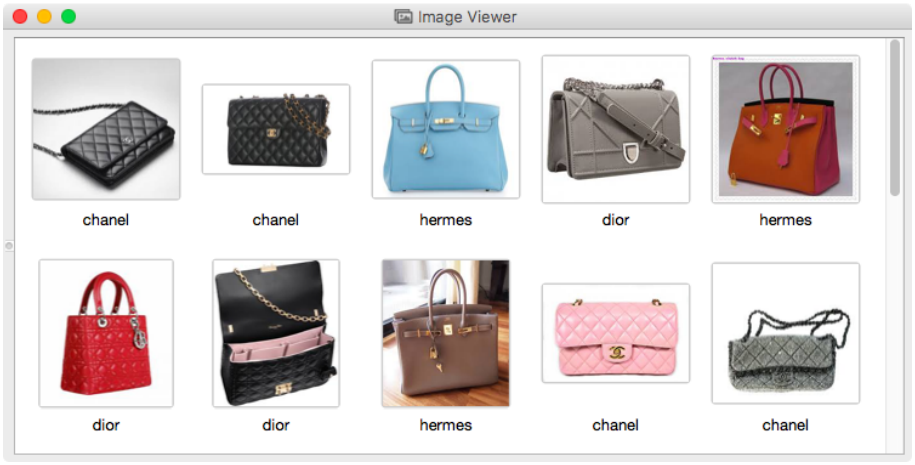
Optimizacija oziroma iskanje primernih matrik P in Q je seveda odvisna tudi od začetne nastavitve matrik. V praksi se nam obnese, da postopek nekajkrat ponovimo, vsakič z drugo inicializacijo, in upoštevamo tisti razcep, pri katerem je RMSE na učni množici, ali pa, še bolje, na validacijski množici, najmanjši.

Poglavje 9

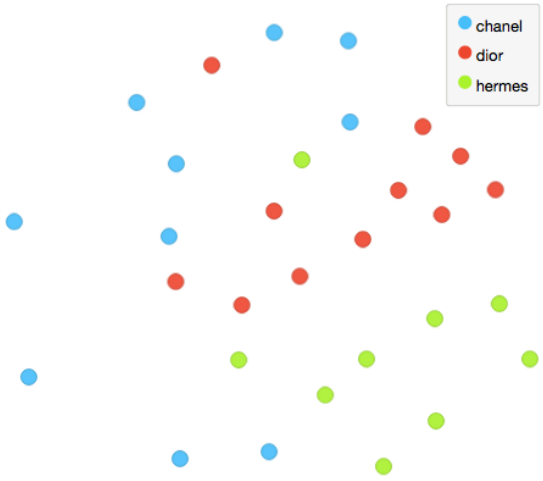
Projekcije in zmanjšanje dimenzionalnosti podatkov

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

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



Slika 9.1: Ženske torbice različnih proizvajalcev.



Slika 9.2: Predstavitev ženskih torbic s slike **??** v dvodimenzionalnem prostoru.

9.1 Metoda glavnih komponent

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

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

Naj bo $x^{(i)}$ primer iz učne množice. Njegova pravokotna projekcija na premico je skalar, $u_1^\intercal x^{(i)} \in \mathbb{R}$. Sama vrednost tega skalarja nam ne pove prav dosti; potrebovali bi neko referenčno točko, od katere bi merili razdalje naših projekcij. To referenčno točko lahko pridobimo kot projekcijo središča podatkov. Naj bo $\bar{x}$ središčna točka podatkov:

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

Projekcija središčne točke na projekcijsko ravnino je $u_1^\intercal \bar{x}$. Primeri v učni množici odstopajo od središčne točke, eni bolj, drugi manj. Projekcijsko premico bi želeli zato poiskati tako, da so ta odstopanja čim bolj zajeta, torej da so projekcije primerov na premico, ki jo določa u_1, čim bolj razpršene. Razpršenost merimo s povprečnim kvadratnim odstopanjem, torej povprečnim kvadratom razdalje projekcije in projekcije središčne točke podatkov. Tako določeni razpršenosti pravimo tudi varianca točk v projekciji:

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

Naš cilj je torej poiskati tak projekcijski vektor u_1, ki maksimizira varianco projekcije $\mathrm{Var}(u_1^\intercal X^\intercal)$. Poigrajmo se malce z enačbo za varianco projekcije:

$$\begin{aligned} \mathrm{Var}(u_1^\intercal X^\intercal) &= \frac{1}{m} \sum_{i=1}^{m} \left(u_1^\intercal x^{(i)} - u_1^\intercal \bar{x} \right)^2 \\ &= \frac{1}{m} \sum_{i=1}^{m} \left(u_1^\intercal x^{(i)} u_1^\intercal x^{(i)} - 2 u_1^\intercal x^{(i)} u_1^\intercal \bar{x} + u_1^\intercal \bar{x} u_1^\intercal \bar{x} \right) \end{aligned} \tag{9.1}$$

Upoštevajmo, da je $u_1^\intercal x$ skalarni produkt dveh vektorjev in da je transponirana vrednost skalarja enaka temu skalarju. Torej je na primer $u_1^\intercal x = (u_1^\intercal x)^\intercal = x^\intercal u_1$. Z upoštevanjem tega se

nam zgornji izraz primerno poenostavi:

$$\begin{aligned}\mathrm{Var}(u_1^\intercal X^\intercal) &= \frac{1}{m}\sum_{i=1}^{m} u_1^\intercal\Big(x^{(i)}x^{(i)\intercal} - 2x^{(i)}\overline{x}^\intercal + \overline{x}\,\overline{x}^\intercal\Big)u_1 \\ &= u_1^\intercal\Big(\frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \overline{x})(x^{(i)} - \overline{x})^\intercal\Big)u_1\end{aligned} \tag{9.2}$$

Produkt $(x^{(i)} - \overline{x})(x^{(i)} - \overline{x})^\intercal$ je matrika velikosti $n \times n$. Matriko sestavljajo produkti vseh možnih parov elementov vektorja $(x^{(i)} - \overline{x})$, torej, v prvi vrstici produkt med prvim elementom in njim samim, pa med prvim in drugim elementom, pa prvim in tretjim in tako naprej. Povprečna vrednost takih matrik, ki jih dobimo s seštevanjem matrik za vsakega od primerov, je kovariančna matrika! Prejšnji stavek smo končali s klicajem zato, ker je kovarianca znan in mnogokrat uporabljan koncept v statistiki in nam pove, kako sta povezani dve naključni spremenljivki, oziroma v našem primeru dva atributa. Označimo kovariančno matriko s S:

$$\mathrm{S} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \overline{x})(x^{(i)} - \overline{x})^\intercal \tag{9.3}$$

V našem primeru imamo n atributov, kar pomeni, da je kovariančna matrika dimenzije $\mathrm{S} = \mathbb{R}^{n\times n}$.

Zapišimo še enkrat dobljeno enačbo za varianco:

$$\mathrm{Var}(u_1^\intercal X^\intercal) = u_1^\intercal \mathrm{S} u_1 \tag{9.4}$$

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

$$f(u_1) = u_1^\intercal \mathrm{S} u_1 + \lambda_1(1 - u_1^\intercal u_1) \tag{9.5}$$

V maksimumu bo odvod zgornje enačbe enak nič:

$$\frac{\partial f(u_1)}{\partial u_1} = \mathrm{S}u_1 - \lambda_1 u_1 = 0 \tag{9.6}$$

kar pomeni,

$$\mathrm{S}u_1 = \lambda_1 u_1 \tag{9.7}$$

Tu je S matrika katere vrednosti poznamo, u_1 je vektor, ki ga iščemo, λ_1 pa skalar. Poznano? Seveda! Zgornjemu pogoju ustreza le tak u_1, ki je lastni vektor matrike S. Ampak korelacijska

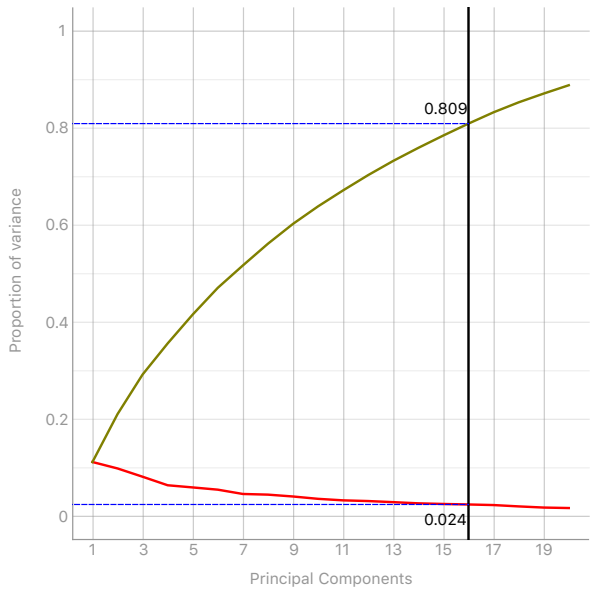
matrika ima več lastnih vektorjev. Kateri med njimi je pravi? Množimo zgornjo enačbo z $u_1^\intercal$:

$$u_1^\intercal S u_1 = u_1^\intercal \lambda_1 u_1 = \lambda_1 u_1^\intercal u_1 = \lambda_1 \tag{9.8}$$

Na levi je izraz za našo varianco $\mathrm{Var}(u_1^\intercal X^\intercal)$, ki jo skušamo maksimizirati. Vemo, da mora biti u_1 lastni vektor kovariančne matrike. Skladno z zgornjo enačbo pa sedaj tudi vemo, da je to lastni vektor z najvišjo pripadajočo lastno vrednostjo.

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

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



Slika 9.3: Graf odvisnosti razložene variance od števila glavnih komponent (zgornja črta) za podatke o torbicah iz slike **??**.

V splošnem nas zanima, koliko najvišje rangiranih dimenzij v novem koordinatnem sis-

temu zares potrebujemo za opis podatkov. Pri tem moramo seveda določiti, kakšen je naš ciljni delež pojasnjene variance. Tipično smo zadovoljni, če izbrano število komponent pojasni vsaj 80% skupne variance. Ker skupno varianco poznamo (enaka je vsoti kvadratov razdalj do centra), je potrebno torej le določiti, koliko začetnih urejenih lastnih vrednosti moramo sešteti, da njihova vsota predstavlja želeni delež razložene variance. Primer grafa, ki kaže odvisnost deleža razložene variance glede na število glavnih komponent kaže slika **??**.

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

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

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

$$Mu = \lambda u \tag{9.9}$$

$$u^\intercal Mu = u^\intercal \lambda u = \lambda u^\intercal u = \lambda \tag{9.10}$$

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

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

9.2 Večrazredno lestvičenje

Pri zmanjšanju dimenzij nam je lahko cilj, da ohranimo razdalje med posameznimi primeri. Tu predpostavimo, da za vsak par primerov znamo med njimi oceniti razdaljo. Se pravi, če imamo primere predstavljene v atributnem zapisu, potem uporabimo recimo Evklidsko razdaljo ali pa kosinusno podobnost. Lepota tehnike, ki jo bomo predstavili v tem razdelku je, da nas ne zanima, kako je bila podobnost oziroma razdalja med primeri ocenjena. Lahko smo na vhodu na primer imeli tekstovne dokumente, za katere razdaljo recimo izračunamo s kakšno kompresijsko tehniko. Ali pa filme, za katere razdaljo izračunamo z ozirom na njihovo gledanost v spletni sposojevalnici. Ali pa čivke, ki jih primerjamo med sabo glede na to, kdo jih je všečkal.

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

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

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

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

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

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

$$\begin{aligned}
\frac{\partial \sigma(\mathbf{X})}{\partial \boldsymbol{\theta}_k} &= \sum_{j \neq i} \frac{\partial \sigma(\mathbf{X})}{\partial d_{ij}} \frac{d_{ij}}{\partial \boldsymbol{\theta}_k} \\
&= -\sum_{j \neq i} 2 \left(\sqrt{(\theta_1^{(i)} - \theta_1^{(j)})^2 + (\theta_2^{(i)} - \theta_2^{(j)})^2} - \delta_{ij} \right) \frac{(\theta_k^{(i)} - \theta_k^{(j)})}{\sqrt{(\theta_1^{(i)} - \theta_1^{(j)})^2 + (\theta 2^{(i)} - \theta_2^{(j)})^2}} \\
&= -\sum_{j \neq i} 2 \left(d_{ij} - \delta_{ij} \right) \frac{(\theta_k^{(i)} - \theta_k^{(j)})}{d_{ij}}
\end{aligned} \tag{9.11}$$

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

torej raje odpovemo.

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

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

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

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

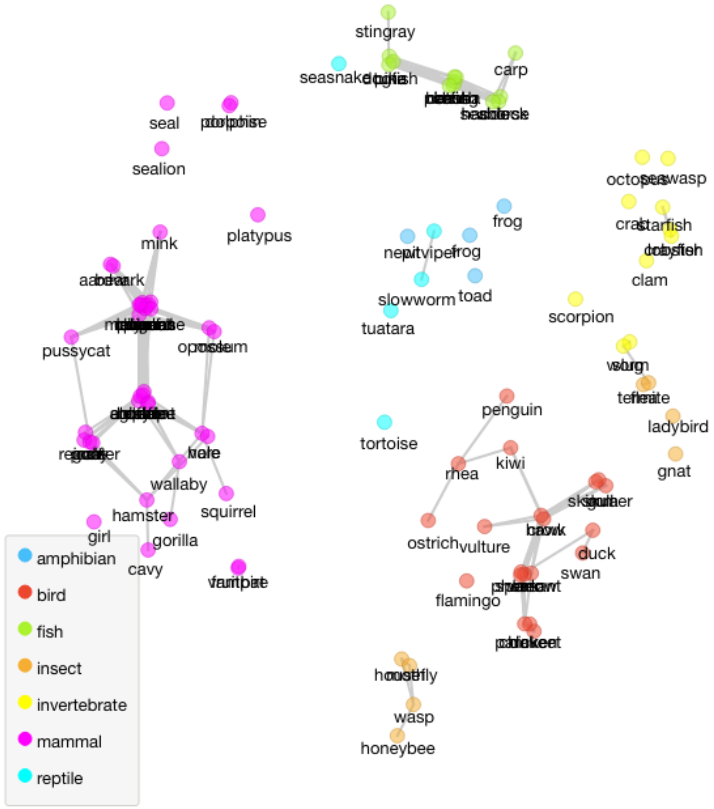
Takšna optimizacija je, tako kot gradientna metoda še vedno iterativna, vendar hitrejša. Koliko hitrejša, je odvisno od tega, kako dobro $g(x, y)$ smo poiskali; $g(x, y)$ se mora čim tesneje prilegati $f(x)$, poleg tega pa jo moramo biti zmožni čim boljše optimizirati.

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

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

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

Čeprav MDS ni uporabljal podatkov o vrstah živali, so različne vrste na zemljevidu dobro ločene. Vidimo tudi nekaj podobnosti med živalmi: morske sesalce je MDS postavil v bližino



Slika 9.4: Zemljevid živali (podatkovna zbirka Zoo). Vrsta živali je označena z barvo. Nekatere oznake so zaradi prekrivanj neberljive in bi razvojnike programa potrebno opozoriti, da v vizualizacijo vključijo optimizacijo postavitve oznak.

rib; dvoživke so blizu rib in nevretenčarjev, predvsem morskih; žuželke so pomešane s ptiči.

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

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

9.3 Stohastična vložitev sosedov

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

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

$$p_{j|i} = \frac{\exp(-\left\|x^{(i)} - x^{(j)}\right\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\left\|x^{(i)} - x^{(k)}\right\|^2 / 2\sigma_i^2)} \tag{9.12}$$

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

Označimo verjetnostne porazdelitve preko vseh sosedov z za primer i in za originalni prostor z P_i in za projekcijski prostor z Q_i. Naša želja je, da bi bili ti porazdelitvi med sabo

čim bolj podobni. Ker ti dve verjetnostni porazdelitve favorizirajo bližnje primere oziroma so za te verjetnosti visoke, podobnost porazdelitev P_i in Q_i pomeni, da je projekcija ohranila bližnje primere za primer i. Mera, ki pove, kako zvesto $q_{j|i}$ sledi $p_{j|i}$ je Kullback-Leibler-jeva divergenca, katere vsoto po primerih želimo minimizirati. Kriterijska funkcija je zato:

$$J(\mathbf{\Theta}) = \sum_i KL(P_i \parallel Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \tag{9.13}$$

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

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

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

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

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

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

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

ter za projekcijski prostor

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

Tudi tu porazdelitvi primerjamo s Kullback-Leiblerjevo divergenco,

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

za katero poiščemo gradient:

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

Zgornja enačba je po strukturi presenetljivo podobna enačbi **??** za gradient pri večrazrednem lestvičenju, kar je glede na drugačen način izpeljave nekoliko presenetljivo, glede na intuitivno razumevanje cilja obeh projekcij pa morda sploh ne. Glavna razlika obeh metod je seveda ocenjevanje razdalj oziroma pojmovanje, oziroma bolje uteževanje sosednosti. Če pri večrazrednem lestvičenju razdalja ni utežena, je pri metodi SNE in izvedenkah stopnja sosednosti eksponencialno padajoča funkcija.

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

Poglavje 10

Povezovalna pravila

Danes vse prodajalne, tako spletne kot te, kamor se moramo sprehoditi ali pa zapeljati z nekim prevoznim sredstvom, beležijo podatke o nakupih. Te, ki tega ne počno, so že propadle ali pa so žal na poti propada. Recimo, prodajalna prehrambenih izdelkov. Oglejmo si en zelo majhen vzorec nakupov (tabela **??**), kjer vsaka vrstica prikazuje, kaj je bilo v nakupovalni košarici. V tem poglavju bomo zadeve zelo poenostavili: kupci bodo ostali anonimni, čas nakupa nas ne bo zanimal, prav tako ne bomo beležili števila nakupljenih stvari. Zanimali nas bodo samo tipi nakupljenih stvari v nakupovalnih košaricah (na primer, mleko, ne pa tri litre mleka, in ne pol štruce kruha).

Nakupovalne košarice smo označili s t_i za i-to nakupovalno košarico. Bolj strokovno vsaki vrstici v tabeli **??** pravimo tudi *transakcija*. Naj bo množica transakcij, katero bi želeli preučiti, $\mathcal{T} = \{t_1, t_2, \ldots t_N\}$. Množico stvari, ki lahko nastopajo v teh transakcijah, označimo z $\mathcal{I} = \{i_1, i_2, \ldots i_M\}$.

Tabela 10.1: Podatki o nakupovalnih košaricah.

nakup	stvari v košarici	poenostavljen zapis
t_1	mleko, kruh	MK
t_2	mleko, sir, zelenjava	MSZ
t_3	kruh, sir, zelenjava	KSZ
t_4	mleko, kruh, sir, zelenjava	MKSZ
t_5	mleko, kruh, sir	MKS
t_6	kruh, sir	KS
t_7	sir	S
t_8	kruh, zelenjava, sir	KZS
t_9	sir, zelenjava	SZ
t_{10}	kruh, sir, zelenjava	KSZ

10.1 Nabori in podnabori

Terkam stvari, kot so na primer {kruh, sir} ali pa {mleko, sir, zelenjava} pravimo *nabor*. Za par naborov X in Y pravimo, da je X podnabor nabora Y, če je vsaka stvar iz nabora X vsebovana v naboru Y. V tem primeru zapišemo, da velja $X \subseteq Y$. Nabor {kruh, sir} je na primer podnabor nabora {kruh, mleko, sir}. Številnost stvari v naboru X označimo $|X|$. V naboru $X = \{\text{kruh}, \text{mleko}, \text{sir}\}$ so tri stvari, $|X| = 3$. Nabor X s številnostjo $|X|$ ima $2^{|X|}$ možnih podnaborov, med katerimi je tudi prazen podnabor $\emptyset$.

10.2 Podpora in pogosti nabori

S $\sigma(X)$ označimo število transakcij, ki vsebuje podnabor X. Podnabor $X = \{\text{mleko}, \text{kruh}\}$ je vsebovan v transakcijah t_1, t_4 in t_5. Število podprtih transakcij za nabor X je zato enako tri, oziroma $\sigma(X) = 3$. Število $\sigma(X)$ formalno določimo s sledečim zapisom:

$$\sigma(X) = |\{t_i | X \subseteq t_i, t_i \in \mathcal{T}\}| \tag{10.1}$$

Iz števila podprtih transakcij za nabor X lahko izračunamo delež podprtih transakcij, torej delež transakcij, ki vsebujejo nabor X:

$$s(X) = \frac{\sigma(X)}{|\mathcal{T}|} = \frac{\sigma(X)}{N} \tag{10.2}$$

Zanima nas, kateri nabori so taki da za njih velja

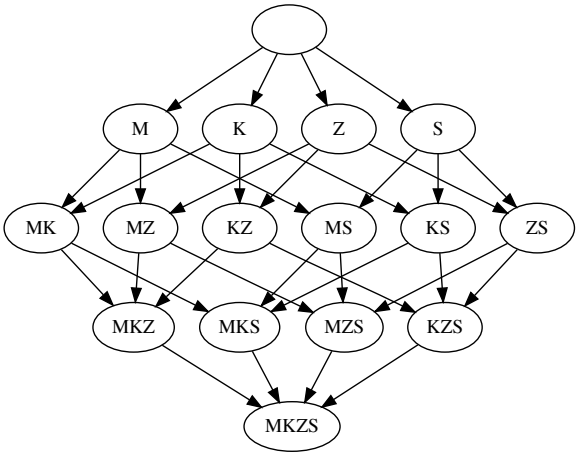
$$s(X) \geq \text{minsupp} \tag{10.3}$$

kjer je minsupp minimalna podpora oziroma parameter, ki ga določi uporabnik. Nabore, kjer velja zgornji izraz, imenujemo *pogosti nabori*.

10.3 Algoritem Apriori

Za iskanje vseh pogostih naborov v množici transakcij bomo razvili algoritem Apriori, ki sta ga sicer predlagala Agrawal in Srikant leta 1994. A preden zapišemo algoritem, razmislimo o nekaj lastnostih prostora vseh možnih naborov. Ti so za naš primer grafično prikazani v mreži naborov (slika **??**), kjer so nabori določenega nivoja povezani z nabori prejšnjega nivoja, če so slednji njihovi podnabori. Tako je vozlišče MKZ povezano z vozlišči MK, MZ in KZ, a ne z vozliščem MS, saj MS ni podnabor MKZ.

Recimo, da iščemo pogoste nabore s podporo vsaj 0.6 (minsupp = 0.6). Najprej za vsak nabor v mreži naborov določimo število transakcij, v katerih je ta nabor prisoten. Za naš primer tako označeno mrežo prikazuje slika **??**. Pogosti nabori morajo biti vsebovani vsaj v



Slika 10.1: Mreža vseh možnih naborov za množico stvari mleko (M), kruh (K), zelenjava (Z) in sir (S). V korenu grafa je prazen nabor.

šestih transakcijah. Iz mreže je razvidno, da to velja za šest naborov, med katerimi je eden prazen.

Iz mreže naborov lahko razberemo nekaj zanimivih zakonitosti. Recimo, nabor M je nepogost. Zaradi tega so nepogosti prav vsi nabori z mlekom, torej vsi nabori, katerih podnabor je M. To je razumljivo, saj vsi ti nabori zahtevajo, da je njhov podnabor tudi M, ta pa je vsebovan v pramajhnem številu transakcij, da bi bili nabori, ki ga vsebujejo, pogosti.

Prav tako razberemo, da podpora nabora ne more presegati podpore kateregakoli njegovega podnabora. Primer: KZS je vsebovan v štirih transakcijah, in to število ne sme presegati števila transakcij, v katerih so vsebovani vsi njegovi možni podnabori, torej tudi ti iz prejšnjega nivoja (KZ, KS, ZS).

Zgornja zapažanja zapišimo v obliki teoremov.

Teorem 1 *Če je nabor pogost, so pogoste tudi vse njegove podmnožice:*

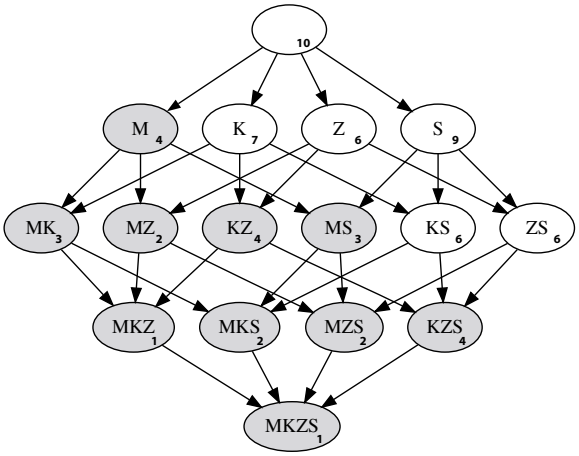
$$s(X) \geq \text{minsupp} \implies s(Y) \geq \text{minsupp}, Y \subseteq X$$

Teorem 2 *Če je nabor nepogost, so nepogosti tudi vsi nabori, ki ga vsebujejo:*

$$s(X) < \text{minsupp} \implies s(Y) < \text{minsupp}, X \subseteq Y$$

Teorem 3 *Podpora nabora nikoli ne presega podpore njegove podmnožnice (antimonotonost, $\mathcal{L} = 2^I$ je močnostna množica, torej množica vseh možnih naborov):*

$$\forall X, Y \in \mathcal{L} : X \subseteq Y \implies s(Y) \leq s(X)$$



Slika 10.2: Mreža pogostih naborov (naborov v praznih vozliščih) za minsupp = 0.6 oziroma za vse nabore, katerih število podpornih transakcij je vsaj 6 ($6 = 0.6 \times 10$). Vozlišča, ki predstavljajo nepogoste nabore, so osenčena.

Iz zgornjega intuitivno sledi algoritem Apriori. Ker nas prazna množica ne zanima, pričnemo lahko s pogostimi nabori prvega nivoja, torej nabori, ki vsebujejo natanko eno stvar (te imenujemo tudi *pogosti 1-nabori*). Od tu dalje s kombinacijo pogostih naborov generiramo *pogoste k-nabore*, kjer k iterativno povečujemo. Ustavimo se, ko na k-tem nivoju ni več pogostih naborov, saj bi to pomenilo, da teh tudi ni v vseh višjih nivojih.

Vhod: množica transakcij in minimalna podpora minsupp
Izhod: množica pogostih naborov

$k = 1$
$F_k = \{i | i \in I \wedge \sigma(i) \geq N \times \text{minsupp}\}$
ponavljaj
  $k \leftarrow k + 1$
  $C_k = \text{kandidati}(F_{k-1})$
  izračunaj podporo naborov v C_k
  $F_k = \{c | c \in C_k \wedge \sigma(c) \geq N \times \text{minsupp}\}$
dokler $F_k = \emptyset$
vrni $\cup_k F_k$

Algoritem Apriori uporablja funkcijo kandidati, ki na podlagi že odkritih pogostih naborov tvori pogoste k-nabore. S podrobnostmi implementacije te funkcije se ne bomo ukvarjali,

omenimo pa samo, da so za hitrost celotnega algoritma precej pomembne. Namreč, izogniti se moramo tvorjenju enakih kandidatov. Recimo, nabor KS lahko tvorimo iz pogostega nabora K in stvari S, ali iz pogostega nabora S in stvari K. Tu omenimo samo nekaj možnosti, ki jih lahko pri tvorjenju kandidatov lahko uporabimo:

- Kandidate za F_k tvorimo iz kombinacije pogostih naborov $F_{k-1} \times F_1$. Problem podvojenih kandidatov rešimo z leksikografsko ureditvijo naborov (na primer MKZ je z MZS, saj je K leksikografsko pred Z).

- Kandidate za F_k tvorimo iz kombinacije pogostih naborov $F_{k-1} \times F_1$, kjer nabora združimo le, če sta različna samo v zadnjem elementu. Tudi tu problem podvojenih kandidatov rešujemo z leksikografsko ureditvijo naborov.

10.4 Povezovalna pravila

Pravila tipa

$$X \rightarrow Y$$

kjer sta X in Y nabora stvari, imenujemo *povezovalna pravilna*. Zahtevamo, da je presečna množica obeh naborov prazna, torej $X \cap Y = \emptyset$. Primer takega pravila je recimo

$$\text{kruh} \rightarrow \text{mleko, sir}$$

pravilo pa pravi, da če bo v košarici kruh, pričakujemo, da bosta tam tudi mleko in sir. Še en primer povezovalnega pravila je

$$\text{mleko, sir} \rightarrow \text{zelenjava}$$

ki pravi, da če bo v košarici mleko, pričakujemo, da bosta tam tudi sir in zelenjava.

Pravila so lahko seveda bolj ali manj v soglasju z učno množico transakcij. Povezovalna pravila ocenjujemo z merami, med katerima sta najbolj znani podpora in zaupanje. *Podporo* že poznamo, poroča pa o delež transakcij, kjer najdemo vse stvari iz povezovalnega pravila:

$$s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N} \tag{10.4}$$

Podpora torej meri pogostost pravila v množici transakcij.

Drugačna mera je *zaupanje*, ki meri, kakšen je delež transakcij, ki vsebujejo desno stran pravila Y med transakcijami, ki vsebujejo levo stran pravila X:

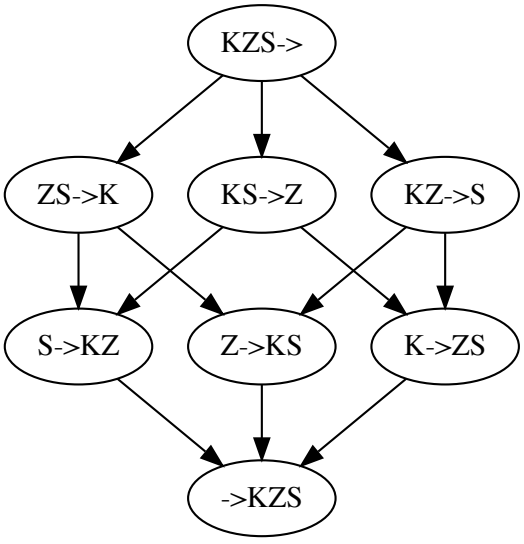
$$c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \tag{10.5}$$

Za pravilo M → SK bo tako podpora enaka 0.2, zaupanje pa enako 0.5 (štiri transakcije vsebujejo mleko, od tega dve poleg mleka še sir in kruh). Podpora pravila Z → S bo 0.6, zaupanje pa 1.0.

Algoritmu za iskanje podpornih pravil predpišemo minsupp in minconf, torej minimalno podporo in minimalno zaupanje. V praksi pa pravila iščemo tako, da najprej poiščemo pogoste nabore, potem pa iz teh izluščimo pravila, katerih zaupanje je dovolj visoko. Pogosti nabori bodo namreč imeli enako podporo kot pravila, ki vsebujejo vse stvari iz pogostega nabora. Iz nabora KZS lahko tako tvorimo $2^3 - 2 = 6$ netrivialnih pravil, kjer je leva oziroma desna stran pravila neprazna:

$$KZ \rightarrow S,\ KS \rightarrow Z,\ ZS \rightarrow K,$$
$$S \rightarrow KZ,\ Z \rightarrow KS,\ K \rightarrow ZS$$

Za dani nabor stvari lahko tvorimo vsa možna povezovalna pravila na podoben način, kot smo tvorili vse možne podnabore. Z eno samo razliko. Tokrat podnabore v mreži uporabimo za desno stran povezovalnega pravila, na levo stran pa damo vse, kar je v naboru, ki ga cepimo v pravilo, še ostalo. Tako nam za nabor KZS mreža na sliki **??** podaja vse možne cepitve oziroma vsa možna pravila.



Slika 10.3: Mreža vseh možnih povezovalnih pravil, ki jih lahko tvorimo iz nabora KZS. Pravili v korenu (prazen nabor na levi strani) in na zadnjem nivoju mreže nista legalni, sta pa vseeno prikazani.

Ostane nam le, da tudi tu poiščemo finto oziroma pristop, kjer je mrežo povezovalnih

pravil graditi iz začetnega vozlišča tako, da kandidate klestimo glede na zahtevano zaupanje. V ta namen si oglejmo dve pravili, ki ju tvorimo iz nabora Z: $X \to Z - X$ in $X' \to Z - X'$. Vsakič smo torej desno stran pravila dobili tako, da smo od nabora Z odstranili stvari na levi strani pravila, ki so enkrat bile X in drugič X'.

Teorem 4 *Naj bosta* X *in* X'*, kjer je* X' *podnabor* X*. Velja:*

$$c(X \to Z - X) < \text{minconf} \implies c(X' \to Z - X') < \text{minconf}$$

Ta teorem moramo dokazati. Spomnimo se, da velja:

$$X' \subset X \implies \sigma(X') \geq \sigma(X)$$

Neenačbo na desni strani pomnožimo z $\sigma(Z)$ in delimo z $\sigma(X')$ ter $\sigma(X)$. Dobimo enačbi za zaupanje obeh pravil, in dokaz zgornjega teorema:

$$\frac{\sigma(Z)}{\sigma(X')} < \frac{\sigma(Z)}{\sigma(X)}$$

Z drugimi besedami: če $X \to Z - X$ ne presega minimalnega zaupanja, tudi $X' \to Z - X'$ ne bo, kjer je X' podnabor X. V naši mreži povezovalnih pravil s slike **??** vidimo, da so pravila s podnabori levih strani postavljena v spodnjih nivojih pravil nivoja nad njimi. Če pravilo ZS→K ne bo ustrezalo pogoju zaupanja, tudi pravili S→KZ in Z→KS, ki iz njega sledita, ne bodo.

Ravno smo torej "izumili" pristop gradnje povezovalnih pravil. Gradimo jih iz pogostega nabora. Uporabimo mrežo pravil, kjer pričnemo s prazno desno stranjo pravila in tej, na vsakem nivoju, dodamo po eno stvar ter na ta način tvorimo vse možne kombinacije. Pravila na nivoju k gradimo s kombinacijo pravil nivoja $k - 1$. Kot pri algoritmu Apriori tudi tu na vsakem nivoju hranimo samo pravila, ki ustrezajo kriteriju zaupanja.

Za velike množice transakcij gradnja povezovalnih pravil kaj lahko privede do nepreglednih množic pravil. To se seveda zgodi pri premalo ostrih pogojih, torej pri premalo visokih minsupp in minconf. Taka pravila je ročno nemogoče pregledati. Pomagamo si tako, da skušamo najprej zmanjšati množico pogostih naborov z dvigovanjem minsupp, potem pa iz nje tvorimo pravila tako, da najprej skušamo postavljati bolj ostre pogoje za zaupanje.