

Igor Kononenko, Marko Robnik Šikonja

# Inteligentni sistemi

Ljubljana, 2010

# Kazalo

Predgovor . . . . .	i
<b>1 Uvod v strojno učenje</b>	<b>1</b>
1.1 Pregled metod strojnega učenja . . . . .	2
Klasifikacija (uvrščanje) . . . . .	2
Regresija . . . . .	5
Logične relacije . . . . .	7
Sistemi (diferencialnih) enačb . . . . .	9
Nenadzorovano učenje - razvrščanje . . . . .	9
Spodbujevano učenje . . . . .	10
1.2 Zgodovina strojnega učenja . . . . .	11
Simbolično učenje pravil . . . . .	11
Nevronske mreže . . . . .	12
Spodbujevano učenje . . . . .	13
Statistične metode . . . . .	14
Formalna teorija naučljivosti . . . . .	15
1.3 Nekateri zgodnji uspehi strojnega učenja . . . . .	15
AM (Automatic Mathematician) . . . . .	15
Eurisko . . . . .	16
Meta-Dendral . . . . .	16
MIS (Model Inference System) . . . . .	17
1.4 Aplikacije strojnega učenja . . . . .	18
Diagnostika proizvodnega procesa . . . . .	18
Medicinska diagnostika in prognostika . . . . .	18
Ocenjevanje zavarovancev in posojilojemalcev . . . . .	19
Klasifikacija slik . . . . .	20
Napovedovanje strukture kemičnih spojin in proteinov . . . . .	20
Igranje iger . . . . .	22
<b>2 Učenje in inteligенca</b>	<b>27</b>
2.1 Kaj je učenje . . . . .	27
Definicija učenja . . . . .	27
Umetna inteligencia . . . . .	28

2.2	Naravno učenje . . . . .	29
	Prirojeno in naučeno. . . . .	30
	Spomin . . . . .	31
	Vrste naravnega učenja . . . . .	33
2.3	Učenje, inteligenca, zavest . . . . .	36
	Količina inteligence . . . . .	36
	Zavest . . . . .	37
	Meje simbolične izračunljivosti, izpeljivosti, opisljivosti . . . . .	37
	Možnost umetne inteligence . . . . .	38
	(Ne)zmožnost umetne zavesti?. . . . .	39
	Znanost in filozofija . . . . .	41
2.4	Zakaj strojno učenje . . . . .	41
<b>3</b>	<b>Osnove strojnega učenja</b> . . . . .	<b>45</b>
3.1	Osnovni principi strojnega učenja . . . . .	46
	Učenje kot modeliranje . . . . .	46
	Princip najkrajšega opisa . . . . .	48
	Inkrementalno (sprotno) učenje . . . . .	50
	Princip večkratne razlage . . . . .	51
	Ocenjevanje verjetnosti . . . . .	53
3.2	Mere za ocenjevanje učenja . . . . .	55
	Klasifikacijska točnost. . . . .	56
	Tabela napačnih klasifikacij . . . . .	58
	Cena napačne klasifikacije. . . . .	58
	Ocenjevanje napovedi verjetnosti razredov. . . . .	59
	Informacijska vsebina odgovora. . . . .	59
	Rob (margin). . . . .	61
	Senzitivnost in specifičnost . . . . .	61
	Krivulja ROC. . . . .	62
	Priklic in preciznost . . . . .	64
	Srednja kvadratna napaka . . . . .	64
	Srednja absolutna napaka . . . . .	65
	Korelacijski koeficient. . . . .	65
	Pristranskost in varianca v strojnem učenju . . . . .	66
3.3	Ocenjevanje učenja . . . . .	68
	Zanesljivost ocene uspešnosti . . . . .	68
	Interval zaupanja. . . . .	69
	Prečno preverjanje. . . . .	69
	Metoda razmnoževanja učnih primerov . . . . .	70
3.4	Primerjanje uspešnosti različnih učnih algoritmov . . . . .	71
	Dva algoritma na eni domeni. . . . .	72
	Bonferronijeva korekcija. . . . .	74
	Dva algoritma na več domenah. . . . .	75
	Več algoritmov na več domenah . . . . .	75

3.5	Kombiniranje algoritmov strojnega učenja . . . . .	77
	Kombiniranje.napovedi različnih.hipotez . . . . .	77
	Povezovanje algoritmov . . . . .	78
	Bagging . . . . .	79
	Boosting . . . . .	79
	Naključni.gozdovi . . . . .	80
	Transdukcija v strojnem učenju . . . . .	80
	Cenovno občutljivo.učenje. . . . .	82
	Oponašanje.funkcije . . . . .	83
	Uporaba klasifikatorjev.v.regresiji in.obratno . . . . .	83
	Kode za popravljanje.napak klasifikacije. . . . .	84
<b>4</b>	<b>Predstavitev znanja in operatorji</b>	<b>89</b>
4.1	Izjavni račun . . . . .	90
	Atributna.predstavitev učnih.primerov. . . . .	90
	Lastnosti atributov.in njihove soodvisnosti . . . . .	90
	Klasifikacijska, regresijska in povezovalna pravila . . . . .	91
	Splošnost pravil . . . . .	92
	Operatorji . . . . .	93
	Prostor hipotez. . . . .	94
	Odločitvena in regresijska drevesa. . . . .	94
4.2	Diskriminantne in regresijske funkcije . . . . .	95
	Linearne funkcije. . . . .	95
	Kvadratne funkcije.in funkcije $\Phi$ . . . . .	96
	Umetne nevronske mreže. . . . .	96
4.3	Verjetnostne porazdelitve . . . . .	99
	Bayesov klasifikator. . . . .	99
	Učni primeri kot verjetnostna porazdelitev. . . . .	100
	Naivni Bayesov klasifikator . . . . .	100
4.4	Učenje kot preiskovanje . . . . .	102
	Izčrpno preiskovanje. . . . .	102
	Omejeno izčrpno.iskanje.(razvaji in omeji) . . . . .	103
	Metoda “najprej najboljši”. . . . .	104
	Požrešno iskanje. . . . .	104
	Iskanje v snopu . . . . .	105
	Lokalna optimizacija . . . . .	105
	Gradientno.iskanje. . . . .	106
	Simulirano.oblajanje. . . . .	106
	Genetski algoritmi. . . . .	107
<b>5</b>	<b>Mere za ocenjevanje atributov</b>	<b>109</b>
5.1	Mere za ocenjevanje atributov v klasifikaciji . . . . .	109
	Mere nečistoče. . . . .	110
	Mere, ki temeljijo nakoličini informacije . . . . .	111

Gini-indeks. . . . .	116
ReliefF . . . . .	117
5.2 Mere za ocenjevanje atributov v regresiji . . . . .	121
Razlika variance pri regresijskih problemih . . . . .	121
Regresijski ReliefF . . . . .	121
MDL v regresiji . . . . .	122
5.3 Izpeljave in dokazi . . . . .	123
<b>6 Predobdelava učnih primerov</b>	<b>131</b>
6.1 Atributna predstavitev kompleksnih struktur . . . . .	131
Atributna predstavitev besedila . . . . .	131
Atributna predstavitev slik . . . . .	132
Atributna predstavitev grafov . . . . .	133
6.2 Diskretizacija zveznih atributov . . . . .	134
Vrste metod za diskretizacijo . . . . .	134
Možne meje med intervali . . . . .	134
Mere za usmerjanje diskretizacije . . . . .	135
6.3 Mehka diskretizacija zveznih atributov . . . . .	135
6.4 Binarizacija atributov . . . . .	136
Dvorazredni problemi . . . . .	137
Večrazredni problemi . . . . .	138
6.5 Spreminjanje diskretnih atributov v zvezne . . . . .	138
6.6 Obravnavanje neznanih vrednosti . . . . .	139
6.7 Vizualizacija . . . . .	139
Vizualizacija enega atributa . . . . .	141
Vizualizacija parov atributov . . . . .	142
Vizualizacija več atributov . . . . .	143
Vizualizacija rezultatov strojnega učenja . . . . .	145
6.8 Izbira podmnožice atributov . . . . .	148
6.9 Izpeljave in dokazi . . . . .	149
<b>7 Simbolično učenje</b>	<b>153</b>
7.1 Učenje odločitvenih dreves . . . . .	153
Gradnja in uporaba odločitvenega drevesa . . . . .	153
Gradnja binarnega odločitvenega drevesa . . . . .	155
Rezanje odločitvenega drevesa . . . . .	155
Obravnavanje neznanih in poljubnih vrednosti . . . . .	156
7.2 Učenje odločitvenih pravil . . . . .	157
Generiranje enega pravila . . . . .	157
Učenje množice pravil . . . . .	157
7.3 Učenje regresijskih dreves . . . . .	158
Gradnja in uporaba regresijskega drevesa . . . . .	158
Rezanje regresijskega drevesa . . . . .	160
7.4 Naivni in delno naivni Bayesov klasifikator . . . . .	161

Naivni Bayesov klasifikator . . . . .	161
Razlaga odločitev naivnega Bayesovega klasifikatorja . . . . .	162
Delno naivni Bayesov klasifikator . . . . .	162
Poskusi v medicinski diagnostiki . . . . .	165
<b>8 Numerične metode</b> . . . . .	<b>171</b>
8.1 Metode najbližjih sosedov . . . . .	171
K-najbližjih sosedov . . . . .	171
Z razdaljo uteženih $k$ -najbližjih sosedov . . . . .	173
Lokalno utežena regresija . . . . .	173
8.2 Linearna regresija . . . . .	174
8.3 Metode podpornih vektorjev . . . . .	175
Osnovni princip metode SVM . . . . .	175
Razširitev na neekskaktno klasifikacijo . . . . .	177
Metoda SVM . . . . .	178
<b>9 Umetne nevronske mreže</b> . . . . .	<b>181</b>
9.1 Uvod . . . . .	181
Preprost primer nevronske mreže . . . . .	181
Porazdeljeni pamnilnik . . . . .	184
Lastnosti nevronskih mrež . . . . .	185
Primeri uporabe . . . . .	187
Analogija z možgani . . . . .	188
Relacija z metodami umetne inteligence . . . . .	190
9.2 Vrste nevronskih mrež . . . . .	191
Topologija nevronske mreže . . . . .	191
Namen nevronske mreže . . . . .	192
Pravilo učenja . . . . .	194
Funkcija kombiniranja vhodov nevrona v izhod . . . . .	196
9.3 Perceptron . . . . .	197
Dvonivojski perceptron . . . . .	197
Večnivojski perceptron . . . . .	200
Slabosti pospološenega pravila delta . . . . .	202
9.4 Izpeljave in dokazi . . . . .	203
<b>10 Verjetnostno modeliranje</b> . . . . .	<b>209</b>
10.1 Bayesove mreže . . . . .	209
Skupna verjetnostna porazdelitev . . . . .	211
Skupinsko filtriranje - bayesovski pristop . . . . .	213
10.2 Sklepanje v času . . . . .	214
Skriti markovski model . . . . .	217
Dinamična Bayesova mreža . . . . .	219
10.3 Učenje v grafičnih verjetnostnih modelih . . . . .	220

<b>11 Obdelava naravnega jezika</b>	<b>223</b>
11.1 Mikrosvetovi . . . . .	224
11.2 Dva pristopa . . . . .	225
Lingvistična analiza jezika. . . . .	226
Sintaktična analiza . . . . .	227
Interpretacija pomena besedila in uporaba znanja o svetu . . . . .	230
11.3 Empirični pristop . . . . .	230
Korpsi . . . . .	232
Predobdelava besedila . . . . .	232
Iskanje dokumentov in pridobivanje informacij . . . . .	233
Klasifikacija dokumentov in tekstovno.rudarjenje . . . . .	239
Povzemanje teksta . . . . .	240
11.4 Trendi v razumevanju naravnega jezika . . . . .	240
<b>12 Evolucijsko računanje</b>	<b>243</b>
12.1 Genetski algoritmi . . . . .	246
Predstavitev osebkov. . . . .	246
Križanje . . . . .	248
Mutacija . . . . .	251
Evolucijski model . . . . .	251
Nadomeščanje osebkov . . . . .	253
Ustavitevni pogoj. . . . .	255
Pomembne podrobnosti . . . . .	255
Zakaj genetski algoritmi delujejo? . . . . .	256
Večkriterijska optimizacija. . . . .	256
Tipične aplikacije . . . . .	257
Orodjarne in knjižnice . . . . .	257
12.2 Genetsko programiranje . . . . .	258
12.3 Inteligenca rojev . . . . .	258
Roj delcev . . . . .	258
Kolonija mravelj . . . . .	260
<b>13 Hevristično preiskovanje</b>	<b>267</b>
13.1 Neinformirano preiskovanje . . . . .	269
V globino. . . . .	269
V širino . . . . .	270
Iterativno poglavljanje . . . . .	271
13.2 Informirano preiskovanje . . . . .	272
Požrešno preiskovanje . . . . .	272
Najprej najboljši . . . . .	273
Algoritem A . . . . .	275
Algoritem A* . . . . .	277
Hevristike . . . . .	280
Razveji in.omeji . . . . .	281

Izboljšave algoritma A* . . . . .	282
13.3 Preiskovanje po principu minimaks . . . . .	284
Rezanje alfa-beta . . . . .	289
Uporaba preiskovanja v igrah . . . . .	290
<b>14 Inteligentni agenti in roboti</b>	<b>293</b>
14.1 Vrste agentov . . . . .	295
14.2 Agentne arhitekture . . . . .	297
14.3 Robotika . . . . .	299
14.4 Porazdeljeno računanje z agenti . . . . .	303
Porazdeljeno upoštevanje omejitev . . . . .	303
14.5 Porazdeljena optimizacija . . . . .	312
Asinhrono dinamično programiranje . . . . .	312
Hevrističen algoritem <i>LRTA*</i> . . . . .	312
<b>15 Spodbujevanje učenje</b>	<b>317</b>
15.1 Komponente spodbujevanega učenja . . . . .	319
15.2 Formalizacija . . . . .	322
15.3 Optimizacijski kriteriji . . . . .	323
15.4 Kriteriji za učenje . . . . .	325
15.5 Markovsko spodbujevanje učenje . . . . .	326
Iskanje strategije pri danem modelu . . . . .	327
Iskanje optimalne strategije . . . . .	330
<b>16 Načrtovanje</b>	<b>335</b>
16.1 Klasično načrtovanje . . . . .	335
Eksplicitna predstavitev . . . . .	336
Predstavitev akcij z značilnostmi . . . . .	337
Predstavitev STRIPS . . . . .	337
Začetna stanja in cilji . . . . .	338
Iskanje načrta . . . . .	338
16.2 Načrtovanje z delnimi urejenostmi . . . . .	339
<b>A Ponovitev verjetnosti</b>	<b>343</b>
A.1 Računanje z dogodki . . . . .	343
A.2 Verjetnost . . . . .	344
A.3 Pogojna verjetnost . . . . .	346
A.4 Slučajne spremenljivke in porazdelitve . . . . .	346
<b>B Slovar nekaterih angleških strokovnih izrazov</b>	<b>349</b>
<b>Stvarno kazalo</b>	<b>359</b>

## Poglavlje 1

# Uvod v strojno učenje

*Prišel bo čas, ko bomo morali pozabiti vse, kar smo se naučili.*

*Ramana Maharshi*

Strojno učenje (*machine learning*) je veja raziskav umetne inteligence, ki je v zadnjih dvajsetih letih močno napredovala, kar se odraža v številnih komercialnih sistemih za strojno učenje in njihovi uporabi v industriji, medicini, ekonomiji, naravoslovnih in tehničnih raziskavah, ekologiji, bančništву, itd. Strojno učenje se uporablja za analizo podatkov in odkrivanje zakonitosti v podatkovnih bazah (podatkovno rudarjenje - *data mining*), za avtomatsko generiranje baz znanja za eksperimentne sisteme, za učenje načrtovanja, igranje iger, za gradnjo numeričnih in kvalitativnih modelov, za razpoznavanje naravnega jezika in prevajanje, klasiifikacijo tekstov in apletno rudarjenje, za avtomatsko ekstrakcijo znanja dinamične kontrole procesov, razpoznavanje govora, pisave, slik itd.

Osnovni princip strojnega učenja je avtomatsko opisovanje (modeliranje) pojavov iz podatkov. Rezultat učenja so pravila, funkcije, relacije, sistemi enačb, verjetnostne porazdelitve ipd., ki so predstavljene z različnimi formalizmi: odločitvenimi pravili, odločitvenimi drevesi, regresijskimi drevesi, Bayesovimi mrežami, nevronskimi mrežami itd. Naučeni modeli poskušajo razlagati podatke, iz katerih so bili modeli zgenerirani, in se lahko uporabijo za odločanje pri opazovanju modeliranega procesa v bodočnosti (napovedovanje, diagnosticiranje, nadzor, preverjanje, simulacije itd.).

Uvodni razdelek v tem poglavju je namenjen kratkemu pregledu metod strojnega učenja. Sledi zgodovinski pregled razvoja različnih smeri strojnega učenja: simboličnega učenja pravil, nevronskih mrež, spodbujevanega učenja, numeričnih metod učenja in formalne teorije naučljivosti. Zatem so podani opisi nekaterih zanimivih zgodnjih sistemov strojnega učenja. Poglavlje se zaključi s pregledom nekaterih aplikacij strojnega učenja.

## 1.1 Pregled metod strojnega učenja

*V splošnem je učeči se stroj vsaka naprava, pri kateri izkušnje iz preteklosti vplivajo na akcije.*

*Nils J. Nilsson*

Metode strojnega učenja delimo glede na način uporabe naučenega znanja: klasifikacija (uvrščanje), regresija, učenje asociacij in logičnih relacij, učenje sistemov (diferencialnih) enačb in razvrščanje (*clustering*). Poleg tega se strojno učenje uporablja pri iterativnem izboljševanju reševanja ciljnega problema z metodami spodbujevanega učenja. Zadnjih dveh področij strojnega učenja (učenja sistemov enačb in razvrščanja) v tem učbeniku sicer ne bomo podrobneje obravnavali, a sta tu vseeno na kratko opisana.

### Klasifikacija (uvrščanje)

Ena najpogostejejših uporab strojnega učenja je klasifikacija ali uvrščanje (*classification*). Naloga *klasifikatorja* je za objekt (problem), opisan z množico atributov (značilk, lastnosti) določiti, kateremu izmed možnih razredov pripada. Atributi so neodvisne zvezne ali diskretne spremenljivke, s katerimi opisujemo objekte, razred pa je odvisna diskretna spremenljivka, ki ji določimo vrednost (izberemo razred) glede na vrednosti neodvisnih spremenljivk. Primer klasifikacijskega problema je postavljanje medicinske diagnoze. Pacient je opisan z zveznimi atributti, kot so starost, višina, teža, telesna temperatura, srčni utrip, krvni pritisk itd, ter z diskretnimi atributti, kot so spol, barva kože, lokacija bolečine, itd. Naloga klasifikatorja je postaviti diagnozo, to je določiti enega izmed možnih razredov. Npr. pacient je lahko zdrav, prehlajen, ima gripo, angino, bronhitis ali pljučnico.

Zato, da lahko klasifikator določi razred, mora imeti na nek način predstavljeno diskretno funkcijo, ki preslika prostor atributov v razred. Ta funkcija je lahko podana vnaprej ali pa je naučena iz podatkov – primerov rešenih problemov v preteklosti. Pri medicinski diagnostiki so to opisi pacientov skupaj z njihovimi diagnozami za vse paciente, ki so se zdravili zadnjih nekaj let. Naloga učnega algoritma je iz množice opisov pacientov z zanimimi diagnozami zgraditi pravilo (izpeljati preslikavo, izračunati funkcijo), ki ga lahko uporabimo za diagnosticiranje novih pacientov.

Klasifikatorje ločimo glede na način predstavitve klasifikatorjeve funkcije. Najbolj počasti klasifikatorji so odločitvena drevesa, odločitvena pravila, naivni Bayesov klasifikator, Bayesove verjetnostne mreže, klasifikator z najbližjimi sosedji, linearna diskriminantna funkcija, logistična regresija, klasifikator po metodi podpornih vektorjev ter usmerjene (večnivojske) umetne nevronske mreže.

**Odločitvena drevesa in pravila** Algoritmi za gradnjo odločitvenih dreves in pravil glede na oceno informativnosti posameznih atributov izbirajo atribute in ustrezne podmnožice njihovih vrednosti za gradnjo odločitvenega drevesa oziroma pravila. Tako dobljene pogoje najpogosteje konjunktivno dodajajo k pogojnemu delu pravila. Sklepni (odločitveni) del pravila vsebuje enega ali več razredov, ki jim pripadajo ustrezni učni primeri. Klasifikacija novega primera poteka tako, da se sproži ustrezno pravilo.

Pri odločitvenih drevesih in pravilih za dani diskretni atribut  $A$  in njegove vrednosti  $V, V_1, \dots, V_m$  imamo konjunktivne pogoje oblike  $A = V$  ali  $A \in \{V_1, \dots, V_m\}$ . Zvezne atrubute je potrebno bodisi vnaprej diskretizirati, ali pa, kar je bolj splošno, so konjunktivni pogoji v pravilih za dani zvezni atribut  $A$  in vrednosti  $V, V_1$  in  $V_2$  oblike  $A > V$  ali  $A \leq V$  oziroma  $V_1 \geq A > V_2$ .

**Bayesov klasifikator** Naloga Bayesovega klasifikatorja je izračunati pogojne verjetnosti za vsak razred pri danih vrednostih (vseh) atrubutov za dani novi primer (problem), ki ga želimo klasificirati. Bayesov klasifikator, ki natančno izračuna pogojne verjetnosti razredov, je optimalen, saj minimizira pričakovano napako. Ker Bayesovega klasifikatorja, ki bi popolnoma pravilno izračunal pogojne verjetnosti razredov, ne poznamo (razen v primerih, ko učna množica pokriva celoten prostor vrednosti vseh atrubutov), je potrebno izračunati približke verjetnosti z vpeljavo določenih predpostavk.

Naivni Bayesov klasifikator predpostavi pogojno neodvisnost atrubutov pri danem razredu. Učno množico nato uporabi oceno vseh potrebnih verjetnosti za izračun pogojne verjetnosti vsakega razreda. Implementacije naivnega Bayesovega klasifikatorja ponavadi predpostavlja samo diskretne atrubute, zato je potrebno zvezne atrubute vnaprej diskretizirati.

Posplošitev naivnega Bayesovega klasifikatorja so Bayesove (verjetnostne) mreže, kjer z usmerjenim acikličnim grafom modeliramo odvisnosti slučajnimi spremenljivkami, ki so v našem primeru atrubuti in razred. S tem implicitno predpostavimo pogojne neodvisnosti med tistimi atrubuti (vozlišči v grafu), ki med seboj niso neposredno povezani. Struktura Bayesove mreže je lahko dana vnaprej (kot predznanje), ali pa jo učni algoritem zgradi na osnovi ocen (ne)odvisnosti atrubutov na učni množici primerov. Pogojne verjetnosti se izračunajo na osnovi učne množice primerov. Acikličnost grafa pripomore k učinkovitosti izračuna pogojnih verjetnosti vseh razredov. Bayesove mreže se uporabljajo tudi za modeliranje verjetnostne distribucije na atrubutno opisani problemski domeni brez razreda (učenju na podatkih brez odvisne spremenljivke - razreda pravimo tudi nenadzorovano učenje).

**Klasifikator z najbližjimi sosedji** Najpreprostejša varianta algoritma najbližjih sosedov ( $k$ -NN) kot znanje uporablja kar množico vseh učnih primerov (učni algoritem si samo zapomni vse primere). Pri klasifikaciji novega primera iz učne množice poišče nekaj najbolj podobnih (najbližjih) primerov. Nov primer klasificiramo v razred, ki mu pripada največ bližnjih primerov. Pri tem je potrebno zaradi ustrezne metrike v prostoru atrubutov normalizirati vrednosti zveznih atrubutov in definirati razdaljo (metriko) med vrednostmi vsakega diskretnega atrubuta.

**Diskriminantne funkcije** Naloga učnega algoritma je izračunati vrednosti koeficientov vnaprej podane diskriminantne funkcije, ki predstavlja hiperploskev med dvema razredoma v prostoru atrubutov. Hiperploskev lahko definiramo samo v zveznem prostoru, torej morajo biti vsi atrubuti zvezni. Če imamo več kot dva razreda, potrebujemo za vsak par razredov eno hiperploskev. Diskriminantne funkcije so lahko linearne, kvadratne, polinomske itd. V primeru linearne diskriminantne funkcije poskušamo pri-

mere iz dveh razredov med seboj ločiti s hiperravnino. Koeficiente določimo tako, da je klasifikacijska napaka čim manjša.

Običajna Fisherjeva linearna diskriminantna funkcija predpostavlja normalno porazdelitev primerov znotraj vsakega razreda in maksimizira razdaljo med povprečnim učnim primerom iz prvega razreda in povprečnim učnim primerom iz drugega razreda. Pri tem upošteva različni varianci razredov in poišče optimalno klasifikacijsko mejo med razredoma.

Pri metodi podpornih vektorjev (*Support Vector Machines - SVM*), ki je bila razvita v devetdesetih letih in je ena najbolj uspešnih metod za klasifikacijo, je potrebno optimizirati več kriterijev:

- maksimizirati je treba razdaljo podpornih vektorjev od ločitvene hiperravnine. Podporni vektorji so tisti učni primeri, ki so najbližje hiperravnini in na ta način določajo rob ob hiperravnini. Torej po tej metodi želimo maksimizirati širino roba.
- ker mnogi klasifikacijski problemi niso eksaktne rešljive z linearno funkcijo, metoda SVM uporablja implicitno transformacijo atributov v (potencialno veliko, lahko tudi neskončno) množico novih atributov s t.i. jedrnimi funkcijami (*kernel functions*). Od izbire ustreznega jedra je odvisna uspešnost metode SVM.
- ker lahko vsak klasifikacijski problem rešimo eksaktno z dovolj zapleteno funkcijo, je treba upoštevati tudi kompleksnost rešitve, kar pomeni minimizacijo vsote uteži vseh atributov.
- kljub nelinearni transformaciji atributov se moramo zadovoljiti z napačno klasifikacijo nekaterih primerov, seveda pa želimo število napačno klasificiranih primerov minimizirati.

Potrebno je torej poiskati ustrezni kompromis med vsemi kriteriji. Metoda SVM uporablja različne optimizacijske algoritme, ki optimizirajo vse kriterije hkrati. Še vedno pa obstajajo parametri, ki utežujejo posamezne kriterije, in je treba njihovo vrednost določiti empirično.

**Umetne nevronske mreže** Algoritmi umetnih nevronskih mrež skušajo oponašati biološke nevronske mreže tako, da nevron abstrahirajo na preprost element, ki zna seštevati utičence vhodnih signala in rezultat normalizirati s pragovno funkcijo. Take preproste umetne nevrone potem povezujemo v poljubno kompleksne umetne nevronske mreže. Za klasifikacijo se najpogosteje uporabljajo usmerjene večnivojske umetne nevronske mreže, ki kaskadno povezujejo več nivojev nevronov: vhodni nevroni (ki ustrezajo atributom), eden ali več nivojev skritih nevronov in izhodni nevroni (ki ustrezajo razredom). Naloga učnega algoritma je nastaviti uteži na povezavah med nevroni (iz katerih vsak nevron izračunava uteženo vsoto) tako, da bo klasifikacijska napaka čim manjša.

Med klasifikacijo novega primera nevronska mreža na vhodu dobi vrednosti atributov (vrednosti vhodnih nevronov) za dani novi primer. Vsak nevron na naslednjem nivoju izračuna svoj izhod kot uteženo vsoto svojih vhodnih signalov. Izračun na zadnjem nivoju določi razred novemu primeru.

**Hibridni algoritmi** Hibridni algoritmi kombinirajo več različnih pristopov, tako da izkoristi prednosti posameznih pristopov in jih združujejo v potencialno boljše algoritme.

Zaradi lepih lastnosti naivnega Bayesovega klasifikatorja se ga pogosto kombinira z drugimi algoritmi, da bi se vsaj delno izognili njegovi naivnosti. Primeri takih algoritmov pri klasifikaciji so:

- *k*-NN naivni Bayes: ideja je za klasifikacijo novega primera uporabiti algoritmom naivnega Bayesa, ki je naučen iz *k* najbližjih sosedov novega primera. S tem se omili naivnost osnovnega (globalnega) naivnega Bayesovega klasifikatorja, saj je le-ta prilagojen lokalni distribuciji, ki implicitno upošteva medsebojno odvisnost atributov. Pri tem mora biti parameter *k* primerno večji kot pri običajnem algoritmu najbližjih sosedov (npr.  $k > 30$ ), saj premajhen *k* ne bi zadostoval za zanesljivo učenje naivnega Bayesa.
- Razvrščanje in naivni Bayesov klasifikator: primere iz vsakega razreda najprej z nekim algoritmom razvrščanja, ki je opisano pozneje, razvrstimo v več podrazredov. Na tako razdrobljenih podrazredih, ki jih je seveda več kot originalnih razredov, zatem učimo naivni Bayesov klasifikator. Smisel je v tem, da prepustimo algoritmu razvrščanja, da (z upoštevanjem odvisnosti med atributti) odkrije lokalno koherentne podprostore, ki se jih lahko naivni Bayes nauči razlikovati med seboj.
- Uporaba naivnega Bayesa v listih odločitvenega drevesa: upamo, da odločitveno drevo, ki upošteva odvisnosti atributov, razbije prostor na podprostore tako, da med preostalimi atributti, ki jih ni na poti od korena do lista, ni močnih odvisnosti in je zato klasifikacija z naivnim Bayesom bolj zanesljiva. Pri tem je treba gradnjo odločitvenega drevesa prej ustaviti (močneje porezati drevo), da je v listih dovolj primerov za učenje naivnega Bayesovega klasifikatorja.
- Uporaba naivnega Bayesa (ali kakega drugega preprostega klasifikatorja) v notranjih vozliščih odločitvenega drevesa: namesto, da uporabimo atributi za gradnjo drevesa, uporabimo preprost klasifikator, tako da ima vozlišče po enega naslednika za vsak razred. Klasifikator se uči na primerih iz vozlišča in zatem klasificira te primere v razrede in s tem jih uvršča po vejah, ki vodijo v naslednike. Naloga klasifikatorja v nasledniku je pravilno klasificirati primere, ki so bili prej napačno klasificirani, tako da ima to vozlišče zopet po enega naslednika za vsak razred. Gradnja drevesa se rekurzivno nadaljuje, dokler niso vsi primeri pravilno klasificirani, ozziroma je izpolnjen kak drug kriterij za ustavitev gradnje (npr. skoraj vsi pravilno klasificirani ali premalo primerov v vozlišču).

**Skupinske metode** Za doseganje večje točnosti se pri napovedovanju namesto enega modela pogosto uporablja množica modelov. Namesto enega modela zgradimo zaporedje modelov tako da spremenjamo parametre učne množice in/ali učnega algoritma. Skupinske metode (*ensemble methods*) sledijo principu večkratne razlage in so podrobnejše opisane v 3. poglavju.

## Regresija

Naloga regresijskega *prediktorja* je za objekt (problem, primer), ki je tako kot pri klasifikaciji opisan z množico atributov (značilk, lastnosti), določiti vrednost odvisne (regresijske) spremenljivke, ki je, za razliko od razreda pri klasifikaciji, zvezna (številska). Kot primer klasifikacijskega problema smo navedli postavljanje medicinske diagnoze. Če namesto diagnoze želimo pacientu določiti npr. indeks težavnosti obolenja, imamo opravka z regresijskim problemom, saj je odvisna spremenljivka zvezna.

Podobno kot pri klasifikaciji mora tudi regresijski prediktor imeti na nek način predstavljeno zvezno funkcijo, ki preslika prostor atributov v napovedano vrednost. Ta funkcija je lahko podana vnaprej ali pa je naučena iz podatkov - primerov že rešenih problemov. Naloga učnega algoritma je torej iz množice opisov primerov z znanimi vrednostmi odvisne spremenljivke izračunati zvezno funkcijo, ki jo lahko uporabimo za določanje vrednosti regresijske spremenljivke novih primerov.

Tudi regresijske prediktorje ločimo glede na način predstavitve regresijske funkcije. Najbolj pogosti regresorji so: regresijska drevesa, linearna regresija, lokalno utežena regresija, regresija po metodi podpornih vektorjev ter usmerjene (večnivojske) umetne nevronske mreže.

**Regresijska drevesa** Algoritmi za gradnjo regresijskih dreves, podobno kot pri odločitvenih drevesih, glede na oceno posameznih atributov v vozliščih izbirajo atribute in ustrezne podmnožice njihovih vrednosti za gradnjo regresijskega drevesa. Vozlišča v drevesu predstavljajo atribute, veje pa vrednosti atributov (konjunktivne pogoje). Listi predstavljajo sklepni del pravila, ki je poljubna zvezna funkcija, ki preslika vrednosti preostalih atributov v vrednost regresijske spremenljivke. Zvezni in diskretni atributi v notranjih vozliščih regresijskega drevesa se obravnavajo tako kot pri odločitvenih (klasifikacijskih) drevesih. V listih imamo opravka z zvezno funkcijo, ki predpostavlja, da so vsi preostali atributi zvezni. Najpogosteje se uporablja točkovna funkcija (povprečna vrednost odvisne spremenljivke pri ustreznih učnih primerih) ter linearna regresijska funkcija.

**Linearna regresija** Linearna regresijska funkcija predpostavlja, da so vsi atributi zvezni (in ustrezno normalizirani) ter da obstaja linearna relacija med odvisno regresijsko spremenljivko in atributi. Treba je le določiti koeficiente linearne funkcije tako, da minimiziramo vsoto kvadratov napak regresijske spremenljivke preko vseh učnih primerov. V ta namen lahko uporabimo bodisi analitično rešitev, bodisi iterativen algoritem za iskanje optimalne funkcije. Rezultat je hiperravnina, ki za vsak primer določa vrednost odvisne spremenljivke. Za razliko od diskriminantne hiperravnine pri klasifikaciji, ki je zgrajena v  $a$ -dimenzionalnem prostoru, kjer je  $a$  število atributov, je tukaj hiperravnina zgrajena v  $a + 1$  dimenzionalnem prostoru, kjer je dodatna dimenzija odvisna spremenljivka. Napaka za nek primer je njegova razdalja od hiperravnine v smeri osi, ki ustreza odvisni spremenljivki.

Pri nelinearnih problemih lahko bodisi uporabimo funkcijo, ki je sestavljena (zlepljena) iz več lokalno linearnih funkcij, bodisi v linearni kombinaciji uporabimo tudi nelinearne člene, ki opisujejo npr. kvadratne, polinomske, logaritemske itd. funkcije.

**Lokalno utežena regresija** Algoritem najbližjih sosedov prilagodimo za regresijo tako, da linearno regresijsko funkcijo izračunamo samo za vrednosti regresijske spremenljivke najbližjih učnih primerov. Tako je linearna regresija prilagojena lokalnim lastnostim prostora učnih primerov v okolici primera, katerega regresijsko vrednost isčemo. Sledi lahko namesto linearne regresije uporabimo kar povprečno vrednost ali pa kako drugo relativno preprosto funkcijo (zapletenih funkcij si ne moremo privoščiti zaradi majhnega števila najbližjih sosedov). Tako kot pri  $k$ -NN tudi tu lahko uporabimo uteževanje primerov z razdaljo.

**Metoda podpornih vektorjev v regresiji** Ker SVM uporablja implicitno transformacijo atributnega prostora s pomočjo jedrnih funkcij, ni potrebo uporabljati nelinearnih funkcij za predstavitev hiperploskve. Metodo SVM lahko prilagodimo za reševanje regresijskih problemov tako, da definiramo rob ob regresijski hiperravnini, znotraj katerega menimo, da je odvisna spremenljivka za vse primere eksaktно napovedana, zunaj njega pa so podporni vektorji, ki določajo potek hiperravnine. Rob ob hiperravnini želimo minimizirati, hkrati pa želimo minimizirati tudi napako na učnih primerih. Podobno kot pri klasifikacijskih SVM je tudi v regresiji treba optimizirati kriterijsko funkcijo, ki upošteva napake napovedi regresijske spremenljivke na učnih primerih ter kompleksnost funkcije (velikosti uteži implicitno generiranih atributov).

**Umetne nevronske mreže** Podobno kot za klasifikacijo se za regresijo najpogosteje uporabljajo usmerjene večnivojske umetne nevronske mreže, ki kaskadno povezujejo vhodne nevronne (ki ustrezajo atributom), en ali več nivojev skritih nevronov in en sam izhodni nevron, ki ustreza regresijski spremenljivki. Naloga učnega algoritma je nastaviti uteži na povezavah med nevroni (iz katerih vsak nevron izračunava uteženo vsoto) tako, da bo regresijska napaka čim manjša.

**Hibridni algoritmi** Tudi pri regresiji različne algoritme ali dele algoritmov kombiniramo ter izkorisčamo prednosti posameznih pristopov in jih združujemo v potencialno boljše algoritme. Primer takega algoritma je že omenjena uporaba lincarne regresije ali lokalno utežene regresije v listih regresijskega drevesa - upamo, da regresijsko drevo razbijuje prostor na podprostore, na katerih preprosti (linearni) regresorji dovolj dobro delujejo. Pri tem je treba, podobno kot pri odločitvenih drevesih, gradnjo regresijskega drevesa ustaviti prej (močneje porezati drevo), da je v listih dovolj primerov za učenje izbranega regresijskega algoritma.

## Asociacije in logične relacije

Pri učenju logičnih relacij je, za razliko od klasifikacije, potrebno namesto diskretnih funkcij izpeljati logično relacijo, na katero lahko gledamo kot na posplošitev diskretnih funkcij. Relacija je za razliko od funkcije večsmerna - ne obstaja ena ciljna (odvisna) diskretna spremenljivka (razred), ampak so spremenljivke (atributi) med seboj enakovredne - včasih so podane vrednosti nekaterih spremenljivk in nas zanimajo vrednosti preostalih spremenljivk in obratno. Lahko so podane vrednosti vseh spremenljivk in želimo samo preveriti, če je relacija v

tem primeru izpolnjena (resnična). Glede na izrazno moč jezika za predstavitev učnih primerov in naučenega znanja pristope k učenju ločimo na učenje asociacij in na induktivno logično programiranje.

**Asociacije** O asociacijah govorimo, če uporabljamo atributni opis. Za razliko od klasifikacije in regresije ni posebne odvisne spremenljivke, ampak je vsaka spremenljivka lahko odvisna oziroma izhodna (nima podane vrednosti) ali neodvisna oziroma vhodna (ima podano vrednost). Zaradi nedoločenosti odvisne spremenljivke včasih pravimo takemu učenju tudi nenanadzorovano učenje (angl. unsupervised learning), čeprav se ta izraz pogosteje uporablja za razvrščanje, ki je opisano pozneje.

V principu se lahko učimo asociacij tako, da problem razbijemo na toliko klasifikacijskih oziroma regresijskih problemov, kolikor je atributov za opis primerov. Za vsak diskretni atribut vpeljemo en klasifikacijski problem, kjer ta atribut nastopa kot odvisna spremenljivka (razred), vsi ostali atributi pa kot neodvisne spremenljivke. Analogno za vsak zvezni atribut vpeljemo en regresijski problem, kjer ta atribut nastopa kot odvisna (regresijska) spremenljivka, vsi ostali atributi pa kot neodvisne spremenljivke. Dobimo toliko učnih problemov, kolikor je atributov. Posebna pristopa k asociacijam sta asociativne nevronske mreže in povezovalna pravila.

**Asociativne nevronske mreže** To so umetne nevronske mreže, kjer so vsi nevroni lahko vhodni in izhodni hkrati. Vsak nevron je povezan z vsakim drugim z dvosmerno povezavo, tako da lahko signal potuje v obe smeri. Uteži na povezavah se pomnožijo s signalom in vsak nevron izračunava uteženo vsoto vseh svojih vhodov. Normalizirana utežena vsota predstavlja stanje nevrona, ki predstavlja vrednost ustreznega atributa. Pri asociativnih nevronskeih mrežah je začetno stanje podano z zanimimi vrednostmi atributov. Zatem vsi nevroni vzopredno in asinhrono izračunavajo svoja nova stanja (kar pomeni da se nevronom z nezanimimi vrednostmi določijo vrednosti, tistim z določenimi vrednostmi pa se lahko vrednosti morda tudi spremenijo). Ko se vrednosti nevronov ne spreminja več, končne vrednosti vseh nevronov (atributov) predstavljajo izhod.

Glede na učni algoritem, ki nastavlja uteži na povezavah, ločimo Hopfieldove in Bayesove nevronske mreže. Učenje Hopfieldove nevronske mreže temelji na korelaciji aktivnosti stanj med posameznimi nevroni. Pri Bayesovih nevronskeih mrežah vsak nevron uporablja pravilo naivnega Bayesovega klasifikatorja za izračun utežene vsote. Temu je prilagojeno učno pravilo, ki izračuna pogojne verjetnosti aktivnosti enega nevrona pri dani aktivnosti drugega nevrona.

Asociativne nevronske mreže se uporabljajo pri problemih prepoznavanja, kjer je ponavadi poznan sam del podatkov in je treba prepozнатi celoto. Npr. pri prepoznavanju osebe je lahko podan del obraza in je treba rekonstruirati cel obraz. Drugo področje uporabe so kombinatorični optimizacijski problemi, kjer parametri (atributi) "tekmujejo" med seboj, tako da povečanje vrednosti enega parametra zmanjša vrednost drugih parametrov in obratno. Mreža iz danega začetnega stanja poišče kompromisno (lokalno optimalno) stanje.

**Povezovalna pravila** Povezovalna pravila (*association rules*) so podobna klasifikacijskim pravilom, le da v sklepnom delu lahko nastopa poljuben (diskretni) atribut, v pogojnem delu pa konjunkcija pogojev iz podmnožice preostalih atributov. Namen učenja povezovalnih pravil na podatkovni bazi je iskanje zanimivih relacij med atributi, ki lahko doprinesejo zanimiva (nova) spoznanja o samih podatkih oziroma o zakonitostih v dani problemski domeni. Naloga učnega algoritma je poiskati vsa povezovalna pravila, ki so dovolj točna in hkrati splošna (se lahko dovolj pogosto uporabijo).

Povezovalna pravila so se najprej uveljavila pri analizi podatkov o nakupovanju izdelkov v trgovinah. Z njihovo pomočjo lahko trgovci načrtujejo razporeditev artiklov na policah. Pravilo npr. pove, da obstaja za tistega, ki izbere moko, orehe in sladkor velika verjetnost, da bo kupil tudi jajca.

**Induktivno logično programiranje (ILP)** Če za opisni jezik uporabljam predikatni račun prvega reda, se najpogosteje omejimo na izraze v programskejem jeziku prolog (Hornovi stavki z negacijo). Učni problem je podan s predznanjem (poljubna množica relacij/programov v prologu) in z množico pozitivnih (pravilnih) in negativnih (nepravilnih) dejstev za ciljno relacijo. Naloga algoritma je izpeljati relacijo/program v prologu, ki pravilno pokriva čim več pozitivnih primerov in čim manj negativnih primerov.

Induktivno logično programiranje se v praksi uporablja na geometrijskih in prostorskih problemih, kot so npr. določanje strukture kemijske spojine, določanje strukture proteinov, iskanje funkcij genov v biologiji ipd. Za ILP so primerni tudi problemi tridimensionalnega načrtovanja in/ali razpoznavanja prostorov, reševanja problemov statike in trdnosti v strojništvu itd.

## Odkrivanje zakonitosti - sistemi (diferencialnih) enačb

Pri modeliraju realnega sveta imamo pogosto opravka z meritvami procesov, ki so medsebojno povezani in odvisni. Z učenjem sistema enačb, ki ustreza meritvam, želimo odkriti, kakšne so te odvisnosti, hkrati pa sistem enačb lahko uporabimo za simulacijo in napovedovanje. Večinoma gre za zvezne spremenljivke ter za numerične relacije in odvisnosti (medtem ko pri induktivnem logičnem programiranju nastopajo tudi diskretne spremenljivke in nas zanimajo predvsem logične relacije in odvisnosti).

Pri učenju sistema enačb je lahko struktura enačb podana vnaprej (kot predznanje), ali pa mora učni algoritem iz učnih podatkov izpeljati poleg vrednosti koeficientov tudi strukturo enačb. Če gre za modeliranje dinamičnih sistemov, katerih stanja se s časom zvezno spremenjajo, je treba izpeljati sistem parcialnih diferencialnih enačb. Naloga učnega algoritma je poiskati čim preprostnejši sistem enačb, ki karseda natančno opisuje vhodne podatke.

Tako modeliranje se uporablja v meteorologiji pri modeliranju vremenskih sistemov, v ekologiji pri modeliranju naravnih podsistemov, v strojništvu pri modeliranju tehničnih procesov itd.

## Nenadzorovano učenje - razvrščanje

Nenadzorovano učenje (*unsupervised learning*) ima drugačno nalogu kot klasifikacija - podani so atributni opisi primerov, razredov pa ne poznamo (od tod izraz nenadzorovano učenje). Naloga učnega algoritma je določiti razrede (roje, skupine, grozde, grupe, gruče). Poleg besede razvrščanje se uporabljajo sinonimi rojenje, grupiranje, gručenje ali grozdenje (*clustering*). Razvrščanje se uporablja pri analizi naravnih in tehnoloških procesov, pri analizi ekonomskih trendov, pri preverjanju konsistentnosti in odvisnosti podatkov itd.

Število želenih razredov je lahko podano vnaprej kot predznanje, ali pa mora primerno število razredov določiti sam učni algoritem. Naloga učnega algoritma je torej določiti relativno majhno število koherentnih razredov, t.j. skupin primerov, ki so si med seboj čim bolj podobni. Podobnost med primeri je odvisna od izbrane metrike, ki je odločilnega pomena za rezultat razvrščanja. Primerena mera podobnosti je lahko del predznanja. Algoritmi tipično uporabljajo enega od treh osnovnih pristopov:

- od spodaj navzgor: na začetku je vsak primer svoj razred. Algoritem iterativno združuje najbolj podobne razrede med seboj, dokler ne ostaneta samo še dva razreda. Zatem uporabnik ali pa sam sistem izbere najustreznejše število razredov.
- od zgoraj navzdol: na začetku vsi primeri pripadajo enemu razredu. Zatem algoritem glede na podobnost iterativno razbija razrede na podrazrede, dokler ne doseže želenega števila razredov.
- na začetku naključno izbere  $n$  primerov kot nosilce razredov, kjer je  $n$  želeno število razredov. Zatem se vsi primeri glede na podobnost razvrstijo v enega od  $n$  razredov. Glede na lastnosti posameznih razredov se izbere  $n$  novih predstavnikov razredov (pri tem lahko en razred da več kot enega predstavnika in kak drug razred nobenega predstavnika). Cel postopek se ponavlja, dokler niso razredi zadovoljivo koherentni.

## Spodbujevano učenje

Spodbujevano učenje (*reinforcement learning*) se ukvarja s problemom, kako avtonomnega agenta v danem okolju naučiti optimalnih akcij za doseg ciljev. Agent dobiva iz okolja podatke o trenutnem stanju ter z akcijami (potezami, odločitvami) vpliva na to, da se stanje okolja spreminja. Poleg opisa stanja agent dobi tudi nagrado ali kazen, ki pa ni nujno posledica zadnje akcije, ampak lahko do nagrade ali kazni vodi zaporedje več, mnogokrat zelo veliko akcij. Ker je povratna informacija zakasnjena, je učenje zaradi tega otežkočeno in počasnejše. Učenec izbira med bolj konzervativnim pristopom in se zanaša na dosedaj naučeno, ali pa poskuša raziskovati in uporablja redkejše akcije ozziroma poteze, da si pridobi izkušnje v novih, neznanih situacijah.

Naloga je sestaviti optimalno strategijo, ki maksimizira ustrezno uteženo vsoto vseh nagrad, ozziroma minimizira uteženo vsoto vseh kazni. Ker želimo čimprej priti do želenega ciljnega stanja, je utežitev nagrad taka, da so pomembnejše takojšnje nagrade. Najpogostejši pristop k spodbujevanemu učenju je iterativno obnavljanje ocen kvalitete možnih akcij iz danega stanja (t. i. učenje Q).

S spodbujevanim učenjem se rešujejo problemi kontrole dinamičnih sistemov (npr. problem upravljanja žerjava in upravljanje robota), optimizacijski problemi ter igranje iger (šah, dama, tarok itd.).

## 1.2 Zgodovina strojnega učenja

*Praktično je nemogoče najti "originalno" idejo, ki še ne bi bila prisotna v zgodnejši znanstveni literaturi. Ko se pojavi potreba po ideji, jo običajno odkrije več skupin naenkrat.*

*James A. Anderson*

Poskusi s prvimi algoritmi strojnega učenja so se pojavili hkrati z nastankom prvih elektronskih računalnikov. Že pri zmetkih strojnega učenja so se pokazale različne smeri razvoja:

- simbolično učenje pravil,
- nevronske mreže,
- spodbujevano učenje,
- numerične metode,
- formalna teorija naučljivosti.

Ostale smeri, kot so: odkrivanje zakonitosti, konstruktivna indukcija ter induktivno logično programiranje, so se razvile nekoliko pozneje in jih v tem razdelku ne obravnavamo.

### Simbolično učenje pravil

Začetki simboličnega učenja pravil segajo v začetek šestdesetih let. Hunt, Martin in Stone so v svoji knjigi leta 1966 predstavili sistem CLS (Concept Learning System) za učenje odločitvenih dreves iz primerov [17]. Osnovna motivacija za razvoj sistema je bilo simuliranje človeškega obnašanja. V knjigi so podrobno opisani izčrpni poskusi z različnimi izpeljankami sistema ter primerjava z učenjem ljudi. Eksperimentalno so potrdili, da obstaja več strategij človeškega učenja in da je ena od njih zelo podobna sistemu CLS.

Že takrat so se ubadali s problemi, ki jih rešujejo današnji sistemi za strojno učenje:

- različno obravnavanje neznanih vrednosti (*unknown values*) in nesmiselnih vrednosti (*inapplicable*),
- cena testiranja atributa,
- določanje najboljšega izmed logično ekvivalentnih pravil,
- ocena klasifikacijske točnosti, vključno s problemom apriorne verjetnosti razredov in apriorne klasifikacijske točnosti,
- definicija preprostosti pravila,
- problem inkrementalnega učenja in povečevanja števila učnih primerov z metodo oken (*windowing*),

- problem hevristične ocene atributa, vključno s predlogom uporabe informacijskega prispevka atributa ter uporabe statistike  $\chi^2$ ,
- problem zveznih in urejenih diskretnih atributov ter problem večvrednostnih atributov,
- problem statistične zanesljivosti pravil in delov pravil.

Sistem CLS so poskušali uporabiti za generiranje diagnostičnih pravil v medicini. Njihove izkušnje so podobne izkušnjam mnogih, ki so pozneje uporabljali sisteme za strojno učenje v medicinski diagnostiki. Zdravnikom je program pomagal organizirati in analizirati njihove podatke, ni pa odkril novih (neznanih) pravil.

Sistem CLS je bil osnova za nastanek sistema ID3 (Iterative Dichotomizer 3), ki ga je razvil Ross Quinlan leta 1979 [34], da bi avtomatsko gradil pravila iz velikih množic podatkov. ID3 je uporabil za gradnjo pravil, ki odločajo, ali je šahovska končnica dobljena ali izgubljena. Z ustrezno definicijo atributov, ki opisujejo pozicijo na šahovnici, je iz začetnih 1,4 milijona različnih opisov pozicij dobil okoli 30.000 različnih opisov. Iz njih je ID3 zgradil odločitveno drevo s 334 vozlišči. Po pregledu drevesa je Quinlan definiral novo množico atributov, tako da je dobil namesto 30.000 različnih opisov samo 428. Novo odločitveno drevo je imelo samo 83 vozlišč in je pravilno klasificiralo vseh 1,4 milijona možnih pozicij.

ID3 je imel močan vpliv na razvoj cele vrste sistemov za gradnjo odločitvenih dreves in pravil, kot so ACLS [33], Assistant [22, 3], C4 [35] ter C4.5 [36]. Neodvisno od ID3 je bil razvit sistem CART [2], ki temelji na podobnih idejah.

Druga veja razvoja so bili algoritmi za neposredno učenje odločitvenih pravil za razliko od posrednega preko odločitvenih dreves. Prvi sistem, ki se je izkazal tudi v praksi, je bil algoritem AQ11, ki so ga uporabili za učenje diagnosticiranja bolezni rastline soje [26]. Klasifikacijska točnost avtomatsko zgrajenih pravil je presegla točnost pravil, ki so jih podali strokovnjaki za diagnosticiranje bolezni soje. Ta uspešna aplikacija je imela močan vpliv na popularnost razvoja in uporabe metod strojnega učenja. Algoritmu AQ11 je sledila serija algoritmov tipa AQ, kot npr. AQ15 [25]. Iz algoritmov tipa AQ je bilo razvitih več sistemov za učenje odločitvenih pravil, od katerih je najbolj znan CN2 [6, 5, 4].

## Nevronske mreže

*Podbobe algoritme je predlagalo kar nekaj raziskovalcev; nekateri so razločno predstavili svoje poglede in rezultate, kar je koristilo vsem.*

*Yoh-Han Pao*

Donald Hebb je leta 1949 v knjigi "The Organization of Behaviour" [13] definiral preprosto pravilo spremenjanja jakosti povezav na sinapsah bioloških nevronov, za katero obstaja precej nevrofizioloških potrditev, da je to osnovno pravilo učenja v možganih. Pravilo pravi, da je jakost (prepustnost) povezave sorazmerna hkratni aktivnosti obeh nevronov. Za to pravilo učenja je značilno, da zadošča omejitvi na *lokalno* informacijo. Prav ideja lokalne informacije je bilo vodilo za razvoj algoritmov učenja umetnih nevronskeh mrež.

Že leta 1958 je Rosenblatt razvil enonivojski perceptron ter osnovno pravilo učenja delta, ki omogoča takšni mreži reševanje linearnih klasifikacijskih problemov [37]. Začetno navdušenje je splahnelo, ko sta Marvin Minsky in Seymour Papert leta 1969 [29] dokazala, da se s pravilom delta ne da reševati nelinearnih problemov, kar je v prejšnji meri vplivalo na zaostanek v razvoju umetnih nevronskih mrež. Kljub temu so se raziskave z nekoliko manjšo zagnanostjo nadaljevale.

Močnejšo spodbudo je področje dobilo po objavi člankov Johna Hopfielda leta 1982 [15, 16], ki je pokazal lepe lastnosti simetrične večsmerne enonivojske mreže. Dokončen zalet pa je področju dalo odkritje posplošenega pravila delta za vzvratno razširjanje napak (*backpropagation of errors*), ki je omogočilo učenje večnivojskih perceptronov in s tem reševanje poljubnih nelinearnih problemov. Čeprav odkritje pripisujejo različnim (neodvisnim) avtorjem, je objava algoritma [38] v knjigi, ki sta jo uredila David Rumelhart in James McClelland leta 1986, dejansko sprožila pravi val raziskav na tem področju.

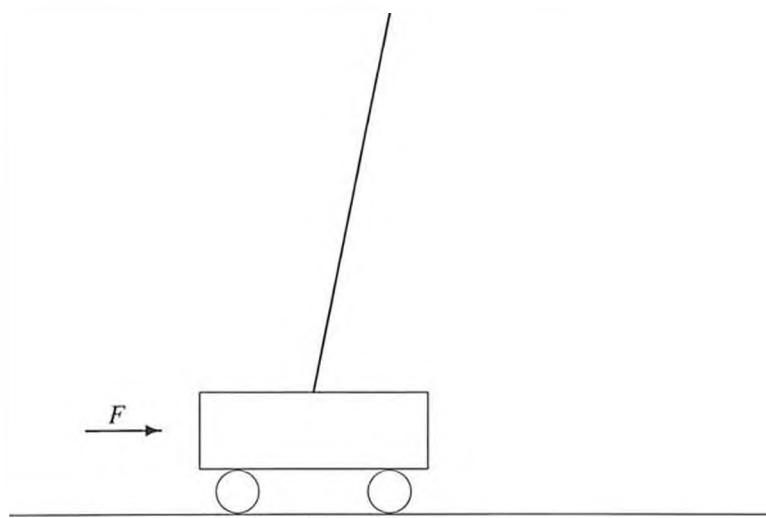
## Spodbujevano učenje

Eden prvih uspešnih poskusov učenja je Samuelov program za igranje dame (*checkers*) iz leta 1959. Samuel [39] je primerjal dva pristopa k učenju igranja. Prvi pristop je temeljil na zapominjanju pozicij na igrальнem polju skupaj z njihovimi ocenami kvalitete (glede na ocene, dobljene s pogledom naprej). Te ocene so bile uporabljene za "podaljševanje" pogleda naprej. Če je algoritem pregledoval prostor pozicij za npr. 3 polpoteze naprej, je lahko pozicije v listih drevesa namesto s hevristično oceno ocenil kar s shranjeno oceno (seveda le, če je bila pozicija shranjena od prej). Na ta način je ocena pozicij lahko vsaj v nekaterih vejah ustrezala pogledu naprej za 6, 9 itd. polpotez. Slabost tega pristopa je poraba pomnilniškega prostora, saj je možnih različnih pozicij mnogo več, kot jih lahko shranimo v pomnilnikih današnjih računalnikov, kaj šele računalnikov iz leta 1959. Kljub temu se je program s pomnilniškim pristopom naučil uspešnih otvoritev igre in relativno dobrega igranja v končnicah, ni pa se uspel naučiti dobre "sredinske" igre.

Drug pristop, ki ga je uporabil Samuel, je temeljil na spodbujevanem učenju (*reinforcement learning*). Samuel je definiral parametre, ki so se uporabljali v kombinirani oceni kvalitete pozicije. Vsak parameter je imel svojo utež in cilj učenja je bil poiskati optimalno nastavitev uteži. Uteži so se spremajale glede na razliko med ocenjeno kvaliteto pozicije in dejansko kvaliteto, dobljeno iz igre, ki je tej poziciji sledila. Algoritem se je, za razliko od prejšnjega pristopa, naučil relativno dobre sredinske igre ter slabe igre ob otvoritvi in v končnicah.

Nekateri raziskovalci so začeli kombinirati spodbujevano učenje z dodatnim preiskovanjem prostora z genetskimi algoritmi. Tako je Bagley leta 1967 uporabil spodbujevano učenje v kombinaciji z genetskimi algoritmi za učenje strategij igre treh kmetov na šahovnici  $3 \times 3$ . To je bilo pionirske delo tudi na področju uporabe genetskih algoritmov [10].

Pionirske delo na področju učenja vodenja dinamičnih sistemov sta opravila Michie in Chambers leta 1968 [27]. Sistem BOXES sta naučila voditi preprost, a netrivialen dinamični sistem vozička in palice. Voziček se je gibal na tračnicah omejene dolžine, palica pa je bila vpeta na vozičku v eni točki z osjo pravokotno na ravnilo, ki jo definirajo sredina tračnic in palica (glej sliko 1.1). Sistem je bil še dodatno omejen z (ekvidistantno) diskretizacijo



Slika 1.1: Dinamični sistem voziček – palica.

časa. Naloga je bila ob vsakem časovnem intervalu poriniti voziček z zunanjim silo konstantne velikosti v levo ali desno tako, da palica ni padla.

Sistem BOXES je definiral diskretiziran prostor možnih situacij in za vsako situacijo (škatlo – od tod ime sistema) poiskal pravilno odločitev (levo ali desno). Algoritem je začel z naključno postavljenimi odločitvami in je odločitve spremenjal ob vsakem padcu palice. Sčasoma se je naučil uspešno voditi voziček, ne da bi palica padla, 10.000 simuliranih časovnih intervalov, kar ustreza 200 sekundam. BOXES je imel močan vpliv na poznejše raziskave učenja vodenja dinamičnih sistemov s spodbujevanim učenjem, induktivnim učenjem in nevronskimi mrežami [43].

## Statistične metode

Statistične metode učenja so verjetno nastarejše, saj so matematične osnove in izpeljave statističnih metod definirali še pred uporabo elektronskih računalnikov. Šele z razvojem računalnikov so te metode postale dostopne širšemu krogu raziskovalcev. Klasično delo, ki obravnava statistične metode učenja, je Nilssonova knjiga "Learning Machines" [31].

Nilsson opisuje metode, ki temeljijo na linearnih diskriminantnih funkcijah, kosoma linearnih in kvadratnih diskriminantnih funkcijah. Prav tako opisuje naivni Bayesov klasifikator ter omenja probleme z zanesljivo aproksimacijo verjetnosti v povezavi z Laplaceovim zakonom zaporednosti. Definicijo naivnega Bayesovega klasifikatorja kot tudi bolj izčrpno analizo bayesovskega pristopa k odločanju opisuje Good že leta 1950 [11, 12]. Metoda najbližjih

sosedov je opisana v članku iz leta 1967 [7].

### Formalna teorija naučljivosti

Kot nadaljevanje teorije rekurzivnih funkcij in teorije izračunljivosti je Gold leta 1967 postavil osnove formalni teoriji naučljivosti [32]. Definicija naučljivosti je analogna definiciji izračunljivosti in logični dokazljivosti, katerima sta temelje postavila Kurt Gödel leta 1931 z dokazom izreka o nepopolnosti logike prvega reda in Alan Turing leta 1936 z dokazom o neodoločljivosti univerzalnega jezika. Prvi izrek pravi, da v vsaki neprotislovni logiki prvega reda obstajajo formule, katerih resničnosti in/ali neresničnosti ni možno dokazati. Drugi izrek pa pravi, da ne obstaja algoritem, ki bi na vprašanje, ali dani algoritem za dani vhod odgovori z "da", znal poiskati odgovor v končnem številu korakov [14, 18]. Čeprav ima teorija naučljivosti le skromen pomen za prakso, je zanimiva kot teoretična osnova področja strojnega učenja.

Iz Goldove teorije naučljivosti je bil leta 1981 razvit eden prvih algoritmov induktivnega logičnega programiranja MIS (*Model Inference System*) kot delajoč, vendar časovno in prostorsko neučinkovit sistem [40].

Sele z razvojem teorije "verjetno približno pravilnega učenja" (*probably approximately correct learning* [44], ki jo na kratko označujemo kot PAC-učenje, je formalna teorija prispevala k boljšemu razumevanju težavnosti problema učenja ter k definirанию bolj realnih možnosti algoritmov učenja nasploh [19].

## 1.3 Nekateri zgodnji uspehi strojnega učenja

*Ljudje so nasploh bolj sposobni za umetnost kakor za znanost. Prva po večjem delu pripadajo njim, druga po večini svetu.*

*Johann Wolfgang Goethe*

### AM (*Automatic Mathematician*)

Lenat [24] je leta 1979 razvil teorijo hevristik, ki jo je podkrepil s sistemom AM. AM je sistem za samostojno odkrivanje teorij in zanimivih konceptov v dani domeni. Začne ("razmišljanje") z množico hevristik, ki vsebujejo znanje o principih znanstvenega raziskovanja, o načinih ocenjevanja pomembnosti izpeljanih konceptov in o strategijah za izbiro poti, ki vodijo do zanimivih zaključkov. Zatem samostojno išče zanimive povezave in koncepte v dani domeni, pri tem pa uporablja strategijo najprej naboljši.

Iz 115 osnovnih konceptov teorije množic (definicija množice, seznama, unije, kompozicije itd.) in 243 hevristik je AM odkril koncept števila, scštevanja, odštevanja, množenja, deljenja, potence, praštevila itd. Podobne uspehe je imel na področju ravninske geometrije. Tipična hevristika je naslednja: "Ujemanje je zanimivo." Koncept množenja je AM odkril na 4 načine: kot večkratno seštevanje, kot analogijo kartezičnemu produktu, kot moč potenčne množice unije dveh množic in kot dolžino seznama, ki ga dobimo, če vsak element prvotnega seznama zamenjamo z elementi drugega seznama. Ker je po več različnih potekh prišel do

istega koncepta, je hevristika ujemanja ocenila tak koncept kot verjetno pomemben. Primer druge hevristike je: "Če je operator zanimiv, definiraj še obratnega." Tako je po množenju odkril še koncept deljenja. Hevristika "Poišči ekstremne primere" mu je omogočila, da je odkril koncept praštevila in nekoliko manj znan koncept maksimalno deljivih števil.

Lenat definira metodologijo raziskovanja takole:

1. Novo znanje lahko razvijemo s pomočjo hevristik.
2. Z novim znanjem potrebujemo nove hevristike.
3. Nove hevristike lahko razvijemo s pomočjo hevristik.
4. Z novim znanjem potrebujemo nove predstavitve znanja.
5. Nove predstavitve znanja lahko razvijemo s pomočjo hevristik.

AM je uspešno opravil prvo točko. Ko je njegovo raziskovanje izčrpalo ozko področje domene, AM ni bil zmožen nadaljevati raziskovanja v pravi smeri in nadgraditi pridobljenega znanja.

## Eurisko

Lenat je nakazal možno avtomatizacijo izvajanja tretje in pete točke v gornjem opisu in njegova teorija hevristik je podlaga za sistem, ki ga je imenoval Eurisko. Eurisko obravnava hevristike kot znanje, jih med učenjem spreminja ter dodaja nove hevristike. Za spreminjanje hevristik uporablja hevristike same, tako da lahko hevristike spreminja same sebe.

Eurisko se je naučil igrati zahtevno igro (Traveller). V tej igri je potrebno sestaviti oboroženo ladjevje, nekaj deset ladij različnega tipa z različnim orožjem na krovu. Pri tem je množica pravil zelo zapletena. Sestava ene ladje zahteva definicijo več kot 50 faktorjev. Eurisko je iz 146 konceptov, vzetih iz igre, po približno 10000 simuliranih bitkah definiral optimalno ladjevje, ki se je popolnoma razlikovalo od ladjevij, ki so jih uporabljali človeški strokovnjaki. Ko se je Eurisko pojavil na turnirju, so se njegovemu ladjevju posmehovali, dokler se ni igra začela. Eurisko je gladko zmagal vse bitke in postal prvak te igre v ZDA. Naslednje leto so pravila igre spremenili in jih objavili šele teden dni pred tekmovanjem. Eurisko je zopet zmagal in naslednjič so prepovedali programom sodelovanje na turnirju.

Drugi uspeh je Eurisko dosegel pri realnem problemu načrtovanja integriranih vezij. Iz osnovnih struktur je sestavil novo strukturo, ki je lahko opravljala dve različni nalogi hkrati. Ta struktura je prispevala k optimalnejšim načrtom integriranih vezij. Eurisko je pri tem "izumu" uporabil hevristiko: "Če je struktura koristna, jo poskusi narediti bolj simetrično." Zaradi kompleksnosti načrtovanja integriranih vezij so načrtovalci težili k čim preprostejšim strukturam in niso prišli na idejo, da bi zgradili strukturo, ki opravlja dve nalogi naenkrat [28].

## Meta-Dendral

Meta-Dendral sta razvila Buchanan in Mitchell leta 1978 [8]. Namenjen je učenju pravil, ki napovedujejo, kakšen bo razpad zaplenenih molekul pri bombardiranju molekul pri masni

spektrografiji. Ta pravila se uporabljo za napovedovanje masnega spektrograma dane molekule. Pri razpoznavanju neznane molekule se masni spektrogram primerja z napovedanimi masnimi spektrogrami različnih znanih molekul. Iz podobnosti spektrogramov se sklepa na strukturo neznane molekule. Meta-Dendral je iz podatkov za nekaj družin molekul z znano strukturo in znanim spektrogramom zgradil pravila, ki do takrat niso bila znana, in je s tem prispeval novo znanje k masni spektrografiji. Podobne uspove je imel Meta-Dendral tudi v nuklearni magnetni resonančni spektroskopiji. Novo znanje je bilo objavljeno v strokovnih revijah na področju kemije.

## MIS (Model Inference System)

Ehud Shapiro [40] je leta 1981 razvil sistem za avtomatsko generiranje teorij v podmnožici predikatnega računa prvega reda, ki ustreza programom v programskej jeziku prolog [1]. Sistem MIS se iz učnih primerov uči opisov relacij, ki jih lahko direktno izvajamo s prologovim interpreterjem. Poleg vhodnih podatkov, ki opisujejo resnične in neresnične elemente ciljnih relacij, uporablja MIS kot predznanje definicije pomožnih relacij, iz katerih potem izpeljuje ciljno relacijo. Sistem med iskanjem opisa teorije generira nove primere – elemente ciljne relacije. Od uporabnika zahteva, da za vsak nov primer opredeli, ali pripada ciljni relacija ali ne.

Poglejmo si primer. Naj bo ciljna relacija “append( $S_1, S_2, S$ )”, ki je resnična, če je seznam  $S$  tak, da ga lahko dobimo s stikom dveh seznamov  $S_1$  in  $S_2$ . MIS je imel na voljo 56 dejstev, za katere je hkrati navedeno, če so resnični (elementi relacije “append”) ali ne. Primeri dejstev so:

```
append([], [a], [a])                je res
append([a,b], [c,d,e], [a,b,c,d,e])  je res
append([a], [], [])                 ni res
```

Iz primerov dejstev je MIS izpeljal pravilno definicijo relacije v prologu:

```
append([], X, X).
append([A|X], Y, [A|Z]) :- append(X, Y, Z).
```

MIS je avtomatsko zgradil mnogo različnih rekurzivnih definicij relacij, izpeljal je pravilo za seštevanje, množenje celih števil itd. Poleg generiranja novih programov lahko algoritem uporabimo tudi za avtomatsko popravljanje obstoječih programov v prologu. Edina zahteva je, da uporabnik pravilno odgovarja na (mnoga) vprašanja, ki mu jih MIS zastavlja.

Uspeh sistema MIS je vzbudil zanimanje za novo vejo strojnega učenja, ki ji pravimo induktivno logično programiranje (*inductive logic programming*) ali na kratko ILP [23]. Zaradi velikanskega preiskovalnega prostora je MIS (in tudi nekateri drugi sistemi ILP) primeren za učenje krajših teorij. Potrebnih je bilo še precej raziskav, da so postali algoritmi ILP praktično uporabni.

## 1.4 Aplikacije strojnega učenja

*Vsemogočnost ne pomeni znati, kako se delajo vse stvari, temveč preprosto delati vse stvari. Tega ni potrebno prevesti v jezik.*

*Alan Watts*

V tem razdelku so opisana nekatera prodročja aplikacij strojnega učenja ter nekaj primerov uspešnih aplikacij.

### Diagnostika proizvodnega procesa

V večini proizvodnih procesov prihaja do različnih odstopanj od običajnega poteka. Pomembna so predvsem odstopanja, ki jih ni mogoče predvideti, ker nadzorni sistem nima popolnega modela (znanja) o celotnem proizvodnem procesu. Z metodami strojnega učenja lahko iz meritev, opravljenih v različnih časovnih obdobjih, izpeljemo model, ki ga lahko uporabimo za napovedovanje obnašanja proizvodnega procesa. Pri tem je bistvena izbira, katere parametre procesa bomo merili. Če nimamo dovolj znanja, je najbolje meriti čimveč parametrov in prepustiti algoritmom strojnega učenja, naj izberejo pomembne.

V neki švedski tovarni papirja so skušali rešiti problem prevelike količine zmečkanega papirja. Ker po več analizah niso uspeli odkriti vzroka, so uporabili strojno učenje. Pred tem so opravili serijo meritev različnih parametrov proizvodnega postopka ter za vsako meritev zabeležili procenček zmečkanega papirja.

Problem so pretvorili v klasifikacijskega, tako da so razred (procent zmečkanega papirja) diskretizirali. Z uporabo algoritma za gradnjo odločitvenih dreves so praktično takoj (v ko-renu drevesa) odkrili parameter, ki je najbolj vplival na količino zmečkanega papirja. Če je bila vrednost tega parametra v določenih mejah, je bila količina zmečkanega papirja bistveno manjša. V proizvodnem procesu so začeli kontrolirati kritični parameter in vzdrževati njegovo vrednost na predpisanim intervalu, kar je prispevalo k ogromnim prihrankom.

V jeklarnah je potrebno iz izmerjenih parametrov oceniti kvaliteto vsake šarže (gmote stopljenega) jekla posebej, ker se na tej osnovi določi, v kakšne namene se bo šarža uporabila. Od pravilne odločitve je odvisna kvaliteta izdelkov in tudi finančna donosnost. Ocenjevanje šarž jekla je delo strokovnjakov z dolgoletnimi izkušnjami. Ker strokovnjak ni vedno dosegljiv (bolezen, vikend, počitnice), proizvodni proces pa se ne sme ustaviti, so v konkretni jeklarni v preteklosti pogosto uporabljali ad-hoc (neoptimalne) odločitve. Na osnovi učnih primerov iz preteklosti so z uporabo odločitvenih dreves izpeljali pravila, ki so bila na neodvisni testni množici celo bolj točna od strokovnjakov. V konkretni jeklarni so zato razviti ekspertni sistem začeli rutinsko uporabljati.

## Medicinska diagnostika in prognostika

Temelj uspešnega zdravljenja po današnji medicinski paradigmi je pravilna diagnoza (vrsta obolenosti). Diagnozo postavi zdravnik glede na simptome pacienta, glede na podatke, ki jih dobi zdravnik pri pregledu pacienta (anamneza), ter iz podatkov različnih laboratorijskih testov, rentgenskih, ultrazvočnih, scintigrafskih slik itd. Podoben je problem prognoze, kjer je potrebno namesto vrste obolenosti napovedati potek oziroma težavnost obolenja.

Na osnovi podatkov bolnikov, ki so se v preteklosti zdravili v določeni kliniki in za katere so znane zanesljive diagnoze ali naknadno ugotovljeni izidi, lahko z algoritmi strojnega učenja izpeljemo pravila, ki se lahko uporabijo za diagnosticiranje oziroma prognoziranje novih bolnikov. Izpeljana pravila lahko nudijo razlago diagnosticiranja, služijo za poučevanje študentov medicine ali pomagajo zdravnikom začetnikom. Z algoritmi strojnega učenja ocenimo tudi pomembnost posameznih parametrov (simptomov ali laboratorijskih izvidov) za samo diagnozo in/ali potek zdravljenja.

V specializirani kliniki so želeli izpopolniti prognostiko bolnic, ki so jim z operacijo odstranili tumor na dojki. Zdravniki so se zavedali, da je njihova prognostika nezanesljiva. Iz podatkov okoli tristo bolnic, za katere je bil znan izid nekaj let po operaciji, smo skušali izpeljati klasifikatorje, ki bi napovedovali potek bolezni nekaj let po operaciji. Izkazalo se je, da noben od podatkov, ki so opisovali pacientke takoj po operaciji, ni bil relevanten za prognozo. Zdravniki so pri prognoziranju te parametre upoštevali kot relevantne in so bile zaradi tega njihove prognoze večinoma naključne in s tem celo slabše od preprostega klasifikatorja, ki upošteva samo apriorne verjetnosti posameznih izidov in ignorira vse podatke o pacientkah.

## Ocenjevanje zavarovancev in posojilojemalcev

*Več je zakonov, bolj revni so ljudje. Bolj natančni so zakoni, več je lopovov in kriminalcev.*

*Lao Ce*

Zavarovalniški agenti pri zavarovanju oseb, nepremičnin, imetja itd. poskušajo ugotoviti, če so zavarovanci preveč tvegani za zavarovalnico. Pri tem uporabljajo formularje, ki jih zavarovanci izpolnijo pred sklepanjem pogodb, ter druge dostopne podatke o klientih. Agenti se naslanjajo na svoje izkušnje in znanje, ki pa ni popolno. Z algoritmi strojnega učenja lahko iz podatkovnih baz o klientih in njihovih zahtevkih po izplačilih zavarovalnih premij izluščimo pravila, ki jih uporabimo za ocenjevanje tveganja pri zavarovanju novih klientov.

Na podoben način banke odobrijo svojim klientom posojila, če so ocenjeni kot dovolj zanesljivi, da bodo posojila dejansko odplačali. Kliente ocenjujejo po osebnih in poslovnih podatkih, ki jih posredujejo bančnim uslužbencem. Pri tem so problematični tisti primeri, kjer ni jasne odločitve o tem, ali je klient zanesljiv ali ne.

V neki banki so uporabljali statistične metode za določitev primernosti klienta. Klientu so posojilo odobrili, če je bila njegova ocena nad vnaprej določenim zgornjim pragom, ter zavrnili, če je bila njegova ocena pod spodnjim pragom. Problematični so klienti z oceno, ki je med pragovoma. Take cliente mora obravnavati strokovnjak, ki da končno odločitev o odobritvi

posojila. Odločili so se za reševanje "nejasnih" klientov s pomočjo strojnega učenja. Iz učnih primerov, ki so opisovali tisoč "nejasnih" klientov skupaj s podatkom, ali je bilo posojilo dejansko odplačano (razred), so izpeljali odločitvena pravila, ki so pravilno napovedovala dve tretini "nejasnih" klientov na neodvisni testni množici, kar je bolje od strokovnjakov. Pravila so omogočala tudi iskanje razlage odločitve, kar je elegantno rešilo obrazložitve zavrnjenim klientom, zakaj je njihova prošnja za posojilo zavrnjena. Banka je zato začela pravila rutinsko uporabljati.

## Klasifikacija slik

Z napredkom digitalnega zajemanja slik in videoposnetkov je količina slikovne informacije v zadnjih nekaj letih drastično narasla, s tem pa tudi potreba po iskanju, razvrščanju in klasifikaciji slik, prepoznavanju objektov na slikah itd. Na mnogih področjih je ročno pregledovanje slik postalo nesprejemljivo počasno in je nujno te postopke vsaj delno avtomatizirati. Strojno učenje uporabimo pri klasifikaciji slik tako, da slike najprej parametriziramo - opišemo z množico numeričnih parametrov. Ko imamo atributni opis slike, uporabimo poljuben algoritmom strojnega učenja, ki se zna učiti iz atributnega opisa. Na ta način lahko slike ravrščamo v razrede, jih klasificiramo, razpoznavamo objekte na slikah ali celo sestavljamo logični opis situacije na sliki. Uporabnost takega pristopa je odvisna predvsem od kvalitetnega atributnega opisa - torej od metode za parametrizacijo slik.

Pri snemanju prstov ljudi s Kirlianovo kamero je posneta korona okoli prsta lahko posebne oblike, če je človek v spremenjenem stanju zavesti (npr. meditira, se zdravi z bioenergijo ali s homeopatskimi pripravki, dela dihalne vaje, je pod vplivom droge ali alkohola, ima psihične težave itd.). Detekcija posebnih oblik koron, ki nakazujejo spremenjeno stanje zavesti, se je izvajala ročno s pomočjo strokovnjakov. Ruski razvijalci in izdelovalci Kirlianove kamere so želeli proces razpoznavanja spremenjenega stanja zavesti avtomatizirati. Iz posnetkov koron ljudi v normalnem stanju in v spremenjenem stanju zavesti so z algoritmi za parametrizacijo slik izpeljali numerične opise koron ter uporabili algoritme strojnega učenja za klasifikacijo. Dobljeni klasifikatorji so bili celo nekoliko boljši od strokovnjakov, zato so se odločili, da bodo razvito metodologijo vgradili v komercialni programski paket za obdelovanje, vizualizacijo in analizo posnetkov koron.

Pri analizi astronomskih fotografij so nedavno ročno pregledovali velike množice slik, da so razlikovali galaksije od zvezd in drugih svetlečih objektov (npr. satelitov), z namenom ustvariti in vzdrževati katalog nebesnih teles. Danes ni nič nenavadnega procesiranje ogromnih baz slik, ki obsegajo stotine TB pomnilnika ( $\text{TeraByte} = 10^{12}$ ). Sredi devetdesetih let so raziskovalci laboratorija Jet Propulsion (JPL) pri Univerzi Caltech v Kaliforniji razvili sistem SKICAT, ki uporablja algoritme za gradnjo odločitvenih dreves, s katerimi so avtomatizirali postopek klasifikacije objektov na astronomskih slikah [9]. Odločitvena drevesa so dosegla preko 90% točnost klasifikacije nebesnih objektov, kar je močno pohitrilo postopke za izdelavo katalogov. Pretežni del kataloga, ki obsega 50 milijonov galaksij in več kot dve milijardi zvezd, so izdelali avtomatsko.

## Napovedovanje strukture kemičnih spojin in proteinov

Odkrivanje boljših inačic aktivnih zdravil sestavlja večino intenzivnega raziskovalnega dela na področju farmacije, kjer aktivno deluje na stotine laboratorijev s kemiki, ki sintetizirajo in testirajo na tisoče spojin. Avtomatsko napovedovanje aktivnosti zdravila na osnovi njegove strukture, seznama sestavin in njihovih lastnosti bi bistveno pohitrlila in pocenila razvoj novih zdravil. Naravna predstavitev znanja na tem področju so logične relacije, kajti atributna predstavitev je preveč pomanjkljiva.

King in sod. [20] so uporabili sistem za induktivno logično programiranje GOLEM za modeliranje aktivnosti sestavnih delov zdravila na osnovi njihove kvantitativne strukture, znane aktivnosti posameznih sestavnih delov ter njihovih kemijskih lastnosti. Aktivnost spojine je realno število na določenem intervalu in se ponavadi modelira s standardnimi statističnimi metodami. GOLEM je dosegel nekoliko boljšo točnost, poleg tega pa, za razliko od statističnih metod, izpeljana pravila opisujejo kemijske zakonitosti in razkrivajo znanje, razumljivo strokovnjakom na tem področju.

Proteini so nizi aminokislin, ki imajo posebno tridimenzionalno obliko. Na vsakem mestu proteinske verige je lahko ena od dvajsetih aminokislin. Definicija niza predstavlja primarno strukturo proteina, medtem ko njegova tridimenzionalna oblika predstavlja sekundarno strukturo proteina. Napovedovanje sekundarne strukture iz primarne strukture je eden najtežjih problemov molekularne biologije. Ker gre za prostorske relacije in različno dolge verige proteinov, je atributna predstavitev neprimerna za ta problem, zato je naravno uporabiti predikatni račun prvega reda.

Muggleton in sod. [30] so uporabili sistem za induktivno logično programiranje GOLEM za napovedovanje sekundarne strukture proteinov iz podane primarne strukture in predznanja, ki opisuje fizikalne in kemijske lastnosti posameznih delov proteinske verige. Postopek učenja je bil iterativen, tako da so naučene relacije dodali kot predznanje novemu krogu učenja. Dosegli so točnost 80%, kar je bolje od najboljšega rezultata do takrat (77%), ki so ga dosegli z nevronskimi mrežami. Pravila, ki jih je izpeljal GOLEM, so razumljiva strokovnjakom, za razliko od netransparentnih napovedi nevronske mreže.

King in sod. [21] so razvili avtonomni robotski sistem Robot Scientist, ki je namenjen avtonomnemu izvajjanju poskusov na področju testiranja hipotez iz funkcijске genomike. Robot načrtuje in izvaja poskuse: s pipeto meša različne tekočine, meri rezultirajoče parametre in jih analizira. Rezultate eksperimenta uporabi sistem sa strojno učenje ASE-progol (naslednik sistema za induktivno logično programiranje Progol; ASE pomeni *Active Selection of Experiments*), ki inducira pravila za usmerjanje načrtovanja poskusov. To je ključni del celotnega sistema, saj je prostor vseh možnih poskusov prevelik in je nujno pametno omejevanje števila poskusov. Primerjava s strokovnjaki je pokazala primerljivo uspešnost.

## Igranje iger

*Viteštvu predstavlja dobro voljo viteza, ki se "igra z lastnim življenjem" v zavesti, da je tudi spopad na življenje in smrt le igra.*

*Alan Watts*

Pri igranju zahtevnih iger, kot sta šah in go, je prostor vseh možnih iger tako velik, da ga ni mogoče v celoti pregledati in izračunati optimalne strategije, tudi če bi imeli mnogo hitrejše računalnike od današnjih. Zato so vse strategije igranja hevristične in temeljijo na izkušnjah mojstrov igre. Prednost računalnikov pred človekom je v hitrosti pregledovanja drevesa igre do velikih globin, kar v nekaterih primerih zadošča, da računalnik doseže ali celo preseže kvaliteto igre najboljših človeških igralcev. Program/računalnik Deep Blue, ki je v šahu dosegel kvaliteto igre, ki je primerljiva s svetovnim prvakom, temelji na hitrosti računanja, hkrati pa uporablja ogromno bazo znanja obstoječih iger.

Pri ighah, kjer je možnih potez v eni poziciji nekaj sto (pri šahu jih je največ nekaj deset), je preiskovanje drevesa igre v globino tudi za najhitrejše računalnike brezupno. V takem primeru je nujno bodisi zakodirati dovolj predznanja o dobrni igri, ki ga lahko dobimo od ljudi strokovnjakov ali iz zgodovine preigranih iger v preteklosti (kar je za tako kompleksne igre skoraj nemogoče), bodisi uporabiti ustrezno strategijo učenja. Primer je uspeh programa TD-gammon v igri backgammon [41, 42]. Za to igro je značilen velik vejitveni faktor v vsakem vozlišču igre (nekaj sto možnih potez v povprečju), ter stohastičnost igre zaradi metanja kocke.

Program TD-gammon uporablja večnivojsko nevronsko mrežo s posebno strategijo spodbujevanega učenja za nastavljanje vrednosti uteži. Zaradi stohastične narave igre je spodbujevano učenje še posebej primerno, saj stohastičnost poskrbi, da se preišče ves preiskovalni prostor in da ne pride do prehitre konvergencije v določene dele prostora iger. Poleg kodiranja pozicije uporablja TD-gammon za ocenjevanje pozicije tudi koncepte, izpeljane iz znanja strokovnjakov. Program začne z naključno nastavljenimi utežmi nevronke mreže in igrat sam s seboj toliko časa, dokler se mreža še lahko kaj nauči. Program je bil razvit v več fazah, tako da so za vsako novo fazo, na osnovi izkušenj prejšnjih faz, načrtovali strukturo in velikost nevronke mreže, parametre učenja in količino učenja. V zadnji fazi je mreža vsebovala 80 nevronov na skritem nivoju in preigrala poldruži milijon iger sama s seboj, preden je učenje skonvergiralo v (lokalni) optimum. Naučena mreža je igrala na nivoju velemojstrov.

Mnogi velemojstri in nekdanji svetovni prvaki v backgammonu so priznali, da je na dolgi rok, zaradi človekove utrudljivosti in čustvene prisstranskoosti, najbrž TD-gammon v prednosti pred njimi. V specifičnih delih igre še vedno ne igra optimalno, kar bi se dalo popraviti z vnašanjem ekspertnega znanja v igro. Po drugi strani je TD-gammon razvil popolnoma nove strategije igre, ki prej niso bile znane in je porušil desetletna verovanja v nekatere hevristične poteze, za katere so mislili, da so optimalne. Do tega je uspel priti, ker se je učil samo iz lastnih izkušenj brez vnaprej podanega znanja o "dobrih strategijah".

## Literatura

- [1] I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, 2000. (3rd Edition).
- [2] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone. *Classification and Regression Trees*. Wadsforth International Group, 1984.
- [3] B. Cestnik, I. Kononenko, I. Bratko. Assistant 86: A knowledge elicitation tool for sophisticated users. V I.Bratko, N. Lavrač, (ur.), *Progress in Machine Learning*. Sigma Press, Wilmslow, England, 1987.
- [4] P. Clark, R. Boswell. Rule induction with CN2: recent improvements,. V Y. Kodratoff, (ur.), *Proc. European Working Session on Learning*, str. 151–163, Porto, March 1991, 1991. Springer Verlag.
- [5] P. Clark, T. Niblett. Induction in noisy domains. V I. Bratko, N. Lavrač, (ur.), *Progress in Machine learning*. Sigma Press, Wilmslow, England, 1987.
- [6] P. Clark, T. Niblett. Learning if then rules in noisy domains. V B. Phelps, (ur.), *Interactions in Artificial Intelligence and Statistical Methods*. Technical Press, Hampshire, England, 1987.
- [7] T. M. Cover, P. E. Hart. Nearest neighbor pattern classification. *IEEE Trans. on Information Theory*, IT-13:21–27, 1967.
- [8] T. G. Dietterich. Learning and inductive inference. V P. Cohen, E.A. Feigenbaum, (ur.), *The Handbook of Artificial Intelligence*, volume 3. Pitman Books Ltd, 1982.
- [9] U. Fayyad. Sky image cataloging and analysis tool. V *Proc. IJCAI-95*, str. 2067–2068, 1995. Montreal.
- [10] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- [11] I. J. Good. *Probability and the Weighing of Evidence*. Charles Griffin, London, 1950.
- [12] I. J. Good. *The Estimation of Probabilities – An Essay on Modern Bayesian Methods*. The MIT Press, Cambridge, 1964.
- [13] D. O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.
- [14] J. E. Hopcroft, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- [15] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *National Academy of Sciences*, 79:2554–2558, 1982.
- [16] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *National Academy of Sciences*, 81:4586–4590, 1984.

- [17] E. Hunt, J. Martin, P. Stone. *Experiments in Induction*. Academic Press, New York, 1966.
- [18] R. Jeffrey. *Formal Logic: Its Scope and Limits*. McGraw-Hill, 1981.
- [19] M. J. Kearns. *The Computational Complexity of Machine Learning*. The MIT Press, 1989.
- [20] R. King., S. Muggleton, R. Lewis, M. Sternberg. Drug design by machine learning: The use of ILP to model the structure-activity relationship of trimethoprim analogues binding to dihydrofolate reductase. *National Academy of Sciences*, 1992.
- [21] R. King., K. Whelan, F. Jones, P. Reiser, C. Bryant, S. Muggleton, D. Kell, S. Oliver. Functional genomic hypothesis generation and experimentation by a Robot Scientist. *Nature*, 427:247–252, 2004.
- [22] I. Kononenko, I. Bratko, E. Roškar. Experiments in automatic learning of medical diagnostic rules. V *Proc. International School for the Synthesis of Expert's Knowledge Workshop*, Bled, Slovenia, 1984.
- [23] N. Lavrač, S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
- [24] D. B. Lenat. The role of heuristics in learning by discovery: Three case studies. V R. Michalski, J.G. Carbonell, T.M. Mitchell, (ur.), *Machine Learning, An Artificial Intelligence Approach*. Tioga, 1983.
- [25] R. Michalski, J. G. Carbonell, T. M. Mitchell, (ur.). *Machine Learning, An Artificial Intelligence Approach*, volume 2. Morgan Kauffman, 1986.
- [26] R. S. Michalski, R. L. Chilausky. Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *Int. Journal of Policy Analysis and Information Systems*, 4:125–161, 1980.
- [27] D. Michie, R. A. Chambers. BOXES: An experiment in adaptive control. V E. Dale, D. Michie, (ur.), *Machine Intelligence 2*. Edinburgh: Oliver and Boyd, 1968.
- [28] D. Michie, R. Johnston. *The Creative Computer: Machine Intelligence and Human Knowledge*. New York: Viking, 1984.
- [29] M. Minsky, S. Papert. *Perceptrons*. Cambridge, MA: MIT Press, 1969.
- [30] S. Muggleton. *Inductive Logic Programming*. Academic Press, 1992.
- [31] N. Nilsson. *Learning Machines*. McGraw-Hill, 1965.
- [32] D. N. Osherson, M. Stob, S. Weinstein. *Systems That Learn*. Bradford Book, The MIT Press, 1986.

- [33] A. Paterson, T. Niblett. *The ACLS User Manual*. Intelligent Terminals Ltd, Glasgow, 1982.
- [34] J. R. Quinlan. Discovering rules from large collections of examples. V D. Michie, (ur.), *Expert Systems in the Microelectronic Age*. Edinburgh University Press, 1979.
- [35] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [36] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [37] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, Washington, DC, 1962.
- [38] D. E. Rumelhart, G. E. Hinton, R. J. Williams. Learning internal representations by error propagation. V D. E. Rumelhart, J. L. McClelland, (ur.), *Parallel Distributed Processing: Foundations*, volume 1. Cambridge: MIT Press, 1986.
- [39] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal*, 3(3), 1959.
- [40] E. Shapiro. Inductive inference of theories from facts. Research report 192, Dept. of Computer Sc., Yale University, New Haven, 1981.
- [41] G. Tesauro. Temporal difference learning of backgammon strategy. V *Int. Machine Learning Conf*, 7 1992. Aberdeen.
- [42] G. Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3), 1995.
- [43] T. Urbančič. *Avtomatizirana sinteza znanja za vodenje sistemov*. Doktorska disertacija, Univerza v Ljubljani, Fakulteta za elektrotehniko in računalništvo, Ljubljana, 1994.
- [44] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

## Poglavlje 2

# Učenje in inteligenco

*Razumeti sebe je kot poskusiti ugrizniti svoje zobe.*

*Alan Watts*

V tem poglavju so poleg definicije učenja in povezanosti z inteligenco nekoliko podrobneje opisani tudi principi učenja v bioloških sistemih. Na koncu so razložene potrebe za razvijanje algoritmov strojnega učenja.

### 2.1 Kaj je učenje

*Učenje določa adaptivne spremembe v sistemu, ki mu omogočajo, da naslednjič reši naloge iste vrste bolj učinkovito.*

*Herbert A. Simon*

Najprej je potrebno definirati učenje in njegovo relacijo z inteligenco, nato nas zanimajo tudi možnosti strojnega učenja in umetne inteligence.

#### Definicija učenja

Pojem *učenje* lahko definiramo z naslednjo splošno situacijo: imamo sistem – učenca, ki mora (želi) opravljati določeno nalogu. Sprva naloga izvršuje slabo ali je sploh ne zna opravljati, sčasoma pa jo z vajo, posnemanjem učitelja ali s poskušanjem in napakami opravlja bolje. Pri tem pojmu *bolje* opravljati naloga lahko pomeni hitreje, natančneje, ceneje ipd. Vaji, posnemanju učitelja in poskušanju z napakami pravimo *učenje*.

Učenec se je naučil opravljati nalogu, če bo isto nalogu lahko naslednjič opravljal enako dobro in se mu ne bo potrebno ponovno učiti. Zato, da lahko opravlja isto nalogu enako dobro, se je sistem – učenec spremenil. Spremembam, ki so rezultat učenja, pravimo pridobivanje *znanja*. Znanje omogoča učencu, da opravlja isto nalogu bolje kot pred učenjem.

Znanje definiramo kot interpretacijo informacije, ki jo nosijo podatki. Znanje je bodisi dano vnaprej (npr. podedovano ali v primeru stroja vkodirano) ali pa je rezultat učenja. Znanje je pravilno, napačno, lahko je pravilno, a neuporabno, pomanjkljivo itd. Vsak podatek je ob definirani interpretaciji znanje. V praksi je zanimivo samo koristno znanje, t.j. znanje, ki omogoča sistemu uspešnejše reševanje nalog iz dane domene.

Učenje srečamo pri skoraj vseh živih bitjih, najbolj pa pride do izraza pri človeku. Učenju živih sistemov pravimo *naravno učenje*. Če pa je učenec stroj – računalnik, pravimo učenju *strojno učenje (machine learning)*. Namens razvoja metod strojnega učenja je poleg razumevanja naravnega učenja in inteligence tudi reševanje problemov s specifičnim znanjem. Tako znanje ponavadi ni dosegljivo ali pa ga poseduje samo omejen krog strokovnjakov. S pomočjo strojnega učenja pod določenimi pogoji učinkovito tvorimo tako znanje, ki ga lahko uporabimo za reševanje problemov.

Celó evolucijo lahko enačimo z učenjem, saj z genetskim križanjem, mutacijo in naravno selekcijo ustvarja sposobnejše sisteme. Za nas bo sicer zanimivejše učenje posameznega osebka, vendar lahko princip evolucije uporabimo tudi pri strojnem učenju s t.i. *genetskimi algoritmi* (glej razdelek 4.4 in poglavje 12).

Strojnega učenja ne smemo zamenjati s *programiranim učenjem*. Programirano učenje je pedagoška metoda, kjer se učenec uči snovi po vnaprej določenem vrstnem redu. Pri tem se njegovo znanje sproti preverja in se poučevanje nadaljuje šele, ko učenec osvoji določeno znanje. Največkrat pri programiranem učenju uporablajo tudi računalnik. Učenju, kjer se računalnik uporablja kot pripomoček pri učnem procesu, pravijo tudi *računalniško podprt učenje*.

## Umetna inteligencia

*Če izločimo zavest, potem se odločanje spremeni v matematično napovedljivo dejavnost, slovečo na informacijah, ki so bile predhodno na voljo.*

*Mortimer Taube*

Pojmi učenje, znanje in inteligencia so tesno povezani. Čeprav ne obstaja splošno veljavna definicija inteligence, jo lahko na grobo opredelimo kot *sposobnost prilaganja okolju in reševanja problemov*. Že v sami definiciji nastopa učenje – prilaganje. Za reševanje problemov je nujno znanje in njegova uporaba.

Dolgoročni cilj raziskav strojnega učenja, ki je (zaenkrat) nedosegljiv, je ustvariti umetni sistem, ki bi z učenjem dosegel ali presegel človekovo inteligenco. Širše področje raziskav, ki ima isti končni cilj, se imenuje *umetna inteligencia (artificial intelligence)*. Raziskave metod umetne inteligence se ukvarjajo z razvojem sistemov, ki se obnašajo inteligentno in ki so sposobni reševati relativno težke probleme. Pogosto te metode temeljijo na oponašanju človekovega načina reševanja problemov. Umetna inteligencia se poleg strojnega učenja ukvarja še z računalniškim zaznavanjem, predstavljivo znanja, razumevanjem naravnega jezika, avtomatskim sklepanjem in dokazovanjem izrekov, logičnim programiranjem, kvalitativnim modeliranjem, razvojem ekspertrih sistemov, igranjem iger, hevrističnim reševanjem problemov, robotiko in kognitivnim modeliranjem.

Pri raziskavah umetne inteligence imajo algoritmi učenja vse pomembnejšo vlogo, saj je potrebno vključevati učenje praktično povsod. S strojnim učenjem se lahko sistemi izpopolnjujejo v računalniškem zaznavanju, razumevanju naravnega jezika, avtomatskem sklepanju in dokazovanju izrekov, hevrističnem reševanju problemov in igranju iger. Na področju logičnega programiranja nastajajo algoritmi za avtomatsko generiranje logičnih programov iz primerov ciljne relacije. Pri raziskavah kvalitativnega modeliranja se metode strojnega učenja uporabljajo za generiranje opisov zapletenih modelov iz primerov obnašanja obravnavanih sistemov. Pri razvoju eksperimentnih sistemov lahko avtomatsko zgradimo bazo znanja iz primerov rešenih problemov. Inteligentni roboti z učenjem izpopolnjujejo svoje postopke reševanja problemov. Današnje kognitivno modeliranje ni mogoče brez upoštevanja algoritmov učenja.

## 2.2 Naravno učenje

*Učenje opredeljujemo kot napredok dejavnosti po izkušnji.*

*Anton Trstenjak*

Človek se uči celo življenje:

- novorojenček se uči gledati, poslušati, razločevati glas in obraz mame in očeta od ostalih ljudi;
- dojenček se uči pomena besed, povezave med vidom in tipom, raznih motoričnih veščin, kot je prijemanje predmetov, plazenje, sedenje;
- otrok se uči hoditi, govoriti besede in pozneje stavke, voziti kolo, plavati;
- učenec se uči brati, pisati, računati, abstraktno razmišljati, logično sklepati, razumeti in govoriti tuj jezik, uriti se v pomnjenju;
- dijak pridobiva novo bolj ali manj splošno opisno znanje, izpopolnjuje se v logičnem in abstraktnem mišljenju, nauči se hevristik za reševanje problemov;
- študent si z učenjem prestrukturira svoje znanje, pridobiva specializirano opisno znanje, nauči se specializiranih hevristik za reševanje določenih vrst problemov, nauči se metod raziskovanja in preizkušanja hipotez;
- na delovnem mestu se človek uči svojega poklica, z delom in izkušnjami izpopolnjuje in razširja svoje znanje.

Učimo se praktično vsak dan, kar hkrati pomeni, da se naše znanje neprestano spreminja, razširja in dopolnjuje. Poleg ljudi se učijo tudi živali. Učljivost je odvisna od razvojne stopnje. V laboratorijih so uspeli učiti celo črve [15, 18].

Z raziskovanjem in razlago naravnega učenja se ukvarjata *psihologija učenja* in *pedagoška psihologija* (npr. [1, 15, 16, 24, 26]). Prva raziskuje in razlaga predvsem principe in zmožnosti učenja, druga pa metode človeškega učenja in poučevanja z namenom izboljšati

učinke učenja. Za nas bo zanimiva predvsem psihologija učenja. Pedagoška psihologija proučuje pozornost in utrujenost pri učenju, pozabljanje, ocenjevanje učencev, odnos učitelja do učenca, motivacijo za učenje ipd. Ti faktorji so zelo pomembni za človeško učenje, manj pomembni pa za strojno učenje.

## Prirojeno in naučeno

*Ego je avtomatični mehanizem, ki ga družba vsadi že v otroštvu, pri čemer je - mogoče - prisoten tudi kanček dednosti.*

*Alan Watts*

Eno osnovnih vprašanj, ki si ga zastavlja psihologija učenja, je razločevanje med prirojenim in naučenim znanjem. To vprašanje je pomembno tudi za strojno učenje: s kakšnim predznanjem (programom) naj se sistem začne učiti oziroma kakšno znanje potrebuje, da se lahko uči.

Marsikatera veščina pri živalih je prirojena in ne naučena. Npr. poskusi s piščanci so pokazali, da je pobiranje zrn prirojeno. Uspešnost pri pobiranju zrn je odvisna samo od starosti piščanca in ne od vaje [24]. Starost je pomembna, ker se s časom organizem razvije v svojo končno (odraslo) podobo, na katero učenje nima vpliva. Temu procesu pravijo *zorenje*. Za določene vrste učenja obstajajo kritična obdobja, ko je učenje možno in/ali nujno za nadaljnji razvoj organizma. Tako npr. podgane, ki so odrasčale v okolju, v katerem se niso mogle naučiti prijemanja in porivanja predmetov, ne gradijo gnezda kljub temu, da so breje in imajo gradbeni material na voljo.

Prirojene veščine so *nagoni*. Tudi pri človeku igrajo pomembno vlogo. Otrok pride na svet s prirojenimi nagnjenji in se uči sam od sebe [25]. Zorenje, ki je prav tako prirojeno, je nujno potrebno za določene oblike človeškega učenja. Učenje je pri človeku odločilnega pomena za njegov obstoj. Čim višje na razvojni lestvici se nahaja živalska vrsta, tem pomembnejšo vlogo v njenem življenju iga učenje. Čim višji je končni nivo učne zmožnosti, ki ga vrsta doseže, tem počasnejše je učenje v otroštvu [1].

Za zaznavno učenje potrebujejo primati več časa kot enostavnejše vrste. Človek ima ob rojstvu prirojene nekatere možnosti razpoznavanja, npr. razlikovanje obrazov od drugih predmetov. Večine zaznavanja se mora šele naučiti: gledati, poslušati itd. Ko so od rojstva slepi po operaciji spregledali, jih je vid dejansko motil in ga niso znali uporabljati. Šele z učenjem so začeli razpoznavati like, obraze itd. [24]. Vid so uporabljali celo slabše kot novorojenčki, kar lahko razložimo z izgubo prirojenih lastnosti med odrasčanjem.

Živo bitje ima določene prirojene sposobnosti, ki odločajo o njegovi sposobnosti za učenje, torej tudi o njegovi inteligenci. Npr. deževnik ima 13 nevronov, čebela pa okoli 900 nevronov, kar ji zadošča, da si zapomni relativno dolgo pot od panja do paše. Čim enostavnejša je naloga, tem večja je verjetnost, da se je bo žival naučila enako hitro kot človek. Težje naloge lahko rešijo samo najbolj razvite živali. Gorilo kot predstavnico ene najbolj razvitih živalskih vrst so naučili razumeti več kot tisoč besed, kar je približno enako besednjaku, ki ga uporablja povprečen Američan [18]. Fizikalne danosti za človeka (lahko bi rekli "strojna oprema") so strnjene v tabeli 2.1. Spoštovanja vredna strojna oprema napeljuje na dialektično razmišljanje [17], da kvantiteta povzroči kvalitativni preskok v zmožnostih inteligenčnega ob-

število nevronov	$10^{11}$
št. nevronov v možganski skorji	$2 \times 10^{10}$
št. poslanih impulzov na nevron	10 impulzov/sek
hitrost pretoka inf. po kanalih	30 bit/sek
št. kemijskih reakcij v možganih	$10^5 - 10^6$ /sek
število povezav / nevron	$10^4$
vseh sinaps	$10^{14} - 10^{15}$
število bolečinskih točk na koži	$1.2 \times 10^6$
št. živčnih vlaken pri očesnem živcu	$1.2 \times 10^6$
kratkoročni spomin (zavest)	7 kazalcev (naslovov)
dolgoročni spomin – samo sinapse	$> 10^{15}$ bitov
dostop do dolgoročnega spomina	2 sek
optimalna zunanja temperatura	18°C
število aminokislin	20
dolžina verige aminokislin	več tisoč
št. možnih različnih proteinov	$> 20^{1000}$
genski zapis	$> 10^9$ bitov = 1G bit
št. proizvedenih proteinskih molekul	15000/sek/nevron

Tabela 2.1: Fizikalne zmožnosti človeških možganov.

našanja.

## Spomin

*Dokazi iz številnih virov kažejo, da si utegnejo možgani zapomniti prav vsako izkušnjo.*

*Peter Russell*

Eden osnovnih pogojev za uspešno učenje je sposobnost pomnenja. Da je spomin osnovni pogoj učenja, nazorno ilustrira primer pacientov, ki zgubijo zmožnost pomnenja [19]. Ne zapomnijo si niti novice, ki so jo slišali pred nekaj minutami in ko isto stvar ponovno slišijo, je to zanje nova informacija. Ti ljudje se obnašajo nespremenjeno do konca življenja. Njihovo znanje se ne spreminja, čas se zanje dejansko ustavi. Tudi čež deset let se vedejo enako in so prepričani, da so deset let mlajši, kot dejansko so. Vsakič znova se začudijo, ko vidijo, da se je njihova okolica spremenila in njihovi kolegi postarali. Njihove reakcije na spremembe so znova in znova enake. Podobni so programom, ki na iste vhode nenehno dajejo enake odgovore.

Možgani so sestavljeni iz velikega števila nevronov (glej podatke za človeške možgane v tabeli 2.1). Nevrone so med seboj povezani s sinapsami, preko katerih pošiljajo impulze. Poleg sprejemanja ter pošiljanja velikega števila impulzov drugim nevronom, so v aktivnost nevrona vključene dejavnosti, za katere obstaja precej nevrofizioloških potrditev, da so osnova spomina. Te dejavnosti so [18] ustvarjanje novih povezav z drugimi nevroni, spremenjanje jakosti povezav na sinapsah in proizvodnja proteinov.

**Ustvarjanje novih povezav z drugimi nevroni** Strukturne spremembe v možganih se dogajajo celo življenje. Z dodajanjem novih povezav in odmiranjem obstoječih se spreminja relacije med posameznimi nevroni, kar spreminja delovanje možganov.

**Spreminjanje jakosti povezav na sinapsah** Jakost povezave na sinapsah vpliva na prepustnost sinapse za impulze, ki jih pošilja en nevron drugemu. S spremjanjem jakosti povezav se spreminja frekvenca impulzov, ki jih sprejema drugi nevron od prvega, kar vpliva na aktivnost drugega nevrona.

Strukturne spremembe ter spremembe jakosti povezav na sinapsah so tudi osnovni princip učenja v algoritmih *umetnih nevronske mreže*, ki so opisane v 9. poglavju. Če bi bili biološki nevroni preprosti elementi (kot so ponavadi umetni nevroni), ki znajo seštevati impulze na vhodu ter pod pogojem zadosti velike vsote (prag) poslati impulz naprej, in če bi uporabljali za pomnjenje samo spremjanje prepustnosti sinaps, imajo naši možgani zadosti kapacitete, da si zapomnijo vse, kar se nam pripeti v življenju [6].

**Proizvodnja proteinov** Neuron proizvede v eni sekundi 15.000 beljakovinskih molekul [10]. Pri tem procesu ima pomembno vlogo ribonukleinska kislina – RNA, ki posnema genetsko kodo dezoksiribonukleinske kislino – DNA. Na ta način nastajajo raznovrstni proteini. Kombinatorika različnih aminokislín omogoča praktično neomejeno število različnih proteinov (glej tabelo 2.1). S poskusi so pokazali, da lahko prenašajo spomin ene živali na drugo žival iste vrste (npr. spomin ene podgane na drugo podgano) ali celo na žival druge vrste (npr. spomin podgane na ribo) tako, da "trenirani" živali iz možganov izločijo proteine in jih vbrizgajo v možgane druge živali [18, 15]. Žival, ki je sprejela proteine, se brez učenja "nauči" istega obnašanja kot "trenirana" žival.

Možnost prenosa spomina med živalmi različnih vrst napeljuje na sklep, da je koda spomina v proteinih univerzalna. Prav tako je vse več potrditev, da je genetska koda v DNA univerzalna, t.j. ista pri vseh živih bitjih [20], obstajajo pa majhna odstopanja, ki jim pravijo dialekti genetske kode.

Lahko bi rekli, da so možnosti človeškega spomina neomejene. Vse več raziskovalcev se strinja, da si (lahko) človek zapomni prav vse, kar se mu pripeti v življenju. Problematično je edino "naslavljjanje spomina", t.j. priklic zapomnjenih podatkov. Znano je, da se lahko ljudje v hipnozi spomnijo podrobnosti dogodkov, ki so se jim pripetili pred davnimi leti in celo v otroštvu. Ljudje s fotografskim spominom si lahko zapomnijo sliko s fotografsko natančnostjo. Npr. nekateri otroci si zapomnijo sliko z različnimi napisimi, še preden se naučijo brati. Pozneje, ko se naučijo brati, lahko iz spomina preberejo napise na slikah.

Ekstremen primer je prav gotovo ruski časnikar Solomon Šereševski, ki so ga poimenovali gospod "S" [18]. Njegova največja težava je bila pozabiti. Zapomnil si je prav vse, kar se mu je pripetilo v življenju. Če ste ga vprašali, kaj je delal npr. 13. septembra pred petimi leti, se je najprej malo zamislil, nato pa vprašal: "Ob kateri uri?" V nekem poskusu je prebral in si zapomnil več kot sto nesmiselnih zlogov. Po osmih letih, ko so ga nenadoma prosili, naj ponovi isti seznam zlogov, ga je ponovil brez napake.

Največjo težavo pri spominjanju imajo ljudje s prikljicom konkretnega podatka, ki ga ne morejo ali ne znajo povezati z drugimi znanimi podatki. Torej manjka naslov. Ker je

naslov v možganih asociacijah, pravimo, da so možgani asociativni pomnilnik in so *vsebinsko* naslovljeni. Zato so tudi sposobnejši prepoznavanja znanih pojmov, slik, melodij itd. kot pa rekonstrukcije, t.j. priklica iz spomina. Kot možgani so tudi nekatere oblike umetnih nevronske mreže vsebinsko naslovljive (glej 9. poglavje).

Ena od lastnosti spomina je *racionalizacija*. Najhitreje pozabimo podrobnosti pojma, slike ali dogodka, medtem ko osnovne obrise in ideje ohranimo. Taka *racionalizacija* je podlaga *generalizaciji*, ko spomin z isto strukturo in idejo zajame vrsto podobnih situacij, ki se razlikujejo v podrobnostih. Podobno se učijo umetne nevronske mreže.

### Vrste naravnega učenja

*Pomnenje je nekaj drugega kot učenje.*

*Peter Russell*

Mnogi raziskovalci so poskušali definirati vrste naravnega učenja, hkrati pa so iskali osnovno vrsto učenja, iz katerega izhajajo vse oblike učenja. Najpogosteje klasifikacije učenja so glede na kompleksnost učenja, glede na gradivo, ki se ga učenec uči, in glede na strategijo učenja.

Po kompleksnosti učnega procesa ločimo [15]:

**Vtiskavanje:** najbolj preprosta oblika učenja, ko se učencu vtiska v spomin določeno znanje, ki ga pozneje več ne spreminja. Tako se učijo npr. komaj zvaljene račke, ki sledijo svoji mami – raci, dokler ne odrastejo. Če namesto race zagledajo npr. človeka, mu bodo prav tako sledile.

**Pogojevanje ali asociiranje:** za osnovno obliko učenja večina psihologov šteje *pogojevanje*. Znani so poskusi ruskega psihologa Pavlova, ki je s pogojevanjem dosegel, da so psi izločali slino ob določenem zвуку. To je dosegel tako, da je vsakič, ko je zazavonil zvonec, pes dobil hrano, kar je povzročilo izločanje sline. Sčasoma je pes začel izločati slino tudi ob zvoku zvonca brez pogleda na hrano. S pogojevanjem se razvije *pogojni refleks* za razliko od prirojenega brezpogojnega refleksa. Rezultat pogojevanja so *asociacije*, ki jih ponavadi definirajo kot mehanične povezave med senzornimi vtiški in njihovimi reakcijami ali kot pomenske zvezce, kar je odvisno od narave dražljaja in situacije, v kateri asociacije nastajajo.

**Verjetnostno učenje:** učenec ima nalogu izbrati pravilno alternativo, pri čemer je generiranje pravilnih alternativ stohastičen proces. Na ta način učenec samo z določeno verjetnostjo izbere pravilno alternativo. V poskusih se je izkazalo, da je v primeru 70% verjetnosti ene alternative in 30% verjetnosti druge alternative, tudi učenec v 70% primerov izbral prvo alternativo. Tako je dosegel v povprečju  $0.7^2 + 0.3^2 = 58\%$  pravilnih izbir (čeprav bi dosegel 70% pravilnih odgovorov tako, da bi vedno izbral prvo alternativo).

**Zapominjanje:** zapominjanje je relativno preprosto učenje, čeprav lahko ljudem zapominjanje velikega števila podatkov dela težave. Preprosto zapominjanje brez razumevanja

pomena ne zahteva dodatnih miselnih procesov in je zato, ker se med učenjem ne vzpostavijo asociacije z znanimi pojmi, tako učenje lahko težavno (za razliko od strojnega zapominjanja, ki je trivialno).

**Učenje s poskusi in napakami:** učenec mora iz začetnega stanja z izbiro ustreznega zaporedja alternativ (akcij, smeri, gibov, odločitev) priti do ciljnega stanja. Proses učenja je na začetku bolj ali manj naključno preiskušanje alternativ ter zapominjanje pravilnih in napačnih odločitev. Z vajo se učenec nauči izbirati bolj perspektivne alternative. Tipičen primer je iskanje izhoda iz labirinta (npr. podagana se po večkratnem poskušanju nauči priti iz labirinta brez napake). Pri tem učenju ima odločilno vlogo spomin, pri zahtevnejših nalogah pa tudi logično sklepanje. Nekoliko drugačno je učenje motoričnih veščin, npr. vožnja kolesa. Tudi tu se učenec uči s poskušanjem in napakami, vendar bi pri tem težko govorili o logičnem sklepanju, čeprav je naučeno reagiranje (refleks) tudi neke vrste (nezavedna) izbira alternative oziroma logični sklep.

**Posnemanje:** gre za reševanje problemov tako, da najprej opazujemo drugega, ki rešuje isto ali sorodno nalogu, in ga zatem posnemamo. S poskusi so pokazali, da se psi in mačke hitreje naučijo skakati čez ovire ali pritiskati na vzzvod, če so pred tem opazovali druge živali pri reševanju istih nalog. Pri zahtevnejših nalogah jim opazovanje ni pomagalo, medtem ko so bolj razvite (inteligentne) živali (npr. opice) tudi pri zahtevnejših nalogah hitreje napredovale z opazovanjem in posnemanjem. Pri zahtevnejšem posnemanju mora učenec razumeti situacijo in vzročno sklepati, kar pride najbolj do izraza pri človeku.

**Učenje z razumevanjem in vpogledom:** je najzahtevnejša oblika učenja, ki vključuje procese zapominjanja, abstraktnega (simboličnega) mišljenja, logičnega sklepanja in vzročnega povezovanja, ki vodi k razumevanju problema. Do vpogleda pride naenkrat, ko učenec odkrije rešitev problema, tako da integrira odnose v dani problemski situaciji. Takega učenja so le v omejenem obsegu sposobne višje razvite živali (npr. opice), daleč najspodbnejši pa je človek.

Glede na gradivo učenja psihologi ločijo naslednje vrste učenja [15]:

- senzorno učenje,
- motorično učenje in
- besedno (semantično) učenje.

Pri vseh oblikah učenja se pojavlja *spontana generalizacija*. Pri senzornem učenju se naučeno generalizira na podobne dražljaje. Če se npr. naučimo reagirati na zvok določene frekvence, potem bomo reagirali na vse podobne zvoke in sicer tem močnejše, čim bližje sta si frekvenci novega in naučenega zvoka. Pri motoričnem učenju se naučeno z določenega dela telesa generalizira na ostale dele telesa, npr. reakcije, naučene z desno roko, se prenesejo na levo roko in v manjši meri celo na noge. Pri semantičnem učenju se naučena reakcija na dane besede prenese na besede, ki so podobne (fonetično) ali imajo soroden pomen (sinonimi). Spet je prenos naučenega tem močnejši, čim večja je podobnost besed.

S poskusi so dokazali tudi obraten proces – *diferenciacijo (specializacijo)*. Živali so s po gojevanjem naučili razlikovati podobne dražljaje, ki jih prvotna generalizacija ni razlikovala. Na ta način se reakcije specializirajo na določene vrste dražljajev. Če so dražljaji preveč podobni (npr. premajhna razlika v frekvencah zvoka, barvi, velikosti, oblik, času itd.), je učenje neuspešno tako pri živalih kot pri ljudeh.

Posebnega pomena je *učenje pojmov*. Pojmi so razredi ali kategorije predmetov, dogodkov ali samih pojmov (v splošnem izkušenj), ki jih iz določenih razlogov enako obravnavamo oziroma imajo neko skupno svojstvo [1]. Pojmi se razlikujejo po konkretnosti/abstraktnosti in po zapletenosti (npr. konjunktivni in disjunktivni pojmi). Bolj konkretni (manj abstraktni) in bolj preprosti pojmi so lažje naučljivi. Znano je, da se človek lažje nauči konjunktivnega kot disjunktivnega pojma (ta ureditev po težavnosti velja tudi za strojno učenje). Vsakemu pojmu ponavadi priredimo tudi ime.

Večina učenja pri otrocih sestoji iz pridobivanja že obstoječih pojmov, tako da jim učitelj podaja (pozitivne) primere in protiprimerne (negativne primere) ter delni opis pojma. Učenje iz primerov je tudi osnovna oblika strojnega učenja. Pravimo mu *induktivno* ali *empirično učenje*. Le redko mora učenec tvoriti nove pojme, kar je težavnejša naloga. Tudi pri strojnem učenju se včasih pojavi potreba po generiranju novega pojma. Takemu učenju pravimo *konstruktivno učenje*.

Pri učenju pojmov uporabljam ljudje različne strategije [1]:

**Iskanje v širino:** pri tej strategiji človek sočasno preiskuša vse konsistentne hipoteze, t.j. opise pojma, ki ustrezajo vsem primerom in ne ustrezajo nobenemu protiprimeru. Taka strategija v polni meri izkoristi vso informacijo, ki je na voljo. Srečamo jo tudi pri strojnem učenju, npr. Model Inference System (MIS) v induktivnem logičnem programiraju [21]. Strategija vodi do optimalne hipoteze pri minimalnem številu učnih primerov in protiprimerov. Za to strategijo je značilno, da je za ljudi težavna in naporna, za stroje pa kompleksna (neučinkovita), ker prostor preiskovanja kombinatorično narašča.

**Iskanje v globino:** pri tej strategiji človek zaporedno postavlja hipoteze in jih preiskuša in spreminja, dokler je to možno, potem pa se vrne na izhodišče in postavi novo hipotezo. Tako učenje je za ljudi lahko, vendar ne vodi nujno k optimalni hipotezi in ponavadi zahteva več učnih primerov. Podobne ugotovitve veljajo za to strategijo pri algoritmih.

**Konzervativno osredotočenje:** hipoteza postane prvi (pozitivni) učni primer. Zatem učenec spreminja hipotezo spreminja tako, da sistematično spreminja eno spremenljivko, dokler ne dobi konsistentne hipoteze. Tudi to strategijo srečamo pri strojnem učenju.

**Osredotočeno ugibanje:** hipoteza postane prvi (pozitivni) učni primer. Zatem učenec spreminja več spremenljivk naenkrat. Taka strategija spominja na stohastične algoritme strojnega učenja (glej razdelek 4.4), kjer algoritem začne z naključno izbrano hipotezo in zatem stohastično spreminja več spremenljivk z namenom izbojšati trenutno hipotezo.

Zanimivo je, da gornje strategije ljudje uporabljamo zavedno ali nezavedno. Pri nezavednem učenju se pogosto zgodi, da človek, ki se je naučil razlikovati primere in protiprimerne nekega pojma, ne zna opisati naučenega pojma niti ne zna obrazložiti poteka razločevanja primerov

od protiprimerov. Ta fenomen srečamo pri strokovnjakih na danem področju, ki le s težavo opišejo znanje, ki ga uporabljajo pri svojem delu. To je problematično pri sestavljanju baz znanja za ekspertne sisteme. Včasih se temu izognemo tako, da znanje avtomatsko zgeneriramo iz primerov z algoritmi za strojno učenje.

### 2.3 Učenje, inteligencia, zavest

*Identičnost med človekom in strojem se ne doseže s tem, da prenesemo človeška svojstva na stroj, marveč s tem, da prenesemo mehanične omejitve na človeka.*

Mortimer Taube

Kot smo že napisali, se inteligencia opredeljuje kot *sposobnost prilagajanja okolju in reševanja problemov*. Danes se večina raziskovalcev strinja, da ni inteligence brez učenja. Samo učenje pa ne zadošča. Da se lahko sistem uči, mora imeti določene sposobnosti, npr. zadostne spominske sposobnosti (pomnilnik), sposobnost sklepanja (procesorske sposobnosti), zaznavne sposobnosti (vhod in izhod) itd. Tudi te sposobnosti same po sebi ne zadoščajo, če niso ustrezno povezane in ne vsebujejo ustreznih algoritmov za učenje. Poleg tega učenje potrebuje neko začetno znanje – predznanje (*background knowledge*), ki je v živih sistemih podedovan. Z učenjem se sposobnost sistema povečuje, torej se povečuje tudi njegova inteligencia. Poenostavljeno lahko zapišemo:

$$\text{i Inteligencia} = \text{strojna oprema} + \text{predznanje} + \text{učenje} + ?.$$

O tem, ali strojna oprema, predznanje in učenje zadoščajo za (umetno) inteligenco, se mnenja razhajajo. Na eni strani so zagovorniki naravne inteligence kot edine možne [2, 23], na drugi strani pa zagovorniki t.i. *močne teze o umetni inteligenci*, ki zagovarjajo možnost ustvariti inteligenten stroj [22].

#### Količina inteligence

*Znanje je pomembno. Toda še veliko pomembnejša je njegova koristna uporaba. Ta je odvisna od srca in uma človeka.*

Dalaj Lama

Sistemov po intelligentnosti ne moremo strogo urediti, saj je potrebno upoštevati različne tipe inteligence (sposobnosti): numerična, besedna in slikovna, prostorska, motorična, spominska, perceptivna, induktivna, deduktivna itd.. V zadnjem času se poudarjata tudi čustvena in duhovna inteligencia [4]. Nekateri avtorji navajajo preko 100 različnih tipov inteligence pri človeku. Nek sistem (človek ali stroj) je lahko boljši v nekaterih tipih inteligence in slabši v ostalih. V okviru umetne inteligence se pričakuje od intelligentnega sistema, da ne bo ekstremno dober samo v enem vidiku, npr. hitrosti in količini spomina, hitrosti računanja ali hitrosti preiskovanja prostora ali v (skoraj optimalnem) igranju iger (kar današnji računačniki zmorejo), temveč da bo (vsaj malo) intelligenten na vseh področjih, ki so značilna za človeka. Zdi

Mentalna vsebina	zavedanje sebe	
	DA	NE
DA	budno stanje	sanjanje
NE	meditacija	spanje

Tabela 2.2: Stanja Zavesti.

se, da je potrebna povezava posameznih tipov inteligenc v smiselno celoto (nadzorni sistem), ki zna preklapljati med različnimi vrstami inteligence. Večina debat o umetni inteligenci pa ne upošteva dovolj še naslednje stopnje, zavesti.

## Zavest

*Če odvzameš vse misli, ostane čista zavest.*

*Ramana Maharshi*

Zavedanje samega sebe, razlikovanje sebe od drugih, zavedanje svojih težav, svojih nalog in svojih (etičnih in moralnih) odgovornosti so gotovo povezani z zavestjo, kaj pa je zavest sama po sebi, je bistveno težje opredeliti. S preučevanjem različnih vidikov zavesti se danes ukvarjajo znanstveniki z mnogih področij: psihologi, psihiatri, nevrofiziologi, fiziki, biologi, biokemiki, računalnikarji, filozofi itd. Na konferenci o zavesti v Tucsonu v Arizoni (*Towards a Science of Consciousness*) vsako leto sodeluje več sto znanstvenikov, ki so v svoje vrste pritegnili tudi spiritualiste in pripadnike različnih duhovnih učenj. Vsak poskuša prikazati izkušnje, povezane s preučevanjem zavesti, iz svojega zornega kota. Splošni vtis je, da nihče nima jasne definicije, kaj zavest sploh je.

Nekateri kvantni fiziki povezujejo zavest s kolapsom valovne funkcije, kar bi lahko razrešilo množe dileme in pojasnilo množe sicer nerazložljive pojave, kot sta telepatija in telekinetsa (glej tudi [7, 8]). Mnogi raziskovalci zagovarjajo materialistično razlago zavesti, češ da je zavest rezultat kompleksnosti sistema in da je nastala v nekem časovnem trenutku evolucije. V zadnjem času je vse več mnenj o subjektivnosti zavesti, ki je zaradi tega nemerljiv in s tem tudi neopisljiv fenomen [27, 11, 9].

Pri ljudeh ločimo več stanj zavesti, eno možno razdelitev prikazuje tabela 2.2. Pri tem je treba poudariti, da meje med budnim stanjem, sanjanjem in spanjem niso ostre, saj se v spanju lahko človek tudi zaveda sam sebe (t.i. lucidno sanjanje). Ali obstaja spanje brez mentalne vsebine, pa pravzaprav ne vemo, saj če se mentalnih vsebin ne spomnimo, še ne pomeni, da jih ni bilo.

Delovanje človekove zavesti lahko razdelimo na več nivojev: čista zavest (brez mentalne vsebine, kar ustreza meditaciji v zgornji tabeli), nadzavest ali spremenjeno stanje zavesti (ki ustreza posebnim sposobnostim, kot so jasnovidnost, telepatija itd.), običajna zavest (budno stanje, mentalna vsebina je odvisna od pozornosti), podzavest (ustreza vsem procesom v zavesti, ki se jih ne zavedamo, v principu pa se jih lahko zavemo, če tja usmerimo pozornost) in nezavest (kar ustreza najverjetnejše samo mrtvemu telesu/organizmu).

## Meje simbolične izračunljivosti, izpeljivosti, opisljivosti

*Resnice se ne da poučevati.*

*Osho*

Teorija izračunljivosti [5] opozarja, da je skoraj zanemarljiv delež problemov, ki si jih lahko formalno zastavimo, rešljiv algoritmično. Turingovih strojev (algoritmov) je števno neskončno mnogo, problemov pa je neštevno neskončno mnogo, kar ustreza moči potenčne množice prejšnje veličine. Prva veličina je enaka moči množice naravnih števil (diskretni svet), druga pa moči množice realnih števil (zvezni svet). Danes uporablja znanost naslednje formalne simbolične jezike za opisovanje (modeliranje) realnosti:

- matematična logika,
- računalniški programski jeziki,
- rekurzivne funkcije in
- formalne gramatike.

Vsi ti formalizmi so po izrazni moči enakovredni in imajo enake omejitve [12, 5]: lahko (delno) opišejo pojave znotraj diskretnega sveta, medtem ko lahko opišejo le tako rekoč zanemarljivo majhen del zveznega sveta. Torej, če je svet dejansko zvezen, je najverjetnejne neopisljiv s katerimkoli formalizmom, ki ga znanost danes uporablja. To bi pomenilo, da znanje, ki nam ga lahko dajo znanost, knjige, učitelji, ni nikoli dokončno, saj je vedno samo približek in ne more popolnoma opisati realnosti.

V smislu ustvarjanja umetne inteligence s pomočjo algoritmov strojnega učenja od začetka nastanka računalnikov do danes ni videti ključnega napredka. Nekaj pomembnih korakov pa vseeno omenimo:

- v prejšnjem poglavju opisani Lenatov AM (Automatic Mathematician),
- uspehi pri igranju iger, kot sta dama in šah,
- umetne nevronске mreže kot model možganov,
- generiranje novega znanja iz podatkov.

Za vse algoritme strojnega učenja, ne glede na to, kako dobro so sprogramirani, veljajo omejitve, ki jih je postavila teorija naučljivosti [14]. Naučljivost temelji na teoriji izračunljivosti, saj je učenec omejen na Turingov stroj. Teorija naučljivosti odgovarja na osnovna vprašanja: ali obstaja program, ki se lahko nauči, po končnem branju iz neomejenega zaporedja besed nad neko abecedo, pravilo, ki bo znalo razlikovati besede iz danega jezika od besed, ki niso iz tega jezika. Če je za dani jezik odgovor pritrilen, pravimo, da je jezik naučljiv. Izkaže se, da je naučljivost tesno povezana z izračunljivostjo in vse omejitve v zvezi z izračunljivostjo veljajo tudi za naučljivost.

## Možnost umetne inteligence

*Učljiva je morala in inteligenta.*

*Anton Trstenjak*

Praktične raziskave metod umetne inteligence se ukvarjajo z razvojem sistemov, ki se obnašajo inteligenčno in ki so sposobni reševati relativno težke probleme. Pogosto te metode temeljijo na oponašanju človekovega načina reševanja problemov. Kot dolgoročni cilj nas

zanima, ali računalnikova inteligenco (sposobnost) lahko doseže oziroma preseže človekovo. Za razumevanje možnosti umetne inteligence, so pomembni vidiki: vpliv učenja na inteligenco, hitrost reševanja problemov, omejitve Turingovega stroja ter oponašanje inteligenčnega obnašanja.

**Vpliv učenja na inteligenco** Z učenjem se sposobnost sistema povečuje, torej se povečuje tudi njegova inteligenco. Človeška inteligenco je dinamična in se skozi življenje spreminja, v glavnem povečuje. Seveda je treba pri opredeljevanju količine inteligence upoštevati, da obstajajo različni vidiki inteligence (sposobnosti).

**Hitrejši je intelligentnejši** Ker je prilagajanje okolju in reševanje problemov boljše (učinkovitejše), če je hitrejše, je inteligenco povezana s hitrostjo in s časom. Vsi inteligenčni testi so časovno omejeni, kot tudi vsi preizkusi znanja. Lahko rečemo, da so v tem smislu hitrejši računalniki intelligentnejši od počasnejših, vzporedno procesiranje pa intelligentnejše od zaporednega.

**Meje inteligence** Če bi bil človek enakovreden Turingovemu stroju (algoritmu), bi teorija izračunljivosti postavila zelo močne omejitve glede njegove inteligence. Če pa privzamemo, da je človek močnejši "stroj" od Turingovega stroja (npr. zvezni in ne diskretni stroj), je njegovo delovanje formalno neopisljivo. To bi pomenilo, da algoritmično umetne inteligence, ki bi popolnoma posnemala človeka, ni mogoče ustvariti.

**Oponašanje inteligence** Tehnologije filma, računalnikov, robotov in navidezne resničnosti so nas prepričale, da se da oponašati prav vse in s tem ustvariti občutek resničnosti.

Če izpustimo zavest, je v principu lahko stroj dovolj inteligenten (sposoben, z dovolj veliko podatkovno bazo, v kateri ima shranjene in rešene vse mogoče situacije, v katerih se lahko znajde), da ustvari občutek umetne inteligence. Če k temu dodamo še izredne procesorske sposobnosti (super parallelizem super hitrih procesorjev), algoritme preiskovanja velikih prostorov ter algoritme strojnega učenja, ki bi mu omogočili izpopolnjevanje lastnih algoritmov in hevristik, bi takemu stroju z vso pravico rekli, da je inteligenten, saj bi v marsikateri nalogi, če že ne kar v vseh "praktičnih" nalogah, prekašal človeka. Seveda pa takemu stroju manjka zavest.

### (Ne)zmožnost umetne zavesti?

*Ne morem reči, da verjamem. Jaz vem! Imel sem izkušnjo, ko me je zaobjelo nekaj, kar je mnogo močnejše od mene, nekaj, čemur ljudje rečejo Bog.*

*Carl Gustav Jung*

Če želimo razmišljati o zavesti (*consciousness*), moramo najprej vsaj okvirno definirati, o čem se bomo pogovarjali. V tem razdelku bomo privzeli, da so opazljivi in merljivi vidiki zavesti (zmožnost ločevanja sebe od okolice, prepoznavanje sebe v ogledalu itd.) pravzaprav del inteligence. Težje opredeljiv je širši pojem zavesti, ki opredeljuje zavest kot subjektivno izkušnjo (v smislu osebne izkušnje v prvi osebi – *1st person experience*), ki je zaradi tega

v principu nemerljiva in neopisljiva. V tem razdelku poskušamo vsaj malo osvetliti ta vidik zavesti, ki se v zadnjem času vse bolj privzema kot temelj zavesti in bistvo človeka [27, 11, 9].

Za nek sistem lahko opredelimo (opazimo in objektivno izmerimo), da ima določene učne sposobnosti in da ima določeno stopnjo inteligence. Za razliko od učenja in inteligence je zavest nekaj posebnega, saj je nujno povezana s subjektivno izkušnjo in je objektivno zunanj opazovalec ne more preveriti. Preveriti sposobnost učenja, pridobljeno znanje in sposobnost intelligentnega prilagajanja okolju in intelligentnega reševanja problemov, je objektivno možno, čeprav ni trivialno (inteligencijski testi so sposobni izmeriti samo določene vidike intelligence, pa še to v precejšnji meri nezanesljivo). Preverjanje zavedanja nekega sistema pa v principu ni možno. Ali je nek (biološki ali umetni) sistem zavesten ali ne, ve samo ta sistem, pa še to samo, če se zaveda. Zunanji opazovalec nima nobenega mehanizma, da bi to ugotovil. O zavesti lahko govorиш samo, če si sam zavesten in če predpostaviš, da so tebi podobni sistemi tudi zavestni. Zavesten sistem se da oponašati z nezavednim sistemom do poljubne (vendar vedno delno nepopolne) natančnosti, zato bi se zunanjega opazovalca v principu dalo prevarati.

Sistem je lahko bolj ali manj intelligenten, vendar brez zavesti (robot, "zombi"), kakor tudi je lahko zavesten, vendar manj intelligenten (manj intelligentni ljudje, živali itd.). Zavest je ključno povezana s pojmi življenje, intelecija in svobodna volja:

**Zavest = življenje?** Človek se zaveda, pes se zaveda, ameba se mogoče tudi do neke mere zaveda. Znanost še vedno ne zna pojasniti nastanka življenja. Obstaja sicer več teorij. Materializem pravi, da je bodisi nastalo po naključju (kar je skrajno neverjetno), bodisi je posledica samoorganizacije in s tem kompleksnosti materije. Po eni teoriji je življenje prišlo iz vesolja (aminokisline na meteoritih), kar vprašanje nastanka samo odmakne. Vitalizem zagovarja tezo, da je življenje ustvarila višja sila (univerzalna zavest).

**Več inteligence omogoča večjo zavest?** Čim sposobnejši je biološki organizem, tem več zavedanja ima lahko osebek. Lahko pa zelo intelligentnemu sistemu, odvzameš zavest ("zombi", robot, pranje možganov, ali pa huda zaslepljenost) in dobiš močno intelligenten sistem, ki se ne zaveda kaj počne.

**Zavest implicira svobodno voljo?** Sistem lahko samo reagira na zunanje dražljaje in so v tem smislu njegovi odzivi determinirani in nezavedni. Zavestni sistem pa se lahko sam, brez zunanjega vzroka, odloči za akcijo (in ne reakcijo), kar pomeni, da ima svobodno voljo. Mnenja o tem, ali svobodna volja sploh obstaja ali ne, so deljena. Zdi pa se logično, da če obstaja zavest, potem obstaja tudi svobodna volja [7, 9].

## Znanost in filozofija

*Prvi požirek iz čaše znanosti te bo spremenil v ateista, a na dnu kozarca te Bog spet čaka.*

*Werner Heisenberg*

Znanost modelira empirične podatke: postavi model (hipotezo, teorijo), ki opisuje meritve. Če model dovolj natančno opisuje podatke, se sčasoma privzame kot (naravni) zakon. Če pride kasneje do drugačnih (bolj natančnih, pod drugačnimi pogoji itd.) meritov, ki se ne skladajo z zakoni, se obstoječi zakoni spremenijo/razširijo, tako da pokrivajo tudi nove meritve. Znanost se pri opisovanju realnosti ormeji na razum, ki je omejen s simbolično reprezentacijo/opisljivostjo, čeprav znanstveniki pri ustvarjalnem raziskovanju uporabljajo (najverjetneje neopisljivo) intuicijo.

Materialistična znanost se ne sprašuje: zakaj vesolje obstaja in čemu je življenje namejeno. Zaradi ignoriranja teh vprašanj mnogi privzamejo, da sta vesolje in življenje nastala po naključju in da sama po sebi nimata globjega smisla.

Filozofija (v prvotnem pomenu besede) poskuša odgovoriti na ti dve vprašanji, vendar pri tem potrebuje razum in srce (intuicijo). Današnja filozofija posnema znanost in izključuje srce (um, etiko). Iz učenj velikih modrecev bi lahko sklepali, da sta tako razum kot srce nujna za zavest. Ali kot je dejal Albert Einstein [3]: "Znanost brez vere je hroma, vera brez znanosti je slepa."

Inteligenco je sposobnost. Umetni sistemi pridobivajo in bodo še pridobivali na sposobnosti. Zavest (v širšem pomenu besede) pa ima najverjetneje globji pomen in namen, zato je zavest nujno povezana z etiko življenja. Inteligenca brez srca je nezavedna inteligenco, sposobna uničevati in uničiti okolje in sebe. Razum je inteligenco brez uma. Kot je dejal častni senator Univerze v Ljubljani, Dalaj Lama: "Potrebujemo izobrazbo in čut moralne etike - to dvoje mora iti skupaj."

Umetna in naravna inteligenco sta orodji, ki ju lahko koristno uporabiš ali zlorabiš, odgovornost pa ostaja na tvoji (za)vesti.

## 2.4 Zakaj strojno učenje

*Pri ustreznem odnosu med človekom in strojem gre za dopolnitev in razširitev, ne pa za posnemanje.*

*Mortimer Taube*

Človek si je vedno želel bolje spoznati samega sebe in ustvariti sisteme, ki bi njegove zmožnosti presegle. Kognitivno modeliranje je namenjeno spoznavanju in razlagi kognitivnih procesov v človeških možganih. Danes si kognitivnih procesov brez učenja ni moč razlagati, zato večina kognitivnih modelov vključuje tudi različne algoritme učenja.

Učenje ljudi se zdi zelo počasno: 20 let šolanja je potrebno, da se lahko magister znanosti začne učiti svojega poklica na delovnem mestu. V naslednjih 20 letih si nabere zadosti izkušenj, da ga lahko imenujemo izkušenega strokovnjaka na nekem ozkem področju. Pri vsem

tem pa svojih izkušenj ne more preprosto prenesti na mlajše kolege. Zanj veljajo vse slabe lastnosti, kot za vsakogar: je pozabljiv, hitro se utrudi, na njegovo delo vplivajo razpoloženje, vreme, bolezen itd.

Enega osnovnih namenov razvoja metod strojnega učenja smo že nekajkrat omenili: avtomatsko generiranje baz znanja za ekspertne sisteme. Ekspertni sistemi so sposobni pomagati strokovnjakom pri njihovem delu, v izjemnih primerih pa lahko strokovnjake tudi nadomestijo (vsaj začasno, npr. če je strokovnjak odsoten, bolan itd.). Za ekspertne sisteme se zah-teva, da znajo uporabniku svoje rešitve tudi obrazložiti in argumentirati. Le takemu sistemu bo človek zaupal in mu prepustil (manj) pomembne odločitve.

Za reševanje zahtevnih specializiranih problemov potrebujejo ekspertni sistemi bazo znanja. Lahko jo sestavimo "ročno" tako, da poskušamo zakodirati pomembne elemente znanja s povzemanjem iz obstoječe literature in spraševanjem strokovnjakov. Tako sestavljanje baze znanja je tipično naporno, zamudno in nekonsistentno.

Druga možnost je avtomatska gradnja baze znanja z algoritmi za induktivno učenje. Znano je, da strokovnjaki lažje opisujejo primere rešenih problemov kot pravila, ki jih uporabljajo pri reševanju problemov. Rešene primere, ki jih je opisal strokovnjak ali zanje obstaja arhivirana baza rešenih problemov, lahko uporabimo za strojno učenje. Rezultat učenja so pravila, ki se uporabijo za reševanje novih problemov z ekspertnim sistemom.

Pri avtomatsko zgrajeni bazi znanja je treba biti pazljiv. Takšno znanje je potrebno preveriti in ovrednotiti. Avtomatsko pridobljeno znanje naj bi bilo človeku razumljivo, zato je pogosto potrebno to znanje popraviti in/ali prilagoditi (npr. spremeniti opisni jezik). Pravilnost avtomatsko zgrajenega znanja je treba ovrednotiti. Najpogostejša oblika ocenjevanja je delež pravilnih odgovorov na novih problemih, ki niso bili na voljo med učenjem.

Ekspertni sistemi imajo prednosti, ki veljajo za računalnik: so neutrudljivi, zanesljivi, ponovljivi, dostopni 24 ur na dan, 365 dni na leto, zakodirano znanje je prenosljivo, pri odločitvah lahko upoštevajo velike podatkovne baze, zbrane v preteklosti.

Kljub prednostim pa ekspertni sistemi ne morejo in v (bližnji) prihodnosti ne bodo izpodrinali strokovnjakov. Primerjava med računalnikovimi in človeškimi lastnostmi je podana v tabeli 2.3. Kljub izredni hitrosti današnjih računalnikov, ki lahko obdelujejo ogromne količine podatkov, človeškega širokega znanja in spomina še zdaleč ne dosegajo. Ljudje zaradi visoke stopnje paralelizma v možganih izredno hitro in z lahkoto rešijo nekatere vrste problemov, ki jih (današnji) računalniki ne morejo rešiti oziroma bi bila rešitev prepočasna.

Glavna prednost ljudi je, da so prilagodljivi in svoje znanje dinamično spreminjajo in izpopolnjujejo. Prav zaradi tega je razvoj metod strojnega učenja tako pomemben. Z ustreznimi algoritmi je potrebno današnje računalnike narediti manj toge in bolj prilagodljive novim situacijam in problemom. To pomeni, da tudi algoritmi ne bodo več statični, ampak se bodo dinamično spreminjali z učenjem. Zavedati se je treba, da bo obnašanje takih učečih se sistemov nenapovedljivo in bo potrebno vgrajevati mehanizme za preverjanje in nadzor naučenega. Nekateri celo napovedujejo, da bodo inteligentni sistemi morali včasih uporabnikom "lagati", zato da bodo v kritičnih trenutkih dosegli optimalno izvajanje [13]. Tak sistem je težko obvladovati, zato je zaželeno, da obstaja možnost popolnega nadzora sistema.

človek	računalnik
pozabljiv potrebuje počitek težko posreduje znanje	zanesljiv, ponovljiv lahko deluje neprekinjeno trivialno prenosljivo znanje
omejen spomin ? počasen ?	velike podatkovne baze ? hiter ?
široko znanje vzporedno procesiranje se uči iz napak dinamično znanje	ozko specializirano znanje večinoma zaporedno proc. isto napako ponavlja statično znanje

Tabela 2.3: Primerjava lastnosti človeka in računalnika.

## Literatura

- [1] R. Borger, A.M. Seaborne. *The Psychology of Learning*. Harmonds worth: Penguin Books, 1966.
- [2] H. L. Dreyfus. *What Computers Can't Do*. New York: Harper & Row, 1972.
- [3] A. Einstein. Science and religion. *Nature*, 146:605–607, 11 1940.
- [4] D. Goleman. *Čustvena inteligenco na delovnem mestu*. Mladinska knjiga, Ljubljana, 2001.
- [5] J. E. Hopcroft, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- [6] T. Kohonen. *Self-Organization and Associative Memory*. Berlin: Springer-Verlag, 1984.
- [7] I. Kononenko. Znanost in duhovnost. *Sodobnost*, 66(2):259–264, 2002.
- [8] I. Kononenko. Znanost preverja fenomene zavesti. V I. Kononenko, I. Jerman, (ur.), *Mind-body studies : proceedings of 6th International Conference on Cognitive Science*, str. 169–172, 2003. ISBN ISBN 961-6303-50-3. Ljubljana, 13-17th October, Institut Jožef Stefan.
- [9] I. Kononenko. Natural and machine learning, intelligence and consciousness. V E. Žerovnik, O. Markič, A. Ule, (ur.), *Philosophical Insights about Modern Science*, str. 239–258. New York: Nova Science Publishers, 2009.
- [10] E. Lausch. *Manipulation – Der Griff nach dem Gehirn*. Deutsche Verlags-Anstalt, Stuttgart, 1972.
- [11] D. Lorimer, (ur.). *The Spirit of Science*. Floris Books, Edinburgh, 1998.
- [12] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill, 1974.

- [13] D. Michie, R. Johnston. *The Creative Computer: Machine Intelligence and Human Knowledge*. New York: Viking, 1984.
- [14] D. N. Osherson, M. Stob, S. Weinstein. *Systems That Learn*. Bradford Book, The MIT Press, 1986.
- [15] V. Pečjak. *Psihologija spoznavanja*. Državna založba Slovenije, Ljubljana, 1977.
- [16] V. Pečjak. *Poti do znanja: metode uspešnega učenja*. Cankarjeva založba, Ljubljana, 1986.
- [17] B. Rudolf. *Dialektika*. ČZP Komunist, Ljubljana, 1977.
- [18] P. Russell. *The Brain Book: Know Your Own Mind and How to Use It*. Routledge and Kegan Paul, London & Hawthorne, New York, 1979.
- [19] O. Sacks. *The Man Who Mistook His Wife for a Hat and other Clinical Tales*. Harper & Row Pub, 1985.
- [20] P. Schauer. *Interferon*. DDU Univerzum, Ljubljana, 1984.
- [21] E. Shapiro. Inductive inference of theories from facts. Research report 192, Dept. of Computer Sc., Yale University, New Haven, 1981.
- [22] A. Sloman. The emperor's real mind: review of Roger Penrose's The Emperor's New Mind: Concerning computers, minds and laws of physics. *Artificial Intelligence*, 56: 355–396, 1992.
- [23] M. Taube. *Computers and Common Sense: The Myth of Thinking Machines*. New York: Columbia University Press, 1961.
- [24] A. Trstenjak. *Problemi psihologije*. Slovenska matica, Ljubljana, 1976.
- [25] A. Trstenjak. *Človek bitje prihodnosti*. Slovenska matica, Ljubljana, 1985.
- [26] L. Žlebnik. *Pedagoška psihologija*. Visoka šola za organizacijo dela, Kranj, 1982.
- [27] B.A. Wallace. *The Taboo of Subjectivity: Towards a New Science of Consciousness*. Oxford University Press, 2000.

## Poglavlje 3

# Osnove strojnega učenja

*Človek in računalnik sta sposobna doseči, česar nobeden od njiju sam ne zmore.*

*Hubert L. Dreyfus*

Imejmo pet učnih primerov, ki so prikazani v tabeli 3.1. Iz teh podatkov se želimo naučiti zakonitosti pojava, ki je te podatke povzročil. Poleg dejanskih pravil, ki so bila uporabljena pri generiraju podatkov:

$$ploščina = 1.\text{stranica} \times 2.\text{stranica}$$

$$obseg = 2 \times 1.\text{stranica} + 2 \times 2.\text{stranica}$$

$$1.\text{stranica} = 2.\text{stranica} \rightarrow \text{kvadrat}$$

so možna pravila, ki bi jih učenec lahko izluščil, tudi npr.:

$$1.\text{stranica} = 5 \text{ ali } 2.\text{stranica} = 5 \rightarrow \text{pravokotnik}$$

$$10 < ploščina < 30 \rightarrow \text{pravokotnik}$$

$$10 < obseg < 20 \rightarrow \text{pravokotnik}$$

$$|obseg - ploščina| < 4 \rightarrow \text{pravokotnik}$$

$$\text{pravokotnik in } 1.\text{stranica} = 3 \text{ ali } 2.\text{stranica} = 3 \rightarrow obseg = 16$$

1.stranica	2.stranica	ploščina	obseg	lik
5	4	20	18	pravokotnik
5	3	15	16	pravokotnik
2	2	4	8	kvadrat
3	5	15	16	pravokotnik
6	6	36	24	kvadrat

Tabela 3.1: Množica učnih primerov.

$$\begin{aligned} ploščina &= 15 \leftrightarrow obseg = 16 \\ kvadrat \rightarrow |obseg - ploščina|/2 &= 1.stranica \end{aligned}$$

Kaj mora voditi učenca, da bo spoznal, da so nekatera od pravil bolj verjetna od drugih? Ali sploh lahko karkoli sklepa iz opisov samo petih učnih primerov? Zakaj je npr. pravilo

$$pravokotnik \text{ in } 1.stranica = 3 \text{ ali } 2.stranica = 3 \rightarrow obseg = 16$$

intuitivno nezanimivo, medtem ko se nam zdi pravilo

$$obseg = 2 \times 1.stranica + 2 \times 2.stranica$$

bolj verjetno?

V tem poglavju odgovarjamo na taka in podobna vprašanja. Opisani so osnovni principi, ki nas vodijo pri načrtovanju sistemov strojnega učenja, kot so princip preprostosti in princip večkratne razlage, ter principe ocenjevanja kvalitete naučene hipoteze.

### 3.1 Osnovni principi strojnega učenja

*Strojno učenje je predpogoj inteligenčnega sistema.*

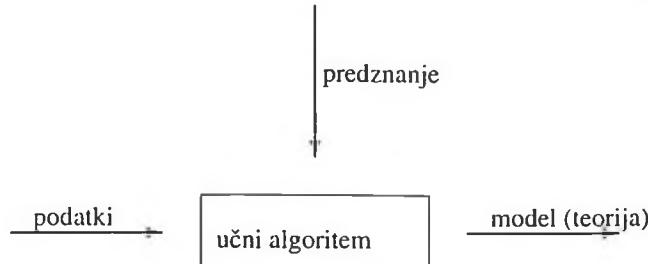
*Alan M. Turing*

V tem razdelku je podana definicija strojnega učenja kot modeliranja podatkov. Opisan je princip najkrajšega opisa, ki pravi, da so najpreprostejše hipoteze, ki razlagajo podatke, najbolj verjetne. Po drugi strani princip večkratne razlage zahteva, da za optimalno napovedovanje uporabimo vse možne hipoteze in ne le najbolj verjetne. Na koncu so opisani še pristopi k ocenjevanju verjetnosti iz podatkov, ki tudi kažejo na to, da je hipoteza tem zanesljivejša, čim več podatkov razлага.

#### Učenje kot modeliranje

Učenje je vsaka sprememba sistema, ki mu omogoča, da opravlja isto nalog bolje. Rezultat učenja je znanje, ki ga sistem uporabi za reševanje novih nalog. Znanje je lahko množica zapomnjenih podatkov, algoritem za reševanje določenih nalog ali pa množica napotkov za bolj učinkovito reševanje nalog. Pogosto bomo pri obravnavi sistemov za strojno učenje ločevali med *učnim algoritmom*, ki iz množice podatkov in predznanja tvori novo znanje (oziroma samo spremeni prejšnje znanje), in med *izvajalnim algoritmom*, ki avtomatsko naučeno znanje uporablja za reševanje novih problemov. Avtomatsko naučenemu znanju bomo pogosto rekli kar *model*. Za model zahtevamo, da čim bolj ustreza vhodnim podatkom in predznanju. Uporabljali bomo tudi izraza *hipoteza* in *teorija*.

Pri uporabi besede *model* se moramo zavedati, da matematična logika uporablja drugačno definicijo modela, ki je intuitivno ravno nasprotna definiciji, ki jo uporabljamo v strojnem



Slika 3.1: Algoritem za strojno učenje.

učenju<sup>1</sup>. Zato bomo pogosto namesto besede "model" uporabljali besedo "hipoteza".

Strojno učenje opredelimo kot opisovanje ali *modeliranje* podatkov. Vhod v sistem za strojno učenje sta množica podatkov ter predznanje, izhod pa opis (model, hipoteza, teorija), ki te podatke skupaj s predznanjem opisuje in pojasnjuje (glej sliko 3.1). Predznanje je ponavadi kar prostor možnih modelov, v katerem bo algoritem iskal tistega, ki čim bolj ustreza vhodnim podatkom, ter kriterij optimalnosti, ki ga bo sistem med iskanjem poskušal izpolniti. Predznanje lahko vsebuje tudi začetno hipotezo, ki je lahko približna rešitev problema, ter množico hevristik, ki služijo za usmerjanje iskanja v bolj obetavne dele prostora.

Strojno učenje lahko predstavimo tudi kot optimizacijski problem. Pri danem prostoru možnih rešitev (modelov) in pri danem kriteriju optimalnosti je treba poiskati tisto rešitev (model), ki zadošča kriteriju optimalnosti oziroma minimizira vrednost kriterijske funkcije. Pri tem je vrednost kriterijske funkcije odvisna od trenutnega modela, predznanja in vhodnih podatkov, ki jih modeliramo. Ker je prostor možnih rešitev ponavadi zelo velik (mnogokrat neskončen), je iskanje optimalne rešitve prezahtevno in se moramo zadovoljiti s čim boljšimi suboptimalnimi rešitvami.

Ločimo naslednje vrste modelov (in s tem tudi zvrsti strojnega učenja):

**Diskrete funkcije** Zaloga napovednih vrednosti je končna neurejena množica. Odvisni spremenljivki pravimo razred. Problemom, katerih model je diskretna funkcija, pravimo *klasifikacijski problemi*. Ko je funkcija naučena, jo uporabljamo za klasifikacijo (uvrščanje), t.j. ugotavljanje vrednosti funkcije pri danih vrednostih neodvisnih spremenljivk. Pogosto je ciljna funkcija mnogolična: isto vrednost domene preslika v več

<sup>1</sup>V logiki je model množice formul (izjav) v danem jeziku definiran kot taka interpretacija jezika, v kateri so vse formule (izjave) iz dane množice resnične [24]. V strojnem učenju pa se za vse vhodne podatke (izjave) predpostavlja, da so resnični, in je model formula, iz katere lahko (do neke mere) izpeljemo vhodne podatke (izjave). Torej je v logiki jezik (formalni sistem) abstrakcija modela. V strojnem učenju je ravno nasprotno: model je abstrakcija podatkov.

različnih vrednosti iz zaloge vrednosti. Ob tem je lahko vsaka vrednost iz zaloge vrednosti utežena, ponavadi z verjetnostmi. Opravka imamo s funkcijami, ki domeno preslikajo v (zvezni) prostor verjetnostnih distribucij zaloge vrednosti. Medicinsko diagnostično pravilo npr. preslika opis stanja pacienta v množico možnih diagnoz, pri čemer je vsaki možni diagnozi pripisana tudi verjetnost.

Mnogi odločitveni problemi, diagnostični problemi, problemi vodenja in problemi napovedovanja se lahko predstavijo kot klasifikacijski problemi. Tipični primeri so medicinska diagnostika in prognostika, napovedovanje vremena, diagnostika industrijskih procesov, klasifikacija izdelkov po kakovosti, vodenje dinamičnih sistemov ipd.

**Zvezne funkcije** Zaloga vrednosti je (potencialno) neskončna urejena množica. Odvisni spremenljivki pravimo regresijska spremenljivka ali zvezni razred. Problemom, katerih model je zvezna funkcija, pravimo *regresijski problemi*. Tako kot pri klasifikaciji tudi tu uporabljamo avtomatsko zgrajeno funkcijo za ugotavljanje vrednosti funkcije pri danih vrednostih neodvisnih spremenljivk. Mnoge probleme napovedovanja in odločitvene probleme lahko opišemo kot regresijske probleme. Primeri so napovedovanje časovnih vrst, vodenje dinamičnih sistemov, ugotavljanje vplivov različnih parametrov na velikost odvisne spremenljivke ipd. Poleg vrednosti funkcije je pogosta zahteva pri regresijskih problemih tudi interval zaupanja. Le-ta verjetnostno opisuje zaupanje v vrednost, ki nam jo model predlaga kot rešitev danega primera.

**Relacije** Avtomatsko zgrajene relacije uporabljamo za preverjanje, če je dana  $n$ -terica objektov element relacije, ali pa jih uporabljamo kot funkcije, pri čemer izberemo enega ali več parametrov za odvisno spremenljivko. Relacije so splošnejše od funkcij, kar pomeni, da je ustrezен prostor možnih relacij mnogo večji kot prostor možnih funkcij. Temu primerno je učenje relacij zahtevnejše, tako glede iskanja rešitev kot glede zahtevanega števila vhodnih podatkov in/ali količine predznanja. Tudi relacije so lahko zvezne (sistemi enačb) ali diskretne (logične relacije). Primeri relacijskega učenja so učenje strukture kemijskih spojin, učenje geometrijskih lastnosti objektov, ugotavljanje splošnih zakonitosti v podatkovni bazi ipd.

## Princip najkrajšega opisa

*Nesmotrno je narediti z več, kar lahko storimo z manj.*

*William of Ockham*

*Narava je zadovoljna s preprostostjo in ne hlini odvečnih vzrokov.*

*Isaac Newton*

*Naredi preprosto, kolikor se le da - vendar ne preprosteje!*

*Albert Einstein*

Pri preiskovanju prostora možnih hipotez, ki ustrezajo vhodnim podatkom in predznanju,

je naloga algoritma za strojno učenje poiskati hipotezo, ki "čim bolj ustreza" vhodnim podatkom in predznanju. Vhodnim podatkom lahko "ustreza" več (tudi neskončno mnogo) hipotez. Zato potrebujemo kriterije, ki bodo ocenjevali kvaliteto hipotez. Znan je princip *Ockhamove britve* (*Occam's Razor Principle*), ki pravi, da je najpreprostejša razlaga najbolj zanesljiva. Kot bomo videli v nadaljevanju, lahko ta princip posplošimo na princip najkrajšega opisa podatkov, ki je ekvivalenten principu maksimalne verjetnosti hipoteze.

Kriteriji za usmerjanje učnega algoritma so različni za različne naloge ter lahko nastopajo v kombinaciji z drugimi kriteriji, npr.:

- maksimizirati klasifikacijsko točnost hipoteze,
- minimizirati povprečno ceno klasifikacijskih napak,
- minimizirati velikost hipoteze,
- maksimizirati prileganje hipoteze vhodnim podatkom,
- maksimizirati razumljivost hipoteze,
- minimizirati časovno zahtevnost klasifikacije,
- minimizirati število parametrov, potrebnih za klasifikacijo,
- minimizirati ceno pridobivanja vrednosti parametrov, potrebnih za klasifikacijo,
- maksimizirati verjetnost hipoteze glede na dano predznanje in vhodne podatke.

Zadnji princip, t.j. *maksimizirati verjetnost hipoteze*, je najsplošnejši in ima lepo lastnost, da zadošča večini zgoraj omenjenih kriterijev. Izkušnjec kažejo, da ima najbolj verjetna hipoteka (v povprečju preko vseh možnih učnih problemov) tudi maksimalno klasifikacijsko točnost, minimalno velikost pri dani klasifikacijski točnosti in se najbolj prilega vhodnim podatkom pri dani klasifikacijski točnosti. Ker je hipoteza najmanjša, je v tem smislu navadno tudi najbolj razumljiva, uporablja minimalno število parametrov, da doseže dano klasifikacijsko točnost, in omogoča najhitrejšo klasifikacijo. Ob predpostavki enake cene vseh klasifikacijskih napak in enake cene za pridobivanje vrednosti posameznih parametrov kriterij maksimalne verjetnosti hipoteze ustreza tudi obema kriterijema, ki upoštevata ceno.

Označimo množico možnih hipotez s  $\mathcal{H}$ , hipotezo s  $H \in \mathcal{H}$ , predznanje z  $B$  in vhodne podatke z  $E$ . V idealnem primeru, ko poznamo verjetnosti  $P(H|E, B)$  posameznih hipotez pri danem predznanju in vhodnih podatkih, je zaželeno poiskati hipotezo, ki maksimizira pogojno verjetnost:

$$H_{opt} = \arg \max_{H \in \mathcal{H}} P(H|E, B) \quad (3.1)$$

Na žalost so (apriorne in pogojne) verjetnosti hipotez neznane in zato gornji kriterij nadomestimo z različnimi hevrističnimi kriteriji, ki se mu poskušajo približati.

Rissanen je predlagal princip najkrajšega opisa (*minimal description length (MDL principle)*), ki je ekvivalenten kriteriju maksimalne verjetnosti [23]. Naj bo  $P(H|B)$  (apriorna)

verjetnost hipoteze  $H$  pri danem predznanju  $B$ . Definirajmo količino informacije  $I(H|B)$ , da zvemo, da je hipoteza  $H$  resnična pri danem predznanju  $B$ , z:

$$I(H|B) = -\log_2 P(H|B) \text{ [bit]}$$

ter analogno temu pogojno količino informacije pri danem predznanju  $B$  in vhodnih podatkih  $E$  z:

$$I(H|E, B) = -\log_2 P(H|E, B) \text{ [bit]}$$

Potem lahko (3.1) zapišemo z:

$$H_{opt} = \arg \min_{H \in \mathcal{H}} I(H|E, B) \quad (3.2)$$

Ker po Bayesovem izreku velja

$$I(H|B, E) = I(E|H, B) + I(H|B) - I(E|B)$$

in ker lahko vzamemo, da je  $I(E|B)$  konstanta, lahko kriterij (3.2) zapišemo kot

$$H_{opt} = \arg \min_{H \in \mathcal{H}} (I(E|H, B) + I(H|B)) \quad (3.3)$$

Torej kriterij zahteva, da poiščemo hipotezo, ki minimizira vsoto dolžin opisov hipoteze in vhodnih podatkov pri dani hipotezi. Dejansko to pomeni, da iščemo kompromis med velikostjo hipoteze  $I(H|B)$  in velikostjo njene napake  $I(E|H, B)$ .

Problem minimizacije dolžine opisa množice podatkov lahko v kontekstu komunikacijske teorije definiramo takole. Oddajnik mora sprejemniku preko komunikacijskega kanala poslati opis množice podatkov. Preko komunikacijskega kanala lahko pošlje sporočilo kot niz ničel in enic. Dolžina sporočila predstavlja število bitov, potrebnih za kodiranje opisa podatkov. Pri kodiranju je treba predpostaviti optimalnost kodiranja, sprejemnik pa mora biti sposoben sporočilo razumeti oziroma dekodirati. To pomeni, da je dekoder sprejemniku znan, ali pa ga mora pošiljalatelj zakodirati in poslati skupaj z opisom podatkov, kar podaljša celotno sporočilo še za dolžino opisa dekoderja. Naloga oddajnika je minimizirati dolžino sporočila.

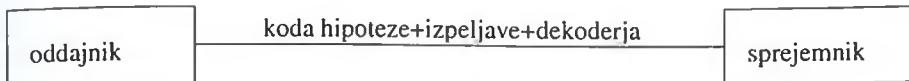
Oddajnik lahko zakodira podatke direktno in jih pošlje sprejemniku, ali pa zakodira hipotezo (model) ter še izpeljave originalnih podatkov iz dane hipoteze (glej sliko 3.2). Če je opis modela skupaj z opisom izpeljave podatkov iz hipoteze krajiš od opisa originalnih podatkov, pravimo, da je hipoteza *kompresivna*:

$$I(H|B) + I(E|H, B) < I(E|B)$$

Za preverjanje tega kriterija ni potrebno poznati verjetnosti podatkov  $P(E|B)$ , ampak zadošča dolžina opisa podatkov  $I(E|B)$ . Dolžino lahko ocenimo tako, da predpostavimo suboptimalno kodirno shemo. Ker je določitev optimalnega kodiranja neizračunljiv problem [23], se zadovoljimo s tem približkom.

Po kriteriju najkrajšega opisa so sprejemljive samo kompresivne hipoteze. V nasprotnem primeru uporabimo originalne podatke kot trivialen model.

Optimalna hipoteza je kratka ( $I(H|B)$ ), vendar nudi dovolj informacije, da lahko z neno pomočjo izpeljemo originalne vhodne podatke z minimalno dodatno informacijo ( $I(E|H, B)$ ). Pri izpeljavi smo predpostavili, da imata oba, sprejemnik in oddajnik, na voljo predznanje  $B$ . Če sprejemnik predznanja nima, lahko predznanje interpretiramo kot del vhodnih podatkov  $E$  in pošljemo še to.



Slika 3.2: Komunikacijski kanal.

### Inkrementalno (sprotno) učenje

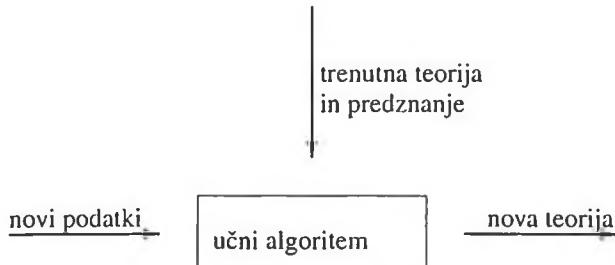
*Narava nikoli ne pove, če so domneve pravilne. Znanstveni uspeh pomeni postaviti pravilno hipotezo, od katere pozneje nikoli ne odstopamo.*

*Osherson in sod. (1986)*

Učni algoritem na sliki 3.1 predpostavlja, da so vsi podatki na voljo že na začetku učenja. Ta scenarij lahko posplošimo tako, da učni algoritem dobiva vhodne podatke drugega za drugim. Po vsakem vhodnem podatku učni algoritem svojo trenutno hipotezo popravi tako, da ustreza (vsem) prej videnim podatkom in novemu podatku (glej sliko 3.3).

Učenju, kjer učenec sproti spreminja teorijo ob spremenjanju novih podatkov, pravimo *inkrementalno (incremental)* ali *sprotno učenje (on-line learning)*. Najpreprostejši inkrementalni algoritem ob vsakem novem vhodnem podatku zavrže prejšnjo hipotezo in začne z učenjem "na novo". Takih algoritmov ne bomo prištevali k inkrementalnim. Inkrementalni algoritem poskuša poiskati najmanjšo potrebno spremembo trenutne teorije, tako da le-ta ustreza vsem do sedaj videnim podatkom. Včasih inkrementalni algoritmi "pozabijo" vhodne podatke, ki so jih že videli in je nova teorija v nasprotju s prejšnjimi podatki. Ta lastnost je prednost v primerih, ko se celoten problem dinamično spreminja, torej se spreminja tudi optimalna hipoteza.

Pričakovali bi, da bo inkrementalno učenje z dodajanjem novih podatkov konvergiralo k optimalni teoriji za obravnavani problem in da se od nekega trenutka naprej teorija ne bo več spreminja kljub novim vhodnim podatkom. Na žalost pa učni algoritem **nikoli** z gotovostjo ne ve, če se je naučil optimalne teorije, saj je lahko vsaka naslednja množica vhodnih podatkov v nasprotju s trenutno teorijo. Kljub temu je v praksi pogosto res, da od nekega trenutka naprej dodajanje novih podatkov ne izboljša naučene teorije.



Slika 3.3: Inkrementalno učenje.

### Princip večkratne razlage

*Če je več teorij konsistentnih z dejstvi, obdrži vse teorije.*

*Epicurus*

*Za vsak posamezen model poišči njegovo napoved in jo obteži z verjetnostjo tega modela. Obtežena vsota takih napovedi je optimalna napoved.*

*Peter Cheeseman*

Princip Ockhamove britve je znan in ga uporablja večina sistemov za strojno učenje predvsem v smislu hevristik, ki dajejo prednost preprostejšim teorijam. Manj znan je *princip večkratne razlage (principle of multiple explanations)*, ki pravi, da je treba obdržati vse hipoteze, ki so konsistentne z vhodnimi podatki [23]. Ta princip je navidezno v nasprotju s principom Ockhamove britve. Dejansko pa se izkaže, da se principa dopolnjujeta. Princip Ockhamove britve lahko uporabimo za iskanje najboljše hipoteze oziroma za iskanje množice relativno dobrih hipotez. Ta princip usmerja učni algoritem v iskanje boljših hipotez. Princip večkratne razlage pa s kombinacijo več verjetnih hipotez daje najboljše rezultate pri uporabi avtomatsko zgrajenih hipotez, torej služi za usmerjanje izvajalnega algoritma.

Gre za povezavo s Centralnim limitnim teoremom, kjer z večanjem števila spremenljivk povprečje konvergira k pravi verjetnosti:

$$P = \lim_{k \rightarrow \infty} \frac{P_1 + P_2 + \dots + P_k}{k}$$

Izvajalni algoritmi najpogosteje uporabljajo samo eno hipotezo, ki se je izkazala za najoptimalnejšo med pregledanimi hipotezami. Naj bo  $\mathcal{R}$  množica rešitev in  $r \in \mathcal{R}$  konkretna rešitev

problema. Naj bo  $P(r|H)$  verjetnost rešitve  $r$ , če je hipoteza  $H$  pravilna. Potem je rešitev, ki jo ponuja optimalna hipoteza, definirana z:

$$r_{opt} = \arg \max_{r \in \mathcal{R}} P(r|H_{opt}) \quad (3.4)$$

Toda to ni optimalna rešitev, saj princip večkratne razlage pravi, da je potrebno uporabiti vse možne hipoteze. Ta princip formaliziramo tako, da odgovore vsake hipoteze obtežimo z verjetnostjo hipoteze. Optimalni izvajalni algoritem uporablja vse (dovolj) verjetne hipoteze:

$$r_{opt} = \arg \max_{r \in \mathcal{R}} \sum_{H \in \mathcal{H}} (P(r|H)P(H|E, B)) \quad (3.5)$$

Pri tem nam princip Ockhamove britve pomaga eliminirati hipoteze z zanemarljivo majhno verjetnostjo  $P(H|E, B)$ . Ostane problem aproksimacije verjetnosti  $P(H|E, B)$  za bolj verjetne hipoteze, ki jih upoštevamo v enačbi (3.5).

Navidezno nasprotje dobimo, če interpretiramo kombinacijo več teorij kot eno (najboljšo) teorijo. Toda v tem primeru smo povečali originalni prostor možnih teorij. Če smo imeli prej  $n$  možnih teorij, je po tej interpretaciji prostor možnih teorij najmanj potenčna množica prejšnjega prostora teorij, ki vsebuje  $2^n$  možnih teorij (z upoštevanjem vseh možnih verjetnostnih distribucij po hipotezah pa je prostor še bistveno večji). Spremeni se tudi optimalno kodiranje teorij in s tem kriterij najkrajše dolžine opisa.

## Ocenjevanje verjetnosti

*Verjetnost je pravzaprav natančnejši opis zdrugega razuma.*

*Laplace*

Pri strojnem učenju se pogosto srečamo z ocenjevanjem verjetnosti. Verjetnost je bodisi dana vnaprej in je vsebovana v predznanju bodisi jo moramo oceniti iz vhodnih podatkov. V slednjem primeru je pogosto potrebno oceniti verjetnost iz majhne množice vhodnih podatkov. Taka ocena verjetnosti je nezanesljiva, zato ji ne smemo preveč zaupati.

Verjetnost dogodka ocenimo s pomočjo relativne frekvence dogodka iz učne množice. Pri tem je optimalna ocena verjetnosti odvisna od predpostavljene apriorne porazdelitve verjetnosti [7]. Pogosto se uporablja porazdelitev  $\beta(a, b)$  zaradi lepih lastnosti, ki omogočijo preproste izpeljave ocen verjetnosti. Porazdelitev  $\beta(a, b)$  ima gostoto definirano z:

$$p(x) = \begin{cases} \frac{1}{B(a,b)} x^{a-1} (1-x)^{b-1} & 0 \leq x \leq 1 \\ 0 & \text{sicer} \end{cases}$$

$a > 0$  in  $b > 0$  sta parametra porazdelitve,  $B(a, b)$  pa je beta funkcija, ki je definirana z:

$$B(a, b) = \int_0^1 x^{a-1} (1-x)^{b-1} dx$$

Parametra  $a$  in  $b$  lahko interpretiramo kot  $a$  uspešnih in  $b$  neuspešnih neodvisnih poskusov. Če je slučajna spremenljivka  $p$  porazdeljena po porazdelitvi  $\beta(a, b)$ , je njeno matematično

upanje enako

$$\text{Exp}(p) = \frac{a}{a+b}$$

kar ustreza relativni frekvenci uspešnih dogodkov ( $a$  uspešnih in  $a+b$  vseh skupaj). Naj bo dana apriorna porazdelitev verjetnosti uspeha  $\beta(a, b)$ . Če je v naših podatkih  $r$  primerov uspešnih in  $n$  vseh primerov, potem je verjetnost uspeha porazdeljena po zakonu  $\beta(a+r, b+n-r)$ . Verjetnost uspeha ocenimo z matematičnim upanjem, ki je podano z naslednjo formulo:

$$p = \frac{r+a}{n+a+b}$$

### *Relativna frekvenca*

Če uporabimo začetno porazdelitev  $\beta(0,0)$ , potem zgornja ocena preide v relativno frekvenco:

$$p = \frac{r}{n} \tag{3.6}$$

Relativna frekvenca ima slabo lastnost, da je pri majhnem številu podatkov ocena ekstremnih verjetnosti 0 in 1 precej pogosta, saj se kaj rado zgodi, da je pri majhnem številu poskusov  $n$ , število uspešnih poskusov enako  $r = 0$  ali  $r = n$ . Taka ocena verjetnosti je pretirano pesimistična/optimistična in se ji raje izogibamo. Če je število podatkov zadosti veliko (npr.  $n > 100$ ), postane relativna frekvenca zanesljiva ocena verjetnosti. V nadaljevanju si bomo ogledali bolj previdne ocene verjetnosti.

### *Laplaceov zakon zaporednosti*

Če uporabimo začetno porazdelitev  $\beta(1,1)$ , ki predstavlja *uniformno apriorno porazdelitev verjetnosti*, potem dobimo *Laplaceov zakon zaporednosti*:

$$p = \frac{r+1}{n+2} \tag{3.7}$$

Ta ocena je bolj previdna od relativne frekvence in zagotavlja, da za vsak  $n \geq 0$  in vsak  $0 \leq r \leq n$  velja  $0 < p < 1$ . To oceno lahko uporabimo, če sta možna dva izida (uspeh in neuspeh). Posplošitev ocene na  $k$  možnih izidov je [26]:

$$p = \frac{r+1}{n+k}$$

Slaba lastnost ocene je, da predpostavlja enakomerno apriorno verjetnost vseh izidov, kar je marsikdaj nesprejemljivo.

### *m-ocena verjetnosti*

Smyth in Goodman [29] ter Cestnik [6] so neodvisno razvili oceno verjetnosti, ki jo imenujemo  $m$ -ocena verjetnosti. Če postavimo  $m = a + b$  in upoštevamo, da je apriorna verjetnost  $p_0$  definirana z

$$p_0 = \frac{a}{a+b}$$

dobimo  $m$ -oceno:

$$p = \frac{r + mp_0}{n+m} = \frac{n}{n+m} \times \frac{r}{n} + \frac{m}{n+m} \times p_0 \quad (3.8)$$

Desna oblika  $m$ -ocene nam kaže, da je  $m$ -ocena sestavljena iz relativne frekvence, ki je utežena z  $n$  (število podatkov), in iz apriorne ocene  $p_0$ , ki je utežena s parametrom  $m$ . Torej lahko  $m$  interpretiramo kot število poskusov, ki podpirajo apriorno verjetnost.

$m$ -ocena je prožna in omogoča izbiro apriorne verjetnosti  $p_0$  ter njeni utežitevi s parametrom  $m$ . Če ni na voljo dovolj predznanja za oceno  $p_0$ , si pomagamo z Laplaceovim zakonom zaporednosti. Parameter  $m$  spremojmo glede na naše predznanje. V praksi se pogosto uporablja vrednost  $m = 2$  [6].

Ostali oceni, relativna frekvence in Laplaceov zakon zaporednosti, sta samo posebna primera  $m$ -ocene. Če nastavimo  $m = 0$ , dobimo relativno frekvenco. Če pa izberemo  $m = k$  in  $p_0 = 1/k$ , pri čemer je  $k$  število možnih izidov, dobimo Laplaceov zakon zaporednosti.

## 3.2 Mere za ocenjevanje učenja

*Vrhovni sodnik vsake fizikalne teorije je eksperiment.*

*Lev Landau in Juriš Rumer*

Pri uporabi strojnega učenja nas zanima, kako uspešno bo izvajalni algoritem z avtomatsko zgrajeno teorijo reševal nove probleme. Posameznim problemom bomo rekli kar primeri problemov ali na kratko *primeri*. Če imamo klasifikacijski problem, želimo vedeti, kako uspešna bo klasifikacija z zgrajeno teorijo. Pri regresiji nas zanima, kako natančne bodo napovedane vrednosti odvisne spremenljivke oziroma s kakšnim zaupanjem lahko verjamemo vrednostim, ki nam jih vrne model. Pri naučenih logičnih relacijah je potrebno oceniti, kolikšen delež novih  $n$ -teric objektov (primerov problemov) bo zgrajena relacija pravilno pokrivala.

Za ocenjevanje uspešnosti avtomatsko zgrajenega znanja ponavadi ločimo razpoložljive podatke, ki opisujejo primere rešenih problemov na dve množici: na učno množico in na testno množico. Učna množica primerov je na voljo algoritmu med učenjem, testna množica pa se uporabi za ocenjevanje uspešnosti zgrajene teorije (glej sliko 3.4).

Pri nekaterih oblikah učenja kvalitete znanja ne moremo oceniti na posameznih primerih odločitev, ampak je potrebno gledati reševanje problema kot celote. Tako je pri problemih odločanja, kjer šele zaporedje odločitev prinese vidne rezultate, npr. vodenje podjetja ali igranje iger. To velja tudi za pravila vodenja dinamičnega sistema, kjer nam šele informacija o dolgotrajni uspešnosti vodenja sistema oceni uspešnost pridobljenega znanja. V tem razdelku se omejimo na ocene, ki upoštevajo samo posamezne primere odločitev. Za analizo uspešnosti klasifikacije se uporabljajo naslednje mere:



Slika 3.4: Ocenjevanje uspešnosti avtomatsko zgrajenih teorij.

- klasifikacijska točnost,
- tabela napačnih klasifikacij,
- cena napačne klasifikacije,
- ocenjevanje napovedi verjetnosti razredov - Brierjeva mera,
- informacijska vsebina odgovora,
- rob (*margin*),
- senzitivnost in specifičnost,
- krivulja ROC ozziroma površina pod njo ter
- priklic in preciznost.

Za analizo uspešnosti regresije se uporabljujo:

- srednja kvadratna napaka,
- srednja absolutna napaka in
- korelacijski koeficient.

Na koncu razdelka je opisana še relacija med pristranskostjo in varianco v strojnem učenju.

### Klasifikacijska točnost

Rešitev vsakega primera klasifikacijskega problema je enolično določen razred iz končne množice  $m_0$  razredov. Uspešnost reševanja klasifikacijskega problema ocenujemo s *klasifikacijsko točnostjo* (*classification accuracy*). Tudi relacijske probleme ocenujemo z isto

mero, če predpostavimo, da je problem pripadnosti relaciji dvorazredni klasifikacijski problem. Klasifikacijska točnost je definirana z:

$$T = \frac{N^{(p)}}{N} \times 100\%$$

kjer je  $N$  število vseh možnih primerov problemov na danem področju in  $N^{(p)}$  število pravilnih rešitev primerov, t.j. število primerov, ki jih dana teorija pravilno klasificira. Klasifikacijsko točnost interpretiramo kot verjetnost, da bo naključno izbran primer pravilno klasificiran.

V realnih domenah je praktično nemogoče izračunati  $T$ , saj ne poznamo pravih rešitev vseh primerov, pa tudi  $N$  je prevelik, če že ne neskončen. Zato klasifikacijsko točnost ocenjujemo na neodvisni testni množici rešenih primerov, če je ta na voljo. Če je  $n_t$  število vseh testnih primerov, je ocena klasifikacijske točnosti dana z:

$$T_t = \frac{n_t^{(p)}}{n_t} \times 100\%$$

Testna množica mora biti neodvisna od učne množice primerov, to je množice rešenih primerov, ki so bili na voljo algoritmu med učenjem. Klasifikacijska točnost na  $n_u$  učnih primerih je podana z:

$$T_u = \frac{n_u^{(p)}}{n_u} \times 100\%$$

in predstavlja optimistično oceno (zgornje mejo) klasifikacijske točnosti. Ni težko sestaviti teorije, ki doseže  $T_u = 100\%$ . Vsaka teorija, ki si pravilno zapomni rešitve (razrede) vseh učnih primerov, ima navidezno visoko uspešnost. Vendar ocena velja samo za učno množico in je zelo slaba ocena dejanske klasifikacijske točnosti. Za teorije, ki imajo  $T_u$  relativno mnogo večji od  $T_t$ , pravimo, da se *preveč prilegajo* učni množici (*overfitting*). Mnogi algoritmi strojnega učenja zahtevajo posebne mehanizme, ki preprečujejo preveliko prileganje teorij učni množici.

Zanimiva je tudi spodnja meja sprejemljive klasifikacijske točnosti, ki jo ocenimo z *večinskim razredom*. Večinski razred je tisti, ki je najbolj zastopan v celotnem prostoru primerov. Ker običajno porazdelitve razredov v celotnem prostoru ne poznamo, jo ocenimo s porazdelitvijo razredov v učni množici. Naj bo  $n_u^{(i)}$  število učnih primerov iz  $i$ -tega razreda. Minimalna (še sprejemljiva) klasifikacijska točnost, ki jo doseže trivialna teorija, ki klasificira vsak primer v večinski razred, je:

$$T_v = \max_i \frac{n_u^{(i)}}{n_u}$$

Da lahko slab klasifikator doseže tudi manjšo klasifikacijsko točnost, nas prepriča naslednji primer. V problemu prognostike ponovitve raka na dojki je zastopanost večinskega razreda 80%. Vendar mnogi klasifikatorji dosegajo klasifikacijsko točnost manjšo od 80%. Razlog je naslednji. Večina parametrov, ki opisujejo paciente, je nepomembnih za prognozo. Klasifikatorji, ki tega ne odkrijejo, uporabljajo te parametre za ocenjevanje napovedi. Ker so parametri nepomembni, je tako klasifikacija bolj ali manj naključna, ponavadi pa obtežena z

tem pa svojih izkušenj ne more preprosto prenesti na mlajše kolege. Zanj veljajo vse slabe lastnosti, kot za vsakogar: je pozabljiv, hitro se utrudi, na njegovo delo vplivajo razpoloženje, vreme, bolezen itd.

Enega osnovnih namenov razvoja metod strojnega učenja smo že nekajkrat omenili: avtomatsko generiranje baz znanja za ekspertne sisteme. Ekspertni sistemi so sposobni pomagati strokovnjakom pri njihovem delu, v izjemnih primerih pa lahko strokovnjake tudi nadomestijo (vsaj začasno, npr. če je strokovnjak odsoten, bolan itd.). Za ekspertne sisteme se zahaja, da znajo uporabniku svoje rešitve tudi obrazložiti in argumentirati. Le takemu sistemu bo človek zaupal in mu prepustil (manj) pomembne odločitve.

Za reševanje zahtevnih specializiranih problemov potrebujejo ekspertni sistemi bazo znanja. Lahko jo sestavimo "ročno" tako, da poskušamo zakodirati pomembne elemente znanja s povzemanjem iz obstoječe literature in spraševanjem strokovnjakov. Tako sestavljanje baze znanja je tipično naporno, zamudno in nekonsistentno.

Druga možnost je avtomatska gradnja baze znanja z algoritmi za induktivno učenje. Znano je, da strokovnjaki lažje opisujejo primere rešenih problemov kot pravila, ki jih uporabljajo pri reševanju problemov. Rešene primere, ki jih je opisal strokovnjak ali zanje obstaja arhivirana baza rešenih problemov, lahko uporabimo za strojno učenje. Rezultat učenja so pravila, ki se uporabijo za reševanje novih problemov z ekspertnim sistemom.

Pri avtomatsko zgrajeni bazi znanja je treba biti pazljiv. Takšno znanje je potrebno preveriti in ovrednotiti. Avtomatsko pridobljeno znanje naj bi bilo človeku razumljivo, zato je pogosto potrebno to znanje popraviti in/ali prilagoditi (npr. spremeniti opisni jezik). Pravilnost avtomatsko zgrajenega znanja je treba ovrednotiti. Najpogostejsa oblika ocenjevanja je delež pravilnih odgovorov na novih problemih, ki niso bili na voljo med učenjem.

Ekspertni sistemi imajo prednosti, ki veljajo za računalnik: so neutrudljivi, zanesljivi, ponovljivi, dostopni 24 ur na dan, 365 dni na leto, zakodirano znanje je prenosljivo, pri odločitvah lahko upoštevajo velike podatkovne baze, zbrane v preteklosti.

Kljub prednostim pa ekspertni sistemi ne morejo in v (bližnji) prihodnosti ne bodo izpodrinali strokovnjakov. Primerjava med računalnikovimi in človeškimi lastnostmi je podana v tabeli 2.3. Kljub izredni hitrosti današnjih računalnikov, ki lahko obdelujejo ogromne količine podatkov, človeškega širokega znanja in spomina še zdaleč ne dosegajo. Ljudje zaradi visoke stopnje paralelizma v možganih izredno hitro in z luhkoto rešijo nekatere vrste problemov, ki jih (današnji) računalniki ne morejo rešiti oziroma bi bila rešitev prepočasna.

Glavna prednost ljudi je, da so prilagodljivi in svoje znanje dinamično spreminjajo in izpopolnjujejo. Prav zaradi tega je razvoj metod strojnega učenja tako pomemben. Z ustreznimi algoritmi je potrebno današnje računalnike narediti manj toge in bolj prilagodljive novim situacijam in problemom. To pomeni, da tudi algoritmi ne bodo več statični, ampak se bodo dinamično spreminjači z učenjem. Zavedati se je treba, da bo obnašanje takih učečih se sistemov nenapovedljivo in bo potrebno vgrajevati mehanizme za preverjanje in nadzor naučenega. Nekateri celo napovedujejo, da bodo inteligentni sistemi morali včasih uporabnikom "lagati", zato da bodo v kritičnih trenutkih dosegli optimalno izvajanje [13]. Tak sistem je težko obvladovati, zato je zaželeno, da obstaja možnost popolnega nadzora sistema.

človek	računalnik
pozabljiv potrebuje počitek težko posreduje znanje	zanesljiv, ponovljiv lahko deluje neprekinjeno trivialno prenosljivo znanje
omejen spomin ? počasen ?	velike podatkovne baze ? hiter ?
široko znanje vzporedno procesiranje se uči iz napak dinamično znanje	ozko specializirano znanje večinoma zaporedno proc. isto napako ponavlja statično znanje

Tabela 2.3: Primerjava lastnosti človeka in računalnika.

## Literatura

- [1] R. Borger, A.M. Seaborne. *The Psychology of Learning*. Harmonds worth: Penguin Books, 1966.
- [2] H. L. Dreyfus. *What Computers Can't Do*. New York: Harper & Row, 1972.
- [3] A. Einstein. Science and religion. *Nature*, 146:605–607, 11 1940.
- [4] D. Goleman. *Čustvena inteligenca na delovnem mestu*. Mladinska knjiga, Ljubljana, 2001.
- [5] J. E. Hopcroft, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- [6] T. Kohonen. *Self-Organization and Associative Memory*. Berlin: Springer-Verlag, 1984.
- [7] I. Kononenko. Znanost in duhovnost. *Sodobnost*, 66(2):259–264, 2002.
- [8] I. Kononenko. Znanost preverja fenomene zavesti. V I. Kononenko, I. Jerman, (ur.), *Mind-body studies : proceedings of 6th International Conference on Cognitive Science*, str. 169–172, 2003. ISBN ISBN 961-6303-50-3. Ljubljana, 13-17th October, Institut Jožef Stefan.
- [9] I. Kononenko. Natural and machine learning, intelligence and consciousness. V E. Žerovnik, O. Markič, A. Ule, (ur.), *Philosophical Insights about Modern Science*, str. 239–258. New York: Nova Science Publishers, 2009.
- [10] E. Lausch. *Manipulation – Der Griff nach dem Gehirn*. Deutsche Verlags-Anstalt, Stuttgart, 1972.
- [11] D. Lorimer, (ur.). *The Spirit of Science*. Floris Books, Edinburgh, 1998.
- [12] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill, 1974.

- [13] D. Michie, R. Johnston. *The Creative Computer: Machine Intelligence and Human Knowledge*. New York: Viking, 1984.
- [14] D. N. Osherson, M. Stob, S. Weinstein. *Systems That Learn*. Bradford Book, The MIT Press, 1986.
- [15] V. Pečjak. *Psihologija spoznavanja*. Državna založba Slovenije, Ljubljana, 1977.
- [16] V. Pečjak. *Pot do znanja: metode uspešnega učenja*. Cankarjeva založba, Ljubljana, 1986.
- [17] B. Rudolf. *Dialektika*. ČZP Komunist, Ljubljana, 1977.
- [18] P. Russell. *The Brain Book: Know Your Own Mind and How to Use It*. Routledge and Kegan Paul, London & Hawthorne, New York, 1979.
- [19] O. Sacks. *The Man Who Mistook His Wife for a Hat and other Clinical Tales*. Harper & Row Pub, 1985.
- [20] P. Schauer. *Interferon*. DDU Univerzum, Ljubljana, 1984.
- [21] E. Shapiro. Inductive inference of theories from facts. Research report 192, Dept. of Computer Sc., Yale University, New Haven, 1981.
- [22] A. Sloman. The emperor's real mind: review of Roger Penrose's The Emperor's New Mind: Concerning computers, minds and laws of physics. *Artificial Intelligence*, 56: 355–396, 1992.
- [23] M. Taube. *Computers and Common Sense: The Myth of Thinking Machines*. New York: Columbia University Press, 1961.
- [24] A. Trstenjak. *Problemi psihologije*. Slovenska matica, Ljubljana, 1976.
- [25] A. Trstenjak. *Človek bitje prihodnosti*. Slovenska matica, Ljubljana, 1985.
- [26] L. Žlebnik. *Pedagoška psihologija*. Visoka šola za organizacijo dela, Kranj, 1982.
- [27] B.A. Wallace. *The Taboo of Subjectivity: Towards a New Science of Consciousness*. Oxford University Press, 2000.

## Poglavlje 3

# Osnove strojnega učenja

*Človek in računalnik sta sposobna doseči, česar nobeden od njiju sam ne zmore.*

*Hubert L. Dreyfus*

Imejmo pet učnih primerov, ki so prikazani v tabeli 3.1. Iz teh podatkov se želimo naučiti zakonitosti pojava, ki je te podatke povzročil. Poleg dejanskih pravil, ki so bila uporabljena pri generiranju podatkov:

$$\text{ploščina} = 1.\text{stranica} \times 2.\text{stranica}$$

$$\text{obseg} = 2 \times 1.\text{stranica} + 2 \times 2.\text{stranica}$$

$$1.\text{stranica} = 2.\text{stranica} \rightarrow \text{kvadrat}$$

so možna pravila, ki bi jih učenec lahko izluščil, tudi npr.:

$$1.\text{stranica} = 5 \text{ ali } 2.\text{stranica} = 5 \rightarrow \text{pravokotnik}$$

$$10 < \text{ploščina} < 30 \rightarrow \text{pravokotnik}$$

$$10 < \text{obseg} < 20 \rightarrow \text{pravokotnik}$$

$$|\text{obseg} - \text{ploščina}| < 4 \rightarrow \text{pravokotnik}$$

$$\text{pravokotnik in } 1.\text{stranica} = 3 \text{ ali } 2.\text{stranica} = 3 \rightarrow \text{obseg} = 16$$

1.stranica	2.stranica	ploščina	obseg	lik
5	4	20	18	pravokotnik
5	3	15	16	pravokotnik
2	2	4	8	kvadrat
3	5	15	16	pravokotnik
6	6	36	24	kvadrat

Tabela 3.1: Množica učnih primerov.

$$\begin{aligned} ploščina &= 15 \leftrightarrow obseg = 16 \\ kvadrat \rightarrow |obseg - ploščina|/2 &= 1.stranica \end{aligned}$$

Kaj mora voditi učenca, da bo spoznal, da so nekatera od pravil bolj verjetna od drugih? Ali sploh lahko karkoli sklepa iz opisov samo petih učnih primerov? Zakaj je npr. pravilo

$$pravokotnik \text{ in } 1.stranica = 3 \text{ ali } 2.stranica = 3 \rightarrow obseg = 16$$

intuitivno nezanimivo, medtem ko se nam zdi pravilo

$$obseg = 2 \times 1.stranica + 2 \times 2.stranica$$

bolj verjetno?

V tem poglavju odgovarjamo na taka in podobna vprašanja. Opisani so osnovni principi, ki nas vodijo pri načrtovanju sistemov strojnega učenja, kot so princip preprostosti in princip večkratne razlage, ter principe ocenjevanja kvalitete naučene hipoteze.

### 3.1 Osnovni principi strojnega učenja

*Strojno učenje je predpogoj inteligenčnega sistema.*

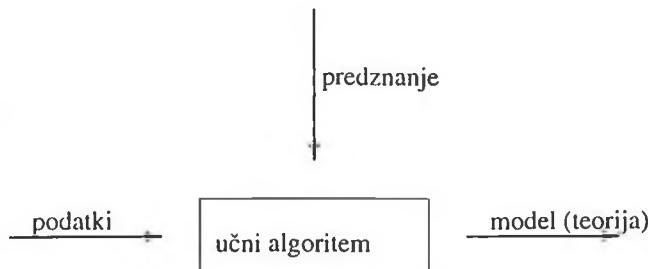
*Alan M.Turing*

V tem razdelku je podana definicija strojnega učenja kot modeliranja podatkov. Opisan je princip najkrajšega opisa, ki pravi, da so najpreprostejše hipoteze, ki razlagajo podatke, najbolj verjetne. Po drugi strani princip večkratne razlage zahteva, da za optimalno napovedovanje uporabimo vse možne hipoteze in ne le najbolj verjetne. Na koncu so opisani še pristopi k ocenjevanju verjetnosti iz podatkov, ki tudi kažejo na to, da je hipoteza tem zanesljivejša, čim več podatkov razlaga.

#### Učenje kot modeliranje

Učenje je vsaka sprememba sistema, ki mu omogoča, da opravlja isto nalog bolje. Rezultat učenja je znanje, ki ga sistem uporabi za reševanje novih nalog. Znanje je lahko množica zapomnjenih podatkov, algoritmom za reševanje določenih nalog ali pa množica napotkov za bolj učinkovito reševanje nalog. Pogosto bomo pri obravnavi sistemov za strojno učenje ločevali med *učnim algoritmom*, ki iz množice podatkov in predznanja tvori novo znanje (oziroma samo spremeni prejšnje znanje), in med *izvajalnim algoritmom*, ki avtomatsko naučeno znanje uporablja za reševanje novih problemov. Avtomatsko naučenemu znanju bomo pogosto rekli kar *model*. Za model zahtevamo, da čim bolj ustrezta vhodnim podatkom in predznanju. Uporabljali bomo tudi izraza *hipoteza* in *teorija*.

Pri uporabi besede *model* se moramo zavedati, da matematična logika uporablja drugačno definicijo modela, ki je intuitivno ravno nasprotna definiciji, ki jo uporabljamo v strojnem



Slika 3.1: Algoritem za strojno učenje.

učenju<sup>1</sup>. Zato bomo pogosto namesto besede "model" uporabljali besedo "hipoteza".

Strojno učenje opredelimo kot opisovanje ali *modeliranje* podatkov. Vhod v sistem za strojno učenje sta množica podatkov ter predznanje, izhod pa opis (model, hipoteza, teorija), ki te podatke skupaj s predznanjem opisuje in pojasnjuje (glej sliko 3.1). Predznanje je ponavadi kar prostor možnih modelov, v katerem bo algoritem iskal tistega, ki čim bolj ustreza vhodnim podatkom, ter kriterij optimalnosti, ki ga bo sistem med iskanjem poskušal izpolniti. Predznanje lahko vsebuje tudi začetno hipotezo, ki je lahko približna rešitev problema, ter množico hevristik, ki služijo za usmerjanje iskanja v bolj obetavne dele prostora.

Strojno učenje lahko predstavimo tudi kot optimizacijski problem. Pri danem prostoru možnih rešitev (modelov) in pri danem kriteriju optimalnosti je treba poiskati tisto rešitev (model), ki zadošča kriteriju optimalnosti oziroma minimizira vrednost kriterijske funkcije. Pri tem je vrednost kriterijske funkcije odvisna od trenutnega modela, predznanja in vhodnih podatkov, ki jih modeliramo. Ker je prostor možnih rešitev ponavadi zelo velik (mnogokrat neskončen), je iskanje optimalne rešitve prezahtevno in se moramo zadovoljiti s čim boljšimi suboptimalnimi rešitvami.

Ločimo naslednje vrste modelov (in s tem tudi zvrsti strojnega učenja):

**Diskrete funkcije** Zaloga napovednih vrednosti je končna neurejena množica. Odvisni spremenljivki pravimo razred. Problemom, katerih model je diskretna funkcija, pravimo *klasifikacijski problemi*. Ko je funkcija naučena, jo uporabljamo za klasifikacijo (uvrščanje), t.j. ugotavljanje vrednosti funkcije pri danih vrednostih neodvisnih spremenljivk. Pogosto je ciljna funkcija mnogolična: isto vrednost domene preslika v več

<sup>1</sup>V logiki je model množice formul (izjav) v danem jeziku definiran kot taka interpretacija jezika, v kateri so vse formule (izjave) iz dane množice resnične [24]. V strojnem učenju pa se za vse vhodne podatke (izjave) predpostavlja, da so resnični, in je model formula, iz katere lahko (do neke mere) izpeljemo vhodne podatke (izjave). Torej je v logiki jezik (formalni sistem) abstrakcija modela. V strojnem učenju je ravno nasprotno: model je abstrakcija podatkov.

različnih vrednosti iz zaloge vrednosti. Ob tem je lahko vsaka vrednost iz zaloge vrednosti utežena, ponavadi z verjetnostmi. Opravka imamo s funkcijami, ki domeno preslikajo v (zvezni) prostor verjetnostnih distribucij zaloge vrednosti. Medicinsko diagnostično pravilo npr. preslika opis stanja pacienta v množico možnih diagnoz, pri čemer je vsaki možni diagnozi pripisana tudi verjetnost.

Mnogi odločitveni problemi, diagnostični problemi, problemi vodenja in problemi napovedovanja se lahko predstavijo kot klasifikacijski problemi. Tipični primeri so medicinska diagnostika in prognostika, napovedovanje vremena, diagnostika industrijskih procesov, klasifikacija izdelkov po kakovosti, vodenje dinamičnih sistemov ipd.

**Zvezne funkcije** Zaloga vrednosti je (potencialno) neskončna urejena množica. Odvisni spremenljivki pravimo regresijska spremenljivka ali zvezni razred. Problemom, katerih model je zvezna funkcija, pravimo *regresijski problemi*. Tako kot pri klasifikaciji tudi tu uporabljamo avtomatsko zgrajeno funkcijo za ugotavljanje vrednosti funkcije pri danih vrednostih neodvisnih spremenljivk. Mnoge probleme napovedovanja in odločitvene probleme lahko opišemo kot regresijske probleme. Primeri so napovedovanje časovnih vrst, vodenje dinamičnih sistemov, ugotavljanje vplivov različnih parametrov na velikost odvisne spremenljivke ipd. Poleg vrednosti funkcije je pogosta zahteva pri regresijskih problemih tudi interval zaupanja. Le-ta verjetnostno opisuje zaupanje v vrednost, ki nam jo model predлага kot rešitev danega primera.

**Relacije** Avtomatsko zgrajene relacije uporabljamo za preverjanje, če je dana  $n$ -terica objektov element relacije, ali pa jih uporabljamo kot funkcije, pri čemer izberemo enega ali več parametrov za odvisno spremenljivko. Relacije so splošnejše od funkcij, kar pomeni, da je ustrezен prostor možnih relacij mnogo večji kot prostor možnih funkcij. Temu primerno je učenje relacij zahtevnejše, tako glede iskanja rešitev kot glede zahtevanega števila vhodnih podatkov in/ali količine predznanja. Tudi relacije so lahko zvezne (sistemi enačb) ali diskretne (logične relacije). Primeri relacijskega učenja so učenje strukture kemijskih spojin, učenje geometrijskih lastnosti objektov, ugotavljanje splošnih zakonitosti v podatkovni bazi ipd.

## Princip najkrajšega opisa

*Nesmotrno je narediti z več, kar lahko storimo z manj.*

*William of Ockham*

*Narava je zadovoljna s preprostostjo in ne hlini odvečnih vzrokov.*

*Isaac Newton*

*Naredi preprosto, kolikor se le da - vendar ne preprosteje!*

*Albert Einstein*

Pri preiskovanju prostora možnih hipotez, ki ustrezajo vhodnim podatkom in predznanju,

je naloga algoritma za strojno učenje poiskati hipotezo, ki "čim bolj ustreza" vhodnim podatkom in predznanju. Vhodnim podatkom lahko "ustreza" več (tudi neskončno mnogo) hipotez. Zato potrebujemo kriterije, ki bodo ocenjevali kvaliteto hipotez. Znan je princip *Ockhamove britve* (*Occam's Razor Principle*), ki pravi, da je najpreprostejša razlaga najbolj zanesljiva. Kot bomo videli v nadaljevanju, lahko ta princip posplošimo na princip najkrajšega opisa podatkov, ki je ekvivalenten principu maksimalne verjetnosti hipoteze.

Kriteriji za usmerjanje učnega algoritma so različni za različne naloge ter lahko nastopajo v kombinaciji z drugimi kriteriji, npr.:

- maksimizirati klasifikacijsko točnost hipoteze,
- minimizirati povprečno ceno klasifikacijskih napak,
- minimizirati velikost hipoteze,
- maksimizirati prileganje hipoteze vhodnim podatkom,
- maksimizirati razumljivost hipoteze,
- minimizirati časovno zahtevnost klasifikacije,
- minimizirati število parametrov, potrebnih za klasifikacijo,
- minimizirati ceno pridobivanja vrednosti parametrov, potrebnih za klasifikacijo,
- maksimizirati verjetnost hipoteze glede na dano predznanje in vhodne podatke.

Zadnji princip, t.j. *maksimizirati verjetnost hipoteze*, je najsplošnejši in ima lepo lastnost, da zadošča večini zgoraj omenjenih kriterijev. Izkušnje kažejo, da ima najbolj verjetna hipoteza (v povprečju preko vseh možnih učnih problemov) tudi maksimalno klasifikacijsko točnost, minimalno velikost pri dani klasifikacijski točnosti in se najbolj prilega vhodnim podatkom pri dani klasifikacijski točnosti. Ker je hipoteza najmanjša, je v tem smislu navadno tudi najbolj razumljiva, uporablja minimalno število parametrov, da doseže dano klasifikacijsko točnost, in omogoča najhitrejšo klasifikacijo. Ob predpostavki enake cene vseh klasifikacijskih napak in enake cene za pridobivanje vrednosti posameznih parametrov kriterij maksimalne verjetnosti hipoteze ustreza tudi obema kriterijema, ki upoštevata ceno.

Označimo množico možnih hipotez s  $\mathcal{H}$ , hipotezo s  $H \in \mathcal{H}$ , predznanje z  $B$  in vhodne podatke z  $E$ . V idealnem primeru, ko poznamo verjetnosti  $P(H|E, B)$  posameznih hipotez pri danem predznanju in vhodnih podatkih, je zaželeno poiskati hipotezo, ki maksimizira pogojno verjetnost:

$$H_{opt} = \arg \max_{H \in \mathcal{H}} P(H|E, B) \quad (3.1)$$

Na žalost so (apriorne in pogojne) verjetnosti hipotez neznane in zato gornji kriterij nadomestimo z različnimi hevrističnimi kriteriji, ki se mu poskušajo približati.

Rissanen je predlagal princip najkrajšega opisa (*minimal description length (MDL) principle*), ki je ekvivalenten kriteriju maksimalne verjetnosti [23]. Naj bo  $P(H|B)$  (apriorna)

verjetnost hipoteze  $H$  pri danem predznanju  $B$ . Definirajmo količino informacije  $I(H|B)$ , da zvemo, da je hipoteza  $H$  resnična pri danem predznanju  $B$ , z:

$$I(H|B) = -\log_2 P(H|B) \text{ [bit]}$$

ter analogno temu pogojno količino informacije pri danem predznanju  $B$  in vhodnih podatkih  $E$  z:

$$I(H|E, B) = -\log_2 P(H|E, B) \text{ [bit]}$$

Potem lahko (3.1) zapišemo z:

$$H_{opt} = \arg \min_{H \in \mathcal{H}} I(H|E, B) \quad (3.2)$$

Ker po Bayesovem izreku velja

$$I(H|B, E) = I(E|H, B) + I(H|B) - I(E|B)$$

in ker lahko vzamemo, da je  $I(E|B)$  konstanta, lahko kriterij (3.2) zapišemo kot

$$H_{opt} = \arg \min_{H \in \mathcal{H}} (I(E|H, B) + I(H|B)) \quad (3.3)$$

Torej kriterij zahteva, da poiščemo hipotezo, ki minimizira vsoto dolžin opisov hipoteze in vhodnih podatkov pri dani hipotezi. Dejansko to pomeni, da iščemo kompromis med velikostjo hipoteze  $I(H|B)$  in velikostjo njene napake  $I(E|H, B)$ .

Problem minimizacije dolžine opisa množice podatkov lahko v kontekstu komunikacijske teorije definiramo takole. Oddajnik mora sprejemniku preko komunikacijskega kanala poslati opis množice podatkov. Preko komunikacijskega kanala lahko pošlje sporočilo kot niz ničel in enic. Dolžina sporočila predstavlja število bitov, potrebnih za kodiranje opisa podatkov. Pri kodiranju je treba predpostaviti optimalnost kodiranja, sprejemnik pa mora biti sposoben sporočilo razumeti oziroma dekodirati. To pomeni, da je dekoder sprejemniku znan, ali pa ga mora pošiljalatelj zakodirati in poslati skupaj z opisom podatkov, kar podaljša celotno sporočilo še za dolžino opisa dekoderja. Naloga oddajnika je minimizirati dolžino sporočila.

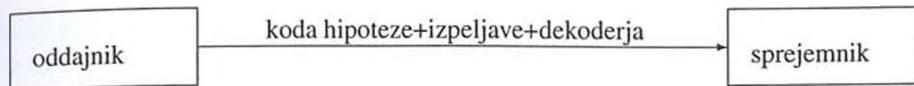
Oddajnik lahko zakodira podatke direktno in jih pošlje sprejemniku, ali pa zakodira hipotezo (model) ter še izpeljave originalnih podatkov iz dane hipoteze (glej sliko 3.2). Če je opis modela skupaj z opisom izpeljave podatkov iz hipoteze krajši od opisa originalnih podatkov, pravimo, da je hipoteza *kompresivna*:

$$I(H|B) + I(E|H, B) < I(E|B)$$

Za preverjanje tega kriterija ni potrebno poznati verjetnosti podatkov  $P(E|B)$ , ampak zadošča dolžina opisa podatkov  $I(E|B)$ . Dolžino lahko ocenimo tako, da predpostavimo suboptimálno kodirno shemo. Ker je določitev optimalnega kodiranja neizračunljiv problem [23], se zadovoljimo s tem približkom.

Po kriteriju najkrajšega opisa so sprejemljive samo kompresivne hipoteze. V nasprotnem primeru uporabimo originalne podatke kot trivialen model.

Optimalna hipoteza je kratka ( $I(H|B)$ ), vendar nudi dovolj informacije, da lahko z njenim pomočjo izpeljemo originalne vhodne podatke z minimalno dodatno informacijo ( $I(E|H, B)$ ). Pri izpeljavi smo predpostavili, da imata oba, sprejemnik in oddajnik, na voljo predznanje  $B$ . Če sprejemnik predznanja nima, lahko predznanje interpretiramo kot del vhodnih podatkov  $E$  in pošljemo še to.



Slika 3.2: Komunikacijski kanal.

### Inkrementalno (sprotno) učenje

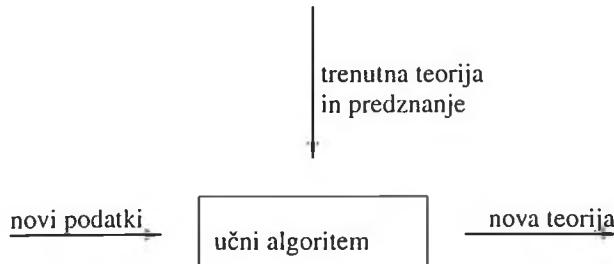
*Narava nikoli ne pove, če so domneve pravilne. Znanstveni uspeh pomeni postaviti pravilno hipotezo, od katere pozneje nikoli ne odstopamo.*

*Osherson in sod. (1986)*

Učni algoritem na sliki 3.1 predpostavlja, da so vsi podatki na voljo že na začetku učenja. Ta scenarij lahko posplošimo tako, da učni algoritem dobiva vhodne podatke drugega za drugim. Po vsakem vhodnem podatku učni algoritem svojo trenutno hipotezo popravi tako, da ustrezta (vsem) prej videnim podatkom in novemu podatku (glej sliko 3.3).

Učenju, kjer učenec sproti spreminja teorijo ob spremenjanju novih podatkov, pravimo *inkrementalno (incremental) ali sprotno učenje (on-line learning)*. Najpreprostejši inkrementalni algoritem ob vsakem novem vhodnem podatku zavriže prejšnjo hipotezo in začne z učenjem "na novo". Takih algoritmov ne bomo prištevali k inkrementalnim. Inkrementalni algoritem poskuša poiskati najmanjšo potrebno spremembo trenutne teorije, tako da le-ta ustrezta vsem do sedaj vidениm podatkom. Včasih inkrementalni algoritmi "pozabijo" vhodne podatke, ki so jih že videli in je nova teorija v nasprotju s prejšnjimi podatki. Ta lastnost je prednost v primerih, ko se celoten problem dinamično spreminja, torej se spreminja tudi optimalna hipoteza.

Pričakovali bi, da bo inkrementalno učenje z dodajanjem novih podatkov konvergiralo k optimalni teoriji za obravnavani problem in da se od nekega trenutka naprej teorija ne bo več spreminja kljub novim vhodnim podatkom. Na žalost pa učni algoritem nikoli z gotovostjo ne ve, če se je naučil optimalne teorije, saj je lahko vsaka naslednja množica vhodnih podatkov v nasprotju s trenutno teorijo. Kljub temu je v praksi pogosto res, da od nekega trenutka naprej dodajanje novih podatkov ne izboljša naučene teorije.



Slika 3.3: Inkrementalno učenje.

### Princip večkratne razlage

*Če je več teorij konsistentnih z dejstvi, obdrži vse teorije.*

*Epicurus*

*Za vsak posamezen model poišči njegovo napoved in jo obteži z verjetnostjo tega modela. Obtežena vsota takih napovedi je optimalna napoved.*

*Peter Cheeseman*

Princip Ockhamove britve je znan in ga uporablja večina sistemov za strojno učenje predvsem v smislu hevristik, ki dajejo prednost preprostejšim teorijam. Manj znan je *princip večkratne razlage (principle of multiple explanations)*, ki pravi, da je treba obdržati vse hipoteze, ki so konsistentne z vhodnimi podatki [23]. Ta princip je navidezno v nasprotju s principom Ockhamove britve. Dejansko pa se izkaže, da se principa dopolnjujeta. Princip Ockhamove britve lahko uporabimo za iskanje najboljše hipoteze oziroma za iskanje množice relativno dobrih hipotez. Ta princip usmerja učni algoritem v iskanje boljših hipotez. Princip večkratne razlage pa s kombinacijo več verjetnih hipotez daje najboljše rezultate pri uporabi avtomatsko zgrajenih hipotez, torej služi za usmerjanje izvajjalnega algoritma.

Gre za povezavo s Centralnim limitnim teoremom, kjer z večanjem števila spremenljivk povprečje konvergira k pravi verjetnosti:

$$P = \lim_{k \rightarrow \infty} \frac{P_1 + P_2 + \dots + P_k}{k}$$

Iзвajalni algoritmi najpogosteje uporabljajo samo eno hipotezo, ki se je izkazala za najoptimalnejšo med pregledanimi hipotezami. Naj bo  $\mathcal{R}$  množica rešitev in  $r \in \mathcal{R}$  konkretna rešitev

problema. Naj bo  $P(r|H)$  verjetnost rešitve  $r$ , če je hipoteza  $H$  pravilna. Potem je rešitev, ki jo ponuja optimalna hipoteza, definirana z:

$$r_{opt1} = \arg \max_{r \in \mathcal{R}} P(r|H_{opt}) \quad (3.4)$$

Toda to ni optimalna rešitev, saj princip večkratne razlage pravi, da je potrebno uporabiti vse možne hipoteze. Ta princip formaliziramo tako, da odgovore vsake hipoteze obtežimo z verjetnostjo hipoteze. Optimalni izvajalni algoritem uporablja vse (dovolj) verjetne hipoteze:

$$r_{opt} = \arg \max_{r \in \mathcal{R}} \sum_{H \in \mathcal{H}} (P(r|H)P(H|E, B)) \quad (3.5)$$

Pri tem nam princip Ockhamove britve pomaga eliminirati hipoteze z zanemarljivo majhno verjetnostjo  $P(H|E, B)$ . Ostane problem aproksimacije verjetnosti  $P(H|E, B)$  za bolj verjetne hipoteze, ki jih upoštevamo v enačbi (3.5).

Navidezno nasprotje dobimo, če interpretiramo kombinacijo več teorij kot eno (najboljšo) teorijo. Toda v tem primeru smo povečali originalni prostor možnih teorij. Če smo imeli prej  $n$  možnih teorij, je po tej interpretaciji prostor možnih teorij najmanj potenčna množica prejšnjega prostora teorij, ki vsebuje  $2^n$  možnih teorij (z upoštevanjem vseh možnih verjetnostnih distribucij po hipotezah pa je prostor še bistveno večji). Spremeni se tudi optimalno kodiranje teorij in s tem kriterij najkrajše dolžine opisa.

## Ocenjevanje verjetnosti

*Verjetnost je pravzaprav natančnejši opis zdrugega razuma.*

### Laplace

Pri strojnem učenju se pogosto srečamo z ocenjevanjem verjetnosti. Verjetnost je bodisi dana vnaprej in je vsebovana v predznanju bodisi jo moramo oceniti iz vhodnih podatkov. V slednjem primeru je pogosto potrebno oceniti verjetnost iz majhne množice vhodnih podatkov. Taka ocena verjetnosti je nezanesljiva, zato ji ne smemo preveč zaupati.

Verjetnost dogodka ocenimo s pomočjo relativne frekvence dogodka iz učne množice. Pri tem je optimalna ocena verjetnosti odvisna od predpostavljenih apriorne porazdelitve verjetnosti [7]. Pogosto se uporablja porazdelitev  $\beta(a, b)$  zaradi lepih lastnosti, ki omogočijo preproste izpeljave ocen verjetnosti. Porazdelitev  $\beta(a, b)$  ima gostoto definirano z:

$$p(x) = \begin{cases} \frac{1}{B(a,b)} x^{a-1} (1-x)^{b-1} & 0 \leq x \leq 1 \\ 0 & \text{sicer} \end{cases}$$

$a > 0$  in  $b > 0$  sta parametra porazdelitve,  $B(a, b)$  pa je beta funkcija, ki je definirana z:

$$B(a, b) = \int_0^1 x^{a-1} (1-x)^{b-1} dx$$

Parametra  $a$  in  $b$  lahko interpretiramo kot  $a$  uspešnih in  $b$  neuspešnih neodvisnih poskusov. Če je slučajna spremenljivka  $p$  porazdeljena po porazdelitvi  $\beta(a, b)$ , je njeno matematično

upanje enako

$$\text{Exp}(p) = \frac{a}{a+b}$$

kar ustreza relativni frekvenci uspešnih dogodkov ( $a$  uspešnih in  $a+b$  vseh skupaj). Naj bo dana apriorna porazdelitev verjetnosti uspeha  $\beta(a, b)$ . Če je v naših podatkih  $r$  primerov uspešnih in  $n$  vseh primerov, potem je verjetnost uspeha porazdeljena po zakonu  $\beta(a+r, b+n-r)$ . Verjetnost uspeha ocenimo z matematičnim upanjem, ki je podano z naslednjo formulo:

$$p = \frac{r+a}{n+a+b}$$

### *Relativna frekvenca*

Če uporabimo začetno porazdelitev  $\beta(0,0)$ , potem zgornja ocena preide v relativno frekvenco:

$$p = \frac{r}{n} \tag{3.6}$$

Relativna frekvenca ima slabo lastnost, da je pri majhnem številu podatkov ocena ekstremnih verjetnosti 0 in 1 precej pogosta, saj se kaj rado zgodi, da je pri majhnem številu poskusov  $n$ , število uspešnih poskusov enako  $r = 0$  ali  $r = n$ . Taka ocena verjetnosti je pretirano pesimistična/optimistična in se ji raje izogibamo. Če je število podatkov zadosti veliko (npr.  $n > 100$ ), postane relativna frekvenca zanesljiva ocena verjetnosti. V nadaljevanju si bomo ogledali bolj previdne ocene verjetnosti.

### *Laplaceov zakon zaporednosti*

Če uporabimo začetno porazdelitev  $\beta(1,1)$ , ki predstavlja *uniformno apriorno porazdelitev verjetnosti*, potem dobimo *Laplaceov zakon zaporednosti*:

$$p = \frac{r+1}{n+2} \tag{3.7}$$

Ta ocena je bolj previdna od relativne frekvence in zagotavlja, da za vsak  $n \geq 0$  in vsak  $0 \leq r \leq n$  velja  $0 < p < 1$ . To oceno lahko uporabimo, če sta možna dva izida (uspeh in neuspeh). Posplošitev ocene na  $k$  možnih izidov je [26]:

$$p = \frac{r+1}{n+k}$$

Slaba lastnost ocene je, da predpostavlja enakomerno apriorno verjetnost vseh izidov, kar je marsikdaj nesprejemljivo.

### *m-ocena verjetnosti*

Smyth in Goodman [29] ter Cestnik [6] so neodvisno razvili oceno verjetnosti, ki jo imenujemo  $m$ -ocena verjetnosti. Če postavimo  $m = a + b$  in upoštevamo, da je apriorna verjetnost  $p_0$  definirana z

$$p_0 = \frac{a}{a+b}$$

dobimo  $m$ -oceno:

$$p = \frac{r + mp_0}{n+m} = \frac{n}{n+m} \times \frac{r}{n} + \frac{m}{n+m} \times p_0 \quad (3.8)$$

Desna oblika  $m$ -ocene nam kaže, da je  $m$ -ocena sestavljena iz relativne ferkvence, ki je utežena z  $n$  (število podatkov), in iz apriorne ocene  $p_0$ , ki je utežena s parametrom  $m$ . Torej lahko  $m$  interpretiramo kot število poskusov, ki podpirajo apriorno verjetnost.

$m$ -ocena je prožna in omogoča izbiro apriorne verjetnosti  $p_0$  ter njeni utežitev s parametrom  $m$ . Če ni na voljo dovolj predznanja za oceno  $p_0$ , si pomagamo z Laplaceovim zakonom zaporednosti. Parameter  $m$  spremenjamamo glede na naše predznanje. V praksi se pogosto uporablja vrednost  $m = 2$  [6].

Ostali oceni, relativna frekvenca in Laplaceov zakon zaporednosti, sta samo posebna primera  $m$ -ocene. Če nastavimo  $m = 0$ , dobimo relativno frekvenco. Če pa izberemo  $m = k$  in  $p_0 = 1/k$ , pri čemer je  $k$  število možnih izidov, dobimo Laplaceov zakon zaporednosti.

## 3.2 Mere za ocenjevanje učenja

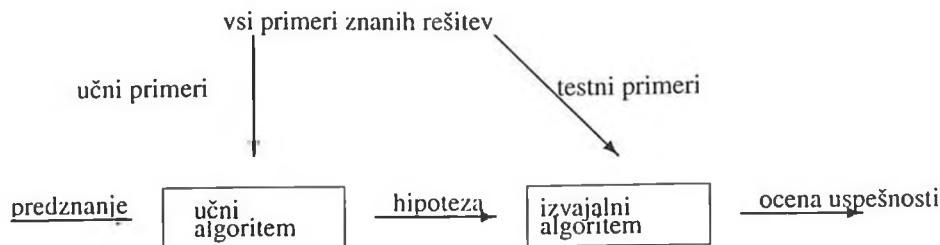
*Vrhovni sodnik vsake fizikalne teorije je eksperiment.*

*Lev Landau in Juriš Rumer*

Pri uporabi strojnega učenja nas zanima, kako uspešno bo izvajalni algoritem z avtomatsko zgrajeno teorijo reševal nove probleme. Posameznim problemom bomo rekli kar primeri problemov ali na kratko *primeri*. Če imamo klasifikacijski problem, želimo vedeti, kako uspešna bo klasifikacija z zgrajeno teorijo. Pri regresiji nas zanima, kako natančne bodo napovedane vrednosti odvisne spremenljivke oziroma s kakšnim zaupanjem lahko verjamemo vrednostim, ki nam jih vrne model. Pri naučenih logičnih relacijah je potrebno oceniti, kolikšen delež novih  $n$ -teric objektov (primerov problemov) bo zgrajena relacija pravilno pokrivala.

Za ocenjevanje uspešnosti avtomatsko zgrajenega znanja ponavadi ločimo razpoložljive podatke, ki opisujejo primere rešenih problemov na dve množici: na učno množico in na testno množico. Učna množica primerov je na voljo algoritmu med učenjem, testna množica pa se uporabi za ocenjevanje uspešnosti zgrajene teorije (glej sliko 3.4).

Pri nekaterih oblikah učenja kvalitete znanja ne moremo oceniti na posameznih primerih odločitev, ampak je potrebno gledati reševanje problema kot celote. Tako je pri problemih odločanja, kjer šele zaporedje odločitev prinese vidne rezultate, npr. vodenje podjetja ali igranje iger. To velja tudi za pravila vodenja dinamičnega sistema, kjer nam šele informacija o dolgotrajni uspešnosti vodenja sistema oceni uspešnost pridobljenega znanja. V tem razdelku se omejimo na ocene, ki upoštevajo samo posamezne primere odločitev. Za analizo uspešnosti klasifikacije se uporabljajo naslednje mere:



Slika 3.4: Ocenjevanje uspešnosti avtomatsko zgrajenih teorij.

- klasifikacijska točnost,
- tabela napačnih klasifikacij,
- cena napačne klasifikacije,
- ocenjevanje napovedi verjetnosti razredov - Brierjeva mera,
- informacijska vsebina odgovora,
- rob (*margin*),
- senzitivnost in specifičnost,
- krivulja ROC oziroma površina pod njo ter
- priklic in preciznost.

Za analizo uspešnosti regresije se uporabljajo:

- srednja kvadratna napaka,
- srednja absolutna napaka in
- korelacijski koeficient.

Na koncu razdelka je opisana še relacija med pristranskoščjo in varianco v strojnem učenju.

### Klasifikacijska točnost

Rešitev vsakega primera klasifikacijskega problema je enolično določen razred iz končne množice  $m_0$  razredov. Uspešnost reševanja klasifikacijskega problema ocenjujemo s *klasifikacijsko točnostjo* (*classification accuracy*). Tudi relacijske probleme ocenjujemo z isto

mero, če predpostavimo, da je problem pripadnosti relaciji dvorazredni klasifikacijski problem. Klasifikacijska točnost je definirana z:

$$T = \frac{N^{(p)}}{N} \times 100\%$$

kjer je  $N$  število vseh možnih primerov problemov na danem področju in  $N^{(p)}$  število pravilnih rešitev primerov, t.j. število primerov, ki jih dana teorija pravilno klasificira. Klasifikacijsko točnost interpretiramo kot verjetnost, da bo naključno izbran primer pravilno klasificiran.

V realnih domenah je praktično nemogoče izračunati  $T$ , saj ne poznamo pravih rešitev vseh primerov, pa tudi  $N$  je prevelik, če že ne neskončen. Zato klasifikacijsko točnost ocenjujemo na neodvisni testni množici rešenih primerov, če je ta na voljo. Če je  $n_t$  število vseh testnih primerov, je ocena klasifikacijske točnosti dana z:

$$T_t = \frac{n_t^{(p)}}{n_t} \times 100\%$$

Testna množica mora biti neodvisna od učne množice primerov, to je množice rešenih primerov, ki so bili na voljo algoritmu med učenjem. Klasifikacijska točnost na  $n_u$  učnih primerih je podana z:

$$T_u = \frac{n_u^{(p)}}{n_u} \times 100\%$$

in predstavlja optimistično oceno (zgornjo mejo) klasifikacijske točnosti. Ni težko sestaviti teorije, ki doseže  $T_u = 100\%$ . Vsaka teorija, ki si pravilno zapomni rešitve (razrede) vseh učnih primerov, ima navidezno visoko uspešnost. Vendar ocena velja samo za učno množico in je zelo slaba ocena dejanske klasifikacijske točnosti. Za teorije, ki imajo  $T_u$  relativno mnogo večji od  $T_t$ , pravimo, da se *preveč prilegajo* učni množici (*overfitting*). Mnogi algoritmi strojnega učenja zahtevajo posebne mehanizme, ki preprečujejo preveliko prileganje teorij učni množici.

Zanimiva je tudi spodnja meja sprejemljive klasifikacijske točnosti, ki jo ocenimo z *večinskim razredom*. Večinski razred je tisti, ki je najbolj zastopan v celotnem prostoru primerov. Ker običajno porazdelitev razredov v celotnem prostoru ne poznamo, jo ocenimo s porazdelitvijo razredov v učni množici. Naj bo  $n_u^{(i)}$  število učnih primerov iz  $i$ -tega razreda. Minimalna (še sprejemljiva) klasifikacijska točnost, ki jo doseže trivialna teorija, ki klasificira vsak primer v večinski razred, je:

$$T_v = \max_i \frac{n_u^{(i)}}{n_u}$$

Da lahko slab klasifikator doscže tudi manjšo klasifikacijsko točnost, nas prepriča naslednji primer. V problemu prognostike ponovitve raka na dojki je zastopanost večinskega razreda 80%. Vendar mnogi klasifikatorji dosegajo klasifikacijsko točnost manjšo od 80%. Razlog je naslednji. Večina parametrov, ki opisujejo paciente, je nepomembnih za prognozo. Klasifikatorji, ki tega ne odkrijejo, uporabljajo te parametre za ocenjevanje napovedi. Ker so parametri nepomembni, je taka klasifikacija bolj ali manj naključna, ponavadi pa obtežena z

pravi razred	napovedani razred			vsota
	C1	C2	C3	
C1	12.3	2.4	8.5	23.2
C2	5.5	58.7	2.1	66.3
C3	0.0	2.0	8.5	10.5
vsota	17.8	63.1	19.1	100.0

Tabela 3.2: Primer matrike zmot.

apriornimi verjetnostmi razredov. Ker sta apriori verjetnosti razredov  $P_1 = 0.8$  in  $P_2 = 0.2$ , je klasifikacijska točnost takega napovedovanja:

$$P_1^2 + P_2^2 = 68\%$$

### Matrika zmot

Klasifikacijska točnost ne pove ničesar o tem, kako dobro so klasificirani primeri iz posameznih razredov, saj je točnost povprečena preko vseh razredov. V praksi nas pogosto zanima uspešnost za vsak razred posebej. Popolno informacijo nudi matrika zmot (*confusion matrix*), t.j. matrika odstotkov napačnih klasifikacij. Tabela 3.2 podaja primer za trirazredni klasifikacijski problem. Na diagonali notranje ( $3 \times 3$ ) matrike so podani odstotki pravilnih klasifikacij. Vsota procentov vsake vrstice predstavlja apriorno verjetnost ustreznega razreda, vsota stolpcev pa delež primerov, klasificiranih v posamezen razred. Iz odstopanja dveh vsot sklepamo na pristranskost klasifikatorja (npr. za ali proti večinskemu razredu).

### Cena napačne klasifikacije

V nekaterih problemih je napačna klasifikacija primerov iz nekaterih razredov hujša napaka kot napačna klasifikacija primerov iz drugih razredov. Če npr. napačno opredelimo nerizičnega pacienta kot rizičnega, je to manjša napaka kot obratno. Zato je v takih klasifikacijskih problemih glavni kriterij uspešnosti povprečna cena klasifikacijske napake (*misclassification cost*). Naj bo  $n_t^{(ij)}$  število testnih primerov, ki pripadajo  $i$ -temu razredu in jih teorija klasificira v  $j$ -ti razred. Velja, da je število pravilno klasificiranih testnih primerov:

$$n_t^{(p)} = \sum_{i=1}^{m_0} n_t^{(ii)}$$

Naj bo podana matrika cen napačnih klasifikacij  $C_{ij}$ , kjer je ponavadi  $C_{ii} = 0$ ,  $i = 1, \dots, m_0$ . Matrika cen ni nujno simetrična, ker za nek  $i$  in  $j$  lahko velja  $C_{ij} \neq C_{ji}$ . Povprečna cena napačne klasifikacije je podana z:

$$C_t = \frac{\sum_{i,j} (C_{ij} n_t^{(ij)})}{n_t}$$

V dvorazrednem problemu običajno klasificiramo primer v bolj verjetni razred. Prag odločitve je 0.5. Če prag odločitve povečamo, damo prednost manj verjetnemu razredu. Na ta način lahko zmanjšamo število dražjih napak (in povečamo število cenejših napak).

### Ocenjevanje napovedi verjetnosti razredov - Brierjeva mera

Mnogi klasifikatorji pri klasifikaciji novega primera namesto razreda vrnejo verjetnostno razdelitev po razredih. V takih primerih se za izračun klasifikacijske točnosti predpostavi, da je primer klasificiran v razred z največjo verjetnostjo. Ta način ocenjevanja klasifikacijske točnosti ne upošteva vse informacije, ki jo klasifikator nudi. Ocena uspešnosti bi morala upoštevati tako apriorne verjetnosti razredov kot tudi verjetnosti, ki jih vrne klasifikator. Informacijska vsebina odgovora, ki upošteva oboje, je opisana v naslednjem podrazdelku. Tukaj pa vpeljimo Brierjevo mero (*Brier score*), ki upošteva samo verjetnosti, ki jih vrne klasifikator [5].

Naj bo  $m_0$  število razredov,  $r^{(j)}$  razred  $j$ -tega testnega primera in  $P'_j(r_i)$ ,  $i = 1..m_0$  verjetnostna distribucija, ki jo vrne klasifikator za  $j$ -ti testni primer. Ciljna (idealna) verjetostna distribucija je torej  $P_j(r^{(j)}) = 1$  in  $P_j(r_i) = 0, r_i \neq r^{(j)}$ . Naslednja mera oceni različnost med ciljno distribucijo in dejansko napovedano distribucijo preko vseh  $n_t$  testnih primerov:

$$\text{Brier} = \text{MSE}_P = \frac{\sum_{j=1}^{n_t} \sum_{i=1}^{m_0} (P_j(r_i) - P'_j(r_i))^2}{n_t} \quad (3.9)$$

Gre za povprečno kvadratno napako napovedi verjetnostne distribucije (*MSE = Mean Squared Error*). V idealnem primeru, če klasifikator vedno natančno, z verjetnostjo  $P'_j(r^{(j)}) = 1$  napove pravilni razred, je mera  $\text{Brier} = 0$ . V najslabšem primeru pa klasifikator napove z verjetnostjo 1 napačen razred in je mera  $\text{Brier} = 2$  (1 dobimo za napačno neklasificiran pravi razred plus 1 za napačno klasifikacijo v nepravi razred). Kvaliteto klasifikatorja ocenimo z:  $1 - \text{Brier}/2$ .

### Informacijska vsebina odgovora

Ocena uspešnosti bi morala upoštevati tako apriorne verjetnosti razredov kot tudi verjetnosti, ki jih vrne klasifikator. Oglejmo si primer. V prognostiki ponovitve raka na dojki klasifikatorji dosegajo okoli 80% klasifikacijske točnosti, pri problemu lokalizacije primarnega tumorja pa dosegajo okoli 45% točnost. Navidezno so v prvem primeru klasifikatorji uspešni, medtem ko je v drugem primeru uspešnost porazna. Sliko nekoliko spremeni podatek, da sta v prvem primeru možna samo 2 razreda, v drugem pa je možnih 22 razredov. Poleg tega je v primeru prognostike ponovitve raka na dojki večinski razred zastopan v 80% primerov, medtem ko v problemu lokalizacije primarnega tumorja samo v 25% primerov. Klasifikator, ki doseže 80% klasifikacijsko točnost pri prvem problemu, je neuporaben, saj je taka klasifikacijska točnost trivialno dosegljiva:  $T_v = 80\%$ . Klasifikacijska točnost 45% v drugem problemu pa je relativno dobra.

Mera, ki oceni težavnost klasifikacijskega problema, je pričakovana količina informacije, ki jo potrebujemo za klasifikacijo enega primera. Tej meri pravimo *entropija razredov*. Če

označimo apriorno verjetnost  $i$ -tega razreda z

$$P(r_i) = \frac{n_u^{(i)}}{n_u}$$

je potrebna količina informacije, da zvemo, da primer pripada  $i$ -temu razredu, enaka:

$$H(r_i) = -\log_2 P(r_i) \quad [bit]$$

in entropija razredov enaka:

$$H(R) = -\sum_{i=1}^{m_0} (P(r_i) \log_2 P(r_i))$$

Pri tem  $R$  označuje odvisno spremenljivko, ki ima zalogo vrednosti  $R \in \{r_1, \dots, r_{m_0}\}$ . Čim večja je entropija, tem težji je klasifikacijski problem.  $H(R)$  doseže maksimum ( $\log_2 m_0$ ), ko so vsi razredi enako verjetni:

$$P(r_i) = \frac{1}{m_0}, \quad i = 1, \dots, m_0$$

ter minimum, ko so vsi primeri iz istega razreda  $r_j$ :

$$P(r_j) = 1, \quad \text{in} \quad P(r_i) = 0, \quad i \neq j$$

Mera, ki upošteva apriorne verjetnosti razredov, je *povprečna informacijska vsebina odgovora* (information score). Naj bo  $r^{(j)}$  pravilni razred  $j$ -tega testnega primera ter  $P'(r^{(j)})$  aposteriorna verjetnost tega razreda za dani testni primer, kot jo vrne klasifikator. Povprečna informacijska vsebina odgovora [19] je definirana z:

$$Inf = \frac{\sum_{j=1}^{n_l} Inf_j}{n_l} \quad [bit] \quad (3.10)$$

in

$$Inf_j = \begin{cases} -\log_2 P(r^{(j)}) + \log_2 P'(r^{(j)}), & P'(r^{(j)}) \geq P(r^{(j)}) \\ -(-\log_2(1 - P(r^{(j)})) + \log_2(1 - P'(r^{(j)}))), & P'(r^{(j)}) < P(r^{(j)}) \end{cases}$$

Če je vrnjena verjetnost pravilnega razreda večja od apriorne verjetnosti, je informacijska vsebina pozitivna, saj je dobljena informacija pravilna. Lahko jo interpretiramo kot razliko med začetno "nevednostjo" in nevednostjo, ki ostane po uporabi klasifikatorja. Bolj formalno je razlika med (apriorno) potrebno informacijo za pravilno klasifikacijo in preostalo (aposteriorno) potrebno količino informacije.

Če je vrnjena verjetnost pravilnega razreda manjša od apriorne verjetnosti, je informacijska vsebina negativna, saj je dobljena informacija napačna. Lahko jo (brez prvega minusa) interpretiramo kot: (apriorna) potrebna informacija za napačno klasifikacijo minus preostala (aposteriorna) potrebna količina informacije za napačno informacijo.

Razliko med klasifikacijsko točnostjo in informacijsko vsebino odgovora ilustriramo z naslednjim primerom. Imejmo dva možna razreda in naj bosta apriorni verjetnosti  $P(r_1) =$

0.8 in  $P(r_2) = 0.2$ . Klasifikator za dani primer vrne aposteriorni verjetnosti  $P'(r_1) = 0.6$  in  $P'(r_2) = 0.4$ . Če je pravilni razred  $r_1$ , potem klasifikacijska točnost tak odgovor ocenjuje kot pravilen, medtem ko ga informacijska vsebina oceni kot napačnega (zavajajočega), saj je negativna. Če pa je pravilni razred  $r_2$ , je odgovor po kriteriju klasifikacijske točnosti napacen, v primeru informacijske vsebine pa pravilen in je informacijska vsebina pozitivna.

V primeru idealnega klasifikatorja, ki vedno vrne verjetnost pravilnega razreda enako 1, je povprečna informacijska vsebina enaka entropiji. Entropija postavlja zgornjo mejo, ki jo informacijska vsebina lahko doseže. V primeru trivialnega klasifikatorja, ki pusti apriorne verjetnosti nespremenjene (in po kriteriju klasifikacijske točnosti klasificira vsak primer v večinski razred), je informacijska vsebina enaka 0. To je spodnja meja za še sprejemljivo uspešnost klasifikatorja.

Včasih je koristno prikazati informacijsko vsebino odgovora, normalizirano z entropijo. Na ta način dobimo *relativno informacijsko vsebino odgovora*:

$$RInf = \frac{Inf}{H(R)} \times 100\% \quad (3.11)$$

### Rob (margin)

Preprosta mera, ki delno upošteva verjetnostno klasifikacijo, je rob (angl. margin). Podana je z razliko med verjetnostjo najbolj verjetnega razreda in verjetnostjo drugega najbolj verjetnega razreda. Rob se uporablja predvsem kot mera zanesljivosti klasifikatorja - večji kot je rob, bolj zanesljiva je odločitev. Npr., pri dvorazrednem problemu je klasifikacija v prvi razred bolj zanesljiva, če so izračunane verjetnosti razredov enake 0.9 in 0.1, kot če so enake 0.6 in 0.4. Rob ne upošteva apriornih verjetnosti razredov.

Pri diskriminantnih funkcijah je zanesljivost klasifikacije podana z oddaljenostjo točke od ločitvene hiperploskeve - čim bolj je točka oddaljena, tem bolj zaupamo, da res pripada razredu, ki mu ustreza ta stran hiperploskeve. Čim večji je rob ob hiperploskvi, ki ne vsebuje nobenega učnega primera, tem bolj zanesljiva je diskriminantna funkcija. Metoda podpornih vektorjev temelji na algoritmu za maksimizacijo roba.

### Senzitivnost in specifičnost

Pri odločjanju so zelo pogosti dvorazredni problemi, kjer se odločamo za ali proti nekemu razredu (npr. pacient je ali ni bolan, klient je ali ni rizičen, nakup je ali ni obetaven itd.). Primere iz enega razreda imenujemo pozitivni primeri, iz drugega razreda pa negativni primeri. Za dvorazredne probleme sta se uveljavili dve meri, senzitivnost (*sensitivity*) in specifičnost (*specificity*), ki sta izpeljani iz štirih osnovnih količin, razvidnih iz tabele števila napačnih klasifikacij za dvorazredni problem (glej tabelo 3.3).

Pomen oznak je naslednji:

P - pozitivni razred; POS - število pozitivnih primerov

N - negativni razred; NEG - število negativnih primerov

n - število vseh primerov

TP (*True Positives*) - število pravilno klasificiranih pozitivnih primerov

pravi razred	napovedani razred		vsota
	P	N	
P	TP	FN	POS=TP+FN
N	FP	TN	NEG=FP+TN
vsota	PP=TP+FP	PN=FN+TN	n = TP+FP+FN+TN

Tabela 3.3: Števila napačnih klasifikacij za dvorazredni problem.

FP (*False Positives*) - število lažnih pozitivnih primerov, torej število negativnih primerov, ki jih je klasifikator zmotno uvrstil med pozitivne

TN (*True Negatives*) - število pravilno klasificiranih negativnih primerov

FN (*False Negatives*) - število lažnih negativnih primerov, torej število pozitivnih primerov, ki jih je klasifikator zmotno uvrstil med negativne

PP (*Predicted Positives*) - število primerov, klasificiranih kot pozitivni

PN (*Predicted Negatives*) - število primerov, klasificiranih kot negativni

Senzitivnost (občutljivost, t.j. verjetnost, da klasifikator zazna pozitivni primer) ocenjuje odstotek pravilno klasificiranih pozitivnih primerov:

$$Senz = \frac{TP}{TP + FN} = \frac{TP}{POS}$$

Specifičnost, ravno nasprotno, ocenjuje odstotek pravilno klasificiranih negativnih primerov:

$$Spec = \frac{TN}{TN + FP} = \frac{TN}{NEG}$$

Pri tem seveda velja, da je klasifikacijska točnost podana z:

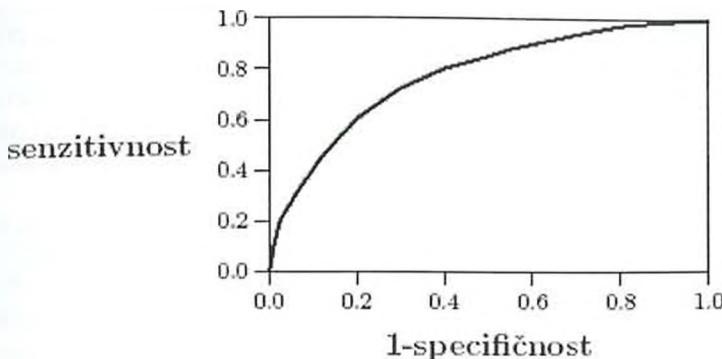
$$T = \frac{TP + TN}{TN + FP + FN + TN} = \frac{TP + TN}{n}$$

Senzitivnost in specifičnost se najpogosteje uporablja v medicini pri ocenjevanju testov. Če ima pri dani problemski domeni (bolezni) laboratorijski test senzitivnost 85% in specifičnost 95% pomeni, da:

- če ima pacient bolezen, potem jo bo test potrdil z verjetnostjo 0.85 (to ne pomeni, da je pri pacientu, ki ga test prepozna kot bolnega, verjetnost bolezni enaka 85% - tako interpretacija ustreza preciznosti, ki je opisana kasneje).

- če pacient nima bolezni, je verjetnost 0.05, da ga bo test napačno opredelil za bolnega.

Meri si med seboj nasprotujeta: če se trudimo povečati senzitivnost, bo trpela specifičnost, in obratno. Mnogokrat se raziskovalci trudijo maksimizirati senzitivnost testa pri fiksirani spodnji meji še sprejemljive specifičnosti. Trivialno je doseči 100% senzitivnost, tako da vse primere klasificiramo v pozitivni razred (in dobimo specifičnost 0%). Enako je trivialno doseči 100% specifičnost, tako da vse primere klasificiramo v negativni razred (in dobimo senzitivnost 0%).



Slika 3.5: Primer tipične krivulje ROC.

## Krivulja ROC

Pri iskanju optimalnega klasifikatorja nas zanima predvsem razmerje med senzitivnostjo in specifičnostjo. Klasična metoda, ki izhaja iz področja teorije odkrivanja signalov, je krivulja ROC (*Reciever Operating Characteristic*) [14]. Krivulja omogoča analizo razmerja med senzitivnostjo in specifičnostjo (glej sliko 3.5). Na vodoravni X-osi je prikazano relativno število napačno klasificiranih negativnih primerov (FP), t.j. 1 – specifičnost. Na navpični Y-osi pa je prikazano relativno število pravilno klasificiranih pozitivnih primerov (TP), t.j. senzitivnost. Vsak klasifikator, ki napove samo en razred, je na sliki prikazan s točko, ki ustreza njegovi senzitivnosti in specifičnosti. Za klasifikatorje, ki vrnejo za vsak primer verjetnostno distribucijo po razredih, dobimo za vsak prag, ki ga izberemo za odločitveno pravilo, drugo točko na grafu. Običajni prag je 0.5 - torej klasificiramo primer v tisti od dveh razredov, ki ima večjo verjetnost (verjetnost, večjo od 0.5).

Če prag spremojmo po krivulji ROC. Prag 0.0 pomeni, da vse primere klasificiramo kot negativne, torej so vsi negativni primeri pravilno klasificirani in vsi pozitivni napačno (senzitivnost = 0.0 in specifičnost = 1.0). Temu ustreza točka v levem spodnjem kotu grafa. Prag 1.0 pomeni, da vse primere klasificiramo kot pozitivne, torej so vsi pozitivni primeri pravilno klasificirani in vsi negativni napačno (senzitivnost = 1.0 in specifičnost = 0.0). Temu ustreza točka v desnem zgornjem kotu grafa. Diagonala, ki povezuje ekstremna primera, ustreza naključnim klasifikatorjem (brez znanja), ki uporabljajo različne pragove. Uporabni klasifikatorji se nahajajo vedno levo zgoraj od diagonale. Idealni klasifikator, ki ima senzitivnost in specifičnost enako 1.0, se nahaja v levem zgornjem kotu grafa. Čim bližje se nahaja tej idealni točki, tem boljši je klasifikator.

Trdimo, da je nek klasifikator boljši od drugega, če ima boljšo senzitivnost ter boljšo ali

enako specifičnost, ali če ima boljšo specifičnost ter boljšo ali enako senzitivnost (torej se nahaja levo in/ali zgoraj v prostoru ROC). V ostalih primerih ne moremo določiti, da je kateri klasifikator boljši.

Če narišemo celotno krivljo ROC za oba klasifikatorja, tako da zvezno spremenimo prag odločitve od 0.0 do 1.0, nam o kvaliteti posameznega klasifikatorja govori ploščina pod krivuljo ROC (*Area Under the ROC Curve, AUC*). Boljši klasifikator je tisti, ki ima večjo ploščino AUC. Izkaže se, da je AUC enaka verjetnosti, da bo klasifikator (ki zna napovedovati verjetnosti) pravilno razločil med pozitivnim in negativnim primerom (t.j. pozitivnemu bo pripisal večjo verjetnost, da je pozitiven).

Krivulja ROC omogoča analizo cen napačnih klasifikacij. Za različna razmerja cen (in različne apriorne verjetnosti ciljnega razreda) so optimalni različni pragi. Krivulja ROC je mogoče uporabljati pri iskanju optimalnega praga za dano cenovno matriko, saj vsaka točka na krivulji ustreza enemu možnemu pragu in, po drugi strani, eni specifičnosti in senzitivnosti. Iz slednjih pa lahko izračunamo ceno klasifikatorjevih napovedi, torej ROC krivulja povezuje prage in cene klasifikacij."

### Priklic in preciznost

Na področju iskanja pomembnih dokumentov (*information retrieval*) se uporablja tri meri priklic (*recall*) in preciznost (*precision*). Priklic je definiran enako kot senzitivnost (ocenjuje odstotek pomembnih odkritih dokumentov glede na vse pomembne dokumente):

$$\text{Priklic} = \frac{TP}{TP + FN} = \frac{TP}{POS}$$

Preciznost ocenjuje odstotek pravilno klasificiranih primerov, ki so bili klasificirani kot pozitivni (ocenjuje odstotek pomembnih dokumentov glede na vse dokumente, ki so bili označeni kot pomembni):

$$\text{Preciznost} = \frac{TP}{TP + FP} = \frac{TP}{PP}$$

Podobno kot s spremenjanjem praga spremenimo razmerje med senzitivnostjo in specifičnostjo, spremenimo tudi razmerje med priklicem in preciznostjo. Na ta način dobimo krivuljo s podobnimi lastnostmi, kot jih ima krivulja ROC.

Če želimo opazovati obe meri istočasno, izračunamo mero F (*F-measure*), ki je podana z:

$$F = \frac{2 \times \text{Priklic} \times \text{Preciznost}}{\text{Priklic} + \text{Preciznost}} = \frac{2TP}{2TP + FP + FN}$$

Vse tri mere, Priklic, Preciznost in mera F ne upoštevajo (zelo velikega) števila pravilno klasificiranih negativnih primerov (TN), saj nas pri iskanju relevantnih dokumentov le-ti ne zanimajo.

### Srednja kvadratna napaka

Pri regresijskih problemih klasifikacijske točnosti ne moremo uporabiti, saj bi bila v večini realnih problemov enaka 0, ker imamo opravka z zveznimi in ne diskretnimi funkcijami.

Mera uspešnosti zveznih funkcij  $\hat{f}$  (modelov) je definirana kot *srednja kvadratna napaka* (*mean squared error; MSE*), t.j. povprečni kvadrat razlike med napovedano vrednostjo  $\hat{f}(i)$  in dejansko vrednostjo  $f(i)$ :

$$E = \frac{1}{n} \sum_{i=1}^n (f(i) - \hat{f}(i))^2 \quad (3.12)$$

Ker je velikost  $E$  odvisna od razpona vrednosti funkcije, uporabljamo relativno srednjo kvadratno napako [5]:

$$RE = \frac{n \times E}{\sum_i (f(i) - \bar{f})^2} \quad (3.13)$$

kjer je  $\bar{f} = \frac{1}{n} \sum_i f(i)$ . Relativna srednja kvadratna napaka je nenegativna in za sprejemljive hipoteze manjša od ena:

$$0 \leq RE \leq 1$$

$RE = 1$  dosežemo s trivialno funkcijo, ki vedno vrne povprečno vrednost  $\hat{f}(i) = \bar{f}$ . Če je za neko funkcijo  $RE > 1$ , je le-ta neuporabna. Idealna funkcija je  $\hat{f}(i) = f(i)$ , katere  $RE = 0$ .

### Srednja absolutna napaka

Druga pogosto uporabljenega mera uspešnosti zveznih funkcij  $\hat{f}$  je *srednja absolutna napaka* (*mean absolute error; MAE*), t.j. povprečna absolutna razlika med napovedano vrednostjo  $\hat{f}(i)$  in dejansko vrednostjo  $f(i)$ :

$$MAE = \frac{1}{n} \sum_{i=1}^n |f(i) - \hat{f}(i)| \quad (3.14)$$

Ker je tudi velikost  $MAE$  odvisna od razpona vrednosti funkcije, lahko uporabljamo relativno srednjo absolutno napako:

$$RMAE = \frac{n \times MAE}{\sum_i |f(i) - \bar{f}|} \quad (3.15)$$

kjer je  $\bar{f}$  definirana kot pri  $RE$  zgoraj. Relativna srednja absolutna napaka je tako kot  $RE$  nenegativna in za sprejemljive hipoteze manjša od ena:

$$0 \leq RMAE \leq 1$$

Tudi  $RMAE = 1$  dosežemo s trivialno funkcijo, ki vrne vedno povprečno vrednost  $\hat{f}(i) = \bar{f}$ .

### Koreacijski koeficient

Koreacijski koeficinet meri statistično korelacijo med dejanskimi vrednostmi  $f(i)$  in napovedanimi vrednostmi  $\hat{f}(i)$  odvisne (regresijske) spremenljivke:

$$\rho = \frac{S_f \hat{f}}{S_f S_{\hat{f}}} \quad (3.16)$$

kjer so:

$$S_{f\hat{f}} = \frac{\sum_{i=1}^n [(f(i) - \bar{f})(\hat{f}(i) - \bar{\hat{f}})]}{n-1}$$

$$S_f = \frac{\sum_{i=1}^n (f(i) - \bar{f})^2}{n-1}$$

$$S_{\hat{f}} = \frac{\sum_{i=1}^n (\hat{f}(i) - \bar{\hat{f}})^2}{n-1}$$

in kjer sta

$$\bar{f} = \frac{1}{n} \sum_i f(i)$$

in

$$\bar{\hat{f}} = \frac{1}{n} \sum_i \hat{f}(i)$$

Korelacijski koeficient ima zgornjo mejo 1 za popolno korelacijo, spodnjo mejo pa -1 za popolno negativno korelacijo. Vrednost 0 ponazarja popolno nekoreliranost. Za praktične regresijske prediktorje so smiselne samo pozitivne vrednosti korelacijskega koeficienta:

$$0 < \rho \leq 1$$

Za razliko od prejšnjih mer, srednje kvadratne in absolutne napake, ki ju je potrebno minimizirati, želimo korelacijski koeficient maksimizirati.

### Pristranskost in varianca v strojnem učenju

Kompromis med kompleksnostjo in točnostjo hipoteze, ki ga določa princip MDL (glej enačbo 3.3), si poglejmo še z drugega zornega kota. Pokazali bomo, da pridemo do enake zahteve za kompromis, če razstavimo napako hipoteze na napako, ki izvira iz učnega algoritma (pristranskost), in na napako, ki izvira iz učnih podatkov (varianco).

Pri strojnem učenju poskušamo aproksimirati ciljno funkcijo. Kvaliteta aproksimacije je odvisna od učnega algoritma in od vhodnih podatkov. V realnih problemih je utopično pričakovati, da bi bila hipoteza popolnoma pravilna - torej imamo praktično vedno opravka z napako. Izjeme so le (umetne) domene s popolno informacijo. Napako hipoteze ocenujemo za konkreten (neoznačen) primer, ali pa za množico primerov.

Naj bo  $t$  dejanska (neznana) vrednost, ki jo želimo napovedati, in  $\hat{t}$  napovedana vrednost. Pri fiksniem učnem algoritmu je napovedana vrednost odvisna od učne množice primerov. Če bi poganjali algoritem na različnih učnih množicah, bi dobili različne napovedi  $\hat{t}$ . Pričakovana napoved ali matematično upanje napovedi  $E[\hat{t}]$  je srednja vrednost napovedi preko vseh možnih učnih množic dane velikosti. Ker je  $E[\hat{t}]$  odvisna od učnega algoritma, je razlika do prave vrednosti  $t$  definirana kot *pristranskost (bias)* učnega algoritma:

$$\text{Pristranskost}(t) = E[\hat{t}] - t$$

Pristranskost je odvisna od vrednosti  $t$ , ki jo želimo napovedati. Če je pristranskost enaka 0, pravimo, da je učni algoritem nepristranski. Pristranskost pomeni sistematično napako, ki je ne moremo popraviti, ne da spremenimo učni algoritem (npr. spremenimo prostor hipotez, spremenimo preiskovalni algoritem). Ker je  $t$  neznana vrednost, pristranskosti ne moremo izračunati. Empirično lahko analiziramo pristranskost algoritma na umetno zgeneriranih podatkih, kjer je  $t$  poznan, ali pa na realni domeni z veliko učnimi primeri uporabimo metodo prečnega preverjanja, opisano v naslednjem razdelku, ali rezerviramo del učnih primerov.

Na konkretnem primeru hipoteza vrne napoved  $\hat{t}$ , ki se v splošnem razlikuje od  $E[\hat{t}]$ . Občutljivost algoritma na učno množico meri varianca (*variance*) napovedi:

$$\text{Varianca}(\hat{t}) = E[E[\hat{t}] - \hat{t}]^2$$

Varanca ni odvisna od dejanske vrednosti  $t$ . Lahko jo ocenimo tako, da s simulacijo večkrat vzorčimo učno množico, zgeneriramo hipotezo in napovedi ter izračunamo varianco dobljenih napovedi  $\hat{t}_i$ . Če imamo več učnih algoritmov z enako pristranskostjo, izberemo tistega z najmanjšo varianco.

Dejanska napaka je sestavljena iz napake zaradi pristranskosti (napaka učnega algoritma) in napake zaradi variance (napaka zaradi učnih primerov). Napaka je podana kot razlika med dejansko vrednostjo  $t$  in napovedano vrednostjo  $\hat{t}$ . Pogosta cenilka napake je povprečna kvadratna napaka:  $E[(\hat{t} - t)^2]$ , ki je enaka vsoti kvadrata pristranskosti in variance, kar pokaže naslednja izpeljava:

$$E[(\hat{t} - t)^2] = E[(\hat{t} - E[\hat{t}] + E[\hat{t}] - t)^2]$$

Če upoštevamo, da je  $t$  konstanta in velja  $E[t] = t$ , ter da je  $E[E[\hat{t}]\hat{t}] = E[\hat{t}]^2$ , dobimo

$$E[(\hat{t} - t)^2] = (E[\hat{t}] - t)^2 + E[(E[\hat{t}] - \hat{t})^2]$$

oziroma

$$E[(\hat{t} - t)^2] = (\text{Pristranskost}(t))^2 + \text{Varianca}(\hat{t})$$

Torej k napaki prispevata tako pristranskost kot varianca. Žal pa sta si pristranskost in varianca nasprotujoča in je treba najti ustrezni kompromis med njima. Če želimo minimizirati varianco, je treba hipotezo poenostaviti (zmanjšati število parametrov). Ekstremen primer je konstantna napoved - hipoteza vedno napove isto vrednost, ne glede na vhodne podatke in ne glede na primer, ki ga želimo rešiti. Varianca take hipoteze je 0, vendar je pristranskost lahko precejšnja (v principu poljubno velika). Če pa želimo minimizirati pristranskost, je treba povečati prostor hipotez, torej povečati število parametrov. To hitro pripelje do pretiranega prileganja učnim primerom (overfitting), kar pomeni, da je hipoteza močno odvisna od učnih primerov in je zaradi tega varianca visoka. V splošnem bo z naraščanjem števila parametrov točnost njihovih ocen slabša - torej bo varianca (razlika od ene učne množice do druge) narasla.

### 3.3 Ocenjevanje učenja

*Pri učenju je vsak dan nekaj pridobljenega, pri sledenju Tau je vsak dan nekaj izgubljenega.*

*Lao Ce*

Da ocenimo kvaliteto učnega algoritma ali kvaliteto naučenega, testiramo hipotezo na neodvisni testni množici. Če je ta dovolj velika, je naša ocena zanesljiva, sicer je potrebno postopek učenja in testiranja, kot ga prikazuje slika 3.4, ponoviti večkrat. V tem razdelku so predstavljene osnovne metode za ocenjevanje kvalitete hipoteze: ocena kvalitete in ocena zanesljivosti tako dobljene ocene v obliki standardne napake in intervala zaupanja. Za ponavljanje postopka učenja in testiranja sta opisani metodi prečnega preverjanja in metoda razmnoževanja učnih primerov. Na koncu je obravnavano še primerjanje uspešnosti različnih algoritmov.

#### Zanesljivost ocene uspešnosti

Kadar imamo na voljo dovolj veliko množico primerov z znano vrednostjo odvisne spremenljivke, postopek na sliki 3.4 izvedemo samo enkrat. Hipotezo zgradimo na dovolj veliki učni množici primerov in jo ocenimo na dovolj veliki neodvisni testni množici. Za ocenjevanje kvalitete  $Q$  uporabimo eno od mer, opisanih v prejšnjem razdelku, ki jo povprečimo preko vseh  $N_t$  testnih primerov:

$$Q = \frac{\sum_{j=1}^{n_t} Q_j}{n_t} \quad (3.17)$$

Zatem ocenimo zanesljivost tako dobljene ocene. Za oceno zanesljivosti se uporablja *standardna napaka* (*standard error*), ki je definirana z [5]:

$$SE(Q) = \sqrt{s^2/n_t} \quad (3.18)$$

kjer je  $s^2$  varianca kvalitete na testni množici:

$$s^2 = \frac{\sum_{j=1}^{n_t} (Q_j - Q)^2}{n_t - 1}$$

Standardno napako interpretiramo kot povprečno napako dane ocene uspešnosti z  $n_t$  testnimi primeri in oceno kvalitete zapišemo v obliki:

$$\text{kvaliteta} = Q \pm SE(Q)$$

Če je napaka normalno porazdeljena, gre za interval zaupanja s stopnjo zaupanja 68% in se dejanska uspešnost z 68% gotovostjo nahaja na intervalu  $(Q - SE(Q), Q + SE(Q))$ . Z večanjem števila testnih primerov se  $SE$  manjša.

V enačbi (3.18) za ocenjevanje kvalitete  $Q$  uporabimo eno od mer, ki je ocenjena s povprečjem preko testnih primerov (enačba (3.17)). Za ostale mere je potrebno posebej izpeljati

standardno napako. Npr. za relativno srednjo kvadratno napako  $RE$  (enačba (3.13)) se standardna napaka izračuna po formuli [5]:

$$SE(RE) = RE \sqrt{\frac{1}{n} \left( \frac{s_1^2}{E^2} - \frac{2s_{12}}{s^2 E} + \frac{s_2^2}{s^4} \right)}$$

kjer je  $E$  srednja kvadratna napaka, podana z enačbo (3.12),  $s^2$  njena varianca:

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (f(i) - \bar{f})^2$$

$s_1^2, s_{12}, s_2^2$  pa se izračunajo po formulah:

$$s_1^2 = \frac{1}{n-1} \sum_{i=1}^n (f(i) - \hat{f}(i))^4 - E^2$$

$$s_{12} = \frac{1}{n-1} \sum_{i=1}^n [(f(i) - \hat{f}(i))^2 (f(i) - \bar{f})^2] - s^2 E$$

$$s_2^2 = \frac{1}{n-1} \sum_{i=1}^n (f(i) - \bar{f})^4 - s^4$$

## Interval zaupanja

Posplošitev standardne napake je *interval zaupanja* (*confidence interval*). Najprej izberemo stopnjo zaupanja  $1 - \alpha$  (npr. 0.9 ali 0.95), ki se interpretira kot verjetnost, da se dejanska kvaliteta, ki smo jo ocenili z  $Q$ , nahaja na danem intervalu. Dejansko je standardna napaka poseben primer, saj gre za polovico intervala zaupanja pri stopnji zaupanja  $1 - \alpha = 68\%$ .

Pri dani stopnji zaupanja izračunamo polovično velikost intervala  $x_\alpha$ :

$$x_\alpha = z_{\frac{\alpha}{2}} SE(Q) \quad (3.19)$$

Pri tem je  $z_{\frac{\alpha}{2}}$  velikost intervala zaupanja za vrednosti, ki so porazdeljene po standardiziranem normalnem zakonu [8]. Npr. za  $\alpha = 0.05$ , je vrednost  $z_{\frac{\alpha}{2}} = z_{0.025} = 1.96$ . V tem primeru lahko s 95% gotovostjo trdimo, da se dejansko povprečje kvalitete  $\bar{Q}$  preko vseh možnih poskusov nahaja na intervalu:

$$Q - x_{0.05} < \bar{Q} < Q + x_{0.05}$$

Za standardno napako velja, da je  $x_{0.32} = SE(Q)$ , oziroma je  $z_{0.16} = 1$ .

Interval zaupanja normalne porazdelitve  $x_\alpha$  se z večanjem števila testnih primerov oži. Čim večje je število testnih primerov, tem bolj zaupamo oceni  $Q$ .

## Prečno preverjanje

Ko imamo na voljo malo vhodnih podatkov, je nesprejemljivo, da prikrajšamo učni algoritem za podmnožico testnih primerov. Za učenje potrebujemo vse razpoložljive primere. Kljub temu želimo oceniti uspešnost zgrajene teorije. Kot smo že omenili, da ocenjevanje uspešnosti hipoteze na učni množici zelo slabo oceno.

Najzanesljivejša metoda v tem primeru je metoda *izloči enega* (*leave-one-out*). Po tej metodi za vsak razpoložljivi rešeni primer zgradimo eno hipotezo. Primer izločimo iz učne množice in iz vseh preostalih primerov zgradimo hipotezo, ki jo zatem uporabimo za reševanje izločenega primera. To ponovimo za vse primere in uspešnost hipoteze, ki smo jo zgradili iz vseh učnih primerov, ocenimo kot povprečno uspešnost zgrajenih hipotez na ustreznih (izločenih) primerih. Zavedati se moramo, da smo uspešnost končne hipoteze ocenjevali z uspešnostjo drugačnih hipotez. Ker je učni algoritem uporabljal zelo podobno učno množico primerov pri vseh zgrajenih hipotezah, velja, da so si hipoteze zelo podobne in imajo zato podobno uspešnost.

Metoda "izloči enega" je lahko časovno nesprejemljiva. Če imamo na voljo  $n$  primerov, moramo zgraditi  $n + 1$  hipotez, namesto ene same ( $n$  hipotez za ocenitev uspešnosti in eno zaključno hipotezo iz vseh učnih primerov). Metodo "izloči enega" lahko posplošimo na "izloči  $n/K$  primerov", ki ji pravimo tudi *K-kratno prečno preverjanje* (*K-fold cross-validation*). Število  $K$  določa število hipotez, ki jih moramo zgraditi. Na začetku množico razpoložljivih primerov razdelimo na  $K$  približno enako močnih podmnožic. Za vsako podmnožico zgradimo hipotezo tako, da za učenje uporabimo unijo preostalih podmnožic. Nato zgrajeno hipotezo uporabimo za reševanje primerov iz dane podmnožice. Uspešnost končne hipoteze, ki jo zgradimo iz vseh razpoložljivih primerov, ocenimo kot povprečno uspešnost vseh  $K$  hipotez na na njihovih podmnožicah testnih primerov. V praksi se najpogosteje uporablja  $K = 10$ .

Bolj zanesljiva različica te metode je *sorazmerno prečno preverjanje* (*stratified cross-validation*). To je prečno preverjanje, kjer ohranjamo približno enako distribucijo razredov v vseh podmnožicah. Na ta način je distribucija razredov v vsaki učni in testni množici približno enaka.

Za prečno preverjanje in metodo izloči enega uporabimo enake formule za izračun standardne napake in intervala zaupanja kot pri testiranju z eno samo dovolj veliko testno množico. Moramo se zavedati, da je ocena približna [5], saj ne testiramo dejanske končne hipoteze, ampak več različnih hipotez, ki so bile zgrajene na nekoliko manjših učnih množicah.

## Metoda razmnoževanja učnih primerov

Če je podatkov za učenje zelo malo (30-50 primerov), postanejo vse do sedaj opisane metode ocenjevanja uspešnosti nezanesljive. Tudi metoda "izloči enega" je pri majhnih množicah podatkov nezanesljiva zaradi majhne množice testnih primerov (ki jih je toliko, kot je učnih). V takem primeru je bolj zanesljiva ocena, če večkrat ponovimo 2-kratno prečno preverjanje (z vsakokratnim naključnim razbitjem množice primerov na dve enako veliki podmnožici). Tipično število potrebnih ponovitev je 100. Povprečna ocena uspešnosti vseh stotih 2-kratnih prečnih preverjanj se je izkazala za bolj zanesljivo kot metoda "izloči enega". Manj zanesljiva je v primerih, ko je dejanska napaka majhna (do 10% pri klasifikacijskih problemih).

Efron [10] je razvil metodo *razmnoževanja učnih primerov* (*bootstrapping*), ki se je izkazala predvsem v primerih, ko ocenjujemo uspešnost modela, dobljenega iz majhnega števila vhodnih podatkov. Po tej metodi približno 200-krat ponovimo naslednji postopek. Iz vseh  $N$  primerov naključno izberemo  $N$  učnih primerov z vračanjem. V povprečju je v tako izbrani učni množici zastopanih samo 63.2% originalnih primerov, od katerih se nekateri tudi večkrat ponovijo. Model zgradimo iz tako dobljenih  $N$  učnih primerov. Za testno množico vzamemo preostalih 36.8% primerov, ki jih ni v učni množici. Povprečni rezultat testiranja preko vseh 200 ponovitev tega postopka je pesimistična ocena uspešnosti modela, ki ga algoritmom učenja zgradi iz vseh primerov, ki so na voljo. Ocena je pesimistična, saj je za učenje uporabljen samo 63% primerov. Oceno napake po tej metodi ponavadi označujejo z  $e_0$ .

Končna ocena napake je linearна kombinacija optimistične napake  $e_u$  modela na celotni množici (celotna množica se uporabi za učenje in za testiranje) in pesimistične ocene  $e_0$ :

$$e = 0.368 \times e_u + 0.632 \times e_0$$

Pri tej metodi je sprejemljivo tudi, če  $e_0$  zamenjamo z oceno napake, ki jo dobimo z večkratno (npr.  $100\times$ ) ponovitvijo 2-kratnega prečnega preverjanja [31].

## 3.4 Primerjanje uspešnosti različnih učnih algoritmov

*Če eden zmaga, morajo vsi drugi zgubiti. Je to sploh zabavno?*

*Marlo Morgan*

Pri uporabi strojnega učenja na različnih področjih želimo včasih primerjati uspešnost različnih algoritmov. Nekateri avtorji nastavljajo parametre sistema toliko časa, dokler njihov sistem ne doseže največje uspešnosti na testnih primerih (ali v povprečju na vseh razbitijih na učno in testno množico). Zatem rezultat primerjajo z uspešnostjo ostalih algoritmov z vnaprej nastavljenimi parametri. Taka primerjava ni dopustna. Nastavljanje parametrov vseh sistemov bi moralno biti opravljeno pred testiranjem sistema na neodvisni testni množici. To pomeni, da je potrebno učno množico razbiti na dve množici: na dejansko učno množico in na *nastavljeno množico* (*validation set*). Zatem lahko nastavljamo parametre algoritma toliko časa, dokler ne dosegнемo najboljših rezultatov na nastavljencih množicah. Ko se odločimo za testiranje na testni množici, je nadaljnje spremnjanje parametrov prepovedano!

Ocenjena uspešnost posameznih algoritmov nam služi za primerjavo uspešnosti algoritmov. Če je razlika uspešnosti relativno majhna, je primerjava lahko nezanesljiva, še posebej če je standardna napaka posameznih ocen uspešnosti relativno velika v primerjavi z razliko uspešnosti. Zato je potrebno uporabiti statistične teste značilnosti odstopanj, s katerimi ocenimo stopnjo zaupanja v ocenjene razlike uspešnosti. Za različne načine ocenjevanja uspešnosti uporabljam različne teste.

Algoritme lahko primerjamo po sledečih tipičnih scenarijih:

- imamo dva algoritma in želimo izbrati boljšega za dano ciljno problemsko domeno,
- imamo dva algoritma in želimo izbrati boljšega na seriji različnih domen,

- želimo primerjati uspešnost enega algoritma (npr. ki smo razvili na novo) z uspešnostjo več drugih algoritmov na seriji različnih domen.

Pri vseh scenarijih velja, da imamo na voljo:

- dovolj veliko število primerov, tako da lahko uporabimo neodvisno testno množico primerov, ne da bi s tem preveč zmanjšali število učnih primerov (npr. pri nekaj tisoč učnih primerih si lahko privoščimo nekaj sto testnih primerov, kar ponavadi zadošča za zanesljivo primerjavo dveh ali več algoritmov); ocena uspešnosti je povprečje uspešnosti na testni množici;
- število primerov je majhno in moramo za primerjanje uspešnosti uporabiti prečno preverjanje ali metodo izloči enega. V obeh primerih je ocena uspešnosti povprečna uspešnost preko vseh testnih primerov. Za oceno pomembnosti razlik uporabljamо tudi povprečne uspešnosti na vsaki od  $K$  testnih množic.

## Dva algoritma na eni domeni

### Prečno preverjanje

V primeru  $K$ -kratnega prečnega preverjanja, kjer oba algoritma uporabljata enaka razbitja na učno in testno množico, se uporablja popravljeni enosmerni (*one-tailed*) parni  $t$ -test [25].  $t$ -test uporabljamо zato, ker je število  $K$  relativno majhno (ponavadi  $K < 30$ ), enosmerni test pa zato, ker nas zanima, če je eden od algoritmов boljši od drugega.

Standardni  $t$ -test poteka po sledečem postopku. Za vseh  $K$  poskusov izračunamo razlike uspešnosti obeh algoritmов:

$$raz_i = \hat{U}_1 - \hat{U}_2 \quad i = 1, \dots, K$$

Za  $raz$  predpostavimo, da je porazdeljena normalno. Preiskusiti želimo hipotezo, da oba algoritma dosegata enako uspešnost ( $\overline{raz} = 0$ ). Izračunamo povprečje razlike:

$$\overline{raz} = \frac{1}{K} \sum_{i=1}^K raz_i$$

in standardno odstopanje

$$s = \sqrt{\frac{1}{K-1} \sum_{i=1}^K (raz_i - \overline{raz})^2} \quad (3.20)$$

Vrstni red primerjanih algoritmов izberemo tako, da je  $\overline{raz} \geq 0$ . Statistika  $t$  je porazdeljena po Studentovem zakonu:

$$t = \frac{\overline{raz}}{s} \sqrt{K} \quad (3.21)$$

Pri dani stopnji zaupanja  $1 - \alpha$  hipotezo, da sta algoritma enako uspešna, zavrzemo, če je  $t > t_{\alpha, K-1}$  (za dvosmerni  $t$ -test bi uporabili mejo  $t_{\alpha/2, K-1}$ ).  $t_{\alpha, K-1}$  določa mejo intervala vrednosti

spremenljivke, porazdeljene po Studentovem zakonu s  $K - 1$  prostostnimi stopnjami. Tipične vrednosti za  $\alpha$  so 0.05, 0.01 in 0.001. Za  $K = 10$  so mejne vrednosti podane v tabeli 3.4.

Uporaba standardnega  $t$ -testa pri prečnem preverjanju ni popolnoma korektna, saj ta predpostavlja neodvisnost poskusov, pri prečnem preverjanju pa učne množice niso neodvisne. Nadeau in Bengio [25] sta izpeljala popravljeni  $t$ -test, ki odpravi to pomanjkljivost. Razlika je, da v enačbi (3.21) spremenimo faktor  $\sqrt{K}$  tako, da inverzni vrednosti parametra  $K$  prištejemo razmerje med številom testnih ( $n_t$ ) in učnih ( $n_u$ ) primerov v vsakem poskusu:

$$t = \frac{\bar{raz}}{s \sqrt{\left(\frac{1}{K} + \frac{n_t}{n_u}\right)}} \quad (3.22)$$

Ker v primeru prečnega preverjanja reda  $K$  na  $n$  primerih velja

$$n_t = n/K$$

in

$$n_u = n_t(K - 1)$$

dobimo:

$$t = \frac{\bar{raz}}{s \sqrt{\left(\frac{1}{K} + \frac{1}{K-1}\right)}} \quad (3.23)$$

Lastnosti popravljenega  $t$ -testa lahko še dodatno izboljšamo, če prečno preverjanje ponovimo večkrat, npr. 10 krat ponovimo 10-kratno prečno preverjanje [2].

V (nezaželenem) primeru, ko primerjana algoritma ne uporablja enakih razbitij na učno in testno množico, se uporablja statistika [8]:

$$t = \frac{(\bar{U}_1 - \bar{U}_2)}{\sqrt{(s_1^2/n_1) + (s_2^2/n_2)}}$$

Pri tem je  $n_j$  število poskusov (razbitij), narejenih z  $j$ -tim algoritmom,

$$\bar{U}_j = \frac{1}{n_j} \sum_{i=1}^{n_j} U_{ji}$$

in

$$s_j = \sqrt{\frac{1}{n_j - 1} \sum_{i=1}^{n_j} (U_{ji} - \bar{U}_j)^2}$$

### Metodi "izloči enega" in zadosti velika neodvisna testna množica

Pri metodi izloči enega in pri metodi testiranja, kjer je na voljo zadosti velika neodvisna testna množica, se uporablja enosmerni (*one-tailed*)  $z$ -test, ker je število testnih primerov  $n$  zadosti veliko. Postopek je podoben kot pri  $t$ -testu opisanem zgoraj. Razlika je, da  $K$  nadomestimo

statistika	stopnje zaupanja $\alpha$		
	0.05	0.01	0.001
$t_{\alpha,9}$	2.262	3.250	4.781
$z_\alpha$	1.960	2.576	3.291
$\chi^2_1$	3.841	6.635	10.828

Tabela 3.4: Tipične mejne vrednosti za statistike  $t_{\alpha,9}$ ,  $z_\alpha$  in  $\chi^2_1$ .

z  $n$  (število poskusov je torej enako številu vseh testnih primerov) ter namesto Studentove porazdelitve upoštevamo normalno porazdelitev. Namesto s črko  $t$  bomo statistiko označevali s črko  $z$ :

$$z = \frac{\bar{r}_{\text{az}}}{s} \sqrt{n}$$

Pri dani stopnji zaupanja  $1 - \alpha$ , hipotezo, da sta oba algoritma enako uspešna, zavrhemo, če je  $z > z_\alpha$  (za dvosmerni  $z$ -test bi uporabili mejo  $z_{\alpha/2}$ ).  $z_\alpha$  določa mejo intervala vrednosti spremenljivke, porazdeljene po normalnem zakonu. Tipične vrednosti za  $\alpha$  so 0.05, 0.01 in 0.001, mejne vrednosti so podane v tabeli 3.4.

Pri klasifikacijskih problemih, kjer nas zanima samo klasifikacijska točnost, lahko uporabimo bolj občutljiv McNemarjev test [11]. Naj sta  $A$  in  $B$  dva klasifikatorja in naj bo  $n_{10}$  število pravilno klasificiranih primerov z algoritmom  $A$ , ki jih je algoritem  $B$  napačno klasificiral. Podobno, naj bo  $n_{01}$  število primerov, pravilno klasificiranih z algoritmom  $B$ , ki jih je algoritem  $A$  napačno klasificiral. McNemarjev test uporablja statistiko:

$$s_M = \frac{(|n_{01} - n_{10}| - 1)^2}{n_{01} + n_{10}}$$

Ničelna hipoteza je, da sta oba algoritma enako točna. Če je hipoteza resnična, bo števec statistike  $s_M$  majhen. Statistika  $s_M$  je porazdeljena po zakonu  $\chi^2$  z eno prostostno stopnjo. Če je ničelna hipoteza resnična, je verjetnost, da je  $s_M > \chi^2_{0.05,1}$ , manjša od 0.05.

## Bonferronijeva korekcija

Če naredimo veliko primerjav različnih algoritmov (npr. z nastavljanjem različnih vrednosti parametrov) na istih podatkih, je končna izbira zmagovalca lahko plod naključja, čeprav je rezultat ocenjen s statičnim testom kot značilen (signifikanten). Enako se lahko zgodi, če primerjamo dva algoritma na mnogih podatkovnih bazah, in je značilna razlika na posameznih podatkovnih bazah lahko zgolj naključna. V takem primeru je za običajni  $t$ -test potrebno uporabiti Bonferronijevo korekcijo [27], ki je za celotno primerjavo mnogo strožja, s tem pa izgubimo moč statističnega testa. Z Bonferronijevo korekcijo zavrhemo ničelno hipotezo (da sta algoritma enako uspešna) le, če je razlika med algoritmoma relativno velika.

Pri značilnosti testa  $\alpha$  in  $N$  neodvisnih poskusih na istih podatkih, je pričakovano število naključno značilnih rezultatov enako  $\alpha \times N$ . Zato Bonferronijeva korekcija zahteva strožjo

vrednost  $\alpha_B$  za dosego enake stopnje značilnosti  $\alpha$ . Če  $N$  krat ponovimo poskus z značilnim rezultatom stopnje  $\alpha_B$ , je skupna stopnja značilnosti enaka:

$$\alpha = 1 - (1 - \alpha_B)^N$$

Od tod dobimo:

$$\alpha_B = 1 - (1 - \alpha)^{\frac{1}{N}} \approx \frac{\alpha}{N}$$

Npr., za 100 primerjav ( $N = 100$ ) in zahtevo, da je stopnja značilnosti  $\alpha = 0.05$ , je Bonferronijseva korekcija stopnje značilnosti  $\alpha_B = 0.0005$

Ker je testiranje z Bonferronijevim korekcijo zelo strogo, je bolje uporabiti močnejše teste za primerjanje dveh algoritmov na več domenah oziroma za primerjanje večih algoritmov na več domenah [9].

## Dva algoritma na več domenah

Za primerjavo uspešnosti dveh algoritmov  $A$  in  $B$  na več podatkovnih bazah se uporablja ne-parametrični Wilcoxonov test rangiranih predznakov (*signed-ranks test*), ki ne predpostavlja normalne porazdelitve. Test rangira absolutne razlike uspešnosti med dvema algoritmoma na  $D$  domenah. Naj bo  $d_i$  razlika uspešnosti med dvema algoritmoma na  $i$ -ti domeni,  $i = 1..D$ :

$$d_i = U_i^A - U_i^B$$

Če je liho število domen z enako uspešnostjo ( $d_i = 0$ ), se ena od teh domen ignorira. Razlike se rangirajo glede na njihove absolutne vrednosti. V primerih, ko je več razlik absolutno enakih, se jim dodeli povprečni rang. Npr., za razlike 0.0, 0.5, 1.7, -1.3, 0.0, 1.0, 0.0 in -1.0 najprej črtamo prvo domeno, rezultate uredimo in dobimo range 5, 1, 2, 6.5, 3.5, 6.5 in 3.5. Posebej se seštejejo rangi pozitivnih in rangi negativnih razlik in obema vsotama se prišteje polovična vsota rangov ničelnih razlik:

$$R^+ = \sum_{d_i > 0} \text{rang}(d_i) + \frac{1}{2} \sum_{d_i=0} \text{rang}(d_i)$$

$$R^- = \sum_{d_i < 0} \text{rang}(d_i) + \frac{1}{2} \sum_{d_i=0} \text{rang}(d_i)$$

Za manjše število domen ( $D < 30$ ) uporabimo statistiko

$$T = \min(R^+, R^-)$$

V večini statističnih učbenikov in v statističnih programih najdemo eksaktne mejne vrednosti za statistiko  $T$ . Npr., za število domen  $D = 14$  in stopnjo zaupanja  $\alpha = 0.05$  lahko ničelno hipotezo, da sta algoritma enako uspešna, zavrzemo, če  $T \leq 21$ .

Za večje število domen uporabimo statistiko, ki je porazdeljena približno normalno:

$$z = \frac{T - \frac{1}{4}D(D+1)}{\sqrt{\frac{1}{24}D(D+1)(2D+1)}}$$

Pri stopnji zaupanja  $\alpha = 0.05$  ničelno hipotezo, da sta algoritma enako uspešna, zavrzemo, če  $z < -1.96$ .

test	število klasifikatorjev									
	2	3	4	5	6	7	8	9	10	
Nemenyi	1.960	2.343	2.569	2.728	2.850	2.949	3.031	3.102	3.164	
$q_{0.05}$	1.645	2.052	2.291	2.459	2.589	2.693	2.780	2.855	2.920	
Bon-Dun	2	3	4	5	6	7	8	9	10	
$q_{0.05}$	1.960	2.241	2.394	2.498	2.576	2.638	2.690	2.724	2.773	
$q_{0.10}$	1.645	1.960	2.128	2.241	2.326	2.394	2.450	2.498	2.539	

Tabela 3.5: Kritične vrednosti za  $q_\alpha$  za dva testa (Nemenyi ter Bonferroni-Dunn) pri več klasifikatorjih in več domenah.

### Več algoritmov na več domenah

Pri primerjavi večih algoritmov je potrebno testirati algoritme na več domenah, če želimo dokazati razlike. Da bi zmanjšali število primerjav, je najbolje, da primerjamo samo en algoritmom z ostalimi (npr. tistega, ki smo ga sami razvili in ga želimo primerjati z drugimi algoritmi). Če primerjamo vsak algoritmom z vsakim, število primerjav močno naraste in je zato večja nevarnost, da so razlike zgolj plod naključja, in je zaradi tega test bolj konzervativ.

Pri primerjanju več algoritmov na več domenah se uporablja neparametrični izboljšani Friedmanov test [17]. Test rangira uspešnost vsakega algoritma na vsaki domeni posebej. V primerih, ko je več algoritmov enako uspešnih, se jim dodeli povprečni rang. Imejmo  $k$  algoritmov in  $D$  domen. Naj bo  $r_i^j$  rang  $j$ -tega algoritma na  $i$ -ti domeni. Povprečni rangi algoritmov so:

$$R_j = \frac{1}{D} \sum_{i=1}^D r_i^j$$

Najprej izračunamo originalno Friedmanovo statistiko:

$$\chi_F^2 = \frac{12D}{k(k+1)} \left( \sum_{j=1}^k R_j^2 - \frac{k(k+1)^2}{4} \right)$$

Zatem izračunamo statistiko, ki je porazdeljena po zakonu  $F$  s prostostnima stopnjama  $(k-1), (k-1) \times (D-1)$ :

$$F_F = \frac{(D-1)\chi_F^2}{D(k-1) - \chi_F^2}$$

Kritične vrednosti uporabimo iz statističnih programov. Npr., pri številu algoritmov  $k = 4$  in številu domen  $D = 14$  ter stopnji zaupanja  $\alpha = 0.05$  ničelno hipotezo zavrzemo, če je  $F_F > F_{3,39} = 2.85$ . Če ničelno hipotezo, da so vsi algoritmi enako uspešni, zavrzemo, potem nadaljujemo s testom, ki pove, kateri pari algoritmov se med seboj statistično značilno razlikujejo.

Če primerjamo vsak algoritmom z vsakim, uporabimo Nemenyijev test, če pa primerjamo vse algoritme z enim algoritmom, uporabimo test Bonferroni-Dunn [9]. Oba testa potrdita

značilnost razlik med algoritmoma  $j_1$  in  $j_2$ , če je razlika med povprečnima rangoma večja ali enaka kritični razliki  $CD$ :

$$|R_{j_1} - R_{j_2}| \geq CD = q_\alpha \sqrt{\frac{k(k+1)}{6D}} \quad (3.24)$$

Kritične vrednosti za  $q_\alpha$  za oba testa so podane v tabeli 3.5.

## 3.5 Kombiniranje algoritmov strojnega učenja

*Bolj kot s prijatelji se učimo strpnosti s pomočjo sovražnikov.*

*Dalaj Lama*

Princip najkrajšega opisa zahteva od učnega algoritma, da poišče kompromis med kompleksnostjo teorije in njeno točnostjo - treba je najti čim bolj preprosto hipotezo, ki še vedno dovolj dobro modelira podatke. Na drugi strani princip večkratne razlage zahteva od napovedovalnega algoritma, da uporabi mnogo čim bolj različnih hipotez, ki dobro modelirajo podatke. Torej mora učni algoritem upoštevati ta kriterij in že med učenjem generirati več čim bolj različnih hipotez. Lahko se odločimo za:

- različne učne algoritme: npr. na istih učnih podatkih zgradimo odločitveno drevo, naučimo naivni Bayesov klasifikator in usmerjeno večnivojsko nevronsko mrežo, shranimo primere za algoritem k-najbližjih sosedov ter izračunamo diskriminantno funkcijo po metodi podpornih vektorjev;
- isti učni algoritem poganjamo večkrat z različnimi nastavitevami parametrov ali nad različno pripravljenimi vhodnimi podatki. Take metode so bagging, boosting, in metoda naključnih gozdov.

Ko zgradimo več različnih hipotez, jih je treba po principu večkratne razlage na nek način skombinirati v enovit napovedovalni algoritem (npr. klasifikator ali regresijski prediktor). Ker verjetnosti hipotez  $P(H|E, B)$  iz enačbe (3.5) v večini primerov ne poznamo, je potrebno izbrati način kombiniranja. V nadaljevanju so opisani načini kombiniranja zgrajenih hipotez.

### Kombiniranje napovedi različnih hipotez

Pri klasifikaciji lahko napovedi klasifikatorjev kombiniramo na naslednje načine:

**Glasovanje (voting):** vsak klasifikator prispeva en glas za izbrani razred; primer klasifikatorja v razred, ki zbere največ glasov.

**Uteženo glasovanje:** vsak klasifikator prispeva verjetnostno distribucijo po vseh razredih - verjetnost razreda je hkrati njegova utež; primer klasificiramo v razred, ki ima največjo vsoto uteži.

**Glasovanje, uteženo z zanesljivostjo napovedi:** ocenimo zanesljivost hipoteze, ponavadi je to kar verjetnost  $P(H|E, B)$  vsake hipoteze H in uporabimo enačbo (3.5). Zanesljivost ocenimo npr. s transdukcijo, ki je opisana pozneje (ocenimo zanesljivost napovedi danega primera), ali pa ocenimo zanesljivost klasifikatorja kot celote (npr. s prečnim preverjanjem ocenjena klasifikacijska točnost).

**Kombiniranje po metodi naivnega Bayesa:** vsak klasifikator prispeva verjetnostno distribucijo po vseh razredih - s pomočjo formule naivnega Bayesa izračunamo verjetnost vsakega razreda, pri čemer en klasifikator obravnavamo kot diskretni atribut (verjetnost razreda pri dani vrednosti atributa je enaka verjetnosti razreda, ki jo je vrnil dani klasifikator); ta izračun predpostavlja neodvisnost klasifikatorjev.

**Naučeno kombiniranje z meta-učenjem:** Na validacijski množici vsak klasifikator vrne napoved za vsak primer. Te napovedi se uporabijo kot atributi za meta učenje. Naloga učnega algoritma je iz napovedi posameznih klasifikatorjev napovedati pravi razred. Kombinacija klasifikatorjev se zgnerira s pomočjo meta klasifikatorja. V praksi se najbolje obnesejo preprosti meta-učni algoritmi, zato da ne pride do prevelikega prileganja učnim primerom. Tej metodi meta učenja pravijo tudi *stacking*.

**Lokalno uteženo glasovanje:** primeru, ki mu želimo določiti vrednost odvisne spremenljivke, najprej poiščemo k najbližjih sosedov iz učne množice. Za vsak prediktor določimo povprečno (lokalno) točnost na teh k učnih primerih. Zatem predikcije na novem primeru utežimo z izračunanimi lokalnimi točnostmi, ali pa preprosto izberemo za predikcijo samo najboljši lokalni prediktor [32].

**Dinamična izbira klasifikatorja:** ta metoda je primerna samo za klasifikacijo. Kot pri lokalno uteženem glasovanju, primeru, ki mu želimo določiti razred, najprej poiščemo k najbližjih sosedov iz učne množice. Z vsakim klasifikatorjem za novi primer ter za vsakega od k bližnjih učnih primerov določimo verjetnostno distribucijo po razredih. Zatem izračunamo podobnosti med klasifikacijo (verjetnostno distribucijo) vsakega od k učnih primerov in novega primera. Z izbranim pragom podobnosti izmed k primerov izberemo najbolj podobno klasificirane učne primere - recimo, da jih je  $l \leq k$ . Zatem za vsak klasifikator izračunamo klasifikacijsko točnost na izbrani podmnožici l učnih primerov. Klasifikacije na novem primeru utežimo z izračunanimi lokalnimi točnostmi, ali pa izberemo za klasifikacijo novega primera samo najboljši lokalni klasifikator [16].

V praksi se najpogosteje uporablja uteženo glasovanje. Nekatere študije kažejo, da najboljše rezultate dobimo pri kombiniranju po metodi naivnega Bayesa [20, 29].

Pri regresiji lahko napovedi različnih prediktorjev kombiniramo tako, da izračunamo povprečje napovedi. Običajno se računa uteženo povprečje, kjer je utež bodisi ocenjena zanesljivost prediktorja kot celote, bodisi ocenjena zanesljivost prediktorja na posameznem primeru. To oceno zanesljivosti lahko dobimo npr. s pomočjo trasdukcije. Tako kot pri klasifikaciji lahko tudi pri regresiji uporabimo naučeno kombiniranje z meta-učenjem.

## Povezovanje algoritmov

Do sedaj smo videli, da lahko algoritme povezujemo vzporedno, tako da kombiniramo njihove napovedi. Funkcija kombiniranja je lahko fiksna ali pa je naučena, kar pomeni, da se meta-učenec nauči kombinirati napovedi različnih algoritmov.

Algoritme lahko povežemo tudi zaporedno: prvi algoritmom se uči ciljne napovedi. Vsak naslednji algoritmom pa se uči napovedovanja napak prejšnjega algoritma. Na ta način zgradimo urejeno serijo hipotez, vsaka napoveduje napake prejšnje hipoteze. V praksi se izkaže, da zaradi prevelikega prileganja učnim primerom niso smiselne dolge zaporedne verige in se ponavadi uporabi veriga dolžine 2: algoritmom, ki rešuje originalni problem, ter korekcijski algoritmom, ki poskuša popraviti napake prvega algoritma. Pomembno je, da se korekcijski algoritmom uči iz klasifikacije primerov, ki jih osnovni algoritmom med učenjem ni videl, ker sicer pride do prevelikega prileganja učni množici. Torej se korekcijski algoritmom uči na validacijski množici primerov.

Principle vzporednega, zaporednega in meta-učenja lahko poljubno kombiniramo med seboj in dobimo različne mreže učnih algoritmov. Več takih kombinacij srečamo pri algoritmih umetnih nevronskih mrež. Zaradi zahteve po čimpreprostejših hipotezah in zaradi nevarnosti prevelikega prileganja učnim podatkom se v praksi ne uporablja kompleksnejšega povezovanja algoritmov.

## Bagging

Izraz bagging pride iz "bootstrap aggregating" [3]. Bootstrapping je metoda razmnoževanja učnih primerov, opisana v tem poglavju. Pri baggingu generiramo serijo različnih učnih množic. Če ima učna množica  $n$  primerov, vsakič  $n$  krat naključno izberemo primer iz učne množice z vračanjem. To pomeni, da se isti učni primer v tako generirani učni množici lahko večkrat ponovi, nekaterih primerov iz originalne učne množice pa generirana množica sploh ne vsebuje (v povprečju je 36.8% takih primerov). Nad vsako tako učno množico poženemo učni algoritmom. Na ta način dobimo veliko število različnih hipotez. Serijo zgeneriranih hipotez uporabimo za napovedovanje odvisne spremenljivke novega primera tako, da kombiniramo napovedi vseh hipotez.

Bagging dobro deluje predvsem pri nestabilnih učnih algoritmih z visoko varianco, kot so odločitvena in regresijska drevesa. Je robusten, saj z večanjem števila modelov ne pride do prevelikega prileganja učni množici.

## Boosting

Ideja metode boosting [28] je uteževanje učnih primerov glede na njihovo težavnost. Metoda predpostavlja, da učni algoritmom zna obravnavati utežene učne primere. Če tega ne zna, potem uteževanje simuliramo tako, da generiramo učno množico podobno kot pri baggingu, le da verjetnostna distribucija za izbiro posameznih primerov ni enakomerna. Primeri z večjo težo imajo večjo verjetnost, da bodo izbrani, in obratno.

V prvi iteraciji učni algoritmom zgradi hipotezo na normalnih učnih primerih - vsak učni primer šteje enako, t.j. ima utež enako 1.0. Hipoteza potem napove odvisno spremenljivko

za vse učne primere. Primerom s pravilno napovedjo zmanjšamo uteži, primerom z napačno napovedjo pa povečamo uteži.

Za uteževanje primerov je pogosto uporabljenca sledča shema. Če je  $e$  napaka hipoteze na učnem primeru, potem se utež primera pomnoži z  $e/(1 - e)$ . Primer z manjšo napako dobi manjšo utež. Na koncu se uteži normalizirajo, tako da je vsota uteži za vseh  $n$  učnih primerov enaka  $n$ .

Naloga naslednje iteracije učenja je usmerjena v obravnavanje težavnejših učnih primerov. Cel postopek ponavljamo, dokler napaka ne postane dovolj majhna, ali pa napaka postane prevelika (preostali učni problem ni rešljiv). Hipoteze s preveliko napako zavržemo, saj ne prispevajo koristnega znanja. Prav tako zavržemo hipoteze s premajhno napako, ker so najpogosteje preveč prilagojene učni množici. Na ta način je zgenerirano urejeno zaporedje hipotez.

Pri napovedovanju se uporabijo vse hipoteze, vendar napovede vsake hipoteze utežimo z njeno točnostjo na uteženi učni množici, iz katere je bila zgenerirana. Bolj točne hipoteze imajo večjo težo pri glasovanju. Treba je izbrati še uteževalno shemo za hipoteze. Ponavadi se uporabi utež  $-\log(f/(1 - f))$ , kjer je  $f$  napaka hipoteze na uteženi učni množici, iz katere je bila hipoteza zgenerirana.

Boosting pogosto dosega boljšo točnost od bagginga in ga lahko uporabimo tudi s stabilnimi učnimi algoritmi z majhno varianco, vendar lahko pride do prevelikega prileganja učni množici. V takih primerih je točnost zaporedja hipotez slabša od točnosti ene same hipoteze.

## Naključni gozdovi

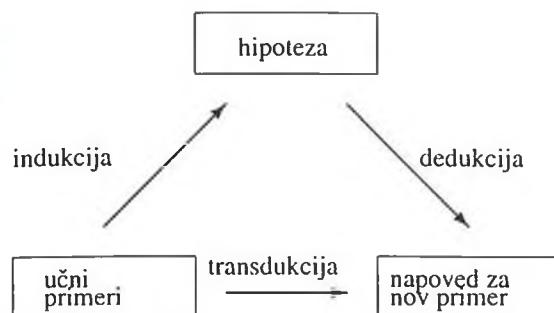
Metoda naključnih gozdov (*random forests*) je namenjena izboljšanju napovedne točnosti drevesnih modelov. Originalno je bila razvita za odločitvena drevesa [4]. Ideja je generirati zaporedje odločitvenih dreves, tako da se pri izbiri najboljšega atributa v vsakem vozlišču naključno izbere majhno število atributov, ki vstopajo v izbor za najboljši atribut. Breiman predlaga naključno izbiro toliko atributov v vsakem vozlišču, kolikor znaša logaritem števila atributov plus 1. Lahko je to število enako kar 1, kar pomeni popolnoma naključno izbiro atributa v vsakem vozlišču vsakega drevesa. Število zgrajenih dreves je ponavadi 100 ali več.

Vsako drevo se uporabi za klasifikacijo novega primera po metodi glasovanja - vsako drevo ima en glas, ki ga nameni razredu, v katerega bi klasificiralo nov primer. Iz vseh glasov dobimo verjetnostno distribucijo po vseh razredih.

Metoda naključnih gozdov je robustna, saj zmanjša varianco drevesnih algoritmov. S to metodo drevesni algoritmom doseže napovedno točnost, primerljivo z najboljšimi algoritmi. Slaba stran metode je, da je razlaga odločitev otežena, saj je množica 100 ali več dreves nepregledna in zato nerazumljiva za uporabnika.

## Transdukcijski strojni učenje

Strojno učenje temelji na induktivnem sklepanju, ki je sklepanje od konkretnega k splošnemu - iz primerov na hipotezo. Uporaba naučene hipoteze za napovedovanje vrednosti odvisne spremenljivke novega primera je deduktivno sklepanje. Transduktivno sklepanje [30] je sklepanje od konkretnega h konkretnemu - iz učnih primerov sklepamo na vrednost odvisne spre-



Slika 3.6: Različne oblike sklepanja pri strojnem učenju.

menljivke novega primera (glej sliko 3.6). Temu sklepanju najbolj utreza leno učenje po metodi najbližjih sosedov, saj algoritom samo shrani učne primere in ne generira hipoteze.

Transdukcijo lahko uporabimo tudi v obratni smeri, tako da v učno množico dodamo nove primere, za katere ne poznamo vrednosti odvisne spremenljivke. Preden dodamo neoznačene primere v učno množico, jih moramo na nek način označiti. Zatem poženemo učni algoritem za generiranje transducirane hipoteze. Poznamo več variant tega pristopa:

- preizkusimo vse možne oznake za nov primer (oziroma za množico novih primerov) in po nekem kriteriju (npr. MDL) izberemo najboljšo transducirano hipotezo in s tem najboljšo označitev novega primera (oziroma množice novih primerov) [15].
- najprej zgeneriramo hipotezo iz učne množice primerov in z njo označimo nov primer (oziroma množico novih primerov). Tako označene nove primere dodamo v učno množico in zgeneriramo transducirano hipotezo. Nove primere sedaj označimo s pomočjo transducirane hipoteze, jih dodamo v originalno učno množico in ponovno zgeneriramo transducirano hipotezo. Postopek ponavljamo, dokler še prihaja do večjih sprememb v hipotezi oziroma v oznakah primerov. Dobljena končna hipoteza določa tudi oznake vsem novim primerom [18].
- želimo oceniti zanesljivost napovedi na novem primeru (za razliko od mer točnosti, ki ocenjujejo točnost na celotnem problemskem prostoru). Najprej zgeneriramo hipotezo iz učne množice primerov, nato s pomočjo hipoteze označimo nov primer. Označen nov primer ali primer z namenoma spremenjeno oznako dodamo v učno množico in zgeneriramo transducirano hipotezo. Nov primer označimo s pomočjo transducirane hipoteze. Razlika med prvotno (netransducirano) in novo (transducirano) oznako nam služi za oceno zanesljivosti originalne hipoteze na novem primeru [22, 1]. Za oceno zanesljivosti pri klasifikaciji lahko uporabimo različne metrike razdalje med verjetnostnimi distribucijami [21], pri regresiji pa normalizirano razdaljo med dvema napovedima [1].
- postopek iz prejšnje alineje lahko uporabimo ne samo za oceno zanesljivosti napovedi

na ciljnem novem primeru ampak tudi za korekcijo te napovedi. Iz smeri spremembe napovedi, do katere pride v transduktivnem koraku, sklepamo na smer potrebne spremembe. Iz ocene zanesljivosti napovedi sklepamo na velikost potrebne spremembe. Empirične analize kažejo, da lahko na ta način dobimo značilno bolj točne napovedi [21, 1].

## Cenovno občutljivo učenje

Pri klasifikacijskih problemih je pogosto potrebno namesto maksimizacije klasifikacijske točnosti minimizirati ceno napačne klasifikacije. Večina algoritmov ne upošteva tega kriterija, čeprav se mnoge da ustrezno prilagoditi za cenovno občutljivo učenje.

Cenovno občutljivo učenje se uporablja pri klasifikacijskih problemih, kjer je en ali več razredov redko zastopanih. Npr., če je v dvorazrednem problemu en razred zastopan samo v 1% primerov, bo večina klasifikatorjev ohranila samo trivialno odločitev, ki klasificira vse primere v drugi razred. Na ta način bo klasifikacijska točnost 99%, kar pa ni tisto, kar uporabnik želi. Pogosto nas zanimajo ravno redki razredi in želimo predvsem pravilno klasificirati primere iz teh razredov.

S spremenjanjem cene napačne klasifikacije pri dvorazrednem problemu (oziroma s spremenjanjem uteži učnim primerom iz različnih razredov) lahko spremojemo razmerje med senzitivnostjo in specifičnostjo in tako zgeneriramo krivuljo ROC za poljubni klasifikator. Da dobimo gladko krivuljo ROC, je potrebno večkratno izvajanje algoritma za vsako nastavljeno razmerje med razredoma in rezultate povprečiti. V ta namen uporabimo sorazmerno prečno preverjanje.

Če algoritem strojnega učenja ne zna upoštevati cene napačnih klasifikacij, lahko uporabimo eno od naslednjih metod:

**Upoštevanje pogojnega tveganja** Najpreprostejša metoda je, da pri klasifikaciji upoštevamo pogojno tveganje. Če pri klasifikaciji novega primera klasifikator vrne verjetnostno distribucijo  $P_i$  po razredih  $i = 1, \dots, m_0$ , je optimalna odločitev definirana s pomočjo cenvne matrike  $C_{ij}$  (cena, če primer iz  $i$ -tega razreda klasificiramo v  $j$ -ti razred). Primer klasificiramo v tisti razred  $j$ , ki minimizira pogojno tveganje:

$$R_j = \sum_{i=1}^{m_0} P_i \times C_{ij} \quad (3.25)$$

**Uteževanje učnih primerov** Preprost trik, ki prisili algoritem, da namesto klasifikacijske točnosti upošteva cene napačnih klasifikacij, je, da ustrezno utežimo učne primere. Primeri iz razredov, katerih napačna klasifikacija je dražja, dobijo sorazmerno večjo utež. Ker algoritem skuša maksimizirati klasifikacijsko točnost, ga bo večja utež pri dražjem razredu prisilila, da minimizira ceno napačne klasifikacije.

Metoda predpostavlja (kot pri boostingu), da učni algoritem zna obravnavati utežene učne primere. Če tega ne zna, uteževanje učnih primerov simuliramo z razmnoževanjem učnih primerov sorazmerno njihovi teži. Npr., če je v dvorazrednem problemu

razmerje uteži 1:20 v prid drugega razreda, je potrebno za vsak učni primer iz drugega razreda zgenerirati 20 kopij tega primera.

Metoda uteževanja primerov je primerna samo za dvorazredne primere, pri večrazrednih primerih pa je uteževanje lahko odvisno samo od pravega razreda, ne more pa upoštevati razreda, v katerega klasificiramo (ni možno upoštevati obeh indeksov v matriki cen  $C_{ij}$ ).

**Spreminjanje razredov učnim primerom** Da dobimo optimalno klasifikacijo, ki minimizira pogojno tveganje (3.25), je potrebno algoritom prepričati, da upošteva pogojno tveganje. To dosežemo s spremenjanjem razredov učnim primerom [13]. Za vsak učni primer najprej z izbranim klasifikatorjem ali pa s pomočjo bagginga ali boostinga določimo verjetnostno distribucijo  $P_i$  po razredih. Zatem učnemu primeru spremeni razred v tistega, ki minimizira pogojno tveganje (3.25). Metoda povzroči, da se spremeni razred tistim učnim primerom, ki so na meji z "dragim" razredom in kjer je bolje napačno klasificirati "poceni" razred in s tem povečati verjetnost pravilne klasifikacije primerom iz "dragega" razreda. Končni klasifikator zgeneriramo na tako spremenjeni učni množici.

## Oponašanje funkcije

Za nekatere težke probleme lahko samo s kompleksnimi algoritmi dosežemo zadovoljivo točnost napovedi. Slaba stran kompleksnih metod, kot sta metoda podpornih vektorjev in nevronske mreže, je ta, da je rezultat netransparenten in težko razumljiv uporabniku. V takem primeru lahko uporabimo princip oponašanja funkcije.

Najprej rešimo klasifikacijski ali regresijski problem z algoritmom, ki daje najboljšo točnost napovedi. Zatem s pomočjo naučenega modela zgeneriramo nove primere, ki jih dodamo k učni množici: vrednosti atributov generiramo naključno (z upoštevanjem distribucij iz učne množice) ter za zgenerirane primere določimo vrednosti odvisne spremenljivke z naučenim modelom. Na ta način dobimo dovolj veliko množico primerov, ki zadošča za učenje lažje razumljivih modelov, npr. odločitvenih ali regresijskih dreves, ki se naučijo oponašati originalni naučeni model.

Z večanjem števila učnih problemov se dobljena funkcija vse bolj približuje funkciji originalno naučenega modela. Veča se tudi kompleksnost transparentnega modela (npr. večje odločitveno drevo), vendar pričakujemo, da bo dobljeni model vseeno bolj transparenten kot originalni model.

## Uporaba klasifikatorjev v regresiji in obratno

Včasih je učinkoviteje reševati regresijske probleme s pomočjo klasifikacijskih algoritmov. Najprej razred z algoritmom za diskretizacijo razbijemo na nekaj intervalov in tako dobimo klasifikacijski problem (vsak interval ustreza enemu razredu). Z algoritmom za klasifikacijo rešimo tako dobljeni klasifikacijski problem. Pri predikciji odgovore klasifikatorja preslikamo nazaj v zvezni razred. Ker so odgovori klasifikatorja verjetnostna porazdelitev, lahko

izračunamo uteženo vsoto srednjih vrednosti intervalov (prej diskretnih razredov), tako da so uteži enake verjetnostim, ki jih je klasifikator priredil posameznim intervalom (razredom).

Postopek diskretizacije napovedi je lahko koristen, ker iznica vpliv šuma odvisne spremenljivke, kar lahko omogoči klasifikatorju, da doseže boljše napovedi, kot bi jih dosegli, če bi neposredno uporabili regresijski algoritem.

Koristen je lahko tudi obratni postopek - uporabiti regresijski algoritem za reševanje klasifikacijskega problema. Če je razredov več kot dva in niso urejeni, razbijemo klasifikacijski problem na več dvorazrednih problemov. To lahko storimo na dva načina:

- za vsak razred  $R$  definiramo dvorazredni klasifikacijski problem: razred  $R$  in razred  $\bar{R}$ , ki združuje vse ostale razrede;
- razrede hierarhično združujemo po podobnosti. To stori strokovnjak iz dane domene, ali pa uporabimo algoritem razvrščanja. Vsaka združitev dveh podrazredov predstavlja en dvorazredni klasifikacijski problem.

Dvorazredni problem rešujemo z regresijskim algoritmom tako, da enemu razredu pripisemo vrednost 0 (ali -1) in drugemu vrednost 1. Napovedi naučenega regresorja vrednosti odvisnih spremenljivk za nove primere zatem sprememimo v diskrette razrede tako, da uporabimo prag (ponavadi 0.5). Če je napovedana vrednost večja od praga, primer klasificiramo v razred 1, sicer v razred 0.

Postopek spremenjanja dvorazrednih problemov v regresijske probleme vpelje v problemski prostor večjo fleksibilnost, saj regresor lahko napove poljubno vrednost na intervalu  $[0, 1]$  in ne samo diskretnih vrednosti 0 in 1. To mu lahko omogoči boljšo točnost, kot če bi neposredno uporabili klasifikacijski algoritem.

### Kode za popravljanje napak klasifikacije

Pri klasifikacijskih problemih z več kot tremi razredi lahko razrede kodiramo s kodami za popravljanje izhodnih napak (*error correcting output codes*). Vsak razred kodiramo z enako dolgim nizom bitov, ki razbijejo klasifikacijski problem na več podproblemov po principu maksimizacije razdalje med kodami razredov. Originalni problem klasifikacije se spremeni v toliko binarnih klasifikacijskih podproblemov, kolikor je dolga koda za kodiranje razredov. V vsakem od binarnih podproblemov so originalni razredi združeni v dva razreda - tisti s kodo 0 v prvega in tisti s kodo 1 v drugega.

Primer kode za klasifikacijski problem s petimi razredi prikazuje tabela 3.6. Vsaka vrstica ustreza enemu razredu. Koda razbije originalni klasifikacijski problem na 15 dvorazrednih klasifikacijskih podproblemov. Vsakega izmed petnajstih podproblemov rešujemo z ločenim klasifikatorjem. Odgovori vseh petnajstih klasifikatorjev se združijo v bitno kodo. Končna klasifikacija se določi tako, da poiščemo vrstico v tabeli, ki se najmanj (v najmanjšem številu bitov) razlikuje od dobljene bitne kode. Npr., če dobimo bitno kodo 100011011000111, potem le-ta najbolj ustreza razredu  $R_3$ , saj se od nje razlikuje v treh bitih. Od  $R_1$  se razlikuje v sedmih bitih, od  $R_2$  v sedmih, od  $R_4$  v enajstih in od  $R_5$  v desetih. V tem primeru je koda,

Vrstica = razred	stolpec														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
R2	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
R3	0	0	0	0	1	1	1	0	0	0	0	1	1	1	1
R4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
R5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Tabela 3.6: Primer kode za popravljanje napak pri klasifikaciji v pet razredov.

če je odgovor  $R3$  pravilen, uspela popraviti tri napačne bite (oz. napačne klasifikacije treh binarnih klasifikatorjev).

Dobra koda za popravljanje izhodnih napak mora imeti naslednji dve lastnosti [12]:

**Ločevanje vrstic:** vsaka vrstica mora biti čim bolj različna od ostalih vrstic, zato da minimiziramo število bitov, ki jih je koda sposobna popraviti. Npr. v tabeli 3.6 je minimalna (in hkrati maksimalna) razlika med dvema vrsticama 8 bitov, kar omogoča popravljanje do treh napačnih odgovorov posameznih dvorazrednih klasifikatorjev.

**Ločevanje stolpcev:** vsak bit v kodi predstavlja en dvorazredni klasifikacijski problem. Do seči moramo, da so dvorazredni klasifikacijski problemi med seboj nekorelirani. Zato morajo biti vsi stolpci med seboj različni. Ker operacija komplementa ohranja nespremenjen dvorazredni klasifikacijski problem (v dvorazrednem problemu lahko razreda zamenjamo med seboj, ne da bi se problem spremenil), morajo biti vsi stolpci različni tudi od komplementov vseh ostalih stolpcev. Koreliranost med dvorazrednimi klasifikacijskimi problemi minimiziramo tako, da maksimiziramo različnost stolpcev med seboj (vključno s komplementi). Pri tem velja, da stolpca samih ničel in njegov komplement (iz samih enic) nista uporabna, saj predstavljata trivialni klasifikacijski problem. V tabeli 3.6 se vsi stolpci razlikujejo od ostalih stolpcev in njihovih komplementov vsaj v enem bitu.

Ker je število uporabnih različnih stolpcev (brez komplementov in brez trivialnega stolpca iz samih ničel ali enic) pri  $r$  razredih enako  $2^{r-1} - 1$ , je koda za popravljanje napak smiselna samo za  $r \geq 4$  (pri treh razredih dobimo samo tri uporabne stolpce in koda ni sposobna popraviti nobene napake).

Pri klasifikaciji novega primera je treba poiskati vrstico v kodi za popravljanje napak, ki je najbolj podobna odgovorom vseh dvorazrednih klasifikatorjev. Če so klasifikatorji verjetnostni, vrnejo verjetnost  $P_j(1)$  razreda 1 in s tem hkrati tudi verjetnost  $1 - P_j(1)$  razreda 0,  $j = 1..k$ , kjer je  $k$  število klasifikatorjev (stolpcev). V tem primeru lahko razdaljo do kod v vrsticah tabele  $W_{i,j}$ , kjer je  $i$  indeks razreda in  $j$  indeks stolpca (klasifikatorja), računamo po

formuli [12]:

$$D = \sum_{j=1}^k |P_j(1) - W_{i,j}|$$

Torej ni potrebno določiti vrednost bita za  $j$ -ti klasifikator (če bi vrednost bita določili, bi enako obravnavali npr. odgovora  $P_j(1) = 0.51$  in  $P_j(1) = 1.0$ ).

Zanimiva je relacija med pristopom s kodami za popravljanje izhodnih napak in kombiniranjem napovedi različnih hipotez, ki je obravnavano na začetku razdelka. Pri slednjem pristopu poskuša vsaka hipoteza napovedati isto funkcijo in z glasovanjem ali kakim drugačnim načinom kombiniranja dobimo končni odgovor. Pri pristopu s kodami za popravljanje napak pa vsak klasifikator skuša napovedati drugačno funkcijo, kar zmanjša koreliranost med klasifikatorji.

## Literatura

- [1] Z. Bosnić. Uporaba transdukcije pri regresijskem napovedovanju. Magistrska naloga, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2003.
- [2] R. R. Bouckaert, E. Frank. Evaluating the replicability of significance tests for comparing learning algorithms. The University of Waikato, New Zealand, 2004.
- [3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 1996.
- [4] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 2001.
- [5] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone. *Classification and Regression Trees*. Wadsforth International Group, 1984.
- [6] B. Cestnik. Estimating probabilities: A crucial task in machine learning. V *European Conf. on Artificial Intelligence 90*, str. 147–149, August 1990.
- [7] B. Cestnik. *Ocenjevanje verjetnosti v avtomatskem učenju*. Doktorska disertacija, Univerza v Ljubljani, Fakulteta za elektrotehniko in računalništvo, Ljubljana, 1991.
- [8] W. Chase, F. Bown. *General Statistics*. John Wiley & Sons, 1986.
- [9] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
- [10] P. Diaconis, B. Efron. Computer-intensive methods in statistics. *Scientific American*, 248:96–108, 1983.
- [11] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.
- [12] T. G. Dietterich, G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

- [13] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. V *Proc. of KDD-99*, str. 155–164. New York: ACM Press, 1999.
- [14] J. P. Egan. Signal detection theory and ROC analysis. *Series in Cognition and Perception*, 1975.
- [15] A. Gammerman, V. Vovk, V. Vapnik. Learning by trasduction. V *Proceedings of 14th Conference on Uncertainty in Artificial Intelligence*, str. 148–155, 1998.
- [16] G. Giacinto, F. Roli. Dynamic classifier selection based on multiple classifier behaviour. *Pattern Recognition*, 34:1879–1881, 2001.
- [17] R. L. Iman, J. M. Davenport. Aproximations of the critical region of the friedman statistic. *Communications in Statistics*, str. 571–595, 1980.
- [18] T. Joachims. Transductive inference for text classification using support vector machines. V *Proceedings 16th International Conference on Machine Learning*, str. 200–209, 1999.
- [19] I. Kononenko, I. Bratko. Information based evaluation criterion for classifier's performance. *Machine Learning*, 6:67–80, 1991.
- [20] I. Kononenko, M. Kovačič. Stochastic generation of multiple rules. V *Proc. Machine Learning Conf.*, Aberdeen, Scotland, 1992.
- [21] M. Kukar. *Ocenjevanje zanesljivosti klasifikacij in cenovno občutljivo kombiniranje metod strojnega učenja*. Doktorska disertacija, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2001.
- [22] M. Kukar, I. Kononenko. Reliable classifications with machine learning. V T. Elomaa, H. Mannila, H. Toivonen, (ur.), *Machine learning: ECML 2002: 13th European Conference on Machine Learning*, str. 219–231. Springer, 2002.
- [23] M. Li, P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer Verlag, 1993.
- [24] J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1984.
- [25] C. Nadeau, Y. Bengio. Inference for the generalization error. *Machine Learning*, 52: 239–281, 2003.
- [26] T. Niblett, I. Bratko. Learning decision rules in noisy domains. V *Expert Systems* 86, 12 1986. Brighton, UK.
- [27] S. L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1:317–328, 1997.
- [28] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

- [29] P. Smyth, R. M. Goodman. Rule induction using information theory. V G. Piatetsky-Shapiro, W. Frawley, (ur.), *Knowledge Discovery in Databases*. MIT Press, 1990.
- [30] V. Vapnik. *The nature of Statistical Learning*. Springer Verlag, 2nd edition, 2000.
- [31] S. M. Weiss, C. Kulikowski. *Computer Systems that Learn*. Morgan Kaufmann, 1991.
- [32] K. Woods, W. P. Kegelmeyer, K. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on PAMI*, 19(4):405–410, 1997.

## Poglavlje 4

# Predstavitev znanja in operatorji

*Trenutno v teorijo informacije ne moremo uvesti nikakega elementa, ki bi nam kaj povedal o vrednosti informacije za človeka. Ta izločitev človeškega elementa je zelo resna omejitev, ki omogoča obravnavati informacijo kot kvantiteto, ki jo je moč fizikalno meriti.*

*Mortimer Taube*

Učni algoritem na vhodu dobi predznanje in učne primere. Zatem preiskuje v prostoru možnih hipotez in kot rezultat vrne zaključno hipotezo. Za predznanje, učne, primere in prostor hipotez potrebujemo ustrezno predstavitev. Predstavitev mora omogočati učinkovito generiranje in uporabo znanja. Nekateri sistemi med izvajanjem dinamično tvorijo znanje za reševanje novih problemov in tudi to znanje mora biti ustrezno predstavljen. Med preiskovanjem prostora hipotez učni algoritem uporablja *operatorje* za spremnjanje trenutne hipoteze ali trenutne množice možnih hipotez.

Znanje lahko predstavimo na različne načine. V grobem ločimo naslednje predstavitve znanja:

- izjavni račun,
- predikatni račun prvega reda,
- diskriminantne in regresijske funkcije ter
- verjetnostne porazdelitve.

V nadaljevanju podrobnejše opišemo najpogostejše predstavitev, v zadnjem razdelku pa na kratko opišemo najpogostejše metode preiskovanja, ki jih uporabljajo sistemi za strojno učenje pri preiskovanju prostora hipotez. Predikatnega računa prvega reda v tem učebniku ne obravnavamo. Zainteresirani bralec naj poseže po dodatni literaturi, npr. [11, 10, 2].

## 4.1 Izjavni račun

*Z opazovanjem primerov lahko razpoznamo metodo.*

*Descartes*

Izjavni račun se uporablja za:

- predstavitev učnih primerov pri klasifikacijskih in regresijskih problemih ter
- predstavitev hipotez pri simboličnem učenju klasifikacijskih in regresijskih pravil.

### Atributna predstavitev učnih primerov

Najpogosteje se pri klasifikacijskih in regresijskih problemih uporablja atributna predstavitev učnih primerov. Atribut je spremenljivka, ki ima določeno množico možnih vrednosti. Atributom pravimo tudi značilke (*attribute, feature*). Vsak učni primer je opisan z vektorjem vrednosti atributov. Atributi so zvezni ali diskretni. Njihovo število je dano vnaprej. Atributno predstavitev definiramo z:

- množico atributov  $A = \{A_i, i = 0..a\}$ ;
- za vsak diskretni atribut  $A_i$  imamo množico možnih vrednosti  $\mathcal{V}_i = \{V_1, \dots, V_{m_i}\}$ ;
- za vsak zvezni atribut  $A_i$  imamo interval možnih vrednosti  $\mathcal{V}_i = [Min_i, Max_i]$ ;
- razred je podan z atributom  $A_0$ : če rešujemo klasifikacijski problem, potem je  $A_0$  diskretni atribut, če pa rešujemo regresijski problem, je  $A_0$  zvezni atribut;
- en učni primer je vektor vrednosti atributov  $u_j = \langle r^{(j)}, v^{(1,j)}, \dots, v^{(a,j)} \rangle$ , pri tem je razred označen z  $r^{(j)} = v^{(0,j)}$ ;
- množica učnih primerov je podana kot množica vektorjev  $\mathcal{U} = \{u_j, j = 1..n\}$ .

### Lastnosti atributov in njihove soodvisnosti

Atributi imajo različne lastnosti, ki so pomembne za strojno učenje:

**Šumski atribut** je skoraj vsak atribut v realnih podatkih. Nekatere vrednosti atributa so napake zaradi tipkarskih napak ali zaradi napak v meritvah.

**Pomanjkljiv atribut** je atribut, ki ima pri nekaterih učnih primerih manjkajoče vrednosti, ki jih mora učni algoritem pravilno upoštevati.

Poleg lastnosti posameznih atributov so pomembne tudi odvisnosti med atributi in razredom (odvisno spremenljivko):

**Naključen atribut** je nepovezan s ciljno spremenljivko in z ostalimi atributi. Tak atribut je nepomemben in ga je najbolje ignorirati, saj samo zamegljuje učni problem.

**Redundanten atribut** je atribut, katerega informacijo vsebuje že drug atribut ali množica atributov. Primer je atribut, ki je kopija drugega atributa.

**Koreliran atribut** je atribut, katerega informacijo delno vsebuje že nek drug atribut ali množica atributov. Bolj ko so atributi korelirani med seboj, večje so njihove soodvisnosti in več je redundancy.

**Močno soodvisni atributi glede na razred** Pri močnih soodvisnostih glede na razred je ciljno funkcijo težko odkriti, saj se atributi šele v kontekstu ostalih atributov pokažejo za pomembne. Pri diskretnih funkcijah je najmočnejša soodvisnost glede na razred v problemih parnosti, od katerih je najpreprostejša funkcija ekskluzivni ali (EXOR), kjer imamo dva binarna atributa  $A_1$  in  $A_2$  ter binarni razred  $A_0$ :

$$A_0 = (A_1 \neq A_2)$$

Enako težka je negacija te funkcije, t.j. ekvivalenca:

$$A_0 = (A_1 = A_2)$$

Posamezen atribut ničesar ne pove o razredu, zato ima večina algoritmov strojnega učenja pri reševanju tega problema težave. Problem parnosti je pospolitev te funkcije na več atributov. Več ko je atributov udeleženih v funkciji parnosti, težji je problem. Algoritem ReliefF, opisan v poglavju 5.1, je sposoben učinkoviti rešiti te vrste probleme.

Da nas lahko odvisnosti med atributi in razredom presenetijo, kaže naslednji primer, znan tudi kot Simpsonov paradoks [7]. Naj bosta  $A$  in  $B$  dve vrsti terapij pri pacientih. Z vsako so zdravili ločeno skupino 100-tih pacientov. Rezultati so sledeči (št. ozdravelih/št. vseh):

terapija A: 50/100

terapija B: 40/100

Navidezno je terapija A boljša od terapije B. Vendar dobimo drugačno sliko, če pogledamo podatke za starost pacientov v obeh skupinah v tabeli 4.1. Tako v skupini starejših kot v skupini mlajših se je izkazala terapija B za boljšo od terapije A. Če pri analizi podatkov ne upoštevamo relevantnih podatkov, ki so lahko skrite spremenljivke (v našem primeru starost), nas lahko podatki hitro zavedejo v napačen sklep.

### Klasifikacijska, regresijska in povezovalna pravila

Klasifikacijska in regresijska pravila so sestavljena iz pogojnega dela in sklepnega dela. Tako pravilom pravimo odločitvena pravila. Pogojni del je ponavadi konjunkcija pogojev  $C_i$ , sklepni del pa vsebuje pogoj  $C_0$ .

$$C_{i_1} \& C_{i_2} \& \dots \& C_{i_l} \Rightarrow C_0$$

starost	terapija		vsota
	A	B	
stari	2/10	30/90	32/100
mladi	48/90	10/10	58/100
vsota	50/100	40/100	90/200

Tabela 4.1: Uspešnost dveh terapij v odvisnosti od starosti pacientov (št. ozdravelih/št. vseh).

Če je atribut  $A_i$  diskreten, potem so pogoji oblike  $C_i = (A_i \in \mathcal{V}'_i)$ , kjer je  $\mathcal{V}'_i$  podmnožica možnih vrednosti atributa  $A_i$ :  $\mathcal{V}'_i \subset \mathcal{V}_i$ . Če je atribut  $A_i$  zvezen, potem so pogoji oblike  $C_i = (A_i \in \mathcal{V}'_i)$ , kjer je  $\mathcal{V}'_i$  podinterval intervala možnih vrednosti atributa  $A_i$ :  $\mathcal{V}'_i = [Min'_i, Max'_i]$ ,  $Min'_i \geq Min_i$ ,  $Max'_i \leq Max_i$ . Pri tem je v regresijskih problemih v sklepnu delu interval možnih vrednosti degeneriran v točko:  $Min'_0 = Max'_0$ . Pri klasifikacijskih problemih pa včasih podmnožica možnih vrednosti v sklepnu delu vsebuje eno samo vrednost  $|\mathcal{V}'_0| = 1$ .

Predstavitev pravil lahko posplošimo tako, da poleg konjunkcije v pogojnem delu pravila dovolimo poljubne logične operatorje: disjunkcijo, negacijo, ekvivalenco itd. Taka posplošitev močno poveča prostor hipotez, kar vpliva na kompleksnost učnega algoritma.

Pomembna je posplošitev sklepnega dela pravila, da razred ni eksaktno določen. Pri klasifikacijskih problemih uporabimo verjetnostno porazdelitev po razredih:  $\{P(r_k), k = 1 \dots m_0\}$ . Pri regresijskih problemih uporabimo interval zaupanja, tako da je  $Min'_0 < Max'_0$ , ali pa goštoto verjetnostne porazdelitve.

Ta posplošitev je uporabna tudi za učne primere, pri katerih razred ni zanesljivo znan. V takem primeru je verjetnostna porazdelitev boljša informacija in omogoča učnemu algoritmu večjo uspešnost, kot če opredelimo, da primer pripada razredu, ki se nam zdi najbolj verjeten.

Pogosteje je posplošitev atributne predstavitev učnih primerov. Če vrednost atributa za učni primer ni zagotovo znana, jo podamo z verjetnostno poradelitvijo, če gre za diskretni atribut, oziroma z gostoto verjetnostne porazdelitve, če gre za zvezni atribut. V primeru, da vrednost atributa ni znana, lahko uporabimo pogojno verjetnostno porazdelitev oziroma pogojno gostoto verjetnostne porazdelitve pri danem razredu, ki mu učni primer pripada.

*Povezovalna pravila* (*association rules*) so posplošitev klasifikacijskih pravil za probleme, kjer nas ne zanima samo ciljna spremenljivka, ampak nas zanimajo povezave (asociacije) med različnimi atributi. Sklepni del povezovalnega pravila tako ne vsebuje pogoja za spremenljivko  $C_0$ , ampak poljubno konjunkcijo pogojev:

$$C_{i_1} \ \& \ C_{i_2} \ \& \ \dots \ \& \ C_{i_l} \Rightarrow C_{j_1} \ \& \ C_{j_2} \ \& \ \dots \ \& \ C_{j_k}$$

V praksi je  $k$  redko večji kot 2.

### Splošnost pravil

Med pravili vpeljemo relacijo bolj splošen  $\prec$  oziroma bolj specifičen  $\succ$ . Pravimo, da je pravilo  $p_1$  bolj splošno kot pravilo  $p_2$ , če lahko iz  $p_1$  izpeljemo  $p_2$ :

$$p_1 \prec p_2 \Leftrightarrow p_1 \models p_2$$

Velja npr.:

$$(a \Rightarrow c) \prec (a \& b \Rightarrow c)$$

ker velja

$$(\neg a \vee c) \models (\neg a \vee \neg b \vee c)$$

Med naslednjimi pari pravil ne moremo določiti relacije bolj splošen oziroma bolj specifičen:

$$\begin{array}{ll} a \Rightarrow c & \text{in } b \Rightarrow c \\ a \& b \Rightarrow c & \text{in } a \& d \Rightarrow c \\ a \Rightarrow c_1 & \text{in } a \& b \Rightarrow c_2 \end{array}$$

Relacija  $\prec$  določa delno urejenost množice pravil. Lahko pa relacijo posplošimo tako, da upoštevamo le pogojne dele pravil:

$$(P \Rightarrow \_) \prec (P' \Rightarrow \_) \Leftrightarrow P' \models P$$

Pravimo tudi, da pogoj  $P$  pokriva pogoj  $P'$ . Velja npr.:

$$(a \Rightarrow \_) \prec (a \& b \Rightarrow \_)$$

ker velja

$$a \& b \models a$$

V tem primeru nas zanima samo, kako velik del prostora pravilo pokriva, ne pa tudi sklepni del pravila.

Na učne primere lahko gledamo kot na zelo specifična pravila, ki v pogojnem delu pravila vključujejo toliko pogojev, kolikor je atributov. Pravimo, da *pravilo pokriva učni primer*, če učni primer ustrezza pogojnemu delu pravila, t.j. če učni primer izpolnjuje pogojni del pravila. Pravimo, da *pravilo pravilno pokriva učni primer*, če razred primera ustrezza sklepнемu delu pravila, t.j. če pravilo za dani primer pravilno napove razred.

## Operatorji

Osnovna operatorja pri učenju simboličnih pravil sta posplošitev (generalizacija) in njej obratna operacija, specializacija. Pogoj pravila, ki uporablja konjunkcijo pogojev, lahko posplošimo na več načinov:

- iz pogoja izpustimo enega ali več konjunktov,
- v pogoju, ki testira pripadnost vrednosti diskretnega atributa dani podmnožici, razširimo podmnožico vrednosti z eno ali z več vrednostmi,
- v pogoju, ki testira pripadnost vrednosti zveznega atributa danemu intervalu, razširimo interval s povečanjem zgornje meje in/ali zmanjšanjem spodnje meje intervala.

Če uporabljamo splošnejšo sintakso, kjer so dovoljene tudi negacija, disjunkcija itd., se poveča tudi število načinov posploševanja pravil (npr. zamenjava konjunkcije z disjunkcijo, disjunktivno dodajanje pogojev itd.).

Pogoj pravila, ki uporablja konjunkcijo pogojev, specializiramo z obratnimi operacijami:

- dodamo en ali več konjunktov v pogoju,
- v pogoju, ki testira pripadnost vrednosti diskretnega atributa dani podmnožici, zmanjšamo podmnožico vrednosti za eno ali več vrednosti,
- v pogoju, ki testira pripadnost vrednosti zveznega atributa danemu intervalu, zožimo interval s povečanjem spodnje meje in/ali zmanjšanjem zgornje meje intervala.

Ob spremembi pogojnega dela pravila se, glede na pokritost množice učnih primerov, spremeni tudi sklepni del pravila. Če rešujemo klasifikacijski problem, se spremeni verjetnostna porazdelitev po posameznih razredih. Če rešujemo regresijski problem, se spremeni interval zaupanja.

## Prostor hipotez

Pri simboličnem učenju pravil je prostor hipotez  $\mathcal{H}$  definiran kot potenčna množica vseh možnih pravil. Ena hipoteza  $H \in \mathcal{H}$  je torej množica pravil. Pomembni sta naslednji lastnosti hipoteze:

**Konsistentnost** Hipoteza  $H$  je konsistentna, če za vsako pravilo  $p \in H$  velja, da pravilno napove razred za vse učne primere, ki jih pokriva pogojni del pravila.

**Kompletnost** Hipoteza  $H$  je kompletна, če za vsak učni primer obstaja pravilo  $p \in H$ , katerega pogojni del ga pokrije.

## Odločitvena in regresijska drevesa

Poseben primer množice odločitvenih pravil je odločitveno drevo (*decision tree*). Odločitveno drevo je sestavljeno iz notranjih vozlišč, ki ustrezajo atributom, vej, ki ustrezajo podmnožicam vrednosti atributov, in listov, ki ustrezajo razredom. Če je razred zvezen, je drevo regresijsko (*regression tree*). Ena pot v drevesu od korena do lista ustreza enemu odločitvenemu pravilu. Pogoji (pari atribut–podmnožica vrednosti), ki jih srečamo na poti, so konjunktivno povezani. Vsako drevo lahko pretvorimo v množico toliko odločitvenih pravil, kolikor ima drevo listov. Po drugi strani pa vsake množice pravil ni možno neposredno pretvoriti v odločitveno drevo.

Predstavitev znanja z odločitvenimi in regresijskimi drevesi posplošimo tako, da namesto enega atributa v notranjem vozlišču nastopa poljubna funkcija atributov. Konstantno vrednost razreda v listih lahko nadomestimo s funkcijo. Npr. pri regresijskih drevesih se pogosto v listih drevesa uporablja linearna kombinacija podmnožice atributov. Taka drevesa so prožnejša za predstavitev znanja od navadnih odločitvenih in regresijskih dreves.

Če ima vsako notranje vozlišče natanko dva naslednika, pravimo, da je drevo binarno. Vsako odločitveno drevo lahko pretvorimo v binarno.

## 4.2 Diskriminantne in regresijske funkcije

*To, čemur roža pravimo, dišalo bi prav tako lepo z imenom drugim.*

*William Shakespeare*

Diskriminantne funkcije se uporabljajo za reševanje klasifikacijskih problemov. Te funkcije implicitno opisujejo mejne ploskve med posameznimi razredi v  $a$ -dimenzionalnem prostoru, kjer je  $a$  število atributov. Naj bo  $V = \langle v_1, \dots, v_a \rangle$  vektor vrednosti vseh atributov  $A_1, \dots, A_a$ . Za vsak razred  $r_k, k = 1 \dots m_0$  potrebujemo eno diskriminantno funkcijo z  $a$  argumenti:  $g_k(v_1, \dots, v_a) = g_k(V)$ , ki preslikava primer iz  $a$ -dimenzionalnega prostora primerov v realno število. Mejna ploskev med razredoma  $r_i$  in  $r_j$  je definirana z enačbo:

$$g_i(V) - g_j(V) = 0$$

Diskriminantne funkcije definirajo  $m_0(m_0 - 1)/2$  mejnih ploskev med razredi.

Naloga učnega algoritma je poiskati (čim preprostješe) funkcije  $g_k, k = 1 \dots m_0$  tak, da za vsak učni primer  $u_j, j = 1 \dots n$ , ki ga zapišemo kot

$$u_j = \langle r^{(j)}, v^{(1,j)}, \dots, v^{(a,j)} \rangle$$

velja, da je vrednost funkcije, ki ustreza dejanskemu razredu  $r^{(j)} = r_i$  večja od vrednosti funkcij za vse ostale razrede  $r_k, k = 1 \dots m_0, k \neq i$ :

$$g_i(v^{(1,j)}, \dots, v^{(a,j)}) > g_k(v^{(1,j)}, \dots, v^{(a,j)})$$

*Regresijske funkcije* se uporabljajo za reševanje regresijskih problemov. Regresijska funkcija preslikava primera v aproksimacijo vrednosti  $v_0$  odvisne spremenljivke  $A_0$ :

$$g(v_1, \dots, v_a) = v'_0$$

Pri tem velja  $v'_0 \approx v_0$ . Regresijska funkcija definira regresijsko hiperploskev v  $(a+1)$ -dimensionalnem prostoru. Naloga učnega algoritma je poiskati čim preprostejšo funkcijo  $g$ , ki čim bolj natančno napove vrednost odvisne spremenljivke.

Če nimamo opravka s simboličnimi pravili, definicije diskriminantnih in regresijskih funkcij predpostavlja, da so atributi zvezni. Binarni (dvovrednostni) diskretni atributi so še sprejemljivi za tako predpostavko, večvrednostni pa ne, ker urejenost med vrednostmi ni smiselna. Zato običajno diskretni atribut z več kot dvema vrednostima nadomestimo s toliko binarnimi atributi, kolikor vrednosti ima diskretni atribut.

Najbolj pogoste diskriminantne in regresijske funkcije so:

- linearne funkcije,
- kvadratne funkcije,
- posplošene linearne funkcije in
- nevronske mreže.

## Linearne funkcije

Linearne funkcije imajo obliko:

$$g(V) = w_1 v_1 + w_2 v_2 + \dots + w_a v_a + w_{a+1}$$

kjer so  $w_i$  koeficienti,  $v_i$  pa vrednosti atributov. Linearne diskriminantne funkcije definirajo mejne hiperravnine:

$$G_{ij}(V) = g_i(V) - g_j(V) = 0$$

$$(w_{i,1} - w_{j,1})v_1 + \dots + (w_{i,a} - w_{j,a})v_a + (w_{i,a+1} - w_{j,a+1}) = 0$$

Če za množico učnih primerov velja, da obstaja konsistentna množica mejnih hiperravnin med razredi, pravimo, da so razredi *linearno ločljivi* (*linearly separable*). Naloga učnega algoritma je za vsak razred  $r_k$  določiti vektor uteži

$$W_k = \langle w_{k,1}, \dots, w_{k,a+1} \rangle$$

Pri regresijskem problemu linearna regresijska funkcija definira regresijsko hiperravnino v  $(a+1)$ -dimenzionalnem prostoru. Regresijskim problemom, ki se dajo zadovoljivo rešiti z linearno regresijsko funkcijo, pravimo *linearni problemi*.

## Kvadratne funkcije in pospoljene linearne funkcije

Kvadratne funkcije imajo obliko:

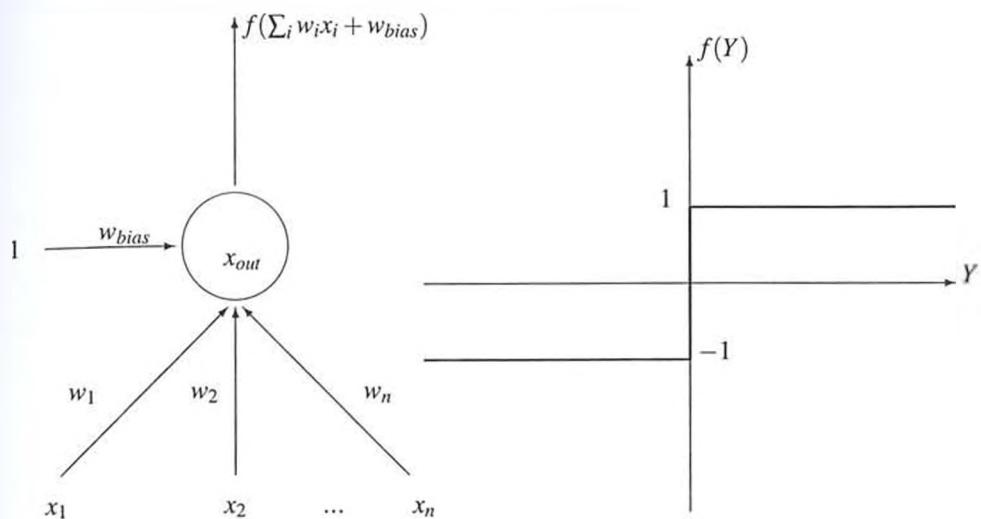
$$g(V) = \sum_{j=1}^a w_{jj} v_j^2 + \sum_{j=1}^{a-1} \sum_{k=j+1}^a w_{jk} v_j v_k + \sum_{j=1}^a w_j v_j + w_{a+1}$$

Vsaka kvadratna diskriminantna ali regresijska funkcija ima  $(a+1)(a+2)/2$  uteži, ki jih mora določiti učni algoritem.

Funkcije  $\Phi$  so pospoljitev linearnih (in kvadratnih funkcij) na poljubne funkcije, ki se dajo izraziti kot linearna kombinacija parametrov, dobljenih iz osnovnih  $a$  atributov:

$$\Phi(V) = w_1 f_1(V) + w_2 f_2(V) + \dots + w_M f_M(V) + w_{M+1}$$

Čim bolj zapletene funkcije dovoljujemo, tem večji je  $M$  in tem zahtevnejši je učni proces. Z večanjem števila parametrov  $M$  kaj hitro pridemo do funkcij, ki se preveč prilagodijo učni množici (niso dovolj splošne) in niso uporabne za klasifikacijo novih primerov oziroma za napovedovanje vrednosti odvisne spremenljivke. Princip najkrajšega opisa (MDL) nas uči, da moramo najti čim preprostejšo množico diskriminantnih funkcij oziroma čim preprostejšo regresijsko funkcijo, če naj bosta klasifikacija in napovedovanje vrednosti odvisne spremenljivke zanesljivi.



Slika 4.1: Umetni nevron in pragovna funkcija.

### Umetne nevronske mreže

Umetne nevronske mreže abstrahirajo delovanje biološkega nevrona na uteženo vsoto vhodnih signalov \$x\_i\$, ki se transformira s pragovno funkcijo \$f\$ v aktivni izhodni signal (\$f = 1\$) oziroma neaktivni signal (\$f = 0\$ ali \$-1\$). Na sliki 4.1 je prikazan umetni nevron, ki izračunava funkcijo:

$$x_{out} = f\left(\sum_i w_i x_i + w_{bias}\right)$$

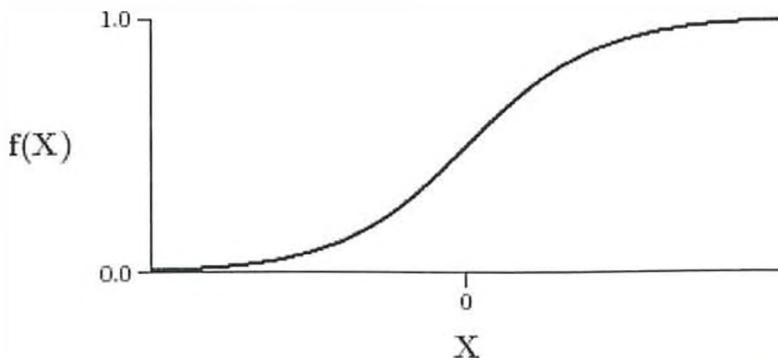
Pragovna funkcija \$f\$ je lahko definirana z:

$$f(X) = \begin{cases} 1 & X > 0 \\ -1 & X \leq 0 \end{cases}$$

Pogosteje se uporablja sigmoidna zvezna in zvezno odvedljiva funkcija (glej sliko 4.2):

$$f(X) = \frac{1}{1 + e^{-X}}$$

Umetne nevrone lahko povezujemo v skoraj poljubne topologije [12, 8]. Najpogosteje se uporablja usmerjene večnivojske umetne nevronske mreže (*feedforward multilayered artificial neural networks*). Nevroni so grupirani v več nivojev: vhodni in izhodni nivo in poljubno število skritih nivojev. Vhodni nivo ima nalogo vhodne signale poslati naprej do vseh nevronov na skritem nivoju (torej ničesar ne izračunava, zato vhodnim vozliščem običajno ne rečemo



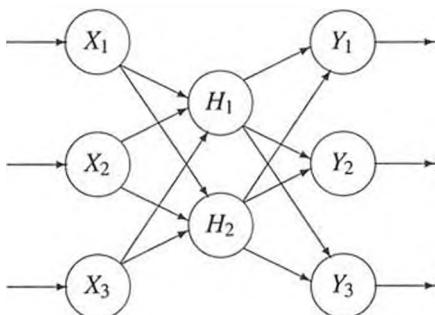
Slika 4.2: Sigmoidna pragovna funkcija  $f(X) = \frac{1}{1+e^{-X}}$ .

nevroni). Vsak nevron iz skritega nivoja sprejme signal od vsakega nevrona iz prejšnjega nivoja in pošlje svoj izhodni signal vsem nevronom naslednjega nivoja. Izhod nevronov na izhodnem nivoju predstavlja izhod nevronske mreže (glej sliko 4.3).

Povezavam med nevroni pravimo *sinapse*. Vsaka sinapsa ima pridruženo utež, ki se pomoži s signalom, ki potuje po sinapsi. Matrika uteži na sinapsah definira funkcijo, ki jo izračunava nevronska mreža. Naloga učnega algoritma je nastaviti uteži tako, da nevronska mreža izračunava želeno funkcijo.

V primeru, da skritih nivojev ni, je nevronska mreža enonivojska. Če rešujemo klasifikacijske probleme, se enonivojska nevronska mreža uporablja za reševanje linearnih klasifikacijskih problemov, t.j. problemov, kjer so razredi linearno ločljivi. Vsak izhodni nevron računa linearno diskriminantno funkcijo, ki ustreza enemu razredu. Nevron z največjim izhodom "zmaga". Izbrani razred za dane vhodne podatke je določen z maksimalnim izhodnim signalom. Če rešujemo regresijski problem, imamo en sam izhodni nevron, ki je pri enonivojski nevronske mreži edini, ki sploh kaj računa. Enonivojsko nevronska mreža lahko uporabljamo za reševanje linearnih regresijskih problemov.

Najpogosteje se v praksi uporabljajo dvonivojske in trinivojske nevronske mreže (z enim oz. dvema skritima nivojem). Za trinivojske nevronske mreže je dokazano, da lahko poljubno natančno aproksimirajo vsako funkcijo. Število vhodnih vozlišč je podano s številom (zveznih) atributov. Število izhodnih nevronov ponavadi ustreza številu razredov za klasifikacijski problem, za regresijski problem pa imamo en sam izhodni nevron. S številom skritih nivojev in s številom nevronov v vsakem skritem nivoju je določena zmožnost opisovanja funkcij z dano nevronska mrežo oziroma je določen razred kompleksnosti funkcij, ki jih lahko aproksimiramo. Z večanjem obeh parametrov se povečuje kompleksnost funkcij, hkrati pa se povečuje tudi število uteži, ki jih mora učni algoritem nastaviti. S tem se



Slika 4.3: Usmerjena dvonivojska umetna nevronska mreža s tremi vhodi, dvema skritima nevronoma in tremi izhodnimi nevroni.

povečuje zahtevnost učenja, pa tudi nevarnost prevelike prilagoditve nevronske mreže učnim primerom. Naloga učnega algoritma je zato poiskati čim preprostejšo nevronske mreže, ki dovolj dobro klasificira učne primere oziroma napove vrednost odvisne spremenljivke.

### 4.3 Verjetnostne porazdelitve

*Naključni pojavi okoli nas se zdijo ireverzibilni samo zato, ker je naše življenje zanemarljivo kratko v primerjavi z ogromnim časom do povratka.*

*Mark Kac in Stanislaw Ulam*

#### Bayesov klasifikator

Naj bo

$P(r_k)$  – apriorna verjetnost razredov  $r_k$ ,  $k = 1, \dots, m_0$ ,

$V = \langle v_1, \dots, v_a \rangle$  – vektor vrednosti atributov,

$d(V)$  – klasifikator, ki preslika atributni opis primera v razred,

$t(V)$  – idealni klasifikator, ki vedno “ugane” pravilni razred,

$P(V)$  – apriorna verjetnost primera z atributnim opisom  $V$ ,

$P(V|r_k)$  – pogojna verjetnost primera z atributnim opisom  $V$  pri razredu  $r_k$ .

Bayesov klasifikator (*Bayesian classifier*)  $d_B(V)$  definiramo kot klasifikator, za katerega je verjetnost napačne klasifikacije minimalna:

$$\forall d(\cdot) : P(d_B(V) \neq t(V)) \leq P(d(V) \neq t(V))$$

Povprečna klasifikacijska točnost Bayesovega klasifikatorja je podana z

$$U_B = P(d_B(V) = t(V))$$

Iz definicije sledi, da je Bayesov klasifikator podan z:

$$d_B(V) = \arg \max_{r_k} P(r_k|V) = \arg \max_{r_k} P(V|r_k)P(r_k)$$

in je njegova klasifikacijska točnost:

$$U_B = \sum_V \max_{k=1}^{m_0} P(r_k|V)P(V)$$

Bayesov klasifikator določa teoretično najboljši klasifikator, ki bi ga lahko učni algoritem dosegel. Ker pa apriornih in pogojnih verjetnostnih porazdelitev ne poznamo, jih lahko samo bolj ali manj natančno aproksimiramo iz učnih podatkov. Učni algoritem je tem boljši, čim bolje aproksimira Bayesov klasifikator.

### Učni primeri kot verjetnostna porazdelitev

Najpreprostejša oblika aproksimacije verjetnostne porazdelitve je dinamična aproksimacija porazdelitve  $P(r_k|V)$  iz shranjenih opisov učnih primerov. Algoritmi, ki kot znanje shranijo kar učne primere, za vsak nov primer, ki ga je potrebno klasificirati, izpeljejo verjetnostno porazdelitev iz shranjenih primerov. Uporabljata se dve osnovni oblici izračuna verjetnostne porazdelitve:

**K-najbližjih sosedov** (*K-nearest neighbors*): algoritom za dani novi primer  $V$  poišče v shranjeni množici učnih primerov  $K$  najbližjih (najbolj podobnih) primerov in oceni verjetnostno porazdelitev

$$P(r_k|V)$$

iz relativne porazdelitve  $K$  primerov po razredih. Ta porazdelitev je lahko dodatno utežena z razdaljami primerov od novega primera.

**Ocena gostote verjetnosti** (*density estimation*): izbere se *jedrna funkcija* (*kernel function*)  $g$ , ki definira vpliv enega primera na gostoto verjetnosti v svoji okolici tako, da vpliv z razdaljo pada. Za novi primer se izračuna verjetnost  $P(V|r_k)$  tako, da se za vse učne primere  $u_i, i = 1 \dots m_k$  iz razreda  $r_k$  izračuna relativna gostota verjetnosti:

$$P(V|r_k) = \frac{1}{m_k} \sum_{i=1}^{m_k} g(V - u_i)$$

Na verjetnostno porazdelitev vplivajo vsi učni primeri, njihov vpliv pa določa jedrna funkcija glede na razdaljo učnega primera  $u_i$  od novega primera. Izračun poenostavimo, če zanemarimo preveč oddaljene primere, ki bi imeli premajhen vpliv.

### Naivni Bayesov klasifikator

Naivni Bayesov klasifikator (naive Bayesian classifier) predpostavlja pogojno neodvisnost vrednosti različnih atributov pri danem razredu [5, 6]. Izpeljemo ga s pomočjo Bayesovega pravila:

$$P(r_k|V) = P(r_k) \frac{P(V|r_k)}{P(V)} \quad (4.1)$$

Ob predpostavki neodvisnosti vrednosti atributov  $v_i$  pri danem razredu  $r_k$ :

$$P(v_1 \& \dots \& v_a|r_k) = \prod_{i=1}^a P(v_i|r_k)$$

dobimo

$$P(r_k|V) = \frac{P(r_k)}{P(V)} \prod_{i=1}^a P(v_i|r_k)$$

Ob ponovni uporabi Bayesovega pravila:

$$P(v_i|r_k) = P(v_i) \frac{P(r_k|v_i)}{P(r_k)}$$

dobimo

$$P(r_k|V) = P(r_k) \frac{\prod_{i=1}^a P(v_i)}{P(V)} \prod_{i=1}^a \frac{P(r_k|v_i)}{P(r_k)}$$

Faktor:

$$\frac{\prod_{i=1}^a P(v_i)}{P(V)}$$

je neodvisen od razreda, zato ga lahko izpustimo in dobimo končno formulo:

$$P(r_k|V) = P(r_k) \prod_{i=1}^a \frac{P(r_k|v_i)}{P(r_k)}$$

Učni algoritem s pomočjo učne množice podatkov aproksimira verjetnosti na desni strani enačbe. Znanje naivnega Bayesovega klasifikatorja je torej tabela aproksimacij apriornih verjetnosti razredov  $P(r_k), k = 1 \dots m_0$  in tabela pogojnih verjetnosti razredov  $r_k, k = 1 \dots m_0$  pri dani vrednosti  $v_i$  atributa  $A_i, i = 1 \dots a$ :  $P(r_k|v_i)$ .

## 4.4 Učenje kot preiskovanje

*Kdor hoče, najde pot. Kdor noče, najde izgovor.*

Arabski pregovor

V tem razdelku na kratko povzamemo metode preiskovanja prostora hipotez. Učni algoritmi uporabljajo različne metode preiskovanja. Za preiskovanje prostora hipotez potrebujemo začetno hipotezo  $H_z$  in množico operatorjev, ki hipotezo transformirajo v množico naslednikov:  $\{H' \mid H_z \rightarrow H'\}$ . S tem je definiran prostor hipotez

$$\mathcal{H} = \{H \mid H_z \rightarrow H\}$$

Z  $\rightarrow$  smo označili nič ali večkratno uporabo operatorjev. Potrebujemo še oceno kvalitete hipoteze  $q$  pri dani množici učnih primerov  $E$  in predznanju  $B$ . Naloga (idealnega) učnega algoritma je poiskati hipotezo  $H_0 \in \mathcal{H}$ , ki maksimizira oceno kvalitete:

$$H_0 = \arg \max_{H \in \mathcal{H}} q(E, B, H)$$

Algoritmi za strojno učenje pri preiskovanju prostora možnih hipotez uporabljajo različne strategije. Osnovne strategije preiskovanja so [1, 3]: izčrpno preiskovanje, omejeno izčrpno iskanje (razveji in omeji), metoda "najprej najboljši", požrešno iskanje, iskanje v snopu, lokalna optimizacija, gradientno iskanje, simulirano ohlajanje ter genetski algoritmi. V enem algoritmu lahko kombiniramo več strategij. Poleg iskanja hipoteze potrebujemo preiskovanje tudi za nekatere druge podprobleme strojnega učenja:

- izbira podmnožice atributov (*feature subset selection*),
- diskretizacija zveznih atributov (*discretization of continuous (numeric) attributes*),
- konstruktivna indukcija (*constructive induction*) - sestavljanje novih atributov,
- nastavitev parametrov (*parameter tuning*).

V nadaljevanju so na kratko opisane posamezne strategije, bolj podrobno in v širšem kontekstu, ki ni nujno vezan na strojno učenje, pa so strategije preiskovanja obravnavane v poglavju 13.

### Izčrpno preiskovanje

Najpreprostejša metoda preiskovanja je izčrpno preiskovanje (*exhaustive search*), ki preišče ves prostor hipotez in izbere najbolje ocenjeno hipotezo. Iskanje v globino (*depth-first search*) ima linearo prostorsko zahtevnost v odvisnosti od globine iskanja. Slaba stran iskanja v globino je, da lahko zgreši rešitev, ki je blizu začetnemu skanju, in mnogo časa izgubi pri preiskovanju globokega (potencialno neskončnega) poddrevesa. Iskanju v neskončnih poddrevesih se izognemo z omejevanjem globine. Pri končnih prostorih lahko uporabimo iskanje v globino brez omejene globine.

Druga izčrpna strategija je preiskovanje v širino (*breadth-first search*). Tu pregledujemo drevo stanj po vrsti po nivojih. Na ta način se izognemo ciklanju in imamo zagotovljeno, da bomo našli najkrajšo pot do rešitve, če ta obstaja. Slaba stran je, da prostorska zahtevnost raste eksponentno z globino iskanja.

Namesto iskanja v širino lahko uporabimo *iterativno poglabljanje* (*iterative deepening*). V tem primeru uporabljam iskanje v globino z omejeno globino, ki jo iterativno povečujemo. Prostorska zahtevnost je enaka kot pri iskanju v globino, način preiskovanja novih stanj pa enak kot pri preiskovanju v širino. Slaba stran je povečan čas preiskovanja, saj moramo za vsako globino še enkrat preiskati vse, kar smo že preiskali.

Na žalost je v večini praktičnih problemov prostor hipotez zelo velik, ponavadi celo neskončen. Če imamo opravka z zveznim prostorom, je prostor neštevno neskončen, kar teoretično pomeni, da se ga niti v neskončnem času ne da vsega pregledati. To pomeni, da je izčrpano preiskovanje najpogosteje neuporabno.

Če imamo npr. deset binarnih atributov in dva razreda, je vseh možnih hipotez (Boolovih funkcij desetih argumentov):

$$2^{2^{10}} > 10^{300}$$

Tudi, če bi algoritem pregledal biljon ( $10^{12}$ ) hipotez v sekundi, bi potreboval več kot  $10^{280}$  let, da bi pregledal vse prostor hipotez.

Zadovoljiti se moramo s preiskovanjem skromne podmnožice celotnega prostora možnih hipotez. Pri tem si pomagamo s hevrističnimi ocenami, ki omejujejo in usmerjajo iskanje. Pogosto pri usmerjanju iskanja uporabimo predznanje, ki je vezano na dano problemsko področje.

### Omejeno izčrpno iskanje (razveji in omeji)

Omejenemu izčrpnemu preiskovanju pravimo tudi metoda "razveji in omeji" (*branch and bound*). Gre za uporabno različico izčrpnega preiskovanja, ki uporablja naslednjo informacijo:

- operator *naslednika* " $\rightarrow$ ", ki *delno uredi prostor hipotez*; ta dodatna zahteva je potrebna zato, da se preiskovanje ne bi v nedogled vračalo do hipotez, ki jih je že pregledal;
- (hevristično) oceno  $q$  kvalitete hipoteze  $H$  pri dani množici učnih primerov  $E$  in danem predznanju  $B$ :

$$q(E, B, H) \in \mathcal{R}$$

- (hevristično) oceno  $\max$ , ki pri dani množici primerov  $E$  in danem predznanju  $B$  oceni zgornjo mejo kvalitete vseh možnih naslednikov dane hipoteze  $H$ :

$$\forall H', H \xrightarrow{*} H' : \max(E, B, H) \geq q(E, B, H')$$

- če algoritem pozna hipotezo  $H_0$ , za katero velja

$$q(E, B, H_0) \geq \max(E, B, H)$$

potem ni potrebno preiskati nobenega naslednika hipoteze  $H$ .

Oceni  $q$  in  $\max$  služita za zavračanje delov prostora hipotez, ki po teh ocenah vsebujejo samo slabše hipoteze od trenutno najboljše. Od tod tudi izvira ime razveji in omeji: dano hipotezo razveji v potencialne naslednike in zavrzi (omeji) neperspektivne naslednike. Omejeno izčrpno iskanje lahko uporablja za osnovo iskanje v širino ali iskanje v globino. Osnovna strategija iskanja v širino je tudi pri omejenem izčrpnem iskanju prostorsko zahtevna. Ker je globina iskanja implicitno omejena z oceno  $\max$ , je zato bolje uporabiti iskanje v globino. Da se iskanje v globino ustavi, nam zagotavlja izbira operatorjev, izbira ocene kvalitete hipoteze  $q$  in ustreznega izbira ocene zgornje meje  $\max$  kvalitete vseh naslednikov hipoteze.

Oglejmo si primer. Pri učenju množic simboličnih konjunktivnih pravil se za delno urejanje prostora hipotez uporablja operator, ki definira relacijo "bolj splošen" oziroma "bolj specifičen". Preprosta ocena kvalitete hipoteze je definirana s:

$$q(E, B, H) = \begin{cases} \text{splošnost}(H) & \forall e \in E : B \& H \models e \\ -\infty & \exists e \in E : B \& H \not\models e \end{cases}$$

Če je hipoteza nekonsistentna z učnimi primeri, je njena ocena najslabša možna, sicer pa je ocena tem boljša, čim bolj splošna je hipoteza. Ta kriterij pri konjunktivnih pravilih ustreza kriteriju čim manjše kompleksnosti pravil. To pomeni, da će med preiskovanjem prostora pravil najdemo konsistentno pravilo, lahko zavrzemo vsa bolj specifična pravila, ker je njihova ocena slabša:

$$\forall e \in E : B \& H \models e \Rightarrow \forall H' : H \longrightarrow H' : q(E, B, H) > q(E, B, H')$$

### Metoda "najprej najboljši"

Algoritem je podoben algoritmu omejenega izčrpnega iskanja, le da v vsakem koraku generira naslednike tiste hipoteze  $H_{\max}$ , ki ima najvišjo oceno zgornje meje kvalitete naslednikov:

$$H_{\max} = \arg \max_H [\max(E, B, H)]$$

Oceni  $q$  in  $\max$  omogočata poleg zavračanja delov prostora hipotez, ki vsebujejo samo slabše hipoteze od trenutno najboljše hipoteze, tudi usmerjanje iskanja v bolj obetavne dele prostora. Čim hitreje najdemo relativno dobro hipotezo  $H_0$ , tem večji del prostora bo z omejenim izčrpnim iskanjem zavrnjen:

$$q(E, B, H_0) \geq \max(E, B, H) \Rightarrow \forall H' : H \longrightarrow H' : \text{zavri } H'$$

Metoda "najprej najboljši" je v splošnem hitrejša kot omejeno izčrpno iskanje, vendar je prostorsko precej bolj zahtevna. Algoritem z omejenim izčrpnim iskanjem realiziramo z iskanjem v globino, ki shrani samo pot od začetne hipoteze  $H_z$  do trenutne hipoteze  $H$  in trenutno najboljšo hipotezo  $H_0$ . Algoritem "najprej najboljši" mora hraniti vse hipoteze, katerih naslednikov še ni generiral. To brez dodatnih omejitve pomeni, da prostorska zahtevnost v najslabšem primeru narašča eksponentno z dolžino poti.

Prostorska zahtevnost algoritma z omejenim izčrpnim iskanjem raste linearno, zato pa časovna zahtevnost raste eksponentno. Časovna zahtevnost algoritma "najprej najboljši" je v najslabšem primeru eksponentna, v najboljšem pa linearne. Učinkovitost obeh algoritmov je odvisna od funkcij  $q$  in  $\max$ .

## Požrešno iskanje

Požrešnemu iskanju (*greedy search*) pravimo tudi metoda "samo najboljši" (best-only search). Algoritem požrešnega preiskovanja uporablja oceno  $\max_1$ , ki upošteva tudi kvaliteto trenutne hipoteze:

$$\max_1(E, B, H) = \max(\max(E, B, H), q(E, B, H))$$

Po tej metodi v vsakem koraku izmed možnih naslednikov trenutne hipoteze  $H$  izberemo samo najboljšo hipotezo  $H_{\max}$ :

$$H_{\max} = \arg \max_{H' \in H \rightarrow H'} \max_1(E, B, H')$$

Vse ostale naslednike hipoteze  $H$  zavrzemo.  $H_{\max}$  pa postane nova trenutna hipoteza. Postopek ponavljamo, dokler ni trenutna hipoteza  $H$  boljša od najboljšega naslednika  $H_{\max}$ :

$$q(E, B, H) > \max_1(E, B, H_{\max})$$

V tem primeru je rezultat  $H$ .

V splošnem požrešni algoritmi ne iščejo samo v prostoru hipotez  $\mathcal{H}$ , ampak v nadprostoru  $\mathcal{D} \supset \mathcal{H}$ , ki vsebuje tudi delne hipoteze. Princip iskanja v prostoru  $\mathcal{D}$  je enak, le da se namesto ocene  $\max_1$  uporablja ocena  $\max_2$ , ki oceni zgornjo mejo kvalitete naslednikov trenutne delne hipoteze  $D$ :

$$\forall H', D \xrightarrow{*} H', H' \in \mathcal{H}, D \in \mathcal{D} : \max_2(E, B, D) \geq \hat{q}(E, B, H')$$

## Iskanje v snopu

Iskanje v snopu (*beam search*) je pospolištev požrešnega algoritma. Namesto ene same najbolj perspektivne hipoteze iskanje v snopu ohranja  $M$  najbolj perspektivnih hipotez. Številu  $M$  pravimo velikost snopa (*beam size*). V enem koraku algoritom poišče naslednike vsake od  $M$  hipotez in izmed naslednikov, vključno s trenutnimi hipotezami, izbere  $M$  najboljših. To ponavlja, dokler ni več možno generirati boljših naslednikov.

Z večanjem parametra  $M$  se povečuje velikost pregledanega prostora in s tem tudi zahtevnost algoritma. Za  $M = 1$  dobimo požrešni algoritmom.

## Lokalna optimizacija

Algoritem lokalne optimizacije se razlikuje od požrešnega algoritma po naslednjem:

- začetna hipoteza  $H_z \in \mathcal{H}$  je naključno generirana,
- lokalna optimizacija išče samo v prostoru hipotez  $\mathcal{H}$ ,
- ponavadi je število možnih operatorjev za generiranje naslednikov večje kot pri požrešnem algoritmu.

Lokalna optimizacija začne iz naključno zgrajene hipoteze in jo poskuša z operatorji lokalno optimizirati. Od množice operatorjev je odvisna kompleksnost algoritma. Ker je algoritem stohastičen, lahko zaporedna izvajanja vrnejo različne hipoteze. Zato ponavadi lokalno optimizacijo poženemo večkrat (odvisno od razpoložljivega časa) in obdržimo najboljšo hipotezo izmed vseh, ki so jih izvajanja odkrila.

### Gradientno iskanje

Gradientno iskanje je lokalna optimizacija v zveznem prostoru hipotez. Namesto uporabe operatorjev in premika v smeri maksimalne kvalitete nove hipoteze se uporablja odvod funkcije kvalitete:

$$q'(E, B, H) = dq(E, B, H)/dH$$

Hipotezo  $H$  spremenimo v smeri, ki lokalno maksimizira vrednost funkcije. Če je hipoteza podana s parametrom  $p$ :  $H = H(p)$ , potem hipotezo spremenimo v smeri, ki maksimizira odvod  $q' = dq/dp$ . Ker nimamo diskretnih operatorjev, potrebujemo parameter  $\eta$ , ki določa velikost koraka:

$$p = p + \eta \frac{dq(E, B, H(p))}{dp}$$

Postopek ponavljamo, dokler kvaliteta hipoteze narašča dovolj hitro, t.j., če je prirastek kvalitete v eni iteraciji večji od  $\varepsilon$ .

### Simulirano ohlajanje

Simulirano ohlajanje (*simulated annealing*) je posplošitev lokalne optimizacije. Osnovna ideja izhaja iz termodinamike. Boltzmanov porazdelitveni zakon pravi, da je verjetnost, da se atom nahaja v nekem stanju z energijo  $E_i$  sorazmerna:

$$e^{-\frac{E_i}{T}}$$

kjer je s  $T$  označena *temperatura* (merjena v joulih). Torej je verjetnost stanja z najmanjšo energijo tem večja, čim nižja je temperatura. Za doseg stanja s čim manjšo energijo, kar ustreza optimalnemu stanju (npr. pravilen kristal), je potrebno snov najprej segreti in zatem počasi ohlajati. Če snov prehitro ohladimo, dobimo suboptimalno stanje (nepravilen kristal). Čim počasneje ohlajamo snov, tem večja je verjetnost za boljše stanje.

Idejo lahko uporabimo za kombinatorično optimizacijo tako, da vpeljemo v algoritem lokalne optimizacije verjetnostno obnašanje [9]. Naslednik trenutne hipoteze ni izbran deterministično ampak stohastično. Čim bolje je ocenjen naslednik trenutne hipoteze, tem večja je verjetnost, da bo izbran. V oceno verjetnosti vpeljemo dodaten parameter, temperaturo  $T$ , ki določa stopnjo stohastičnosti. Čim višja je temperatura, tem bolj naključno se algoritem giblje po prostoru hipotez. Čim nižja je temperatura, tem bolj je izvajanje deterministično. Ko je temperatura  $T = 0$ , postane algoritem determinističen in se obnaša kot lokalna optimizacija.

Algoritem začne z naključno hipotezo in se na začetku pri visoki temperaturi stohastično giblje po prostoru hipotez. Izmed vseh naslednikov hipoteze  $H$  naključno izbere enega naslednika  $H'$ . Če je kvaliteta naslednika boljša od originalne hipoteze:  $q(E, B, H') > q(E, B, H)$ ,

ga sprejme za novo trenutno hipotezo  $H_0$  z verjetnostjo  $P = 1$ , sicer pa z verjetnostjo, ki je sorazmerna kvaliteti naslednika in obratno sorazmerna temperaturi:

$$P(H_0 = H') = e^{\frac{q(E, B, H') - q(E, B, H)}{T}}$$

Temperatura se počasi niža, dokler ne postane zanemarljivo nizka. Čim počasneje se temperatura niža, tem večji del prostora bo algoritem pregledal in tem večja je verjetnost, da bo našel optimalno stanje. Ponavadi se uporablja geometrično spremicanje temperature:

$$T' := \lambda T, \quad \lambda < 1$$

Na koncu, ko je  $T = 0$  se izvede še deterministična lokalna optimizacija.

## Genetski algoritmi

Genetski algoritmi [4] (genetic algorithms) temeljijo na idejah evolucije in naravne selekcije. V genetskem algoritmu ena hipoteza ustreza enemu *osebku*. Osebek je kodiran kot niz simbolov, ki jim pravimo tudi *geni*. Genetski algoritom začne z naključno izbrano množico osebkov – hipotez. V vsaki iteraciji iz dane populacije stohastično generira naslednjo populacijo (generacijo). Pri tem uporablja transformacije, ki jih srečamo tudi v biološki genetiki:

- reprodukcija: čim boljši je osebek, tem večja je verjetnost, da bo prispeval naslednike v naslednji populaciji,
- križanje (*crossover*): osebek nastane iz dveh osebkov, tako da od vsakega podeduje del genetskega zapisa,
- mutacija: vrednost gena se naključno (z majhno verjetnostjo) spremeni.

Ključni del uporabe genetskega algoritma je kodiranje osebkov. Čim bolj primerno je kodiranje, predvsem glede izbranega načina križanja, tembolje se bo obnašal genetski algoritom. Genetski algoritmi so podrobnejše obravnavani v 12.poglavlju.

## Literatura

- [1] A.V. Aho, J.E. Hopcroft, J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, MA, Menclo Park, CA, London, Amsterdam, 1983.
- [2] I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, 2000. (3rd Edition).
- [3] T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to algorithms*. The MIT Press, 1990.
- [4] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

- [5] I. J. Good. *Probability and the Weighing of Evidence*. Charles Griffin, London, 1950.
- [6] I. J. Good. *The Estimation of Probabilities – An Essay on Modern Bayesian Methods*. The MIT Press, Cambridge, 1964.
- [7] D. Hand, H. Mannila, P. Smyth. *Principles of Data Mining*. MIT Press, 2001.
- [8] S. Haykin. *Neural Networks: A Comprehensive Foundation*. New York: Macmillian College Publ. Comp, 1994.
- [9] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
- [10] I. Kononenko, M. Kukar. *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing, 2007. Chichester, UK.
- [11] N. Lavrač, S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
- [12] D. E. Rumelhart, J. L. McClelland, (ur.). *Parallel Distributed Processing: Foundations*, volume 1. Cambridge: MIT Press, 1986.

## Poglavlje 5

# Mere za ocenjevanje atributov

*Človekove iznajdbe ne nastanejo po naključju.*

*Werner Heisenberg*

Pri iskanju hipoteze je osnovna naloga algoritma oceniti pomembnost atributa za dani učni problem, če imamo opravka z atributnim opisom, oziroma pomembnosti literala, če imamo opravka z relacijskim opisom. V tem poglavju so opisane ocene za ocenjevanje atributov v klasifikacijskih in v regresijskih problemih. Uporaba mer za ocenjevanje pomembnosti literalov v relacijskih problemih je analogna uporabi teh mer za ocenjevanje binarnih atributov. V naslednjih dveh razdelkih so podane lastnosti mer brez dokazov, v zadnjem razdelku pa so za zahtevnejšega bralca zbrane izpeljave in dokazi.

### 5.1 Mere za ocenjevanje atributov v klasifikaciji

*Težki problemi v življenu se vedno začnejo kot preprosti.*

*Lao Ce*

Funkcije, opisane v tem razdelku, so namenjene usmerjanju iskanja v klasifikacijskih problemih. To so:

- informacijski prispevek,
- razmerje informacijskega prispevka,
- razdalja dogodkov,
- mera najkrajšega opisa (MDL),
- J-ocena,

- Gini-indeks ter
- ReliefF.

Ponovimo opis atributne predstavitev, ki je dana z:

- množico atributov  $A = \{A_i, i = 0 \dots a\}$ ;
- za vsak diskretni atribut  $A_i$  imamo množico možnih vrednosti  $\mathcal{V}_i = \{V_1, \dots, V_{m_i}\}$ ;
- za vsak zvezni atribut  $A_i$  imamo interval možnih vrednosti  $\mathcal{V}_i = [Min_i, Max_i]$ ;
- razred je podan z atributom  $A_0$ : če rešujemo klasifikacijski problem, potem je  $A_0$  diskretni atribut, če pa rešujemo regresijski problem, je  $A_0$  zvezni atribut;
- učni primer je vektor vrednosti atributov  $u_j = \langle r^{(j)}, v^{(1,j)}, \dots, v^{(a,j)} \rangle$ , pri tem je razred označen z  $r^{(j)} = v^{(0,j)}$ ;
- množica učnih primerov je podana kot množica vektorjev  $\mathcal{U} = \{u_j, j = 1 \dots n\}$

## Mere nečistoče

Kot osnova za oceno pomembnosti atributa v klasifikacijskem problemu (oziroma literala v relacijskem problemu) se uporablajo mere nečistoče (*impurity measure*). Naj bodo  $r_1, \dots, r_{m_0}$  možni razredi (vrednosti diskretnega atributa  $A_0$ ). Mera nečistoče je funkcija  $\phi$  nad verjetnostno porazdelitvijo  $P(r_k) \geq 0, k = 1 \dots m_0, \sum_k P(r_k) = 1$ , ki ima naslednje lastnosti:

1.  $\phi$  ima edini maksimum v točki, kjer so vsi dogodki enako verjetni:  $P(r_k) = 1/m_0, k = 1 \dots m_0$ .
2.  $\phi$  ima  $m_0$  minimumov v točkah, kjer je eden od dogodkov gotov, ostali pa imajo verjetnost 0, torej je izid že znan:  $P(r_k) = 1, P(r_l) = 0, l = 1 \dots m_0, l \neq k$ .
3.  $\phi$  je simetrična funkcija svojih argumentov (je neobčutljiva na vrstni red argumentov), kar v našem primeru pomeni, da mera ne dela razlik med razredi.
4.  $\phi$  je konkavna:  $\frac{\partial^2 \phi}{\partial P(r_k)^2} < 0, k = 1 \dots m_0$
5.  $\phi$  je zvezna in zvezno odvedljiva funkcija svojih argumentov.

Pomembnost diskretnega atributa  $A_i$  z množico vrednosti  $\mathcal{V}_i = \{V_1, \dots, V_{n_i}\}$  za dani učni problem ocenimo kot pričakovano *zmanjšanje nečistoče*, ko poznamo vrednost atributa:

$$q(A_i) = \phi(P(r_1), \dots, P(r_{m_0})) - \sum_{j=1}^{n_i} P(V_j) \phi(P(r_1|V_j), \dots, P(r_{m_0}|V_j))$$

Pri tem apriorne verjetnosti razredov  $P(r_k)$  in vrednosti atributa  $P(V_j)$  ter pogojne verjetnosti razredov pri dani vrednosti atributa  $P(r_k|V_j)$  ocenimo iz porazdelitev v učni množici.

Lastnosti mere pomembnosti atributa so:

**Nenegativnost:** pomembnost atributa  $q(A_i)$  je vedno nenegativna:

$$q(A_i) \geq 0$$

in je enaka 0 samo, če so apriorna in vse pogojne porazdelitve enake:

$$q(A_i) = 0 \Leftrightarrow \forall j \forall k : P(r_k) = P(r_k | V_j)$$

Glej dokaz na strani 123.

**Maksimum:**  $\max(q(A_i)) = \phi(P(r_1), \dots, P(r_{m_0}))$ . Funkcija doseže maksimum, ko atribut idealno razločuje razrede, t.j., ko velja:

$$\phi(P(r_1 | V_j), \dots, P(r_k | V_j), \dots, P(r_{m_0} | V_j)) = \text{minimum}, \quad j = 1 \dots n_i$$

oziroma velja  $P(r_k | V_j) = 1, P(r_l | V_j) = 0, l = 1 \dots m_0, l \neq k, j = 1 \dots n_i$ .

**Dodajanje vrednosti atributu:** če atribut  $A_i$  z  $n_i$  vrednostmi spremenimo v atribut  $A'_i$  tako, da vrednost  $V_{n_i}$  zamenjamo z dvema možnima vrednostima  $V'_{n_i}$  in  $V'_{n_i+1}$ , potem velja:

$$q(A'_i) \geq q(A_i)$$

Glej dokaz na strani 124.

Lastnost, da se z večanjem števila vrednosti pomembnost atributa kvečjemu poveča, kaže na to, da ocena pomembnosti atributa  $q$  apriori daje prednost atributom z več vrednostmi. To je v praksi nezaželena lastnost, saj je zanesljivost aproksimacije verjetnosti za večvrednostne attribute manjša. Zato je pogosto potrebno v mero pomembnosti vpljati normalizacijo, ki upošteva število vrednosti atributa.

## Mere, ki temeljijo na količini informacije

Največ mer, ki se uporabljajo v strojnem učenju, temelji na količini informacije. Potrebna *količina informacije*, da izvemo, da je izid poskusa dogodek  $X_i$ , je definirana kot negativni dvojiški logaritem verjetnosti dogodka [16] in jo izražamo v bitih:

$$I(X_i) = -\log_2 P(X_i)$$

Povprečni pričakovani količini informacije, da zvemo, kateri izmed nezdružljivih izidov  $X_i, i = 1 \dots m_0, \sum_i P(X_i) = 1$ , se je zgodil, pravimo *entropija* dogodka:

$$H(X) = -\sum_i P(X_i) \log_2 P(X_i)$$

Osnovna mera nečistoče je entropija. Glej dokaz na strani 125.

Ker so vsi logaritmi z osnovo 2, osnove v nadaljevanju ne bomo pisali. Vpeljimo notacijo:  
 $n$  – število učnih primerov,  
 $n_k$  – število učnih primerov iz razreda  $r_k$ ,

$n_{\cdot j}$  – število učnih primerov z  $j$ -to vrednostjo danega atributa  $A$ ,  
 $n_{kj}$  – število učnih primerov iz razreda  $r_k$  in z  $j$ -to vrednostjo danega atributa  $A$ .

Vpeljimo še aproksimacije verjetnosti iz učne množice primerov:

$$p_{kj} = n_{kj}/n,$$

$$p_{k\cdot} = n_{k\cdot}/n,$$

$$p_{\cdot j} = n_{\cdot j}/n,$$

$$p_{k|j} = p_{kj}/p_{\cdot j} = n_{kj}/n_{\cdot j}.$$

Vpeljimo naslednje entropije:

$H_R$  – entropija razredov:

$$H_R = - \sum_k p_{k\cdot} \log p_{k\cdot}$$

$H_A$  – entropija vrednosti danega atributa:

$$H_A = - \sum_j p_{\cdot j} \log p_{\cdot j}$$

$H_{RA}$  – entropija produkta dogodkov razred–vrednost atributa:

$$H_{RA} = - \sum_k \sum_j p_{kj} \log p_{kj}$$

$H_{R|A}$  – pogojna entropija razreda pri dani vrednosti atributa:

$$H_{R|A} = H_{RA} - H_A = - \sum_j p_{\cdot j} \sum_k \frac{p_{kj}}{p_{\cdot j}} \log \frac{p_{kj}}{p_{\cdot j}}$$

$$H_{R|A} = - \sum_j p_{\cdot j} \sum_k p_{k|j} \log p_{k|j}$$

Ker velja  $H_X \geq 0$ , velja  $H_{RA} \geq H_{R|A}$ , prav tako veljata relaciji  $H_{RA} > H_R$  in  $H_{RA} > H_A$ .

### Informacijski prispevek atributa

Klasična mera za pomembnost atributa je informacijski prispevek atributa (*information gain*), ki so ga predlagali že Hunt s sodelavci leta 1966 [3]. Informacijski prispevek atributa je definiran kot prispevek informacije atributa k določitvi vrednosti razreda:

$$Gain(A) = H_R + H_A - H_{RA} = H_R - H_{R|A} \quad (5.1)$$

Ker je entropija mera nečistoče, veljajo za informacijski prispevek lastnosti, ki veljajo za funkcijo pomembnosti atributa:

$$0 \leq Gain(A) \leq H_R$$

Če atributu  $A$  neko vrednost razbijemo na dve vrednosti in tako dobimo atribut  $A'$ , velja:

$$Gain(A') \geq Gain(A)$$

### Razmerje informacijskega prispevka

Lastnost informacijskega prispevka, da kvaliteta atributa s številom vrednosti atributa kvenčjemu raste, je nezaželena. Zato je Quinlan [11] definiral razmerje informacijskega prispevka (*gain-ratio*):

$$GainR(A) = \frac{Gain(A)}{H_A} \quad (5.2)$$

Normalizacija informacijskega prispevka z entropijo vrednosti atributa odpravi problem precenjevanja večvrednostnih atributov, zato pa zdaj ocena precenjuje atribute, ki imajo zelo majhno entropijo vrednosti  $H_A$ . Quinlan predlaga, da se pri izbiri upoštevajo samo atributi  $A_i$ , ki imajo informacijski prispevek večji od povprečnega:

$$Gain(A_i) \geq \frac{\sum_{j=1}^a Gain(A_j)}{a}$$

### Razdalja dogodkov

Mantaras je leta 1989 [10] vpeljal mero  $D$  razdalje dogodkov (dejansko je isto mero definiral že Rajske leta 1961 [12]), ki jo zapišemo v obliki mere bližine dogodkov, ki ocenjuje pomembnost atributa:

$$1 - D(R, A) = \frac{Gain(A)}{H_{RA}} \quad (5.3)$$

Zato, da je mera  $D$  razdalja, mora izpolnjevati naslednje pogoje:

1.  $D(R, A) \geq 0$ ; ta pogoj je izpolnjen, saj velja:

$$D(R, A) = 1 - \frac{Gain(A)}{H_{RA}} = \left(1 - \frac{H_R}{H_{RA}}\right) + \left(1 - \frac{H_A}{H_{RA}}\right)$$

Nenegativnost velja, ker je  $H_{RA} \geq H_A$  in  $H_{RA} \geq H_R$ .

2.  $D(R, A) = 0 \Leftrightarrow R = A$ .  $D(R, A) = 0$ , če velja:

$$H_R + H_A = 2H_{RA}$$

Lastnost velja, ker je  $H_{RA} \geq H_R$  in  $H_{RA} \geq H_A$  in velja  $H_{RA} = H_R \Rightarrow H_{A|R} = 0$  in  $H_{RA} = H_A \Rightarrow H_{R|A} = 0$

3.  $D(R, A) = D(A, R)$  je očitno res.

4. Trikotniška neenakost:  $D(R, A_1) + D(A_1, A_2) \geq D(R, A_2)$ . Dokaz te lastnosti je na strani 125.

Funkcija  $1 - D$  zadovoljivo reši problem večvrednostnih atributov in nima težav z majhnimi vrednostmi imenovalca ( $H_{RA}$ ), kot jih ima  $GainR$ .

## MDL

Po principu najkrajšega opisa [13, 9] lahko definiramo, da je atribut tem pomembnejši, čim bolj je kompresiven. Imejmo naslednji problem prenosa podatkov po komunikacijskem kanalu [7]. Oba, pošiljatelj in sprejemnik, poznata vse možne vrednosti  $V_j$  danega atributa  $A$ , število možnih razredov  $m_0$  in za vsak (učni) primer  $u_i$  poznata vrednosti atributa:  $v^{(i)}$ . Samo pošiljatelj pozna pravilne razrede za vse primere. Naloga pošiljatelja je poslati informacijo o razredih za vse primere, tako da je sporočilo čim krajše, t.j. kodirano s čim manjšim številom bitov.

Pošiljatelj eksplisitno zakodira razred za vsak primer ali pa uporabi atribut, tako da za vsako vrednost atributa zakodira razrede primerov, ki imajo to vrednost atributa. V prvem primeru mora pošiljatelj poleg kodiranih razredov poslati tudi porazdelitev primerov po razredih, ki omogoča sprejemniku dekodiranje opisov razredov. V drugem primeru obstaja za vsako vrednost atributa drugačna porazdelitev po razredih. To pomeni, da je potrebno za vsako vrednost atributa poslati poleg opisov razredov primerov z dano vrednostjo atributa tudi opis porazdelitve.

Število bitov, ki so potrebni za zakodiranje razredov primerov z dano verjetnostno porazdelitvijo, lahko aproksimiramo z entropijo  $H_R$ , pomnoženo s številom primerov  $n$ , čemur prištejemo število bitov, potrebnih za zakodiranje porazdelitve po razredih. Ker imamo  $n$  primerov in  $m_0$  razredov, je število različnih porazdelitev enako:

$$\binom{n+m_0-1}{m_0-1}$$

Interpretacija izraza je, da če razdelimo interval  $1 \dots n$  na  $m_0$  podintervalov, od katerih so lahko nekateri prazni, je to enako, kot če razbijemo interval  $1 \dots n+m_0$  na  $m_0$  nepraznih podintervalov; vseh možnih različnih mej je  $n+m_0-1$ , izbrati pa moramo  $m_0-1$  mej.

Na ta način dobimo oceno števila bitov, ki jih potrebujemo za kodiranje razredov vseh  $n$  primerov:

$$Prior\_MDL' = nH_R + \log \binom{n+m_0-1}{m_0-1}$$

Ocena števila bitov za kodiranje razredov po posameznih vrednostih atributa je vsota takih členov po vseh vrednostih atributa  $A$ :

$$Post\_MDL'(A) = \sum_j n_{.j} H_{R|j} + \sum_j \log \binom{n_{.j} + m_0 - 1}{m_0 - 1}$$

$$Post\_MDL'(A) = nH_{R|A} + \sum_j \log \binom{n_{.j} + m_0 - 1}{m_0 - 1}$$

Pomembnost atributa  $MDL'(A)$  definiramo kot kompresivnost atributa, t.j. kot razliko dolžin kode brez in z uporabo atributa, normalizirano s številom učnih primerov:

$$MDL'(A) = \frac{Prior\_MDL' - Post\_MDL'(A)}{n} =$$

$$Gain(A) + \frac{1}{n} \left( \log \binom{n+m_0-1}{m_0-1} - \sum_j \log \binom{n_j+m_0-1}{m_0-1} \right) \quad (5.4)$$

Pri izpeljavi smo uporabili  $H_R$ , ki definira optimalno kodiranje samo, če je število zakodiranih besed poljubno. Če je število besed (učnih primerov) znano, lahko uporabimo optimalnejše kodiranje. Potrebno je izračunati število vseh možnih klasifikacij  $n$  učnih primerov. Število le-teh je enako:

$$\binom{n}{n_1, \dots, n_{m_0}}$$

Z optimalnim kodiranjem dobimo:

$$Prior\_MDL = \log \binom{n}{n_1, \dots, n_{m_0}} + \log \binom{n+m_0-1}{m_0-1}$$

in

$$Post\_MDL(A) = \sum_j \log \binom{n_j}{n_{1j}, \dots, n_{mj}} + \sum_j \log \binom{n_j+m_0-1}{m_0-1}$$

Na ta način dobimo:

$$\begin{aligned} MDL(A) &= \frac{Prior\_MDL - Post\_MDL(A)}{n} \\ &= \frac{1}{n} \left( \log \binom{n}{n_1, \dots, n_{m_0}} - \sum_j \log \binom{n_j}{n_{1j}, \dots, n_{mj}} + \right. \\ &\quad \left. + \log \binom{n+m_0-1}{m_0-1} - \sum_j \log \binom{n_j+m_0-1}{m_0-1} \right) \end{aligned} \quad (5.5)$$

Izkazalo se je, da je ocena  $MDL$  najprimernejša glede pristranskosti pri ocenjevanju večvrednostnih atributov [7]. Če je  $MDL < 0$ , potem je atribut nekompresiven in zatorej neuporaben.

### J-ocena

Do sedaj smo definirali mere za ocenjevanje pomembnosti atributa kot celote. Pogosto pa želimo oceniti pomembnost ene vrednosti atributa. V ta namen bi lahko spremenili definicijo mere pomembnosti:

$$q_1(V_j) = \phi(P(r_1), \dots, P(r_{m_0})) - \phi(P(r_1|V_j), \dots, P(r_{m_0}|V_j))$$

Vendar ima tako definirana mera več slabosti:

- lahko je negativna in
- ne razlikuje med enakimi in permutiranimi porazdelitvami, npr.:

$$P(r_1|V_j) = P(r_1), \dots, P(r_{m_0}|V_j) = P(r_{m_0}) \Rightarrow q_1(V_j) = 0$$

in

$$P(r_1|V_j) = P(r_{m_0}), \dots, P(r_{m_0}|V_j) = P(r_1) \Rightarrow q_1(V_j) = 0$$

Smyth in Goodman [17] sta definirala  $J$ -oceno za ocenjevanje ene vrednosti atributa oziroma za ocenjevanje pomembnosti odločitvenega pravila kot celote. Mera oceni informacijsko vsebino pogoja:

$$J_j = p_{.j} \sum_k p_{k|j} \log \frac{p_{k|j}}{p_k} \quad (5.6)$$

Mero lahko interpretiramo kot pričakovano količino informacije, ki jo dobimo od vrednosti  $V_j$  danega atributa za klasifikacijo. Mera ima lepe lastnosti:

**Nenegativnost**  $J_j \geq 0$ . Glej dokaz na strani 126.

**Zgornja meja  $J$ -ocene** Ker je odvod funkcije  $J_j$  enak 0 tudi v primeru, ko je za nek  $k_0$ :  $p_{k_0|j} = 1$  in  $\forall k \neq k_0$ :  $p_{k|j} = 0$ , ima funkcija v tej točki ekstrem. Zato lahko zgornjo mejo ocenimo z [17]:

$$J_j \leq p_{.j} \left( \min \left\{ \max_k -\log p_{k,.}, -\log p_{.j} \right\} \right) \leq -p_{.j} \log p_{.j} \leq 0.53 \text{ bit}$$

Pri tem smo uporabili tudi enakost:

$$\frac{P(X|Y)}{P(X)} = \frac{P(Y|X)}{P(Y)}$$

**Relacija z informacijskim prispevkom** Velja naslednja enakost:

$$\sum_j J_j = Gain(A)$$

kar lahko preprosto izpeljemo:

$$\sum_j J_j = \sum_j \sum_k p_{kj} \log \frac{p_{kj}}{p_k p_{.j}} = H_A + H_R - H_{RA}$$

Poleg zgoraj naštetih lastnosti velja, da je statistika

$$2nJ_j \log_e 2 = 1.3863 nJ_j$$

kjer je  $n$  število učnih primerov, porazdeljena približno po zakonu  $\chi^2$  [2]. Torej jo lahko uporabljam za ocenjevanje verjetnosti, da porazdelitev  $p_{kj}$  ni dobljena naključno. Test uporabimo pri ocenjevanju zanesljivosti ene vrednosti atributa ali pa celega odločitvenega pravila. V nadaljevanju so opisani testi za ocenjevanje zanesljivosti celega atributa.

### Gini-indeks

Breiman in sod. [1] uporabljajo Gini-indeks za izbiro atributov v odločitvenem drevesu. Apriori Gini-indeks je mera nečistoče, definirana z (glej dokaz na strani 127):

$$Gini\_prior = \sum_k \sum_{l \neq k} p_k p_l = 1 - \sum_k p_k^2$$

$A_1$	$A_2$	$A_3$	R
0	0	0	0
0	0	1	0
0	1	1	1
0	1	0	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	0	0

Tabela 5.1: Primer učnega problema.

Pomembnost atributa  $Gini(A)$  je definirana kot razlika med apriornim in pričakovanim aposteriornim Gini-indeksom:

$$Gini(A) = \sum_j p_{\cdot j} \sum_k p_{k|j}^2 - \sum_k p_k^2. \quad (5.7)$$

$Gini(A)$  ima lastnosti, ki izvirajo iz definicije mere nečistoče:

**Nenegativnost:**  $Gini(A) \geq 0$

**Maksimalna vrednost:**

$$Gini(A) = Gini\_prior \Leftrightarrow \forall j : \exists ! k : p_{k|j} = 1$$

**Večanje števila vrednosti atributa:**  $Gini(A)$  kvečjemu naraste. Ta lastnost je nezaželena, saj  $Gini(A)$  precenjuje večvrednostne atribute.

## Relieff

Do sedaj opisane mere za ocenjevanje kvalitete atributa predpostavljajo apriorno in pogojno neodvisnost atributov pri danem razredu. Zaradi te predpostavke so vse mere "kratkovidne". Imejmo naslednji preprost primer: trije binarni atributi, dva možna razreda ter osem učnih primerov, ki so podani v tabeli 5.1.

Pravilna teorija je  $R = (A_1 \neq A_2)$ , torej sta pomembna le  $A_1$  in  $A_2$ . Vendar bi vse do sedaj obravnavane mere ocenile kot najbolj pomemben atribut  $A_3$ . Razlog je v tem, da sta atributa  $A_1$  in  $A_2$  pogojno močno odvisna med seboj, in če opazujemo vsakega posebej, sta navidezno popolnoma nepomembna za dani klasifikacijski problem. Tako bi dobili  $Gain(A_1) = GainR(A_1) = 1 - D(A_1, R) = Gini(A_1) = 0$ , medtem ko bi bile vse mere pri oceni atributa  $A_3$  pozitivne. Neformalno, kot vidimo iz tabele primerov, sta pri obeh vrednostih  $A_2$  oba razreda enako verjetna, torej nam poznavanje (samo) vrednosti  $A_2$  ne pove ničesar o razredu. Enako velja za  $A_3$ , medtem ko oba skupaj že popolnoma določita razred.

Kira in Rendell [5, 4] sta razvila algoritem RELIEF za ocenjevanje atributov v dvorazrednih klasifikacijskih problemih, ki učinkovito reši problem odvisnosti atributov. Osnovna

ideja algoritma je, da za vsak učni primer poišče najbližji primer iz istega razreda (najbližji zadelek) in najbližji primer iz nasprotnega razreda (najbližji pogrešek). Na ta način oceni kvaliteto atributa glede na lokalne značilnosti razločevanja razredov. Ravno lokalnost pa vključuje v oceno tudi ostale attribute. Tako RELIEF implicitno ocenjuje attribute v odvisnosti od ostalih atributov. Osnovni algoritem je sledeč:

#### Algoritem RELIEF

*Vhod:* za vsak učni primer vektor vrednosti atributov in razreda, parameter  $m$

*Izhod:* vektor ocen kakovosti atributov  $W$

```

postavi vse uteži  $W[A] = 0.0$ ;
for i = 1 to m do
    naključno izberi učni primer  $R_i$ ;
    poišči najbližji zadelek  $H$  in najbližji pogrešek  $M$ ;
    for  $A = 1$  to  $a$  do
         $W[A] = W[A] - diff(A, R_i, H)/m + diff(A, R_i, M)/m;$ 
    end for;
end for;
end RELIEF

```

Pri tem funkcija  $diff(A_i, primer1, primer2)$  za dani atribut  $A_i$  vrne razliko vrednosti tega atributa pri dveh primerih:

$$diff(A_i, u_j, u_k) = \begin{cases} \frac{|v^{(i,j)} - v^{(i,k)}|}{Max_i - Min_i}, & A_i \text{ je zvezni} \\ 0, & v^{(i,j)} = v^{(i,k)} \wedge A_i \text{ je diskreten} \\ 1, & v^{(i,j)} \neq v^{(i,k)} \wedge A_i \text{ je diskreten} \end{cases}$$

RELIEF oceni naslednjo razliko verjetnosti [6]:

$$Relief(A_i) =$$

$$\begin{aligned} P(diff(A_i, \cdot, \cdot) = 1 | \text{najbližji primer je iz nasprotnega razreda}) - \\ - P(diff(A_i, \cdot, \cdot) = 1 | \text{najbližji primer je iz istega razreda}) \\ = P(diff(A_i, \cdot, \cdot) = 0 | \text{najbližji primer je iz istega razreda}) - \\ - P(diff(A_i, \cdot, \cdot) = 0 | \text{najbližji primer je iz nasprotnega razreda}) \end{aligned}$$

Če izpustimo pogoj bližine, dobimo funkcijo, ki je zelo podobna Gini-indeksu (glej izjavlo na str. 127):

$$\begin{aligned} Relief(A_i) &= \frac{\sum_j p_{\cdot j}^2 \times Gini'(A_i)}{\sum_k p_{k \cdot}^2 (1 - \sum_k p_{k \cdot}^2)} \\ &= konstanta \times \sum_j p_{\cdot j}^2 \times Gini'(A_i) \end{aligned} \quad (5.8)$$

kjer je  $Gini'(A)$  zelo podoben razlik med apriornim in aposteriornim Gini-indeksom:

$$Gini'(A) = \sum_j \left( \frac{p_{\cdot j}^2}{\sum_j p_{\cdot j}^2} \times \sum_k p_{k|j}^2 \right) - \sum_k p_k^2. \quad (5.9)$$

Edina razlika je, da namesto faktorja

$$\frac{p_{\cdot j}^2}{\sum_j p_{\cdot j}^2}$$

v Gini-indeksu nastopa faktor:

$$\frac{p_{\cdot j}}{\sum_j p_{\cdot j}} = p_{\cdot j}$$

Bistvena razlika med Reliefom in Gini-indeksom pa je v faktorju, ki nastopa pred  $Gini'$  v imenovalcu:

$$\sum_j p_{\cdot j}^2$$

Faktor predstavlja apriorno verjetnost, da imata dva primera isto vrednost za dani atribut. Faktor implicitno normalizira Reliefove ocene, tako da ocena  $Relief(A_i)$  ne precenjuje več vrednostnih atributov, kot je empirično pokazano [7].

Osnovni RELIEF zanesljivo ocenjuje zvezne in diskretne atrinute, ki so močno odvisni med seboj. Npr. v množici atributov lahko RELIEF pravilno oceni x atributov, ki definirajo za učenje zelo težavno funkcijo parnosti reda x. Časovna zahtevnost algoritma je  $O(m \times n \times a)$ , kjer je a število atributov, n število učnih primerov ter m število iteracij v algoritmu RELIEF. Za izboljšanje zanesljivosti ocen RELIEF-a lahko zanka preteče vseh n učnih primerov. Ker je to pogosto časovno preveč zahtevno, ponavadi izberemo ustrezno majhno število iteracij  $m \ll n$ , v katerih primere izbiramo naključno (brez vračanja). Tipična vrednost parametra m je med 30 in 200.

Da bi izpopolnili osnovni algoritem, smo razvili inačico ReliefF [6]. ReliefF vsebuje naslednje razširitve:

**Neznane vrednosti atributov** Pri ocenjevanju atributov lahko uporablja tudi nepopolne podatke. V ta namen je posložena funkcija  $diff$  tako, da izračuna verjetnost, da imata dva primera različno vrednost danega atributa. Če sta obe vrednosti podani, potem je za diskretne atrinute vrednost vedno 0 ali 1. Pri manjkajočih vrednostih ločimo naslednjia primera:

- prvi primer ( $u_j$ ) nima vrednosti atributa:

$$diff(A_i, u_j, u_k) = 1 - p_{v(i,k)}|_{r(j)}$$

- oba primera nimata vrednosti atributa:

$$diff(A_i, u_j, u_k) = 1 - \sum_{l=1}^{n_l} (p_{V_l|r(j)} \times p_{V_l|r(k)})$$

Pri neznanih vrednostih zveznih atributov si pomagamo z diskretizacijo.

**Šumni podatki** Najpomebniji del algoritma RELIEF išče najbližji zadetek in najbližji pogrešek danega primera. Šum v razredu in/ali vrednostih atributa bistveno vpliva na izbor najbližjih zadetkov/pogreškov. Da bi povečali zanesljivost iskanja najbližjih pogreškov/zadetkov, razširjeni algoritem ReliefF za vsak učni primer poišče  $k$  najbližjih zadetkov in  $k$  najbližjih pogreškov ter upošteva povprečje njihovih prispevkov. Izkaže se, da ta preprosta razširitev bistveno izboljša zanesljivost ReliefF-ovih ocen. Tipična vrednost parametra  $k = 5 \dots 10$ .

**Večrazredni problemi** Poslošitev na ocenjevanje atributov pri klasifikacijskih problemih z več kot dvema razredoma zahteva, da namesto  $k$  najbližjih pogreškov iz nasprotnega razreda ReliefF poišče po  $k$  najbližjih pogreškov iz vsakega razreda. Prispevki posameznih razredov so uteženi z apriorimi verjetnostmi razredov (v spodnjem algoritmu je faktor utežitve  $p_{razr.}/(1 - p_r)$ ; razr je razred najbližjega pogreška,  $r$  pa razred izbranega primera; utežitveni faktor predstavlja prispevke razreda najbližjega pogreška relativno glede na vse razrede, ki niso enaki  $r$ ).

Sedaj lahko zapišemo celotni algoritem ReliefF:

#### Algoritem ReliefF

*Vhod:* za vsak učni primer vektor vrednosti atributov in razreda, parametra  $m, k$

*Izhod:* vektor ocen kakovosti atributov  $W$

```

postavi vse uteži  $W[A] = 0.0$ ;
for i = 1 to m do
    naključno izberi učni primer  $R_i$ ;
    poišči  $k$  najbližjih zadetkov  $H_j$ ;
    for vsak razred  $C \neq razred(R_i)$  do
        iz razreda  $C$  poišči  $k$  najbližjih pogreškov  $M_j(C)$ ;
    end for;
    for A = 1 to a do
         $W[A] = W[A] - \sum_{j=1}^k diff(A, R_i, H_j) / (m \cdot k) +$ 
         $\sum_{C \neq razred(R_i)} \left[ \frac{P(C)}{1 - P(razred(R_i))} \sum_{j=1}^k diff(A, R_i, M_j(C)) \right] / (m \cdot k);$ 
    end for;
end for;
end ReliefF
```

## 5.2 Mere za ocenjevanje atributov v regresiji

*Narava ne dovoli modrim, da bi imeli želje ali ambicije, niti ne dovoli neumnim, da jih izpolnijo ali da se jih znebijo.*

*Sri Sri Ravi Shankar*

V regresijskih problemih se za ocenjevanje kvalitete atributov in kvalitete hipoteze uporabljajo pričakovana razlika variance, regresijski ReliefF in princip najkrajšega opisa (MDL).

### Razlika variance pri regresijskih problemih

Pri regresijskih problemih namesto mere nečistoče uporabljam varianco odvisne spremenljivke, ki je definirana kot povprečni kvadrat napake:

$$s^2 = \frac{1}{n} \sum_{k=1}^n (r^{(k)} - \bar{r})^2$$

pri tem je  $\bar{r}$  povprečna vrednost zveznega razreda med  $n$  učnimi primeri:

$$\bar{r} = \frac{1}{n} \sum_{k=1}^n r^{(k)}$$

Relacijo z mero nečistoče podaja naslednja enakost. Če v dvorazrednem klasifikacijskem problemu enemu razredu priredimo številsko vrednost 0, drugemu razredu pa vrednost 1, velja:

$$Gini\_prior = 2s^2$$

Glej dokaz na strani 128.

Za ocenjevanje pomembnosti atributa  $A_i$  se uporablja (nenegativna) pričakovana razlika variance:

$$ds^2(A_i) = \frac{1}{n} \sum_{k=1}^n (r^{(k)} - \bar{r})^2 - \sum_{j=1}^{n_i} \left( p_{.j} \frac{1}{n_{.j}} \sum_{k=1}^{n_{.j}} (r_j^{(k)} - \bar{r}_j)^2 \right) \quad (5.10)$$

Pri tem je  $r_j^{(k)}$  vrednost odvisne spremenljivke  $k$ -tega primera, ki ima  $j$ -to vrednost atributa  $A_i$  ter  $\bar{r}_j$  povprečna vrednost odvisne spremenljivke primerov z  $j$ -to vrednostjo atributa  $A_i$ .

$$\bar{r}_j = \frac{1}{n_{.j}} \sum_{k=1}^{n_{.j}} r_j^{(k)}$$

### Regresijski ReliefF

V regresijskih problemih je napovedana vrednost zvezna, zato ne moremo uporabiti najbližjih pogreškov (primerov iz nasprotnega razreda) in zadetkov (primerov iz istega razreda) kot pri

navadnem algoritmu ReliefF. Regresijski ReliefF ali RReliefF [14, 8, 15] zato uporablja neke vrste "verjetnost, da dva primera pripadata različnima razredoma". To verjetnost modeliramo z relativno razdaljo med vrednostima odvisne spremeljivke obeh primerov.

Za oceno kvalitete atributa v enačbi (5.12) potrebujemo verjetnost, da primera pripadata istemu razredu, če imata isto vrednost atributa, in apriorno verjetnost, da primera pripadata istemu razredu. Enačbo lahko preoblikujemo tako, da potrebujemo verjetnost, da imata primera različen razred, če imata različne vrednosti atributa:

$$Relief(A_i) = \frac{P_{diff|cl|diff} P_{diff}}{P_{diffcl}} - \frac{(1 - P_{diff|cl|diff}) P_{diff}}{1 - P_{diffcl}} \quad (5.11)$$

Pri tem je  $P_{diff}$  apriorna verjetnost, da imata primera različno vrednost atributa, in  $P_{diffcl}$  apriorna verjetnost, da imata primera različen razred. Algoritem RReliefF mora torej oceniti verjetnosti iz enačbe (5.11). V psevdokodi algoritma je odvisna spremeljivka označena z *vrednost*:

#### Algoritem RReliefF

*Vhod:* za vse primere vektor vrednosti atributov in napovedane vrednosti, parametra  $m$  in  $k$   
*Izhod:* vektor ocen kakovosti atributov  $W$

```

postavi vse  $N_{dC}$ ,  $N_{dA}[A]$ ,  $N_{dC\&dA}[A]$ ,  $W[A]$  na 0;
for i = 1 to m do
    naključno izberi primer  $R_i$ ;
    izberi k primerov  $I_j$ , ki so najbližji  $R_i$ ;
    for j = 1 to k do
         $N_{dC} = N_{dC} + diff(vrednost, R_i, I_j)/k$ ;
        for A = 1 to a do
             $N_{dA}[A] = N_{dA}[A] + diff(A, R_i, I_j)/k$ ;
             $N_{dC\&dA}[A] = N_{dC\&dA}[A] + diff(vrednost, R_i, I_j) \cdot diff(A, R_i, I_j)/k$ ;
        end for;
    end for;
end for;
for A = 1 to a do
     $W[A] = N_{dC\&dA}[A]/N_{dC} - (N_{dA}[A] - N_{dC\&dA}[A])/(m - N_{dC})$ ;
end for;
end RReliefF
```

Časovna zahtevnost algoritma RReliefF je enaka časovni zahtevnosti originalnega algoritma RELIEF in je  $O(m \times n \times a)$ . Najzahtevnejša operacija v glavni zanki je izbira  $k$  najbližjih sosedov. Zanjo je potrebno izračunati  $n$  razdalj, kar lahko storimo v  $O(n \times a)$  korakih. To je časovno najbolj zahtevno, saj lahko kopico zgradimo v  $O(n)$  korakih,  $k$  najbližjih sosedov pa zatem iz nje izločimo v  $O(k \log n)$  korakih, kar je praktično vedno manj kot  $O(n \times a)$ .

Ker tako ReliefF kot RReliefF izračunavata kvaliteto atributov po enačbi (5.11), nam ta enačba daje enoten pogled na problem ocenjevanja atributov v klasifikaciji in regresiji.

## MDL v regresiji

Za uporabo principa najkrajšega opisa (MDL) v regresiji, moramo izbrati ustrezeno kodiranje realnih števil, ki ga uporabimo za kodiranje regresijskih vrednosti (ali vrednosti zveznih atributov) in napake (odstopanja od prave vrednosti regresijske spremenljivke). Izberi kodiranja lahko močno vpliva na obnašanje ocene MDL. Ker lahko kodiranje realnega števila zahteva neskončno kodo, se omemjimo na fiksno natančnost (število decimalk) in zatem kodiramo realno število kot naravno število.

Pri kodiranju regresijske spremenljivke in zveznih atributov imajo vse vrednosti enako dolžino kode, ki je podana z

$$\log_2 \frac{\text{Interval}}{\text{natančnost}}$$

Vidimo, da izberi natančnosti vpliva na dolžino kode.

Pri kodiranju napake je zaželeno, da imajo manjša števila, ki predstavljajo manjšo napako, krajšo kodo. V tem primeru naravna števila kodiramo z Rissanenovo kodo [13]:

$$\begin{aligned} Rissanen(0) &= 1 \\ Rissanen(n) &= 1 + \log_2 n + \log_2(\log_2 n) + \dots + \log_2(2.865064\dots) \end{aligned}$$

kjer vsota vsebuje le pozitivne člene. Ideja tega kodiranja je, da za kodiranje števila  $n$  potrebujemo  $\log_2 n$  bitov, za kodiranje stavka, ki predstavlja dolžino kode, potrebujemo  $\log_2(\log_2 n)$  bitov itd.

Ko izberemo ustrezeno kodiranje, ocenujemo kvaliteto atributa kot pri klasifikaciji. Namesto porazdelitve po razredih kodiramo povprečno vrednost odvisne spremenljivke (na prvi zgoraj opisan način) in odstopanje od nje za vsak učni primer (na drugi zgoraj opisan način).

## 5.3 Izpeljave in dokazi

**Nenegativnost pomembnosti atributa** To lastnost pokažemo z odvajanjem. Pri tem upoštevamo, da velja

$$P(r_k) = \sum_j P(V_j)P(r_k|V_j)$$

in

$$\frac{\partial P(r_k)}{\partial P(r_k|V_j)} = P(V_j)$$

Prvi odvod funkcije  $q$  kaže na to, da pomembnost atributa  $q$  doseže ekstrem v točki  $P(r_k) = P(r_k|V_j)$ , saj je takrat odvod enak 0:

$$\begin{aligned} \frac{\partial q}{\partial P(r_k|V_j)} &= \frac{\partial \phi(P(r_1), \dots, P(r_k), \dots, P(r_{m_0}))}{\partial P(r_k)} P(V_j) - \\ &- P(V_j) \frac{\partial \phi(P(r_1|V_j), \dots, P(r_k|V_j), \dots, P(r_{m_0}|V_j))}{\partial P(r_k|V_j)} \end{aligned}$$

Ker je funkcija  $\phi$  konkavna velja

$$\frac{\partial^2 \phi}{\partial P(r_k)^2} < 0, \quad k = 1 \dots m_0$$

Zato je drugi odvod funkcije  $q$  v točki  $P(r_k) = P(r_k|V_j)$  pozitiven, kar pomeni da je to minimalna vrednost funkcije  $q$ , in je s tem lastnost dokazana:

$$\begin{aligned} \frac{\partial^2 q}{\partial P(r_k|V_j)^2} &= \frac{\partial^2 \phi(P(r_1), \dots, P(r_k), \dots, P(r_{m_0}))}{\partial P(r_k)^2} P(V_j)^2 - \\ &- P(V_j) \frac{\partial^2 \phi(P(r_1|V_j), \dots, P(r_k|V_j), \dots, P(r_{m_0}|V_j))}{\partial P(r_k|V_j)^2} \\ \frac{\partial^2 q}{\partial P(r_k|V_j)^2} &= -C(P(V_j)^2 - P(V_j)) \geq 0 \end{aligned}$$

□

**Dodajanje vrednosti atributu poveča njegovo pomembnost** Zopet si pomagamo z odvajanjem funkcije  $q(A'_i) - q(A_i)$ . Odvod je enak 0 v primeru, ko je  $P(r_k|V'_{n_i}) = P(r_k|V'_{n_i+1}) = P(r_k|V_{n_i})$ ,  $k = 1 \dots m_0$ . Ker velja:

$$P(r_k|V_{n_i}) = \frac{P(r_k|V'_{n_i})P(V'_{n_i}) + P(r_k|V'_{n_i+1})P(V'_{n_i+1})}{P(V_{n_i})}$$

velja zato:

$$\frac{\partial P(r_k|V_{n_i})}{\partial P(r_k|V'_{n_i})} = \frac{P(V'_{n_i})}{P(V_{n_i})}$$

Torej je odvod funkcije  $q(A'_i) - q(A_i)$  enak:

$$\begin{aligned} \frac{\partial(q(A'_i) - q(A_i))}{\partial P(r_k|V'_{n_i})} &= \\ P(V_{n_i}) \frac{\partial \phi(P(r_1|V_{n_i}), \dots, P(r_k|V_{n_i}), \dots, P(r_{m_0}|V_{n_i}))}{\partial P(r_k|V_{n_i})} \frac{P(V'_{n_i})}{P(V_{n_i})} - \\ &- \frac{\partial \phi(P(r_1|V'_{n_i}), \dots, P(r_k|V'_{n_i}), \dots, P(r_{m_0}|V'_{n_i}))}{\partial P(r_k|V'_{n_i})} P(V'_{n_i}) \end{aligned}$$

Drugi odvod funkcije je pozitiven, kar pomeni, da v točki  $P(r_k|V'_{n_i}) = P(r_k|V'_{n_i+1}) = P(r_k|V_{n_i})$ ,  $k = 1 \dots m_0$ , doseže funkcija minimum:

$$\frac{\partial^2(q(A'_i) - q(A_i))}{\partial P(r_k|V'_{n_i})^2} = CP(V'_{n_i}) \left( 1 - \frac{P(V'_{n_i})}{P(V_{n_i})} \right) \geq 0$$

□

**Entropija je mera nečistoče**

1. Maksimum:

$$H(X) = - \sum_{i=1}^{m_0-1} P(X_i) \log_2 P(X_i) - \left( 1 - \sum_{i=1}^{m_0-1} P(X_i) \right) \log_2 \left( 1 - \sum_{i=1}^{m_0-1} P(X_i) \right)$$

$$\frac{\partial H}{\partial P(X_k)} = -\log_2 P(X_k) - \log_2 e + \log_2 \left( 1 - \sum_{i=1}^{m_0-1} P(X_i) \right) + \log_2 e$$

Prvi odvod je 0 v točki:  $P(X_k) = 1/m_0$ 

Drugi odvod je v tej točki negativen (povsod je negativen):

$$\frac{\partial^2 H}{\partial P(X_k)^2} = -\log_2 e \left( \frac{1}{P(X_k)} + \frac{1}{(1 - \sum_{i=1}^{m_0-1} P(X_i))} \right) < 0$$

2. Minimum:

$$H(X) \geq 0 \wedge H(X) = 0 \Leftrightarrow (P(X_i) = 0 \vee P(X_i) = 1)$$

3.  $H$  je simetrična.4.  $H$  je konkavna.5.  $H$  je zvezna in zvezno odvedljiva funkcija.

□

**Trikotniška neenakost za mero razdalje** Izrazimo funkcijo  $D$  drugače:

$$D(R, A_1) = \frac{H_{A_1|R} + H_{R|A_1}}{H_{A_1|R}}$$

Najprej dokažimo naslednje [10]:

$$H_{R|A_1} + H_{A_1|A_2} \geq H_{R|A_2}$$

Ker velja  $H_{R|A_1} \geq H_{R|A_1 A_2}$ , lahko zapišemo:

$$H_{R|A_1} + H_{A_1|A_2} \geq H_{R|A_1 A_2} + H_{A_1|A_2} = H_{R A_1|A_2} \geq H_{R|A_2}$$

Sedaj dokažimo še naslednjo relacijo:

$$\frac{H_{R|A_1}}{H_{R A_1}} + \frac{H_{A_1|A_2}}{H_{A_1 A_2}} \geq \frac{H_{R|A_2}}{H_{R A_2}}$$

Iz  $H_{R A} = H_{R|A} + H_A$  dobimo:

$$\frac{H_{R|A_1}}{H_{R A_1}} + \frac{H_{A_1|A_2}}{H_{A_1 A_2}} = \frac{H_{R|A_1}}{H_{R|A_1} + H_{A_1}} + \frac{H_{A_1|A_2}}{H_{A_1|A_2} + H_{A_2}} \geq$$

$$\begin{aligned} &\geq \frac{H_{R|A_1}}{H_{R|A_1} + H_{A_1|A_2} + H_{A_2}} + \frac{H_{A_1|A_2}}{H_{R|A_1} + H_{A_1|A_2} + H_{A_2}} = \\ &= \frac{H_{R|A_1} + H_{A_1|A_2}}{H_{R|A_1} + H_{A_1|A_2} + H_{A_2}} \end{aligned}$$

Sedaj iz relacije

$$H_{R|A_1} + H_{A_1|A_2} \geq H_{R|A_2}$$

sledi dokaz lastnosti

$$\frac{H_{R|A_1} + H_{A_1|A_2}}{H_{R|A_1} + H_{A_1|A_2} + H_{A_2}} \geq \frac{H_{R|A_2}}{H_{R|A_2} + H_{A_2}} = \frac{H_{R|A_2}}{H_{RA_2}}$$

Isto lastnost lahko zapišemo tudi kot

$$\frac{H_{A_2|A_1}}{H_{A_2A_1}} + \frac{H_{A_1|R}}{H_{A_1R}} \geq \frac{H_{A_2|R}}{H_{RA_2}}$$

Če seštejemo obe relaciji s permutiranimi argumenti, dobimo zahtevano lastnost:

$$\frac{H_{A_2|A_1} + H_{A_1|A_2}}{H_{A_2A_1}} + \frac{H_{A_1|R} + H_{R|A_1}}{H_{A_1R}} \geq \frac{H_{A_2|R} + H_{R|A_2}}{H_{RA_2}}$$

□

**Nenegativnost J-ocene** To lastnost lahko pokažemo z odvajanjem. Zapišimo  $J$ -oceno nekoli drugače:

$$J_j = p_{xj} \left( \sum_{k=1}^{m_0-1} p_{kj} \log \frac{p_{kj}}{p_{k.}} + \left( 1 - \sum_{k=1}^{m_0-1} p_{kj} \right) \log \frac{1 - \sum_{k=1}^{m_0-1} p_{kj}}{1 - \sum_{k=1}^{m_0-1} p_{k.}} \right)$$

Sedaj odvajamo:

$$\frac{\partial J_j}{\partial p_{xj}} = \log \frac{p_{xj}}{p_x} + \log e - \log \frac{1 - \sum_{k=1}^{m_0-1} p_{kj}}{1 - \sum_{k=1}^{m_0-1} p_{k.}} - \log e$$

Odvod je enak 0, ko velja:

$$\forall k : p_{kj} = p_k.$$

V tej točki velja tudi  $J_j = 0$ . Za dokaz nenegativnosti zadošča, da pokažemo, da je v tej točki minimum funkcije  $J_j$ , to pomeni, da je drugi odvod pozitiven:

$$\frac{\partial^2 J_j}{\partial p_{xj}^2} = \frac{1}{p_{xj}} + \frac{1}{1 - \sum_{k=1}^{m_0-1} p_{kj}} > 0$$

□

$$P_{\text{samecl}|eq\_val} = \sum_j \left( \frac{p_{.j}^2}{\sum_j p_{.j}^2} \times \sum_k p_{k|j}^2 \right)$$

dobimo:

$$\begin{aligned} Relief(A_i) &= \frac{P_{eq\_val} \times Gini'(A_i)}{P_{\text{samecl}}(1 - P_{\text{samecl}})} \\ &= \frac{\sum_j p_{.j}^2 \times Gini'(A_i)}{\sum_k p_{.k}^2 (1 - \sum_k p_{.k}^2)} \\ &= \text{konstanta} \times \sum_j p_{.j}^2 \times Gini'(A_i) \end{aligned} \quad (5.13)$$

kjer je

$$Gini'(A) = \sum_j \left( \frac{p_{.j}^2}{\sum_j p_{.j}^2} \times \sum_k p_{k|j}^2 \right) - \sum_k p_{.k}^2 \quad (5.14)$$

□

**Relacija variance z Gini-indeksom** Lastnost dokaže nasledenja izpeljava:

$$Gini\_prior = p_1.p_2 + p_2.p_1 = \frac{2n_1.n_2}{n^2}$$

$$\bar{r} = \frac{n_1. \times 0 + n_2. \times 1}{n} = \frac{n_2.}{n}$$

$$\begin{aligned} s^2 &= \frac{1}{n} \sum_{k=1}^n (r^{(k)} - \bar{r})^2 \\ &= \frac{1}{n} \left( \sum_{k=1}^{n_1.} \left( 0 - \frac{n_2.}{n} \right)^2 + \sum_{k=1}^{n_2.} \left( 1 - \frac{n_2.}{n} \right)^2 \right) \\ &= \frac{1}{n} \left( n_1. \left( \frac{n_2.}{n} \right)^2 + n_2. \left( \frac{n_1.}{n} \right)^2 \right) \\ &= \frac{n_1.n_2.(n_1.+n_2.)}{n^3} \\ &= \frac{n_1.n_2.}{n^2} \\ &= \frac{Gini\_prior}{2} \end{aligned}$$

□

## Literatura

- [1] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone. *Classification and Regression Trees*. Wadsforth International Group, 1984.
- [2] P. Clark, T. Niblett. Induction in noisy domains. V I. Bratko, N. Lavrač, (ur.), *Progress in Machine learning*. Sigma Press, Wilmslow, England, 1987.
- [3] E. Hunt, J. Martin, P. Stone. *Experiments in Induction*. Academic Press, New York, 1966.
- [4] K. Kira, L. Rendell. The feature selection problem: traditional methods and new algorithm. V AAAI-92, 7 1992. San Jose, CA.
- [5] K. Kira, L. Rendell. A practical approach to feature selection. V D. Sleeman, P. Edwards, (ur.), *Int. Conf. on Machine Learning*, str. 249–256, Aberdeen, Scotland, 1992. Morgan Kaufmann.
- [6] I. Kononenko. Estimating attributes: Analysis and extensions of RELIEF. V L. De Raedt, F. Bergadano, (ur.), *European Conf. on Machine Learning*, str. 171–182, Catania, Italy, 1994. Springer Verlag.
- [7] I. Kononenko. On biases in estimating multivalued attributes. V *Int. Joint Conf. on Artificial Intelligence IJCAI-95*, str. 1034–1040, 1995. Montreal, August 20–25.
- [8] I. Kononenko, M. Robnik-Šikonja, U. Pompe. ReliefF for estimation and discretization of attributes in classification, regression, and ILP problems. V A. Ramsey, (ur.), *AIMSA-96*, str. 31–40. IOS Press, 9 1996. Sozopol, Bulgaria.
- [9] M. Li, P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer Verlag, 1993.
- [10] R. L. Mantaras. ID3 revisited: A distance based criterion for attribute selection. V *Int. Symp. Methodologies for Intelligent Systems*, Charlotte, North Carolina, U.S.A., 1989.
- [11] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [12] C. Rajska. A metric space of discrete probability distributions. *Information and Control*, 4:373–377, 1961.
- [13] J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1993.
- [14] M. Robnik-Šikonja, I. Kononenko. An adaptation of Relief for attribute estimation in regression. V *Proc. Int. Conf. on Machine Learning ICML-97*, str. 296–304, Nashville, 1997.
- [15] M. Robnik-Šikonja, I. Kononenko. Theoretical and empirical analysis of ReliefF and RReliefF. *Machine Learning*, 53:23–69, 2003.

- [16] C. E. Shannon, W. Weaver. *The Mathematical Theory of Communications*. Urbana: The University of Illinois Press, 1949.
- [17] P. Smyth, R. M. Goodman. Rule induction using information theory. V G. Piatetsky-Shapiro, W. Frawley, (ur.), *Knowledge Discovery in Databases*. MIT Press, 1990.

## Poglavlje 6

# Predobdelava učnih primerov

*Velika inteligenca je dvorenzen meč. Lahko jo uporabimo konstruktivno ali destruktivno... Inteligenca je pravilno vodena šele potem, ko um prizna neizogibnost duhovnih zakonov.*

*Sri Yukteswar*

Večina algoritmov strojnega učenja ima različne omejitve glede dovoljenih opisov učnih primerov. V tem poglavju so najprej predstavljene metode za atributno predstavitev kompleksnih struktur, ki omogočajo tudi učenje iz besedil, slik in grafov, zatem pa metode za spremenjanje atributnega opisa primerov: diskretizacija zveznih atributov, mehka diskretizacija zveznih atributov, binarizacija atributov, spremenjanje diskretnih atributov v zvezne in obravnavanje neznanih vrednosti. Na koncu poglavja sta opisana vizualizacija, kot pomemben pristop k razumevanju in analizi tako vhodnih podatkov kot tudi rezultatov strojnega učenja, ter postopki za izbiro podmožice atributov. Konstruktivna indukcija, ki je tudi namenjena spremenjeni predstavitvi učnih primerov, pa presega namen tega učbenika. Zainteresirani bralec lahko poseže po bolj poglobljeni literaturi, npr. [7].

### 6.1 Atributna predstavitev kompleksnih struktur

*Razum ne more nikoli prepozнати нити устварити лепоте.*

*Eckhart Tolle*

Atributna predstavitev primerov je definirana v razdelku 4.1, kjer so tudi opisane lastnosti atributov in njihove medsebojne odvisnosti. Tu pa si oglejmo, kako predstaviti z atributnim opisom kompleksne strukture, kot so besedila, slike in grafi. Kompleksne strukture se da naravno opisati v predikatnem računu 1. reda. Če želimo uporabiti atributno predstavitev, kjer je število atributov dano vnaprej in je enako za vse učne primere, je opis kompleksnih struktur samo približen in povzema pomembne lastnosti strukture.

## Atributna predstavitev besedila

Z razvojem računalniške tehnologije in medmrežja je količina dostopnih tekstov zelo velika. Ročno iskanje pomembnih dokumentov je brezupno in je v ta namen nujna računalniška podpora. Za iskanje relevantnih dokumentov (*information retrieval*) ter za klasifikacijo tekstov na spletu ali v ločenih podatkovnih bazah se lahko uporabi strojno učenje. Pred uporabo metod strojnega učenja je treba učne primere, torej tekstovne dokumente, ustrezno opisati.

Za opis besedil se uporablja t.i. *vreča besed* (*bag of words*). Vsaka možna beseda predstavlja en atribut, njegova vrednost za dani dokument pa je število pojavitev te besede v besedilu. Ker je število različnih besed in s tem število atributov reda  $10^5$  in več, skušamo število uporabljenih besed zmanjšati. Izvržemo veznike (npr. in, ter, ali), ločilne besede, nekatere pogoste besede (npr. je) in splošne besede, ki ne nosijo informacije o vsebini besedila. Poleg tega besede nadomestimo s koreni besed. Npr. za besedi "pretepati" in "pretepač", kot tudi za vse njune oblike (spregative, sklone itd.), kot npr. pretepam, pretepali, pretepača, pretepačev itd., se uporabi beseda "pretep". Včasih celo sinonime štejejo kot enake besede.

Kljub zmanjšanju števila atributov je njihovo število v vreči besed še vedno zelo veliko, hkrati pa so vrednosti atributov povečini 0. Zato ponavadi vrečo besed predstavimo s seznamom atributov, ki imajo neničelno vrednost. Zaradi velikega števila atributov je potrebno izbrati ustrezni in učinkovit algoritem strojnega učenja. Za klasifikacijo besedil se najpogosteje uporablja v te namene prilagojeni naivni Bayesov klasifikator in metoda podpornih vektorjev, ki dajeta tudi najboljše rezultate.

## Atributna predstavitev slik

Podobno kot pri besedilih je tudi pri razpoznavanju in klasifikaciji slik ter pri iskanju podobnih slik v različnih bazah in na svetovnem spletu potrebno slike opisati z numeričnimi parametri - atributi. Za opis (parametrizacijo) slik se uporabljajo metode, kot so npr. metode za opis tekstur ali specializirane metode za posamezna področja uporabe:

**Statistike 1. in 2. reda** se izračunajo iz verjetnosti, da ima naključno izberana pika na sliki določeno lastnost [6]. Te verjetnosti so odvisne od posameznih točk in ne od njihovih odnosov s sosednjimi točkami. Primeri atributov na osnovi statistik prvega reda so povprečna svetlost, standardno odstopanje svetlosti, entropija, momenti itd.

Statistike 2. reda se izračunajo iz verjetnosti, da imata dve naključno izbrani točki na neki razdalji določene lastnosti. Primeri atributov na osnovi statistik drugega reda so kontrast, homogenost, korelacija in varianca.

**Analiza glavnih komponent** (*Principal Component Analysis - PCA*) je matematični postopek, ki pretvori večje število koreliranih spremenljivk v manjše število nekoreliranih spremenljivk, ki jim pravimo glavne komponente. PCA sta za procesiranje slik prvič uporabila Sirovich in Kirby [12]. Znana je uporaba PCA za razpoznavanje obrazov [14]. Vsaka slika iz učne množice se predstavi kot linearna kombinacija lastnih vektorjev. Metoda poskuša najti glavne komponente iz množice lastnih vektorjev.

**Atributi, izpeljani s pomočjo povezovalnih pravil:** tako kot lahko uporabimo povezovalna pravila za opis relacij med vrednostmi atributov v podatkovni bazi, jih uporabimo tudi

za opis relacij med točkami na slikah [1, 10]. Povezovalno pravilo je sestavljenko kot konjunkcija pogojev v pogojnem in v sklepnom delu pravila. V primeru slik so pogoji lastnosti točk, ki so na določeni oddaljenosti od začetne točke. Na ta način povezovalno pravilo statistično opiše relacije med sosednjimi točkami.

**Domeni prilagojeni atributi:** za posebne domene lahko izpeljemo prilagojene attribute, ki so bolj informativni kot splošni opisi. Taki so npr. numerični parametri za opis slik kapljic vode ali krvi ter posnetkov koron pri slikanju človeških prstov s Kirlianovo kamero [8, 11]. Primeri atributov so ploščina korone, nivo šuma na sliki, koeficient oblike, fraktalna dimenzija, koeficient svetlosti, odstopanje svetlosti, število ločenih fragmentov na sliki itd.

### Atributna predstavitev grafov

Grafi so naravna predstavitev znanja za mnoge realne probleme. Z njimi ponazorimo dogodke, ki se dogajajo po nekem časovnem zaporedju (npr. grafi za ponazoritev načrtov velikih projektov). Lahko pa grafi ponazarjajo relacije med podatki/opisi objektov, kot je npr. povezanost dokumentov na svetovnem spletu. Primeri kompleksnejših grafov, ki jih je bilo za strojno učenje potrebno prevesti v atributni opis, so opisani v [2].

Za atributni opis grafov izluščimo splošne lastnosti grafov, kot so npr.

- število vozlišč in število povezav v grafu,
- povprečna vstopna stopnja vozlišča (število povezav, ki vstopajo v vozlišče),
- povprečna izstopna stopnja vozlišča (število povezav, ki izstopajo iz vozlišča),
- minimalna, maksimalna in povprečna dolžina acikličnih poti v grafu,
- povprečne lastnosti dokumentov v vozliščih,
- število ciklov v grafu,
- povprečne lastnosti dokumentov v sosednih vozliščih, itd.

Za vsako domeno je potrebno definirati specializirane attribute, ki povzemajo pomembnejše lastnosti podatkov, ki jih opisuje njihov graf.

## 6.2 Diskretizacija zveznih atributov

*Bog, daj mi vedrost, da bom sprejel stvari, ki jih ne morem spremeniti, pogum, da bom spremjal tiste, ki jih lahko, in modrost, da bom prepoznal razliko.*

Sv. Frančišek Asiški (citat je uradno pripisan Reinholdu Niebuhrju)

Mnogi algoritmi za strojno učenje znajo uporabljati samo diskrette atribute. Zanje je potrebno zvezne atribute diskretizirati vnaprej ali med samim učenjem. Diskretizacija zveznega atributa pomeni, da se interval vseh možnih vrednosti atributa razbije na določeno število podintervalov, ki predstavljajo diskrette vrednosti tega atributa. Diskretizacija atributa pomeni kvečjemu izgubo informacije, saj novi atrribut ne ločuje med vrednostmi atributa iz istega podintervala. Zato mora algoritem za diskretizacijo določiti:

- optimalno število intervalov ter
- optimalne meje vseh intervalov,

tako da se ohrani čimveč informacije za dani problem. V nadaljevanju se omejimo na klasifikacijske probleme. Diskretizacija atributov v regresijskih problemih poteka na podoben način.

Posebej pomembna je izbira števila intervalov. Čimveč je intervalov, manj informacije je izgubljene, zato pa je aproksimacija verjetnostnih porazdelitev znotraj intervalov manj zanesljiva. Čimmanj je intervalov, več koristne informacije se lahko izgubi, raste pa zanesljivost aproksimacij verjetnostnih porazdelitev po razredih. Število želenih intervalov se lahko določi avtomatsko, če je na voljo ustrezna ocenitvena funkcija, ali ročno s parametrom, ki določa želeno število intervalov.

### Vrste metod za diskretizacijo

V praksi se uporablja dva osnovna pristopa k diskretizaciji, ki temeljita na požrešnem algoritmu: *od spodaj navzgor (bottom-up)* ali *od zgoraj navzdol (top-down)*. V obeh primerih je potrebno učne primere najprej urediti po vrednostih zveznega atributa.

Pri metodi od spodaj navzgor začnemo s toliko intervali, kolikor je učnih primerov. Zatem združujemo po dva sosednja intervala, ki sta si najbolj podobna po verjetnostni porazdelitvi razredov, relativno glede na dano funkcijo kvalitete atributa  $q(A)$ . Združevanje nadaljujemo, dokler kvaliteta atributa narašča oziroma dokler ne dobimo samo dveh intervalov. Časovna zahtevnost te metode je reda  $O(n^2)$ , kjer je  $n$  število učnih primerov.

Pri metodi od zgoraj navzdol začnemo s celotnim intervalom vrednosti in v vsakem koraku izberemo mejo, ki maksimizira kvaliteto atributa. Tudi pri tej metodi je časovna kompleksnost v najslabšem primeru reda  $O(n^2)$ , vendar je kompleksnost zaradi relativno majhnega števila intervalov  $k$  v primerjavi s številom učnih primerov  $n$  dejansko manjša in jo lahko ocenimo z  $O(kn)$ .

### Možne meje med intervali

Pri diskretizaciji upoštevamo kot potencialne meje med intervali samo meje med primeroma, ki pripadata različnima razredoma. Naj bo

$$\phi(P(r_1), \dots, P(r_{m_0}))$$

mera nečistoče verjetnostne porazdelitve razredov  $r_k$ . Če uporabljamo za oceno kvalitete atributa  $A$  pričakovano zmanjšanje nečistoče pri uporabi meje, ki definira podintervala z označkama  $V_1$  in  $V_2$ :

$$q(A) = \phi(P(r_1), \dots, P(r_{m_0})) - \sum_{j=1}^2 P(V_j) \phi(P(r_1|V_j), \dots, P(r_{m_0}|V_j))$$

velja, da ima funkcija  $q$  med možnimi mejami znotraj zaporedja učnih primerov iz istega razreda maksimum na začetku ali na koncu zaporedja. To lastnost pokažemo tako, da je drugi odvod funkcije  $q$  glede na premik meje vedno pozitiven. Ker je funkcija  $q$  konveksna, ima zato maksimum na začetku ali na koncu intervala. Glej dokaz na strani 149.

### Mere za usmerjanje diskretizacije

Za usmerjanje diskretizacije zveznega atributa lahko uporabimo katerokoli mero za ocenjevanje kvalitete atributa. Velja, da pričakovano zmanjšanje nečistoče, t.j. kvaliteta  $q$ , z dodajanjem novih mej kvečjemu naraste. To neposredno sledi iz lastnosti, da pomembnost atributa  $q$  kvečjemu naraste, če mu dodamo vrednost. Zato informacijski prispevek in Gini-indeks nista primerni meri za diskretizacijo. Primernejše so funkcije, ki ne precenjujejo večvrednostnih atributov, npr. mera bližine  $I - D$ ,  $MDL$  in  $ReliefF$ .

## 6.3 Mehka diskretizacija zveznih atributov

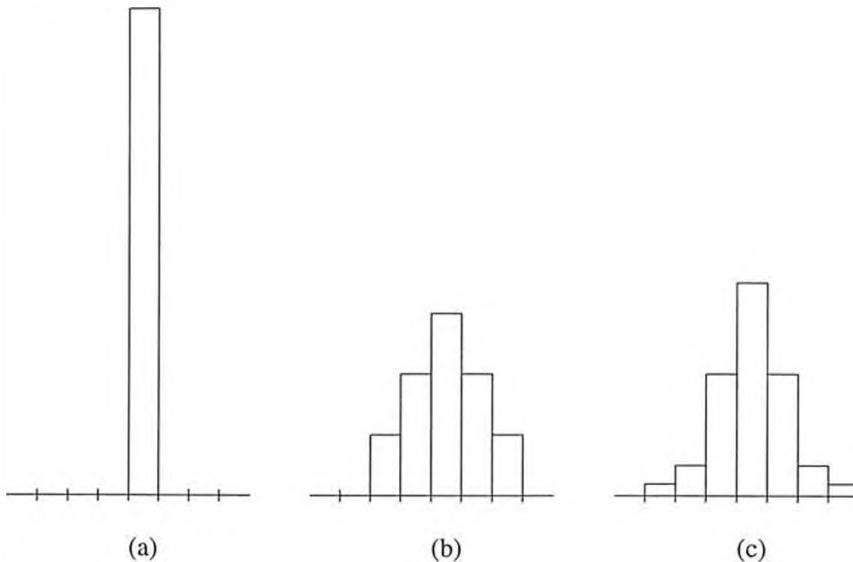
*Edini hudiči na tem svetu so tisti, ki se podijo v našem srcu, in v njem bi morali izbojevati vse svoje bitke.*

*Mahatma Gandhi*

Pri diskretizaciji zveznih atributov meja med intervaloma predstavlja mejo med dvema logično različnima odločtvama. Primeri, ki imajo vrednost zveznega atributa blizu meje, se obravnavajo enako kot primeri, katerih vrednost je daleč a na isti strani meje, in drugače kot primeri, ki so blizu vendar na drugi strani meje. Če npr. diskretiziramo zvezni atribut z mejo 0.30, tako da dobimo intervala [0.00, 0.30] in [0.31, 1.00], se primera z vrednostima atributa 0.00 in 0.30 obravnavata enako, medtem ko se primera z vrednostima 0.30 in 0.31 obravnavata različno.

Tej pomanjkljivosti trde diskretizacije se izognemo, če uporabimo mehko diskretizacijo zveznih atributov. V tem primeru meja ni fiksna točka, ampak je raztegnjena in predstavlja

verjetnost, da primer pripada eni oziroma drugi strani meje. Na ta način modeliramo negotovost pripadnosti eni oziroma drugi strani meje. Vsak učni primer ima za tako dobljeni diskretni atribut vrednost, podano kot verjetnost pripadnosti različnim intervalom, t.j. verjetnostno porazdelitev posameznih diskretnih vrednosti. Pri trdi diskretizaciji imamo stopničasto porazdelitev, kot jo prikazuje slika 6.1a, pri mehki diskretizaciji pa lahko uporabimo različne monotone porazdelitve. Primera sta na sliki 6.1b in c. Druga možnost je, da namesto mehkih



Slika 6.1: Trda in dve varianti mehke diskretizacije pri danem učnem primeru.

mej med intervali uporabimo mehke vrednosti originalnega zveznega atributa. Namesto ene točke na realni osi ima v tem primeru vsak učni primer podano samo najbolj verjetno vrednost, gostota verjetnosti pa je podana s funkcijo, ki monotono pada levo in desno od najbolj verjetne vrednosti. Ko atribut diskretiziramo z uvedbo trdih mej, se primeru pripiše vrednost diskretenega atributa z verjetnostjo, ki je enaka ploščini pod verjetnostno porazdelitvijo na danem intervalu.

## 6.4 Binarizacija atributov

*Vzrok za nasilje je strah.*

*Jiddu Krishnamurti*

Mnogi algoritmi za strojno učenje odločitvenih dreves gradijo binarna drevesa. V ta namen pred izbiro atributa za koren poddrevesa atributte binariziramo, tako da iz vsakega atributa dobimo več binarnih atributov, t.j. diskretnih atributov z dvema vrednostima. Izmed binarnih atributov zatem izberemo najboljšega.

Za gradnjo binarnih odločitvenih dreves je več razlogov.

- Z gradnjo binarnih dreves se poveča verjetnost, da bo drevo manjše in s tem optimalnejše.
  - Več različnim vrednostim večvrednostnega atributa lahko ustreza enako ali podobno poddrevo, kar poznamo kot problem podvojevanja poddreves (*replication problem*). Z binarizacijo atributa upamo, da se bodo enaka ali podobna poddresova združila v eno poddrevo.
  - Pri gradnji binarnih odločitvenih dreves se množica učnih primerov v vsakem koraku razbije na dve podmnožici. To preprečuje razdrobljenost učne množice, do katere pride, če uporabimo atribut z mnogo vrednostmi. Na ta način je vsaka odločitev oz. aproksimacija verjetnosti iz učne množice statistično bolj podprtta.
- Nekatere ocene atributov (informacijski prispevek, Gini-indeks) precenjujejo večvrednostne attribute. Z binarizacijo se izognemo temu problemu.
- Nekatere ocene atributov znajo ocenjevati samo binarne attribute.

Pri binarizaciji zveznega atributa uporabimo samo prvi korak diskretizacije od zgoraj navzdol. Poisčemo namreč tisto mejo znotraj intervala možnih vrednosti danega atributa, ki maksimizira pomembnost atributa.

Pri binarizaciji diskretnega atributa z več kot dvema vrednostima sta možna dva pristopa.

- Za vsako vrednost atributa generiramo en binarni atribut tako, da vse ostale vrednosti združimo. Število novih binarnih atributov je enako številu vrednosti originalnega diskretnega atributa  $m$ .
- Generiramo toliko binarnih atributov, kolikor je možnih različnih razbitij množice vrednosti atributa na dve disjunktni podmnožici. Če ima originalni atribut  $m$  vrednosti, dobimo

$$\frac{1}{2} \sum_{k=1}^{m-1} \binom{m}{k} = 2^{m-1} - 1$$

novih binarnih atributov.

Drugi pristop je splošnejši, vendar je za diskrete attribute z mnogo vrednostmi časovno prezahteven. V posebnih primerih se izognemo časovni zahtevnosti, ne da bi izgubili optimalkost. V takih primerih obstaja algoritem s sprejemljivo časovno zahtevnostjo, ki najde optimalno binarno različico danega večvrednostnega diskretnega atributa. V ostalih primerih uporabimo učinkovit algoritem za iskanje suboptimalne binarizacije.

## Dvorazredni problemi

Pri dvorazrednih problemih lahko za poljubno mero nečistoče odkrijemo optimalno binarizacijo atributa  $A$  z  $m$  vrednostmi v času  $O(m)$ . Vrednosti atributa  $V_j, j = 1 \dots m$  uredimo po naraščajočih pogojnih verjetnostih prvega razreda:

$$P(r_1|V_1) \leq P(r_1|V_2) \leq \dots \leq P(r_1|V_m)$$

Pomembnost binariziranega atributa  $A$ :

$$q(A) = \phi(P(r_1), \dots, P(r_{m_0})) - \sum_{j=1}^2 P(\mathcal{V}_j) \phi(P(r_1 | \mathcal{V}_j), \dots, P(r_{m_0} | \mathcal{V}_j))$$

je največja za podmnožici  $\mathcal{V}_j, j = 1, 2$ , za kateri velja:

$$\mathcal{V}_1 = \{V_1, \dots, V_l\}, \quad \mathcal{V}_2 = \{V_{l+1}, \dots, V_m\}$$

pri čemer je  $0 < l < m$ . Dokaz te trditve najdemo v knjigi Breimana in sod. [3]. Torej pri dvorazrednih problemih zadošča, da pregledamo  $m - 1$  razbitij diskretnega atributa namesto vseh  $2^{m-1} - 1$ .

### Večrazredni problemi

Pri večrazrednih problemih in kadar v dvorazrednih problemih ne želimo uporabljati mere nečistoče, uporabimo požrešni algoritem za iskanje suboptimalne binarizacije atributa:

1. Izberi eno vrednost, ki maksimizira kvaliteto atributa z združenimi vsemi ostalimi vrednostmi.
2. Ponavljam, dokler se kvaliteta atributa ne spreminja več:
  - (a) Če v drugi množici obstaja vrednost, ki premaknjena v prvo množico poveča kvaliteto atributa, premakni iz druge množice v prvo tisto vrednost, ki maksimizira kvaliteto binarnega atributa.
  - (b) Če v prvi množici obstaja vrednost, ki premaknjena v drugo množico poveča kvaliteto atributa, premakni iz prve množice v drugo tisto vrednost, ki maksimizira kvaliteto binarnega atributa.

## 6.5 Spreminjanje diskretnih atributov v zvezne

*Umetnost se zaključi v posameznih delih, znanost se nam kaže nikoli zaključena.*

*Johann Wolfgang Goethe*

Nekatere metode za strojno učenje ne znajo obravnavati diskretnih atributov in predpostavljajo, da so vsi atributi zvezni. Take so npr. linerna in druge regresije, diskriminantne funkcije in nevronske mreže.

Če ima atribut dve vrednosti, ga lahko obravnavamo kot poseben primer zveznega atributa. Eni vrednosti pripisemo vrednost 0, drugi pa vrednost 1. Tako postaneta vrednosti urejeni in lahko na njima uporabljamo aritmetične operacije in funkcije.

Če so vrednosti večvrednostnega diskretnega atributa urejene (t.i. ordinalni atribut), jih oštevilčimo in obravnavamo kot vrednosti zveznega atributa. Sicer pa je večvrednostni diskretni atribut potrebno spremeniti v toliko binarnih atributov (z vrednostmi 0 in 1), kolikor

ima originalni atribut različnih vrednosti. Za neko vrednost originalnega atributa bo imel binarni atribut, ki ustreza tej vrednosti, vrednost 1, vsi ostali binarni atributi pa vrednost 0. Slaba stran te transformacije je veliko število medsebojno odvisnih novih atributov. Včasih je dobro zmanjšati število binarnih atributov za ena, t.j. če ima diskretni atribut  $m$  vrednosti, ga pretvorimo v  $m - 1$  binarnih atributov in je vrednost zadnjega binarnega atributa podana implicitno z vrednostmi 0 pri vseh  $m - 1$  binarnih atributih.

## 6.6 Obravnavanje neznanih vrednosti

*Prava modrost starega človeka je v tem, da zna smrt vključiti v svoj življenjski in delovni načrt kot končni vrh svojega življenja in dela.*

*Anton Trstenjak*

V realnih učnih problemih se pogosto srečamo z manjkajočimi podatki. Če učnemu primeru vrednost atributa  $A_i$  manjka, ga učna metoda lahko ignorira ali pa poskuša bodisi nadomestiti manjkajočo vrednost z najbolj verjetno vrednostjo ali pa uporabi verjetnostno porazdelitev po posameznih vrednostih atributa. Pri izračunu verjetnosti vrednosti atributa se uporablja pogojne verjetnosti pri danem razredu primera  $r_k$ , če je atribut  $A_i$  diskreten:

$$P(V_j|r_k), \quad j = 1 \dots m_i$$

ozioroma pogojne gostote verjetnosti pri danem razredu primera, če je atribut zvezen. Če želimo nadomestiti manjkajoče vrednosti z najbolj verjetnimi vrednostmi, lahko uporabimo nek učni algoritem za učenje preslikave vrednosti ostalih atributov in razreda v atribut, katerega vrednost manjka. Na ta način dobimo toliko učnih problemov, kolikor je atributov, ki nimajo podanih vrednosti za vsak učni primer.

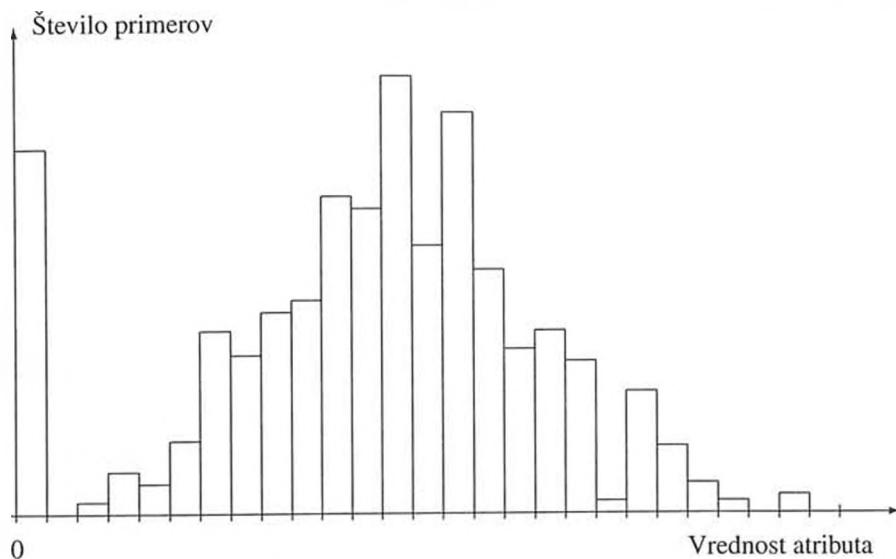
Poleg neznanih lahko v podatkih nastopajo tudi druge posebne vrednosti. Npr. poljubna (*don't care*) vrednost, kjer primer razmnožimo na toliko primerov, kolikor je možnih različnih vrednosti za dani atribut, ali nesmiselna (*inapplicable*) vrednost (npr. pri moških pacientih so nesmiselne vrednosti, ki se nanašajo na ženske spolne organe).

## 6.7 Vizualizacija

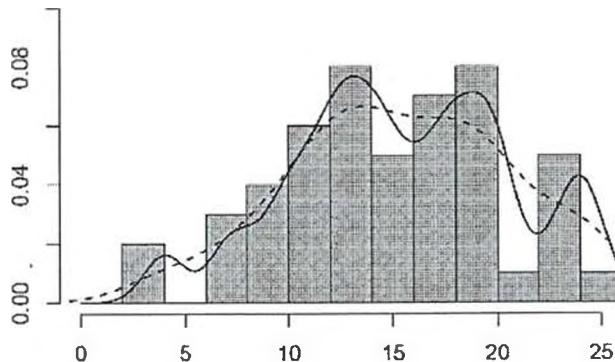
*Nezavedanje osnovne enosti organizma in okolja je resna in nevarna halucinacija.*

*Alan Watts*

Človek zaradi narave svojih možganov veliko hitreje procesira vizualno kot numerično informacijo. Zato je vizualizacija podatkov zelo pomembna pri pregledovanju in interpretaciji podatkov, pa tudi pri urejanju podatkov, iskanju pomembnih relacij, ugotavljanju kvalitet in pomanjkljivosti v podatkih ipd. Vizualizacija je dejansko transformacija podatkov v preglenejšo obliko. Pojem preglednosti je subjektiven in od posameznika je odvisno, kaj je



Slika 6.2: Primer histograma.



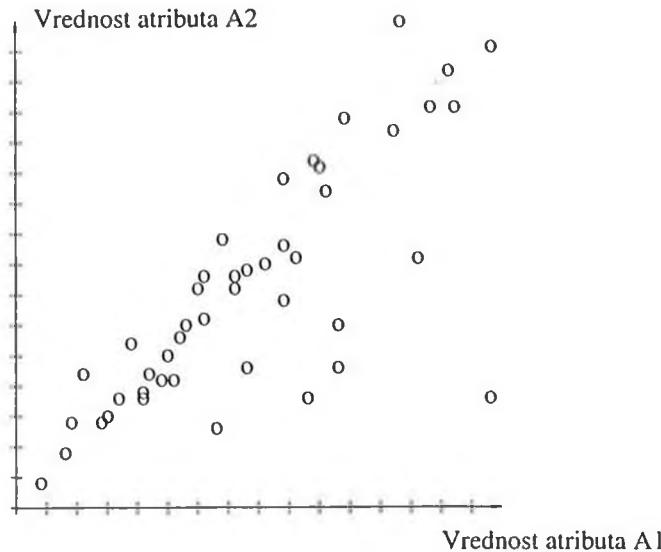
Slika 6.3: Primer dveh različnih glajenj histograma za prikaz zvezne distribucije primerov. Polna črta prikazuje šibkejše glajenje, črtkana pa močnejše glajenje.

najpreglednejša oblika predstavitev podatkov. Vseeno pa veljajo splošni principi, ki jih lahko uporabimo za pregledno vizualizacijo [13].

Pri strojnem učenju je pomembno, da ustrezno pripravimo vhodne podatke. Postopek učenja načrtujemo na osnovi poznavanja podatkov. To pomeni, da moramo podatke pregledati, urediti, spoznati njihove lastnosti in jih transformirati v obliko, ki je najprimernejša za učenje. Pri tem je v neprecenljivo pomoč strokovnjak iz problemske domene, če je na voljo

in če nam bo posvetil dovolj časa. Ne glede na strokovnjakovo poznavanje domene pa se je treba spoznati z lastnostmi podatkov in njihovo kvalitetu, pri čemer si pomagamo z različnimi metodami vizualizacije.

Prav tako je pomembno razumeti in interpretirati vmesne in končne rezultate strojnega učenja, pri čemer nam prav tako lahko veliko pomaga strokovnjak za domeno. Vizualizacija tukaj olajša dialog s strokovnjakom in pohitri razumevanje smiselnosti in pomena naučenega, olajša pa tudi ocenjevanje dobrijene hipoteze.

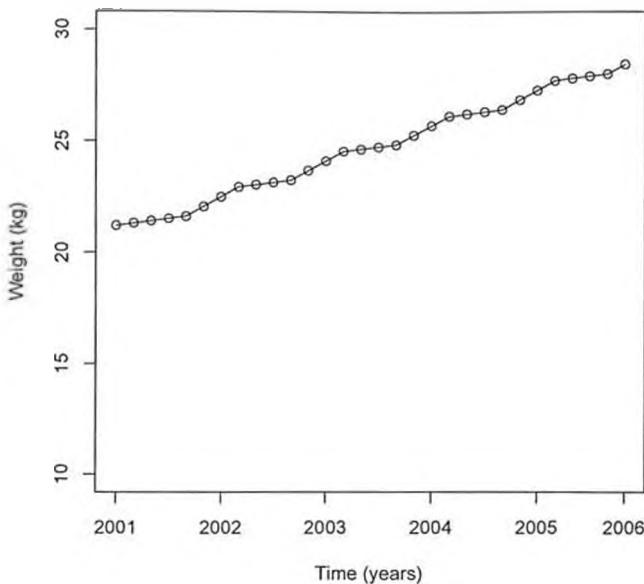


Slika 6.4: Primer razsevnega diagrama, ki prikazuje skoraj linearno odvisnost med dvema zveznima atributoma.

### Vizualizacija enega atributa

Za vizualizacijo porazdelitve vrednosti enega atributa oziroma ene spremenljivke se najpogosteje uporabljojajo histogrami, ki ponazarjajo pogostost pojavljanja vrednosti spremenljivke. Histogrami vizualizirajo diskretne spremenljivke, zvezne pa je potrebno diskretizirati. Iz njih razberemo distribucijo primerov po vrednostih spremenljivke. Opozorilo nas na neneavadne lastnosti distribucije, ki lahko nakazujejo posebnosti domene, na napake v podatkih ali na specifično kodiranje podatkov. Primer histograma zvezne spremenljivke je podan na sliki 6.2. Iz slike je razvidno neneavadno veliko število primerov z vrednostjo blizu 0. V danem primeru se je izkazalo, da gre za kodiranje, v katerem so manjkajoče (neznané) vrednosti atributa označili z ničlo.

Histogram spremenimo v zvezno distribucijo z različnimi algoritmimi glajenja. Ponavadi se uporabljojajo Gaussove jedrne funkcije. Od širine jedrne funkcije je odvisna jakost glajenja. Prave jakosti glajenja ni lahko določiti. Preveč glajenja skrije pomembno informacijo, pre-



Slika 6.5: Primer časovnega grafa.

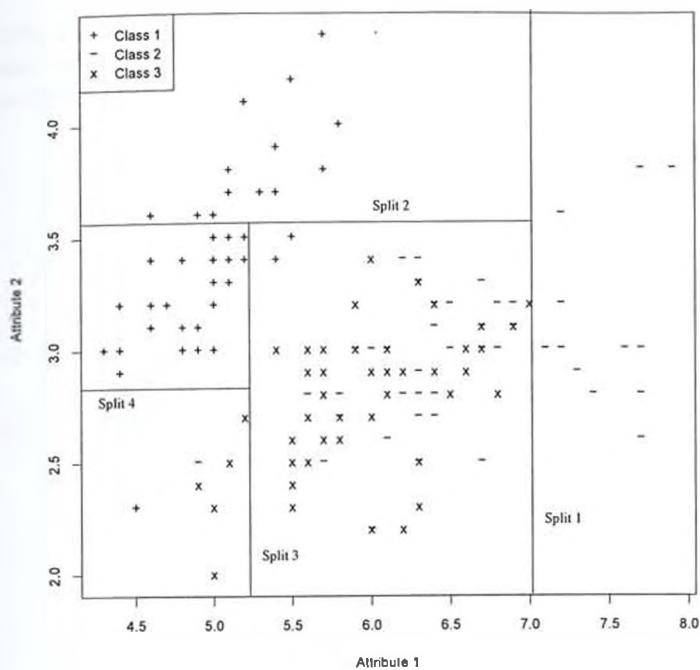
malо glajenja pa ne povzame podatkov, ampak pokaže nepomembne podrobnosti. Primera različnega glajenja na istih podatkih sta prikazana na sliki 6.3.

### Vizualizacija parov atributov

Pri vizualizaciji porazdelitve vrednosti dveh atributov nas zanima medsebojna odvisnost atributov. V ta namen se uporablja razsevni diagram (*scatterplot*), ki v dvodimenzionalnem prostoru prikazuje po eno točko za vsak učni primer. Primer vizualizacije skoraj linearne odvisnosti med dvema spremenljivkama je podan na sliki 6.4.

Če je eden od atributov čas, lahko prikažemo časovni graf. Ti grafi pokažejo zanimive lastnosti vrednosti atributa v odvisnosti od časa. Na sliki 6.5 [5] je razviden nepričakovani stopničasti vzorec za vrednosti atributa. Graf prikazuje merjenje teže otrok v različnih časovnih obdobjih in nenavadno bi bilo, da bi teža otrok naraščala po stopničasti krivulji. Izkazalo se je, da je vzorec posledica površnih meritev. Otroci so bili tehtani oblečeni. Pri pari meritev je bila prva meritev opravljena pozimi, ko so otroci bolj oblečeni, druga pa poleti, kar je prispevalo k nenavadnemu vzorcu, ki nima nič skupnega z dejanskim pridobivanjem teže pri otrocih.

Včasih želimo pri klasifikacijskih problemih vizualizirati odvisnosti med razredom in dvema atributoma. V tem primeru je razsevni diagram namesto iz enakih točk sestavljen



Slika 6.6: Primer razsevnega diagrama, ki prikazuje rezultat učenja na trorazrednem problemu z odločitvenim drevesom. "Split" pomeni razbitje učne množice na dva dela glede na vrednosti atributa v korenju ustreznega poddrevesa.

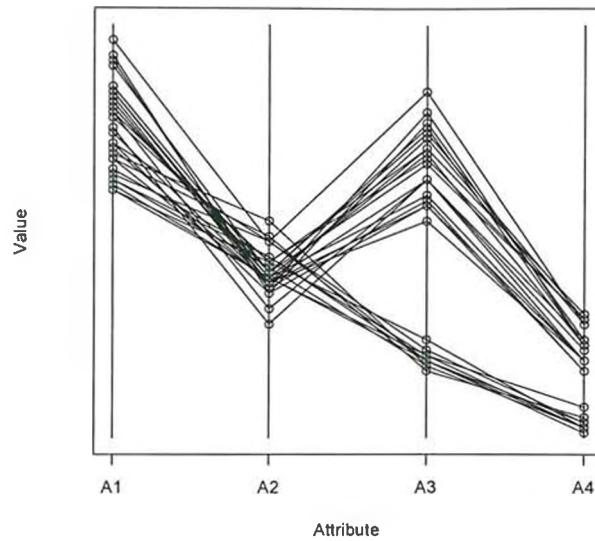
iz različnih simbolov, tako da so primeri iz istega razreda predstavljeni s svojim simbolom. Na sliki 6.6 je prikazan razsevni diagram za tri razrede, ki so predstavljeni s tremi simboli (krogec, križec in zvezdica).

### Vizualizacija več atributov

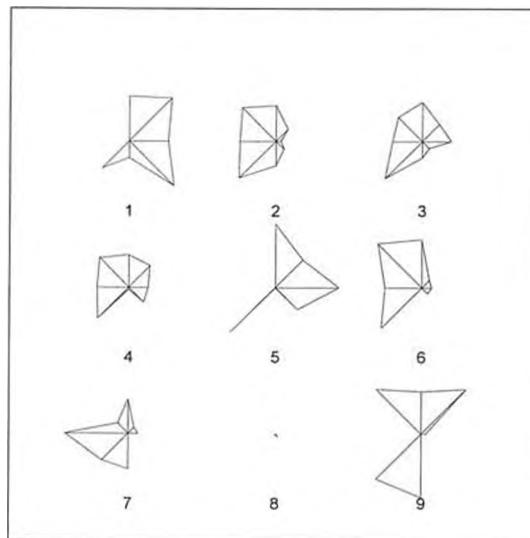
Vizualizacije več kot dveh atributov se lotimo s prikazom razsevnih diagramov za vse pare atributov. Na ta način dobimo matriko razsevnih diagramov s prazno diagonalo, ki jo lahko uporabimo za zapis imen atributov. To seveda ni prava vizualizacija vseh atributov naenkrat, je pa pogosto dovolj informativna, saj nas najpogosteje zanimajo ravno interakcije parov atributov.

Drug način je prikaz vrednosti vseh atributov naenkrat za vse učne primere. Vrednosti prikažemo na vzporednem grafu, kjer je za vsak atribut ena navpična. En učni primer je prikazan kot lomljena črta preko vseh navpičnih osi. Iz vzporednega grafa lahko sklepamo na korelacijo med spremenljivkami in ugotovimo, če v učni množici nastopajo značilne podobne podmnožice primerov, kot tudi korrelacije med podmnožicami primerov. Primer vzporednega grafa za 4 zvezne spremenljivke je prikazan na sliki 6.7.

Z zvezdastimi grafi en učni primer prikažemo z zvezdo, sestavljeni iz toliko krakov,

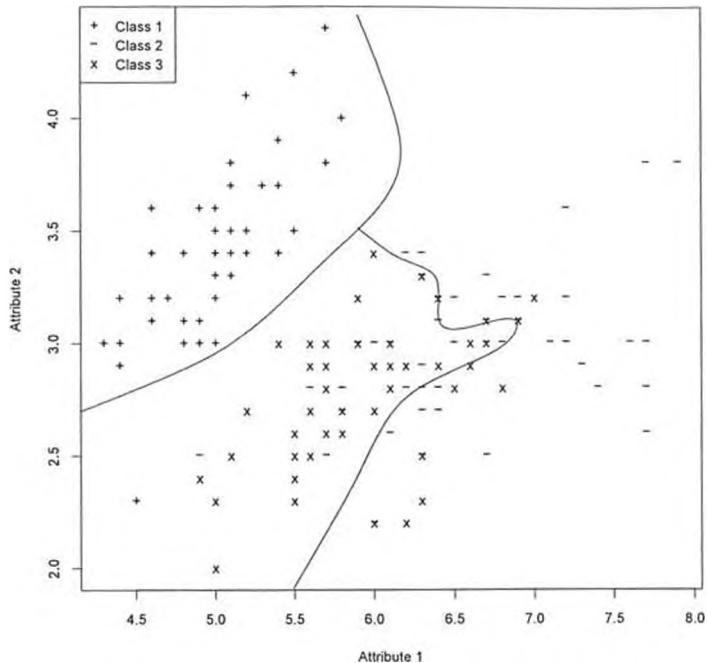


Slika 6.7: Primer vzporednega grafa, ki prikazuje vrednosti štirih zveznih atributov.



Slika 6.8: Primer zvezdastega grafa.

kolikor je atributov. Vrednost atributa je predstavljena z dolžino ustreznega kraka. Vrhovi sosednjih krakov so med seboj povezani. Zvezdasti grafi nazorno pokažejo podobnost med primeri kot podobne zvezde. Primer zvezdastega grafa je na sliki 6.8.



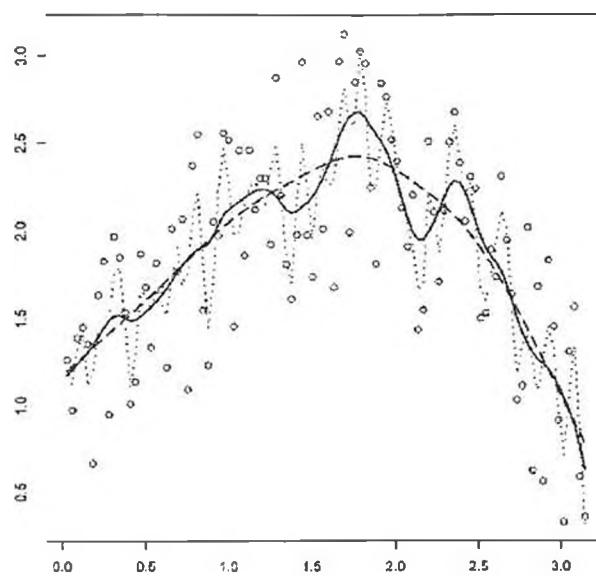
Slika 6.9: Klasifikacijski problem iz slike 6.6, rešen z nevronske mrežo.

Lahko uporabimo tudi inovativne načine vizualizacije, kot so npr. obrazi. Ljudje imamo dober spomin za obraze in na obrazih hitro opazimo določene lastnosti. Abstrahirane obraze parametriziramo s toliko parametri, kolikor imamo atributov. En obraz predstavlja en učni primer. Vrednost atributa določa lastnost obraza (npr. nasmejan ali resen, velike ali majhne oči, oblika glave, poševnost obrvi itd.). Prepoznavanje obrazov in s tem ustreznost vizualizacije je tudi kulturno pogojeno.

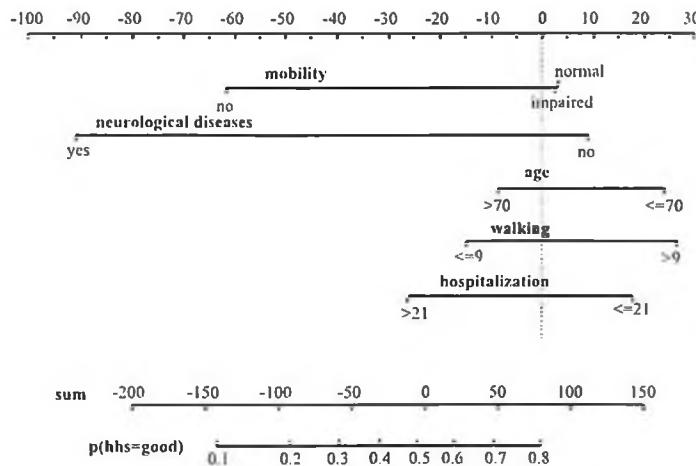
Poleg vizualizacije odvisnosti med atributi z eksplisitnimi vrednostmi atributov pri učnih primerih lahko odvisnost med atributi najprej izračunamo in zatem vizualiziramo ocenjene odvisnosti, ki so povprečene preko vseh učnih primerov.

### Vizualizacija rezultatov strojnega učenja

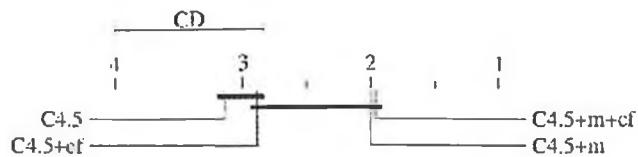
Rezultat klasifikatorja si lahko predstavljamo kot hiperploskev v toliko dimenzionalnem prostoru, kolikor imamo atributov v danem klasifikacijskem problemu. Hiperploskev ločuje primere različnih razredov med seboj. Najpreprosteje je vizualizirati posamezne dele hiperploskev, tako da uporabimo dvodimenzionalne podprostore in opazujemo lastnosti klasifikatorja, ki je v tem primeru vizualiziran s krivuljo, ki loči primere iz različnih razredov. Z



Slika 6.10: Razmerje med kompleksnostjo in natančnostjo regresorja. Točkasta krivulja predstavlja najkompleksnejši model, polna srednjega in črtkana najmanj kompleksnega.



Slika 6.11: Primer nomograma, ki prikazuje klasifikacijo z varianto naivnega Bayesovega klasifikatorja [9].



Slika 6.12: Vizualizacija značilnosti razlik uspešnosti več algoritmov.

dvodimenzionalno predstavljivjo lahko naenkrat opazujemo klasifikacijo na dveh atributih. Primer mej med razredi, ki jih določa odločitveno drevo, je na sliki 6.6. Za isti klasifikacijski problem so meje med razredi, ki se jih nauči nevronska mreža, podane na sliki 6.9.

Rezultat regresorja je hiperploskev v toliko dimenzionalnem prostoru, kolikor je atributov, plus ena za regresijsko spremenljivko. Torej je možna hkratna vizualizacija samo enega atributa v odvisnosti od regresijske (odvisne) spremenljivke. Na sliki 6.10 je prikazan rezultat treh različno kompleksnih regresorjev na istih učnih podatkih. Iz slike je razviden kompromis med kompleksnostjo in natančnostjo regresorja.

Poleg naučene funkcije je pri simboličnem učenju tudi vizualizacija strukture modela. Prikaz odločitvenega ali regresijskega drevesa je lahko interaktivni in lahko na različnih nivojih podrobnosti prikazuje pomembne podatke o drevesu. Npr. v vozlišču s klikom pregledamo število učnih primerov, distribucijo odvisne spremenljivke, ocenjeno kvaliteto atributov itd.

Pri uporabi modela za reševanje novih problemov, to je za napovedovanje vrednosti odvisne spremenljivke, je pomembna transparentnost napovedi. Algoritmi nudijo različne razlage odločitve. Pri naivnem Bayesovem klasifikatorju odločitev prikažemo kot vsoto informacijskih prispevkov posameznih atributov (glej enačbo (7.1)). Razlagu vizualiziramo kot histogram informacijskih prispevkov posameznih atributov. Taka predstavitev omogoča uporabniku hiter vpogled v klasifikacijski postopek in pomaga pri pregledu atributov, ki odločilno vplivajo na končno klasifikacijo.

Možina [9] je razvil metodo za vizualizacijo naivnega Bayesovega klasifikatorja z nomogrami. Prirejeni naivni Bayesov klasifikator uporablja namesto verjetnosti razmerje verjetnosti (*odds*). Ta različica naivnega Bayesa omogoča elegantno razlagu odločitve z nomogramom, ki je ilustriran na sliki 6.11. Prikazan je izračun verjetnosti razreda ( $P(hhs=good)$ ) preko skale z vsoto točk (*sum*). Za vsak atribut so prikazane njegove vrednosti na skali točk (za ta primer od -100 do 30). Vsaka vrednost atributa prispeva določeno število točk v vsoto, ki enolično določa razmerje verjetnosti in s tem verjetnost razreda. Če je vrednost atributa neznana, je prispevek atributa 0 točk.

Pri primerjanju uspešnosti algoritmov se uporabljam različni testi značilnosti (signifikantnosti), opisani v razdelku 3.4. Za ugotavljanje značilnosti razlik uspešnosti lahko navedemo tabelo vrednosti statističnih testov med pari klasifikatorjev, kar je precej nepregledno. Primerjavo lahko vizualiziramo [4], kot je to prikazano na sliki 6.12. Primerjajo se štiri variante algoritma C4.5. Na vrhu je prikazana velikost mejne vrednosti razlike uspešnosti  $CD$  za Nemenyijev test iz enačbe (3.24). Variante, katerih uspešnosti se neznačilno razlikujejo, so

povezane z debelo črto. Iz slike je torej razvidno, da se uspešnost osnovne variante C4.5 značilno razlikuje od uspešnosti variant C4.5+m+cf ter C4.5+m, medtem ko se uspešnost variante C4.5+cf ne razlikuje značilno od nobene primerjane variante. Slika tudi nazorno prikazuje, pri katerih algoritmih se uspešnosti bolj oziroma manj razlikujejo.

## 6.8 Izbira podmnožice atributov

*Poslušnost se drži pravil, ljubezen pa ve, kdaj jih lahko prekrši.*

*Anthony de Mello*

Za učinkovito strojno učenje je pogosto potrebno iz celotne množice atributov izbrati ustrezeno (čim optimalnejšo) podmnožico atributov. Izbira podmnožice atributov (*feature subset selection*) je potrebna, kadar je vseh atributov preveč ali kadar množica atributov vsebuje nepotrebne (nepomembne, naključne) in/ali redundantne attribute, ki vplivajo na slabše rezultate učenja. Poiskati želimo podmnožico atributov, ki optimizira izbrani kriterij (npr. kompresivnost hipoteze, napovedno natančnost hipoteze itd.). Ker je vseh možnih podmnožic atributov preveč ( $2^n$ , če je  $n$  število vseh atributov), je potrebno, za izbiro čim bolj optimalne podmnožice atributov, uporabiti približne rešitve.

Najpreprostejša in najhitrejša je metoda filtriranja atributov. Z ocenitveno funkcijo (npr. MDL ali (R)ReliefF) ocenimo kvaliteto atributov in izberemo  $k$  najboljših atributov, kjer  $k$  izberemo vnaprej, ali pa določimo prag, ki odreže slabše ocenjene attribute.

Zanesljivejša vendar počasnejša je metoda notranje optimizacije (*wrapper*), kjer algoritmom strojnega učenja uporablja za optimizacijo notranje prečno preverjanje, t.j. prečno preverjanje brez testnih primerov. Učne primere razdelimo na začasne učne in začasne testne primere, zato da ocenimo optimizacijski kriterij. Postopek ponavljamo z različnimi nastavtvami parametrov učenja, dokler ne najdemo (lokalnega) optimuma. V našem primeru izbira parametrov pomeni izbiro podmnožice atributov.

Za preiskovanje prostora podmnožic atributov ponavadi uporabljamo kar požrešno iskanje, lahko pa izberemo kompleksnejšo metodo iskanja, če je število atributov majhno. Najpogosteje se uporablja iskanje naprej (*forward search*) in iskanje nazaj (*backward search*):

- iskanje naprej poteka tako, da začnemo s prazno množico atributov in v enem koraku dodamo bodisi naključno izbrani atribut (hitrejša varianta), če se vrednost optimizacijskega kriterija izboljša, bodisi dodamo atribut, ki maksimizira optimizacijski kriterij (počasnejša varianta);
- iskanje nazaj poteka tako, da začnemo s celotno množico atributov in v enem koraku odstranimo bodisi naključno izbrani atribut (hitrejša varianta), če se vrednost optimizacijskega kriterija izboljša, bodisi odstranimo atribut, ki z odstranitvijo maksimizira optimizacijski kriterij (počasnejša varianta);
- lahko pa uporabimo mešano iskanje (v enem koraku atribut dodamo ali zбриšemo) - v tem primeru je začetno stanje naključna podmnožica atributov.

Korak iskanja se ponavlja, dokler še lahko izboljšamo vrednost optimizacijskega kriterija.

## 6.9 Izpeljave in dokazi

### Pri diskretizaciji meja med intervali ne loči istorazrednih primerov

Preden odvajamo, ugotovimo, da je mera nečistoče  $\phi$  dejansko funkcija  $m_0 - 1$  argumentov, saj je verjetnost zadnjega razreda enolično določena z verjetnostmi ostalih razredov. Torej lahko zapišemo:

$$q(A) = \phi(P(r_1), \dots, P(r_{m_0-1})) - \sum_{j=1}^{m_0-1} P(V_j) \phi(P(r_1|V_j), \dots, P(r_{m_0-1}|V_j))$$

Naj obstaja v učni množici primerov, urejeni po vrednostih danega zveznega atributa, zaporedje  $Max$  primerov iz razreda  $r_{m_0}$ . Z  $x \in 0 \dots Max$  označimo možne meje med temi primeri. Začetna meja naj bo na začetku zaporedja ( $x=0$ ). Uporabili bomo naslednjo notacijo:

$n_{..} = n$  – število učnih primerov,

$n_k$  – število učnih primerov iz razreda  $r_k$ ,

$n_{.1}$  – število učnih primerov levo od meje  $x = 0$

$n_{k1}$  – število učnih primerov iz razreda  $r_k$  in levo od meje  $x = 0$

$n_{k2}$  – število učnih primerov iz razreda  $r_k$  in desno od meje  $x = 0$

$$p_k = n_k/n,$$

$$p_{.1} = (n_{.1} + x)/n,$$

$$p_{.2} = 1 - (n_{.1} + x)/n = (n - n_{.1} - x)/n,$$

$$p_{k|1} = n_{k1}/(n_{.1} + x),$$

$$p_{k|2} = n_{k2}/(n - n_{.1} - x).$$

Sedaj odvajamo:

$$\frac{\partial q}{\partial x} = -\frac{\phi(p_{1|1}, \dots, p_{m_0-1|1})}{n} + \frac{n_{.1} + x}{n(n_{.1} + x)^2} \sum_{k=1}^{m_0-1} n_{k1} \frac{\partial \phi(p_{1|1}, \dots, p_{m_0-1|1})}{\partial p_{k|1}} +$$

$$\frac{\phi(p_{1|2}, \dots, p_{m_0-1|2})}{n} - \frac{n - n_{.1} - x}{n(n - n_{.1} - x)^2} \sum_{k=1}^{m_0-1} n_{k2} \frac{\partial \phi(p_{1|2}, \dots, p_{m_0-1|2})}{\partial p_{k|2}}$$

$$\frac{\partial q}{\partial x} = -\frac{\phi(p_{1|1}, \dots, p_{m_0-1|1})}{n} + \frac{1}{n(n_{.1} + x)} \sum_{k=1}^{m_0-1} n_{k1} \frac{\partial \phi(p_{1|1}, \dots, p_{m_0-1|1})}{\partial p_{k|1}} +$$

$$\frac{\phi(p_{1|2}, \dots, p_{m_0-1|2})}{n} - \frac{1}{n(n - n_{.1} - x)} \sum_{k=1}^{m_0-1} n_{k2} \frac{\partial \phi(p_{1|2}, \dots, p_{m_0-1|2})}{\partial p_{k|2}}$$

$$\frac{\partial^2 q}{\partial x^2} = \frac{2 - 2}{n(n_{.1} + x)^2} \sum_{k=1}^{m_0-1} n_{k1} \frac{\partial \phi(p_{1|1}, \dots, p_{m_0-1|1})}{\partial p_{k|1}} -$$

$$\begin{aligned}
 & -\frac{1}{n(n_{.1}+x)^3} \sum_{k=1}^{m_0-1} n_{k1}^2 \frac{\partial^2 \phi(p_{1|1}, \dots, p_{m_0-1|1})}{\partial p_{k|1}^2} + \\
 & + \frac{2-2}{n(n-n_{.1}-x)^2} \sum_{k=1}^{m_0-1} n_{k2} \frac{\partial \phi(p_{1|2}, \dots, p_{m_0-1|2})}{\partial p_{k|2}} - \\
 & - \frac{1}{n(n-n_{.1}-x)^3} \sum_{k=1}^{m_0-1} n_{k2}^2 \frac{\partial^2 \phi(p_{1|2}, \dots, p_{m_0-1|2})}{\partial p_{k|2}^2}
 \end{aligned}$$

Ko poenostavimo, vidimo, da je drugi odvod povsod pozitiven, ker je funkcija  $\phi$  konkavna in je njen drugi odvod povsod negativen:

$$\begin{aligned}
 \frac{\partial^2 q}{\partial x^2} = & -\frac{1}{n(n_{.1}+x)^3} \sum_{k=1}^{m_0-1} n_{k1}^2 \frac{\partial^2 \phi(p_{1|1}, \dots, p_{m_0-1|1})}{\partial p_{k|1}^2} - \\
 & - \frac{1}{n(n-n_{.1}-x)^3} \sum_{k=1}^{m_0-1} n_{k2}^2 \frac{\partial^2 \phi(p_{1|2}, \dots, p_{m_0-1|2})}{\partial p_{k|2}^2}
 \end{aligned}$$

Torej je funkcija  $q$  konveksna:

$$\frac{\partial^2 q}{\partial x^2} > 0$$

in ima maksimum na eni od mejnih točk:  $x = 0$  ali  $x = Max.$   $\square$

## Literatura

- [1] M. Bevk. Analiza tekstur s strojnim učenjem. Magistrska naloga, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, 2003.
- [2] J. Brank, J. Leskovec. The download estimation task on KDD Cup 2003. *SIGKDD Explorations*, 5(2):160–162, 2003.
- [3] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.
- [4] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
- [5] D. Hand, H. Mannila, P. Smyth. *Principles of Data Mining*. MIT Press, 2001.
- [6] B. Julesz, E. N. Gilbert, L. A. Shepp, H. L. Frisch. Inability of humans to discriminate between visual textures that agree in second-order-statistics. *Perception*, 2:391–405, 1973.
- [7] I. Kononenko, M. Kukar. *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing, 2007. Chichester, UK.

- [8] K. Korotkov. *Aura and Consciousness*. State Editing & Publishing Unit "Kultura", 1998. St.Petersburg, Russia.
- [9] M. Možina, J. Demšar, M. Kattan, B. Zupan. Nomograms for visualization of naive Bayesian classifier. V *ECML-04*. Springer Verlag, 2004. Pisa, Italy.
- [10] J. A. Rushing, H. S. Ranagath, T. H. Hinke, S. J. Graves. Using association rules as texture features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, str. 845–858, 2001.
- [11] A. Sadikov. Računalniška vizualizacija, parametrizacija in analiza slik električne izpraznitve plinov. Magistrska naloga, Univerza v Ljubljani, 2002.
- [12] L. Sirovich, M. Kirby. A low-dimensional procedure for the characterisation of human faces. *Journal of the Optical Society of America*, str. 519–524, 1987.
- [13] R. Spence. *Information Visualization*. Addison-Wesley, 2001.
- [14] M. Turk, A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, str. 71–86, 1991.

## Poglavlje 7

# Simbolično učenje

*Kako nespametno je prizadevati si za posvetne stvari. Veliko modreje se je pred bogovi pokazati iskreno in zvesto v popolni preprostosti.*

*Mark Avreljij*

V tem poglavju so opisane metode za učenje odločitvenih dreves in pravil ter metode za učenje regresijskih dreves. Na koncu poglavja sta opisana naivni in delno naivni Bayesov klasifikator.

### 7.1 Učenje odločitvenih dreves

*Če sodiš ljudi, nimaš časa, da bi jih ljubil.*

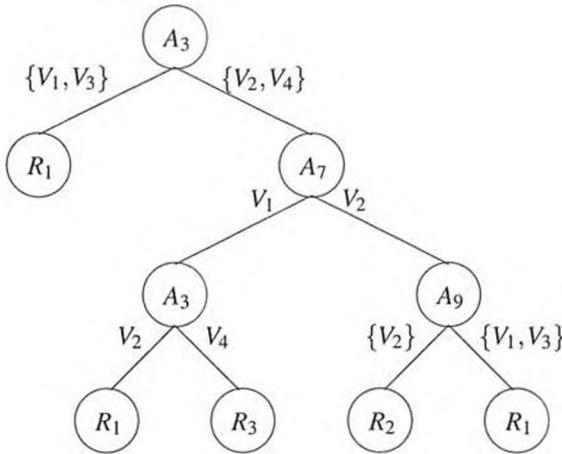
*Mati Tereza*

Pri gradnji odločitevenih dreves (decision trees) se uporablja atributna predstavitev učnih primerov z diskretnim razredom. Na kratko jo ponovimo. Vsak učni primer je opisan z vektorjem vrednosti atributov. Atributi so lahko zvezni ali diskretni in njihovo število je dano vnaprej. Atributna predstavitev je podrobnejše opisana na strani 110.

#### Gradnja in uporaba odločitvenega drevesa

Odločitveno drevo je sestavljeno iz notranjih vozlišč, ki ustrezajo atributom, vej, ki ustrezajo podmnožicam vrednosti atributov, in listov, ki ustrezajo razredom. Pot v drevesu od korena do lista ustreza odločitvenemu pravilu. Pri tem so pogoji (pari atribut – podmnožica vrednosti) konjunktivno povezani. Primer odločitvenega drevesa je podan na sliki 7.1.

Osnovni algoritem učenja odločitvenih dreves je sledeč [6]:



Slika 7.1: Primer odločitvenega drevesa.

Če je izpolnjen ustaviljni pogoj,

potem ustvari list, ki vključuje vse učne primere;

sicer

izberi "najboljši" atribut  $A_i$ ;

razdeli primere glede na vrednosti atributa  $A_i$ ;

za vsako vrednost  $V_j$  atributa  $A_i$  ponovi:

rekurzivno zgradi poddrevo z ustreznou podmnožico učnih primerov;

Pri tem je ustaviljni pogoj lahko:

- dovolj "čista" učna množica (npr. vsi ali večina primerov iz istega razreda),
- premalo učnih primerov za zanesljivo nadaljevanje gradnje drevesa,
- zmanjkalo je (dobrih) atributov.

Ključnega pomena je izbira "najboljšega atributa". Za izbiro atributa se najpogosteje uporabljajo mere: informacijski prispevek [14, 10, 3], razmerje informacijskega prispevka [15], Gini-indeks [1] in ReliefF [9], ki so opisane v razdelku 5.1.

Odločitveno drevo predstavlja klasifikacijsko funkcijo, ki je hkrati simbolični opis in povzetek zakonitosti v dani problemski domeni. Zato je drevo ponavadi zanimivo za strokovnjaka iz dane problemske domene, saj lahko iz drevesa razbere zakonitosti in strukturo.

Zgrajeno drevo uporabimo za klasifikacijo novih primerov. Od korena potujemo po ustreznih vejah do lista. List vsebuje informacijo o številu učnih primerov iz posameznih razredov.

Iz frekvenc učnih primerov ocenimo verjetnostno porazdelitev razredov. Pri tem lahko uporabimo  $m$ -oceno verjetnosti. Koristen podatek je tudi število učnih primerov v listu, ki kaže na zanesljivost ocene verjetnostne porazdelitve razredov.

Poseben primer je prazen list, ki ne ustreza nobenemu učnemu primeru in zato ne omogoča ocene verjetnosti posameznih razredov. V takem primeru ponavadi uporabimo naivni Bayesov klasifikator, ki upošteva samo atribut in njihove vrednosti, ki se nahajajo na poti od korena do lista. Verjetnostna porazdelitev po razredih se z naivnim Bayesovim klasifikatorjem izračuna že med gradnjo drevesa in je pri klasifikaciji fiksna.

### Gradnja binarnega odločitvenega drevesa

Ker je zanesljivost ocene kvalitete atributa odvisna od števila učnih primerov, se učna množica ne sme prehitro zmanjšati. Zato algoritmi gradijo binarna odločitvena drevesa. Algoritem se razširi še z binarizacijo atributov pred izbiro najboljšega atributa:

Če je izpolnjen ustaviti pogoj,

potem ustvari list, ki vključuje vse učne primere;  
sicer

generiraj vse binarne verzije vseh atributov;  
izberi "najboljši" binarni atribut  $A_i$ ;  
razdeli primere glede na (dve) vrednosti atributa  $A_i$ ;

**za vsako** od dveh vrednosti  $V_1$  in  $V_2$  ponovi:

rekurzivno zgradi poddrevo z ustrezno podmnožico učnih primerov;

Pri tem je vseh binarnih verzij diskretnega atributa z  $m$  vrednostmi  $2^m - 1$ . Tej kombinatorični eksploziji se izognemo s požrešnim algoritmom, ki lokalno optimizira kvaliteto binarne verzije atributa. Najprej izberemo najboljšo binarno verzijo, ki loči eno vrednost originalnega atributa od vseh ostalih vrednosti. Ponavljamo dodajanje ene vrednosti, dokler kvaliteta atributa narašča. Pri binarizaciji zveznega atributa upoštevamo vse možne meje na danem intervalu vrednosti.

Gradnja binarnega odločitvenega drevesa omogoča, da lahko uporabimo za ocenjevanje atributov tudi informacijski prispevek in Gini-indeks, ki precenjujeta večvrednostne atribute. Z binarizacijo dobimo manjša drevesa, ki imajo ponavadi tudi boljšo klasifikacijsko točnost. Primer na sliki 7.1 je binarno drevo. Iz slike se vidi, da lahko isti atribut ( $A_3$ ) nastopa večkrat na poti od korena do lista drevesa.

### Rezanje odločitvenega drevesa

Nižji nivoji drevesa so nezanesljivi, kajti vozliščem ustreza le majhno število učnih primerov. Ustaviti pogoj poskuša ustaviti gradnjo, ko le-ta postane nezanesljiva ali nepotrebna. Ker je nezanesljivost težko vnaprej oceniti, se pogosto gradnja nadaljuje ne glede na zanesljivost ocene atributa in se drevo *naknadno poreže* (*postpruning*). Osnovni algoritem naknadnega rezanja je sledeč [2]:

**Za vsa notranja vozlišča od spodaj navzgor ponavljam:**

- oceni povprečno pričakovano napako klasifikacije v poddrevesih;
- oceni pričakovano napako klasifikacije v trenutnem vozlišču;
- če je povprečna pričakovana napaka poddreves večja od pričakovane napake vozlišča, potem poreži poddrevesa in spremeni vozlišče v list.

Za ocenjevanje napake lahko uporabljamo  $m$ -oceno verjetnosti, ki je opisana v razdelku 3.1.

Druga možnost je uporaba principa MDL. Pri tem se namesto klasifikacijske napake ocenjuje dolžina kodiranja drevesa in porazdelitev razredov učnih primerov v listih, podobno kot smo opisali ocenjevanje atributov z oceno MDL v razdelku 5.1. Dolžina kodiranja porazdelitve v listih se oceni po formuli *Prior\_MDL*.

Za vsako notranje vozlišče primerjamo dolžino kodiranja porazdelitve učnih primerov v vozlišču, če poddrevesa porežemo in vozlišče postane list, z vsoto dolžin kodiranja poddreves povečano za 1 bit, ki pove, da poddreves nismo porezali. Če je dolžina kodiranja porazdelitve v vozlišču manjša od vsote dolžin kodiranj poddreves, potem poddrevesa porežemo. Dolžina kodiranja lista je enaka dolžini kodiranja porazdelitve razredov učnih primerov v listu. Vozlišča v drevesu pregledujemo od spodaj navzgor (od listov proti korenemu).

Prednost metode MDL pri naknadnem rezanju dreves pred  $m$ -oceno je v tem, da metoda MDL ne zahteva nastavitev nobenega parametra in skoraj optimalno poreže drevesa. Zato je metoda primerna za prvi približek optimalne teorije. Z eksperimentiranjem z različnimi vrednostmi parametra  $m$  lahko uporabnik po potrebi prilagodi velikost drevesa.

## Obravnavanje neznanih in poljubnih vrednosti

Učnim primerom včasih manjka vrednost enega ali več atributov. Med gradnjo odločitvenega drevesa za primer, ki za izbrani atribut nima znane vrednosti, izračunamo pogojno verjetnost, da primer ustreza eni ali drugi veji:

$$P(V_i|R) = \frac{N_{V_i,R} + mP(R)}{N_R + m}$$

kjer je  $P(R)$  apriorna verjetnost razreda  $R$ , ki mu primer pripada. Nato primer pošljemo v obe veji, v vsaki pa ga utežimo z ustrezno pogojno verjetnostjo. Primeri z zanimimi vrednostmi vseh atributov od vozlišča do korena imajo utež 1, primeri z neznano vrednostjo pa utež, ki je manjša od 1. Pri zveznih atributih sta  $V_1$  in  $V_2$  v zgornji formuli intervala pod in nad izbrano mejo.

Ta princip lahko posplošimo tudi na "poljubne" vrednosti atributov. Če namesto vrednosti nekaterih atributov nastopa "poljubna" vrednost (*don't care*), potem tak primer ob začetku gradnje utežimo s produktom števil vrednosti atributov, pri katerih ima poljubno vrednost. To je ekvivalentno razmnoževanju učnih primerov v toliko kopij, kolikor je možnih vrednosti za atribut s poljubno vrednostjo. To velja za diskretne attribute. Pri zveznih je potrebno hevristično določiti utež.

Če ima npr. učni primer poljubno vrednost binarnega atributa in poljubno vrednost atributa s tremi možnimi vrednostmi, je njegova utež na začetku gradnje enaka 6. Če se med

gradnjo izbere atribut s tremi vrednostmi, ima primer od takrat naprej utež 2. Če se med gradnjo izbereta oba atributa, ima primer od takrat naprej utež enako 1.

## 7.2 Učenje odločitvenih pravil

*Po končani vojni je problem z zmagovalcem. Misli, da je pravkar dokazal, da se vojna in nasilje splačata. Kdo ga bo sedaj naučil lekcije?*

*A.J. Muste, mirovni aktivist*

Vsako odločitveno drevo lahko spremenimo v množico pravil. Pot od korena do lista ustreza enemu odločitvenemu pravilu. Vsake množice odločitvenih pravil pa ni možno neposredno združiti v drevo. Zato je predstavitev znanja s pravili prožnejša od predstavitve z drevesi.

### Generiranje enega pravila

Pri gradnji enega pravila se za ocenjevanje kvalitete pravila zaradi lepih lastnosti uporablja predvsem *J*-ocena [17, 4]. Učenje poteka od splošnega k specifičnemu. Pravila lahko gradimo za vsak razred posebej, tako da so primeri iz tega razreda pozitivni, iz ostalih pa negativni, ali pa jih gradimo sproti za vse razrede. Na začetku je pravilo prazno in pokriva vse učne primere, tako pozitivne kot negativne, oziroma primere iz vseh razredov. Sklepni del takega pravila je distribucija po razredih, ki ustreza distribuciji učne množice. V enem koraku se pravilo specializira na vse možne načine tako, da se v pogojni del pravila konjunktivno doda pogoj *atribut = vrednost* za diskretne attribute, oziroma *atribut > vrednost* ali *atribut < vrednost* za zvezne attribute.

Zatem se s hevristično oceno (kot je npr. *J*-ocena) oceni najboljše kandidate glede na število pokritih primerov in distribucijo pokritih primerov po razredih. Če se uporablja iskanje v snopu, se obdrži več kandidatov (velikost snopa), če pa uporabljamo požrešni algoritem, obdržimo samo najboljšega (kar je enako snopu velikosti 1).

Postopek specializacije kandidatov ponavljamo, dokler ni izpolnjen ustavitevni pogoj (dovolj visoka točnost pravila ali premalo učnih primerov, ki jih pravilo pokriva). Izmed vseh kandidatov se izbere najboljši.

### Učenje množice pravil

Algoritmi za učenje pravil ponavadi uporabljajo *prekrivni algoritem (covering algorithm)* [5, 4] :

**Dokler** učna množica ni prazna in je možno zgenerirati dobro pravilo, **ponavljam**  
generiraj odločitveno pravilo;  
iz učne množice izloči primere, ki jih generirano pravilo pravilno klasificira;  
**konec zanke;**  
če nisi uspel zgenerirati dobrega pravila, **potem** zgeneriraj večinsko pravilo;

Večinsko pravilo, ki se doda na koncu, klasificira v večinski razred primere, ki jih ne pokriva nobeno drugo pravilo. Na ta način dobimo urejeno množico pravil, ki jih pri klasifikaciji uporabljamo v istem vrstnem redu, kot so bila generirana.

Druga možnost je, da gradimo redundantna pravila [17, 11], t.j. pravila, ki lahko pokrijejo iste učne primere. To dosežemo npr. z naključnim generiranjem pravil, ki jih zatem z optimizacijo popravimo. Množico redundantnih pravil uporabljamo pri klasifikaciji novih primerov tako, da odgovore pravil, ki pokrivajo primer, kombiniramo v končni odgovor. Odgovore lahko kombiniramo z glasovanjem, uteženim glasovanjem ali z naivnim Bayesovim klasifikatorjem.

**Glasovanje:** vsako pravilo glasuje za en razred. Primer klasificiramo v razred z največ glasovi.

**Uteženo glasovanje:** vsako pravilo prispeva za razred toliko točk, kolikor je vsota uteži učnih primerov iz razreda, ki jih pravilo pokriva. Primer klasificiramo v razred z največ točkami.

**Naivni Bayesov klasifikator:** vsako pravilo prispeva pogojno verjetnost vsakega razreda pri danem pravilu. Po naivnem Bayesovem klasifikatorju se izračuna pogojna verjetnost vsakega razreda pri vseh pravilih. Pri tem se predpostavi pogojna neodvisnost pravil pri danem razredu.

### 7.3 Učenje regresijskih dreves

*V časih, kot je tale, je koristno vedeti, da so bili vedno časi, kot je tale.*

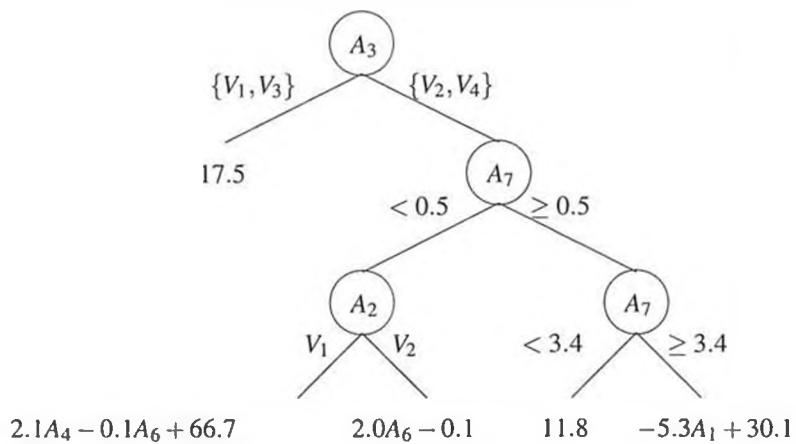
*Paul Harvey*

Pri gradnji regresijskih dreves (*regression trees*) se uporablja atributna predstavitev učnih primerov z zvezno odvisno spremenljivko. Vsak učni primer je opisan z vektorjem vrednosti atributov. Atributi so lahko zvezni ali diskretni in njihovo število je dano vnaprej.

#### Gradnja in uporaba regresijskega drevesa

Regresijsko drevo je sestavljeno iz notranjih vozlišč, ki ustrezajo atributom, vej, ki ustrezajo podmnožicam vrednosti atributov, in listov, ki ustrezajo funkcijam, ki preslikajo vektor vrednosti atributov v odvisno spremenljivko. Najpreprostejša funkcija v listih je konstanta, pogosto pa se uporablja linearна funkcija na podmnožici zveznih atributov. Pot v drevesu od korena do lista ustreza enemu pravilu. Pri tem so pogoji (pari atribut – podmnožica vrednosti), ki jih srečamo na poti, konjunktivno povezani. Primer regresijskega drevesa je podan na sliki 7.2.

Osnovni algoritem učenja regresijskih dreves je podoben algoritmu za gradnjo odločitvenih dreves:



Slika 7.2: Primer regresijskega drevesa.

Če je izpolnjen ustavilveni pogoj,

**potem**

ustvari list, ki vključuje učne primere;

generiraj funkcijo, ki modelira učne primere v listu;

**sicer**

izberi "najboljši" atribut  $A_i$ ;

razdeli primere glede na vrednosti atributa  $A_i$ ;

za vsako vrednost  $V_j$  atributa  $A_i$  ponovi:

rekurzivno zgradi poddrevo z ustrezno podmnožico učnih primerov;

Ustavilveni pogoj mora upoštevati tudi kvaliteto funkcije, ki modelira učne primere v listu. Model učnih primerov v listu je lahko:

**konstanta:** izračunamo povprečno vrednost odvisne spremenljivke vseh učnih primerov v listu;

**linearna funkcija:** napoved učnih primerov modeliramo z regresijsko premico (glej razdelek 8.2.);

**poljubna funkcija:** napoved učnih primerov modeliramo s poljubno funkcijo, pri čemer je pri iskanju ustrezne funkcije potrebno upoštevati njeno kompleksnost v odvisnosti od števila učnih primerov.

Kot pri odločitvenih drevesih, je tudi tu ključnega pomena izbira "najboljšega atributa". Za izbiro atributa se pri gradnji regresijskih dreves uporablja meri razlika variance [1] in regresijski ReliefF [16], ki sta opisani v razdelku 5.1. Uporabimo lahko tudi oceno kvalitete funkcije, s katero bi modelirali primere v naslednjih trenutnega vozlišča.

Podobno kot pri odločitvenih drevesih obravnavamo neznane in poljubne vrednosti tudi v regresijskih drevesih. Ker je napoved zvezna, pogojnih verjetnosti vrednosti atributa pri danem razredu ne moremo neposredno izračunati in si pomagamo z diskretizacijo razreda.

Tudi regresijsko drevo predstavlja funkcijo, ki je hkrati simboličen opis in povzetek zakonitosti v problemski domeni. Zato je drevo ponavadi zanimivo za strokovnjaka iz problemske domene, saj lahko iz njega razbere zakonitosti in strukturo.

Zgrajeno drevo uporabimo za določanje vrednosti odvisne spremenljivke novih primerov. Od korena potujemo po ustreznih vejah do lista. Vrednost odvisne spremenljivke za dani primer napovemo s funkcijo, ki se tam nahaja.

Zanesljivost ocene kvalitete atributa je odvisna od števila učnih primerov, zato ni dobro, da se učna množica prehitro razdeli na majhne podmnožice. Algoritmi pogosto gradijo binarna regresijska drevesa. Algoritem se v tem primeru, tako kot pri odločitvenih drevesih, razširi z binarizacijo atributov pred izbiro najboljšega atributa.

Gradnja binarnega regresijskega drevesa omogoča, da za ocenjevanje atributov uporabimo razliko variance, ki precenjuje večvrednostne attribute. Z binarizacijo dobimo manjša drevesa, ki imajo ponavadi boljšo točnost. Primer s slike 7.2 je binarno drevo.

## Rezanje regresijskega drevesa

Zaradi nezanesljivosti nižjih nivojev drevesa, kjer vozliščem ustreza majhno število učnih primerov, ustavitevi pogoji poskušajo ustaviti gradnjo, ko le-ta postane nezanesljiva ali nepotrebna.

Tako kot pri odločitvenih drevesih se tudi tu gradnja pogosto nadaljuje ne glede na zanesljivost ocene atributa in se drevo *naknadno poreže*. Osnovni algoritem naknadnega rezanja je enak kot pri odločitvenih drevesih. Za ocenjevanje pričakovane napake  $e$  v vozlišču lahko uporabimo prirejeno  $m$ -oceno:

$$e = \frac{n}{N+m} e_v + \frac{m}{N+m} e_k$$

kjer je  $N$  število primerov v vozlišču,  $e_v$  povprečna napaka modela (funkcije), ki bi ga dobili, če bi bilo vozlišče list, in  $e_k$  povprečna napaka istega modela na vseh učnih primerih. Primerjamo pričakovano napako v vozlišču, če postavimo list, in pričakovano napako, če pustimo poddrevesa. Če je pričakovana napaka poddreves večja, jih porežemo. To ponavljamo za vsa vozlišča v drevesu od spodaj navzgor vse do korena.

Druga možnost je uporaba principa MDL, podobno kot pri odločitvenih drevesih. Namente napake se ocenjuje dolžina kodiranja drevesa in dolžina kodiranja napake na učnih primerih v listih. Ker imamo opravka z realnimi števili, je ključnega pomena izbira načina kodiranja in parametrov, kot je natančnost števil, ki močno vplivajo na jakost rezanja drevesa.

## 7.4 Naivni in delno naivni Bayesov klasifikator

*Lahko rečeš, da sem sanjač, toda nisem edini!*

*John Lennon*

### Naivni Bayesov klasifikator

Naivni Bayesov klasifikator (*naive Bayesian classifier*) predpostavlja pogojno neodvisnost vrednosti različnih atributov pri danem razredu. Osnovno formulo naivnega Bayesovega klasifikatorja smo izpeljali v razdelku 4.3. Izpeljemo jo s pomočjo Bayesovega pravila:

$$P(r_k|V) = P(r_k) \prod_{i=1}^a \frac{P(r_k|v_i)}{P(r_k)}$$

Naloga učnega algoritma je s pomočjo učne množice podatkov aproksimirati apriorne verjetnosti razredov  $P(r_k), k = 1 \dots m_0$  in pogojne verjetnosti razredov  $r_k, k = 1 \dots m_0$  pri dani vrednosti  $v_i$  atributa  $A_i, i = 1 \dots a$ :  $P(r_k|v_i)$ . Za ocenjevanje apriornih verjetnosti se uporablja Laplaceov zakon zaporednosti:

$$P(r_k) = \frac{n_k + 1}{n + m_0}$$

kjer je  $n_k$  število učnih primerov iz razreda  $r_k$  in  $n$  število vseh učnih primerov. Za ocenjevanje pogojnih verjetnosti se uporablja  $m$ -ocena:

$$P(r_k|v_i) = \frac{n_{k,i} + mP(r_k)}{n_i + m}$$

kjer je  $n_{k,i}$  število učnih primerov iz razreda  $r_k$  in z vrednostjo  $i$ -tega atributa  $v_i$  ter  $n_i$  število vseh učnih primerov z vrednostjo  $i$ -tega atributa  $v_i$ .

Pri učenju (izračunu pogojnih verjetnosti) lahko primere, ki nimajo vrednosti za dani atribut, preprosto izpustimo iz formule. Naivni Bayesov klasifikator pri klasifikaciji uporabi vse atrbute s podanimi vrednostmi. Atributi, katerih vrednosti za dani primer ne poznamo, preprosto ignoriramo.

Naivni Bayesov klasifikator uporabljamo neposredno pri diskretnih atrributih, pri zveznih atrributih pa je potrebno atrribut najprej diskretizirati. Pri tem najbolje deluje mehka diskretizacija zveznih atrributov. Pri mehkih mejah primer ne pripada samo enemu intervalu, ampak več intervalom, vsakemu z določeno verjetnostjo. S tem se ohranja množica primerov, ohranja se informacija o urejenosti intervalov, s parametrom "mehkosti" pa kontroliramo razdrobljenost atrributa na vrednosti.

V praksi je naivni Bayesov klasifikator kljub naivnosti dostikrat uspešen. Izkaže se, da je pogojna neodvisnost pogosto sprejemljiva predpostavka. Oglejmo si dva primera.

- Simptomi pri pacientih so odvisni od bolezni (diagnoze, razreda) in so pri dani diagnozi med seboj relativno neodvisni. Zato je naivni Bayesov klasifikator v mnogih medicinskih diagnostičnih problemih med najuspešnejšimi [8].

- Umetni problem, ki se pogosto uporablja za testiranje učnih algoritmov, je problem prikazovalnika cifer LED. Vsaka cifra je sestavljena iz podmnožice sedmih segmentov. Pri tem šum podaja verjetnost okvare na enem segmentu. Naloga algoritma je pri dani podmnožici pričlanjanih segmentov ugotoviti cifro. Ker je okvara enega segmenta neodvisna od okvar drugih, je v tem problemu vrednost atributa (pričlan ali ugasnjen segment) pri danem razredu neodvisna od vrednosti ostalih atributov. Zato v tem problemu naivni Bayesov klasifikator dosega optimalno klasifikacijsko točnost.

Drugi razlog uspešnosti je, da je ocenjevanje verjetnosti relativno zanesljivo in zato ne pride do prevelikega prilagajanja učni množici. Tretji razlog pa je, da tudi kadar predpostavka o pogojni neodvisnosti ni izpolnjena v celoti, ima naivni Bayes še vedno "rezervo", namreč razponi ocen verjetnosti med najbolj verjetnim in ostalimi razredi so ponavadi dovolj veliki, da napaka zaradi predpostavke o neodvisnosti ne uspe "pokvariti" vrstnega reda verjetnosti razredov.

Zato se naivni Bayesov klasifikator obnaša dobro tudi na problemih, kjer predpostavka o pogojni neodvisnosti atributov ne drži popolnoma. Kjer pa obstajajo močne odvisnosti med atributi, naivni Bayesov klasifikator odpove. V tem primeru si lahko pomagamo z delno naivnim Bayesovim klasifikatorjem.

### Razlaga odločitev naivnega Bayesovega klasifikatorja

Z logaritmiranjem osnovne formule naivnega Bayesovega klasifikatorja dobimo:

$$-\log_2 P(r_k|V) = -\log_2 P(r_k) - \sum_{i=1}^d (\log_2 P(r_k|v_i) - \log_2 P(r_k)) \quad (7.1)$$

Negativni logaritem verjetnosti dogodka interpretiramo kot količino informacije, ki je potrebna, da izvemo, da se je dogodek zgodil. Faktor  $\log_2 P(r_k|v_i) - \log_2 P(r_k)$  interpretiramo kot razliko med apriorno in apostериорно potrebnou količino informacije, da zvemo, da primer pripada razredu  $r_k$ . Z drugimi besedami, ta faktor predstavlja informacijski prispevek vrednosti  $v_i$  i-tega atributa za razred  $r_k$ .

Celotno formulo interpretiramo kot razliko med apriorno količino informacije, ki je potrebna, da primer razvrstimo v razred  $r_k$  ( $-\log_2 P(r_k)$ ), in vsoto informacijskih prispevkov posameznih atributov. Torej odločitve naivnega Bayesovega klasifikatorja obrazložimo z vso-tami informacijskih prispevkov atributov za posamezne razrede, če je informacijski prispevek pozitiven, ozziroma proti posameznim razredom, če je informacijski prispevek negativen. Taka razlaga odločitve se v mnogih domenah izkaže za razumljivo strokovnjakom [8].

### Delno naivni Bayesov klasifikator

Delno naivni Bayesov klasifikator (*semi-naive Bayesian classifier*) [7] skuša poiskati močne odvisnosti med vrednostmi različnih atributov in jih ustrezno upoštevati. Poleg spodaj opisanega algoritma obstaja več različic delno naivnega Bayesovega klasifikatorja, glej npr. pregledni članek [19].

A	B	C
1	1	0
1	0	1
0	1	1
0	0	0

Tabela 7.1: Problem XOR.

Po definiciji sta dogodka  $A$  in  $B$  neodvisna, če velja:

$$P(AB) = P(A)P(B) \quad (7.2)$$

Dogodka sta tem bolj odvisna, čim večja je razlika med  $P(AB)$  in produktom  $P(A)P(B)$ . V ekstremnem primeru imamo  $A = B$ , kjer je  $P(AB) = P(A) = P(B)$  ali  $A = \bar{B}$ , kjer je  $P(AB) = 0$ . Ker nas zanima odvisnost dogodkov glede na razred  $C$ , je za nas zanimiva tudi pogojna odvisnost. Dogodka  $A$  in  $B$  sta neodvisna glede na razred  $C$ , če velja:

$$P(AB|C) = P(A|C)P(B|C) \quad (7.3)$$

Dogodka sta tem bolj odvisna, čim večja je razlika med  $P(AB|C)$  in produktom  $P(A|C)P(B|C)$ . V ekstremnem primeru imamo  $C = A \vee B = (A \neq B)$  ali  $C = (A = B)$ , kjer je  $P(AB|C) = 0$  ali  $P(AB|C) = P(A|C) = P(B|C)$ .

XOR (ekskluzivni "ali", glej tabelo 7.1) je klasičen primer nelinearnega problema, ki ga ne moreta rešiti niti enonivojski perceptron niti naivni Bayesov klasifikator. Z večnivojsko nevronske mrežo in posplošenim pravilom delta je XOR mogoče rešiti. Mreža pri tem potrebuje čez 500 prehodov čez učno množico in topologija mreže mora biti ustreznata.

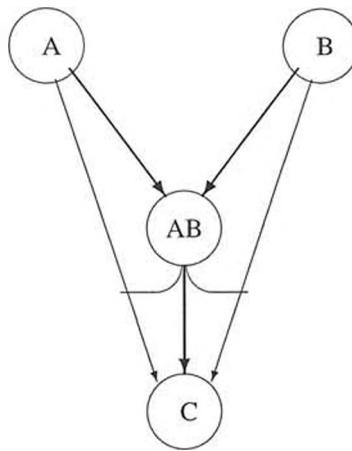
Problem XOR lahko rešimo na tri načine:

- učne primere si zapomnimo: ta rešitev je primerna samo, če vnaprej vemo, da v podatkih obstaja odvisnost XOR in je uporabna samo za eksaktne domene;
- razbijemo razrede na podrazrede: s tem implicitno vpeljemo disjunkcijo; ta rešitev ima podobne pomanjkljivosti kot prejšnja;
- združimo attribute; v primeru večnivojske nevronske mreže nevron iz skritega nivoja predstavlja nov koncept, definiran z nevroni iz predhodnega nivoja.

Za naivni Bayesov klasifikator je najbolj primerno združevanje vrednosti atributov. Če se ugotovi, da sta vpliva dveh vrednosti atributov  $A$  in  $B$  na razred  $C$  med seboj odvisna, lahko ti dve vrednosti združimo (glej sliko 7.3).

### Združevanje vrednosti atributov

Sestavimo algoritem, ki bo ugotavljal odvisnosti med vplivi vrednosti atributov na razrede. Naloga je poiskati *kompromis med nenaivnostjo in zanesljivostjo aproksimacije verjetnosti*.



Slika 7.3: Združevanje dveh odvisnih vrednosti.

Pri računanju verjetnosti razreda  $C$  vplivata vrednosti  $A$  in  $B$  s faktorjema:

$$\frac{P(C|A)}{P(C)} \cdot \frac{P(C|B)}{P(C)} \quad (7.4)$$

če ju obravnavamo neodvisno, in s faktorjem

$$\frac{P(C|AB)}{P(C)} \quad (7.5)$$

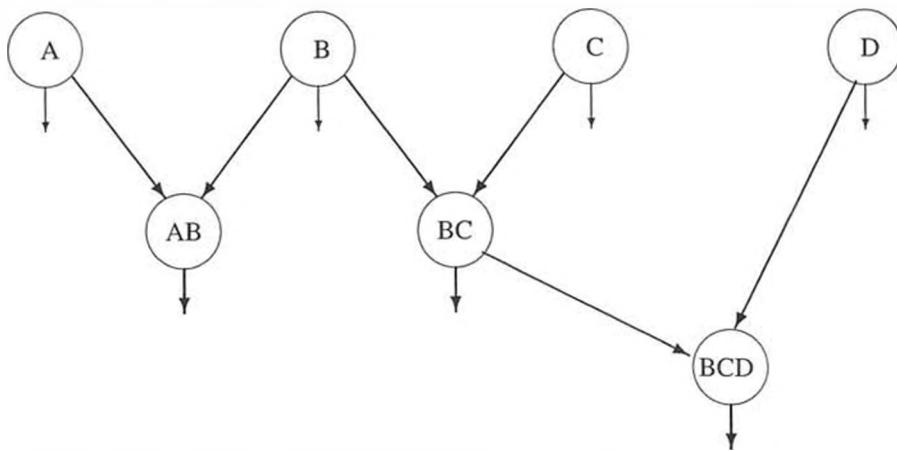
če ju združimo. Za združitev vrednosti  $A$  in  $B$  v vrednost  $AB$  sta potrebna dva pogoja: vrednosti izrazov (7.4) in (7.5) se morata dovolj razlikovati, hkrati pa mora biti aproksimacija verjetnosti  $P(C|AB)$  dovolj zanesljiva. Za oceno smiselnosti združitve dveh vrednosti bomo uporabili izrek Čebiševa [18], ki navzdol omejuje verjetnost, da se relativna frekvenca  $f$  po  $n$  narejenih poskusih razlikuje od dejanske apriorne verjetnosti  $p$  za manj kot  $\varepsilon$ :

$$P(|f - p| \leq \varepsilon) > 1 - \frac{p(1-p)}{\varepsilon^2 n} \quad (7.6)$$

Spodnja meja verjetnosti je sorazmerna številu narejenih poskusov  $n$  in kvadratu  $\varepsilon$ , ki je ocenjevana razlika med  $f$  in  $p$ . V našem primeru nas zanima zanesljivost aproksimacije

$$P(C|AB) \approx \frac{n_{CAB}}{n_{AB}}$$

Število poskusov  $n$  iz enačbe (7.6) je enako številu primerov, ko sta se hkrati zgodila dogodka  $A$  in  $B$ , to je  $n_{AB}$ . Ker apriorne verjetnosti  $p$  ne poznamo, vzamemo za aproksimacijo desne strani neenačbe najslabši primer, to je  $p = 0.5$  (takrat ima produkt  $p(1-p)$  največjo vrednost). Preostane nam še, da določimo vrednost  $\varepsilon$ .



Slika 7.4: Primer delno naivnega Bayesovega klasifikatorja v obliki večnivojske Bayesove nevronske mreže.

Ker nas poleg zanesljivosti aproksimacije zanima tudi, če je aproksimacija izraza (7.5) dovolj različna od vrednosti izraza (7.4), je za  $\epsilon$  smiselna vrednost, ki je proporcionalna tej razliki. Ker bo nova vrednost  $AB$  vplivala na vse razrede  $C$ , vzamemo za  $\epsilon$  povprečno razliko med izrazoma (7.4) in (7.5) po vseh razredih:

$$\epsilon = \sum_C P(C) \left| P(C|AB) - \frac{P(C|A)P(C|B)}{P(C)} \right| \quad (7.7)$$

Potrebno je izbrati še prag  $\tau$  za verjetnost (7.6), nad katerim je smiselno združiti dve vrednosti. V praksi se giblje vrednost tega praga med 0.5 in 1.0. Pravilo za združitev dveh vrednosti  $A$  in  $B$  se torej glasi: združi  $A$  in  $B$ , če je verjetnost večja ali enaka  $\tau$ , da se bo dejanski (neznan) prispevek nove vrednosti  $AB$  v povprečju po vseh razredih razlikoval od uporabljenega prispevka za manj, kot je razlika med uporabljenim prispevkom in prispevkom  $A$  in  $B$  brez združitve:

$$1 - \frac{1}{4\epsilon^2 n_{AB}} \geq \tau$$

Vrednosti različnih atributov lahko združujemo, dokler zanesljivost aproksimacije verjetnosti ne postane premajhna. Na ta način dobimo večnivojski Bayesov klasifikator, kot ga prikazuje slika 7.4.

### Poskusi v medicinski diagnostiki

Uporabnost naivnega in delno naivnega Bayesovega klasifikatorja pokažemo pri reševanju štirih medicinskih diagnostičnih problemov: lokalizacija primarnega tumorja, prognostika ponovitve raka na dojki, diagnostika obolenj ščitnice in diagnostika revmatoloških obolenj. Podatke smo dobili iz Univerzitetnega kliničnega centra v Ljubljani. Podatki predstavljajo

domena	prim.tumor	rak na dojki	ščitnica	revmat.
št. primerov	339	288	884	355
št. razredov	22	2	4	6
št. atributov	17	10	15	32
št. vred./atribut	2.2	2.7	9.1	9.1
št. manjk.pod/prim.	0.7	0.1	2.7	0.0
večinski razred	25%	80%	56%	66%
entropija razr. (bit)	3.64	0.72	1.59	1.70
točn. zdravnikov	42%	64%	64%	56%

Tabela 7.2: Opis podatkov za štiri medicinske diagnostične probleme.

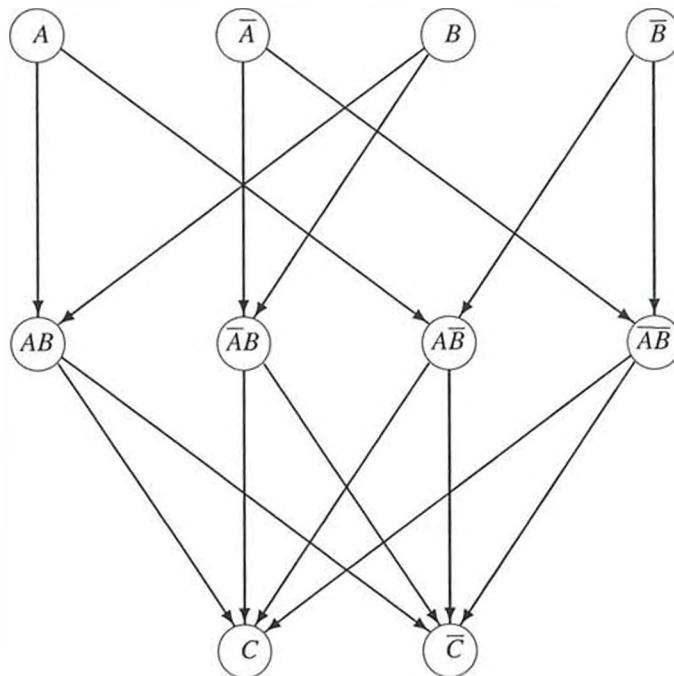
opise pacientov, ki so se zdravili na Onkološkem inštitutu v Ljubljani, v Kliniki za nuklearno medicino v Ljubljani oziroma v Revmatološki kliniki Bolnice Petra Držaja v Ljubljani.

Vsek pacient je opisan z množico atributov. Vsak atribut lahko zavzema določeno število vrednosti. Za vsakega pacienta je znana tudi končna diagnoza. Naloga mreže je naučiti se diagnosticiranja novih pacientov na osnovi učnih primerov, to je pacientov z znanimi dia-gnozami. Osnovne značilnosti štirih množic podatkov so podane v tabeli 7.2. Diagnostična pravilnost zdravnikov specialistov je povprečje pravilnosti štirih specialistov iz dane domene na naključno izbrani testni množici pacientov. Pravilnost diagnosticiranja zdravnikov je bila testirana v Univerzitetnem kliničnem centru v Ljubljani. Pravilnost zdravnikov skupaj s številom razredov in entropijo grobo ocenjuje težo problema. Očitno je problem lokacije primarnega tumorja najtežji, saj je možnih 22 razredov, entropija pa je 3.64 bita. Najlažji problem je prognostika raka na dojki, kjer sta dva razreda in entropija 0.72.

Število atributov približno pove, kako natančno so opisani primeri. Povprečno število vrednosti na atribut po eni strani govori o natančnosti opisa, po drugi strani pa skupaj s številom učnih primerov in številom razredov o zanesljivosti relativnih frekvenc, dobljenih iz učnih primerov. Povprečno število manjkajočih podatkov na en učni primer govori o (ne)popolnosti podatkov, ki so bili na voljo. Najbolj nepopolni podatki so v domeni diagnosticiranja obojenj ščitnice, kjer pri enem primeru v povprečju manjkajo vrednosti treh atributov. Najbolj popolni podatki so v domeni revmatologije, kjer od 355 primerov samo pri štirih manjka vrednost enega atributa.

Delež primerov iz večinskega razreda pomeni točnost klasifikatorja, ki klasificira vsak primer v večinski razred. Tak preprost klasifikator bi močno presegel zdravnike v revmatologiji (za 10%) in v prognostiki raka na dojki (za 16%). To kaže na šibko informativnost atributov, pa tudi na to, da klasifikacijska točnost ni zadostni dober način merjenja uspešnosti klasifikacije. Zato smo merili tudi informacijsko vsebino odgovorov klasifikatorjev. To merilo iznica vpliv apriornih verjetnosti razredov in obravnava tudi nepopolne in neeksaktne odgovore.

V poskusih smo uporabljali izčrpno preiskovanje prostora parov vrednosti atributov. Število prehodov preko učne množice je približno enako številu vrednosti atributov. Algoritem je sledeč:



Slika 7.5: XOR, rešen z delno naivnim Bayesovim klasifikatorjem.

nauči naivni Bayesov klasifikator nad celotno učno množico;

**za vsako vrednost  $A$  vsakega atributa ponavljam**

**za vsako drugo vrednost  $B$  drugih atributov ponavljam**

dodaj novo vrednost  $AB$ ;

za novo vrednost izračunaj verjetnosti;

zavri novo vrednost, če je  $\frac{1}{4\varepsilon^2 n_{AB}} > 0.5$

Problem XOR, podan v tabeli 7.1, zgornji algoritem reši tako, da združi štiri pare vrednosti, kot jih prikazuje slika 7.5. Zaradi zanesljivosti aproksimacij verjetnosti potrebujemo vsaj osem učnih primerov, ki jih dobimo tako, da učno množico podvojimo.

V poskusih smo uporabljali  $m$ -oceno verjetnosti. Rezultati eksperimentov z zgoraj opisanim algoritmom v štirih medicinskih domenah so podani v tabeli 7.3. V tabeli so rezultati delno naivnega Bayesa primerjeni z naivnim Bayesom, s sistemom za induktivno gradnjo odločitvenih dreves Asistent [3] in zdravniki specialisti. V tabeli je poleg klasifikacijske pravilnosti in informacijske vsebine odgovorov podano tudi povprečno število dodanih združenih vrednosti.

Iz tabele je razvidno, da je v diagnostiki primarnega tumorja in prognostiki ponovitve raka na dojni združevanje vrednosti atributov nepotrebno, kar se sklada z mnenjem zdravnikov, da

	primarni tumor (%)	rak na dojki (%)	ščitnica (%)	revmato- logija (%)
zdravniki	42	1.22	64	0.05
Asistent	44	1.38	77	0.07
naivni Bayes	51	1.58	79	0.18
delno naivni B.	51	1.58	79	0.18
št. združenih vr.	0.4		1.5	32.3
				17.9

Tabela 7.3: Rezultati delno naivnega Bayesovega klasifikatorja v štirih medicinskih domenah, primerjani z ostalimi klasifikatorji.

so atributi med seboj neodvisni. V diagnostiki ščitnice in v revmatologiji z združevanjem vrednosti dobimo nekoliko boljše rezultate kot z naivnim Bayesom. Sistem Asistent dosega boljše rezultate pri diagnostiki ščitnice zaradi ustreznješega obravnavanja zveznih atributov, ki so za (delno) naivni Bayesov klasifikator vnaprej diskretizirani. Sistem Asistent je v tej domeni dosegel boljšo diagnostično točnost, ker med gradnjo drevesa določi meje zveznim atributom in učne množice ne razdrobi preveč zaradi zanesljivejše aproksimacije verjetnosti.

Algoritem za učenje delno naivnega Bayesovega klasifikatorja skuša poiskati kompromis med "nenainvostjo" in zanesljivostjo aproksimiranja verjetnosti. Čim večja je "nenainvost," tem slabša je zanesljivost aproksimiranja verjetnosti in obratno. S spremenjanjem praga verjetnosti zanesljive aproksimacije (ki je bil v naših poskusih postavljen na 0.5) usmerjamo algoritom k manjši zanesljivosti aproksimacij verjetnosti in manjši naivnosti v *eksaktih domenah* oziroma k bolj zanesljivi aproksimaciji verjetnosti in večji naivnosti v *mehkih domenah*.

Odločitvena drevesa so zaradi nenaivnosti in nezanesljive aproksimacije verjetnosti bolj primerna za eksakte domene, medtem ko je naivni Bayes zaradi zanesljivejše aproksimacije verjetnosti primeren za mehke domene, v katerih je atributi definirali strokovnjak. Človek tipično definira relativno neodvisne atributi zaradi lažjega (linearnega) razmišljanja. Zato je predpostavka o neodvisnosti atributov v takih domenah sprejemljiva.

Prednost delno naivnega Bayesovega klasifikatorja pred odločitvenim drevesom je tudi v obravnavanju manjkajočih podatkov. Če nimamo podatka za atribut v drevesu, postane klasifikacija v precejšnji meri nezanesljiva. Delno naivni Bayes atributa brez vrednosti preprosto ne upošteva pri klasifikaciji. Pri klasifikaciji upošteva vse razpoložljive atributе, medtem ko v odločitvenem drevesu tipično nastopa le manjši del razpoložljivih atributov. Zdravniki so mnenja, da pravila v odločitvenih drevesih ponavadi vsebujejo premalo atributov, preskromno opisujejo pacienta in je zaradi tega diagnosticiranje nezanesljivo [13].

Naivni Bayesov klasifikator lahko implementiramo tudi z usmerjeno ali večsmerno *enonivojsko* Bayesovo nevronsko mrežo (BNM) [12]. Delno naivni Bayesov klasifikator lahko implementiramo z usmerjeno *večnivojsko* BNM. Usmerjena BNM je dejansko osnovni naivni Bayesov klasifikator v obliki usmerjenega grafa. Vhodni nevroni predstavljajo posame-

zne vrednosti atributov, izhodni nevroni pa razrede. Na vhodu so podane vrednosti vhodnih nevronov (t.j. atributov) in mreža v enem koraku izračuna aktivacijske nivoje izhodnih nevronov.

## Literatura

- [1] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone. *Classification and Regression Trees*. Wadsforth International Group, 1984.
- [2] B. Cestnik. *Ocenjevanje verjetnosti v avtomatskem učenju*. Doktorska disertacija, Univerza v Ljubljani, Fakulteta za elektrotehniko in računalništvo, Ljubljana, 1991.
- [3] B. Cestnik, I. Kononenko, I. Bratko. Assistant 86: A knowledge elicitation tool for sophisticated users. V I.Bratko, N. Lavrač, (ur.), *Progress in Machine Learning*. Sigma Press, Wilmslow, England, 1987.
- [4] P. Clark, T. Niblett. Induction in noisy domains. V I. Bratko, N. Lavrač, (ur.), *Progress in Machine learning*. Sigma Press, Wilmslow, England, 1987.
- [5] P. Clark, T. Niblett. Learning if then rules in noisy domains. V B. Phelps, (ur.), *Interactions in Artificial Intelligence and Statistical Methods*. Technical Press, Hampshire, England, 1987.
- [6] E. Hunt, J. Martin, P. Stone. *Experiments in Induction*. Academic Press, New York, 1966.
- [7] I. Kononenko. Semi-naive Bayesian classifier. V Y. Kodratoff, (ur.), *European Working Session on Learning'91*, str. 206–219. Springer-Verlag, 1991. Porto, March 4–6.
- [8] I. Kononenko. Inductive and Bayesian learning in medical diagnosis. *Applied Artificial Intelligence*, 7:317–337, 1993.
- [9] I. Kononenko. *Logično programiranje in drugi principi programskih jezikov*. Fakulteta za računalništvo in informatiko, Ljubljana, 1997.
- [10] I. Kononenko, I. Bratko, E. Roškar. Experiments in automatic learning of medical diagnostic rules. V *Proc. International School for the Synthesis of Expert's Knowledge Workshop*, Bled, Slovenia, 1984.
- [11] I. Kononenko, M. Kovačič. Stochastic generation of multiple rules. V *Proc. Machine Learning Conf.*, Aberdeen, Scotland, 1992.
- [12] I. Kononenko, M. Kukar. *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing, 2007. Chichester, UK.
- [13] V. Pirnat, I. Kononenko, T. Janc, I. Bratko. Medical estimation of automatically induced decision rules. V *2nd Europ. Conf. on Artificial Intelligence in Medicine*, str. 24–36, 1989. City University, London, August 29–31.

- [14] J. R. Quinlan. Discovering rules from large collections of examples. V D. Michie, (ur.), *Expert Systems in the Microelectronic Age*. Edinburgh University Press, 1979.
- [15] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [16] M. Robnik-Šikonja, I. Kononenko. An adaptation of Relief for attribute estimation in regression. V *Proc. Int. Conf. on Machine Learning ICML-97*, str. 296–304, Nashville, 1997.
- [17] P. Smyth, R. M. Goodman. Rule induction using information theory. V G. Piatetsky-Shapiro, W. Frawley, (ur.), *Knowledge Discovery in Databases*. MIT Press, 1990.
- [18] A. Vadnal. *Elementarni uvod v verjetnostni račun*. Državna založba Slovenije, Ljubljana, 1979.
- [19] F. Zheng, G.I. Webb. A comparative study of semi-naive Bayes methods in classification learning. V *Proceedings of the 4th Australasian Data Mining Workshop (AusDM05)*, str. 141–156, 2005.

## Poglavlje 8

# Numerične metode

*Razum lahko odgovori na vprašanje, toda domišljija ga mora postaviti.*

*Ralph Gerard*

V tem poglavju so opisane nekatere osnovne numerične metode strojnega učenja: metode najbližjih sosedov, linearna regresija in metode podpornih vektorjev.

### 8.1 Metode najbližjih sosedov

*Morali bi delati dobro tako spontano, kot konj teče ali čebela dela med, ali trta rodi grozdje leto za letom, ne da bi razmišljala o grozdju, ki ga je rodila.*

*Mark Avrelij*

Učenje z metodami najbližjih sosedov pomeni shranitev vseh ali pa podmnožice učnih primerov. Ko je potrebno klasificirati novi primer, poiščemo podmnožico *podobnih* učnih primerov in jih uporabimo za napoved razreda novega primera. Ker učenja pri teh metodah skorajda ni, pravimo tej vrsti učenja tudi *leno učenje* (*lazy learning*). Glavnina procesiranja je potrebna pri klasifikaciji novega primera in zato je časovna zahtevnost klasifikacije precej večja kot pri drugih metodah učenja.

Metode najbližjih sosedov se uporablajo za reševanje klasifikacijskih in regresijskih problemov. V tem razdelku so opisane navadna metoda *k-najbližjih sosedov*, metoda z razdaljo uteženih *k-najbližjih sosedov* ter lokalno utežena regresija.

#### *k-najbližjih sosedov*

Pri najpreprostejši različici algoritma *k-najbližjih sosedov* ali na kratko *k-NN* (*k-nearest neighbors*) shranimo vse učne primere. Ko želimo napovedati razred  $r_x$  novemu primeru  $u_x$ ,

poščemo med učnimi primeri  $k$  najbližjih  $u_1, \dots, u_k$  in pri klasifikaciji napovemo večinski razred, t.j. razred, ki mu pripada največ najbližjih sosedov:

$$r_x = \arg \max_{r \in \{V_1, \dots, V_{m_0}\}} \sum_{i=1}^k \delta(r, r^{(i)})$$

kjer je

$$\delta(a, b) = \begin{cases} 1, & a = b \\ 0, & a \neq b \end{cases}$$

Pri regresiji napovemo povprečno vrednost odvisne spremenljivke za  $k$  najbližjih sosedov:

$$r_x = \frac{1}{k} \sum_{i=1}^k r^{(i)}$$

$k$  je parameter, ki ga običajno nastavimo na neko liho število, npr. 1, 7, 15 ali 31 (z lihim številom se izognemo neodločeni klasifikaciji v dvorazrednih problemih). Če v učnih podatkih ni napak, se bo najbolje obnesel algoritem 1-NN. Če so v učnih primerih napake, s povečevanjem parametra  $k$  povprečimo napovedi več bližnjih primerov in s tem zmanjšamo verjetnost, da je vseh  $k$  učnih primerov napačnih. Po drugi strani pa z večanjem števila  $k$  povečujemo možnost, da h klasifikaciji prispevajo tudi učni primeri, ki niso dovolj podobni novemu primeru. Zato je potrebno za vsak problem posebej eksperimentalno določiti optimalni  $k$ .

Velja poudariti, da parameter  $k$  ne določa velikosti okolice novega primera, znotraj katere izbiramo učne primere, ampak se okolica dinamično spreminja v odvisnosti od gostote učnih primerov v podprostoru primerov. S fiksno velikostjo okolice bi v nekaterih delih prostora dobili preveč bližnjih sosedov, v drugih delih prostora pa nobenega. Ker gostota učnih primerov služi za oceno gostote verjetnosti v prostoru primerov, s parametrom  $k$  algoritem  $k$ -NN elegantno reši problem gostejših oziroma redkejših delov prostora.

Algoritem  $k$ -NN je občutljiv na izbrano metriko pri računanju razdalj med novim primerom in učnimi primeri. Ponavadi se uporablja evklidska razdalja. Vsi zvezni atributi se normalizirajo na interval  $[0, 1]$ . Razdalja med dvema vrednostima je enaka absolutni razliki med njima. Za diskretne atribute je razdalja med različnima vrednostima 1, med enakima vrednostima pa 0. Torej je razdalja med dvema primeroma podana z:

$$D(u_l, u_j) = \sqrt{\sum_{i=1}^a d(v^{(i,l)}, v^{(i,j)})^2}$$

kjer za zvezni atribut  $A_i$  velja:

$$d(v^{(i,l)}, v^{(i,j)}) = |v^{(i,l)} - v^{(i,j)}|$$

in za diskretnega:

$$d(v^{(i,l)}, v^{(i,j)}) = \begin{cases} 0, & v^{(i,l)} = v^{(i,j)} \\ 1, & v^{(i,l)} \neq v^{(i,j)} \end{cases}$$

Pogosto se uporablja uteževanje vpliva atributov na razdaljo med dvema primeroma tako, da pomembnejši atributi bolj vplivajo na razdaljo. Atribute ocenimo z mero za ocenjevanje pomembnosti atributov  $q$ . Razdalja med dvema primeroma se izračuna po formuli:

$$D(u_l, u_j) = \sqrt{\sum_{i=1}^a q(A_i) d(v^{(i,l)}, v^{(i,j)})^2}$$

### Z razdaljo uteženih $k$ -najbližjih sosedov

Bolj robustna različica algoritma  $k$ -NN uporablja uteževanje vpliva učnih primerov na napoved. Vpliv utežujemo z razdaljo učnih primerov od novega primera. Pri tem lahko razdalja vpliva linearne, kvadratne, eksponentne itd. Funkciji vpliva učnega primera glede na razdaljo pravimo tudi *jedrna funkcija* (*kernel function*). Če izberemo kvadratni vpliv razdalje, dobimo pri klasifikaciji:

$$r_x = \arg \max_{r \in \{v_1, \dots, v_{m_0}\}} \sum_{i=1}^k \frac{\delta(r, r^{(i)})}{D(u_x, u_i)^2}$$

in pri regresiji:

$$r_x = \frac{\sum_{i=1}^k \left[ r^{(i)} / D(u_x, u_i)^2 \right]}{\sum_{i=1}^k [1 / D(u_x, u_i)^2]} \quad (8.1)$$

Če uporabljamo uteževanje učnih primerov, ni potrebno omejevati števila najbližjih sosedov na  $k$  in na napoved vplivajo vsi učni primeri. Vpliv oddaljenih učnih primerov bo zanemarljiv v primerjavi z bližnjimi učnimi primeri.

### Lokalno utežena regresija

Pri regresijskih problemih osnovno idejo uteževanja z razdaljo posplošimo tako, da namesto povprečenja v enačbi (8.1) uporabimo poljubno regresijsko funkcijo skozi  $k$  najbližjih sosedov, npr. linearne, kvadratne, večnivojski perceptron (nevronske mreže) itd. Lokalno utežena regresija sestavlja lokalno aproksimacijo ciljne funkcije v okolini novega primera. Ta aproksimacija se uporabi za napoved vrednosti funkcije za dani primer. Najpogosteje se uporablja linearne lokalne utežene regresije, ki je linearne regresije, uporabljeni na  $k$  najbližjih sosedih (glej razdelek 8.2). Ker je parameter  $k$  relativno majhen, je število učnih primerov, ki služijo za sestavljanje lokalne aproksimacije ciljne funkcije, majhno. Zaradi nevarnosti prevelikega prileganja učnim primerom si ne moremo privoščiti kompleksnih modelov, ki bi eksaktne modelirali vseh  $k$  učnih primerov. Za vsak novi primer je potrebno ponovno poiskati  $k$  najbližjih sosedov in ponovno aproksimirati funkcijo, kar je lahko časovno potratno.

## 8.2 Linearna regresija

*Bodi ti sam, kdo drug je bolj primeren?*

*Frank J. Giblin*

Naj bo  $\mathbf{v}^T = \langle 1, v^{(1)}, \dots, v^{(a)} \rangle$  vektor vrednosti vseh atributov  $A_1, \dots, A_a$  razširjen z elementom 1. Pri linearni regresiji je funkcija linearna kombinacija vrednosti vseh (ali ustrezne podmnožice) atributov:

$$\hat{r} = g(v^{(1)}, \dots, v^{(a)})$$

$$\hat{r} = w_0 + \sum_{i=1}^a w_i v^{(i)} = \mathbf{w}^T \mathbf{v}$$

Naloga je določiti vektor  $\mathbf{w}$  parametrov  $w_i, i = 0 \dots a$  tako, da minimiziramo vsoto kvadratov napake (*SSE - Sum of Squared Errors*) napovedi preko vseh učnih primerov:

$$SSE = \sum_{j=1}^n (r^{(j)} - \hat{r}^{(j)})^2$$

$$SSE = \sum_{j=1}^n \left( r^{(j)} - w_0 - \sum_{i=1}^a w_i v^{(i,j)} \right)^2$$

Poleg analitične rešitve za iskanje minimuma  $SSE$ , ki je podana spodaj, lahko za generiranje linearne regresijske funkcije uporabljammo tudi pravilo delta za učenje dvonivojskega perceptra (glej razdelek 9.3).

Z  $\mathbf{V}$  označimo matriko vseh učnih primerov:

$$\mathbf{V} = \begin{bmatrix} 1 & v^{(1,1)} & \dots & v^{(a,1)} \\ 1 & v^{(1,2)} & \dots & v^{(a,2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & v^{(1,n)} & \dots & v^{(a,n)} \end{bmatrix}$$

ter z  $\mathbf{r}$  vektor odvisnih spremenljivk učnih primerov:

$$\mathbf{r}^T = \langle r^1, \dots, r^n \rangle$$

Izkaže se [2], da minimalni  $SSE$  dobimo, če velja:

$$\mathbf{w} = (\mathbf{V}^T \mathbf{V})^{-1} \mathbf{V} \mathbf{r}$$

Zato da lahko izračunamo  $(\mathbf{V}^T \mathbf{V})^{-1}$ , mora veljati  $n > a + 1$  in noben učni primer ne sme biti linearna kombinacija ostalih učnih primerov.

## 8.3 Metode podpornih vektorjev

*Dokler ne razširi krog sočutja na vsa živa bitja, človek ne bo našel miru.*

*Albert Schweitzer*

Metode podpornih vektorjev (*SVM - Support Vector Machines*) so bile razvite v devetdesetih letih [3] in so med najbolj uspešnimi metodami za klasifikacijo in regresijo. Večina algoritmov strojnega učenja teži k minimalnemu številu atributov ter poišče ustrezeno podmnožico pomembnih atributov, nad katerimi zgradi model. Pri metodi SVM uporabimo vse razpoložljive attribute, tudi malo pomembne, in jih z linearo kombinacijo uporabimo za napovedovanje odvisne spremenljivke. Pri SVM je pomembno predvsem, na kakšen način kombinirati attribute, zato je izbira atributov manj pomembna, saj bo sama metoda z ustrezeno kombinacijo izluščila želeno informacijo.

Metode SVM so primerne za učenje na velikih množicah primerov, opisanih z velikim številom (manj) pomembnih atributov. Metode dosegajo visoko točnost napovedi. Njihova slaba stran je (podobno kot pri nevronskih mrežah), da je interpretacija naučenega težavna, prav tako pa tudi razлага posamezne odločitve.

V nadaljevanju si poglejmo SVM, namenjen klasifikaciji. V osnovi so klasifikacijske metode SVM namenjene razločevanju dveh razredov. Če imamo več razredov, postopek ponovimo za vsak razred, ki ga skušamo ločiti od ostalih. Pri klasifikaciji novi primer klasificiramo v razred z najvišjo vrednostjo odločitvene funkcije (vrednost  $w_0 \cdot u - b_0$  v enačbi (8.6)).

### Osnovni princip metode SVM

Osnovna ideja je v danem prostoru (originalnih ali transformiranih) atributov postaviti optimalno hiperravnino. Če so primeri linearo ločljivi, potem je možnih več hiperravnin, ki ločujejo dva razreda med seboj, kot to prikazuje slika 8.1.

Optimalna hiperravnina je tista, ki je enako in najbolj oddaljena od najbližjih primerov iz obth razredov. Najbližnjim primerom optimalne hiperravnine pravimo *podporni vektorji*, razdalji hiperravnine od podpornih vektorjev pa *rob (margin)*. Torej je optimalna hiperravnina tista, ki ima maksimalni rob.

Naj bo učna množica podana s pari

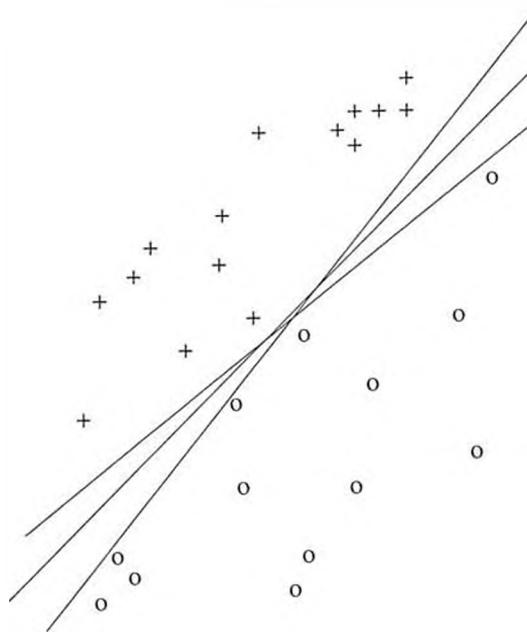
$$(u_j, y_j), j = 1..n$$

kjer je  $y_j = 1$ , če primer pripada prvemu razredu, in  $y_j = -1$ , če pripada drugemu razredu.  $n$  je število učnih primerov,  $a$  pa število atributov. Vsi atributi morajo biti zvezni, torej  $u_j \in R^a$ ,  $j = 1..n$ . Enačba za hiperravnino je podana z:

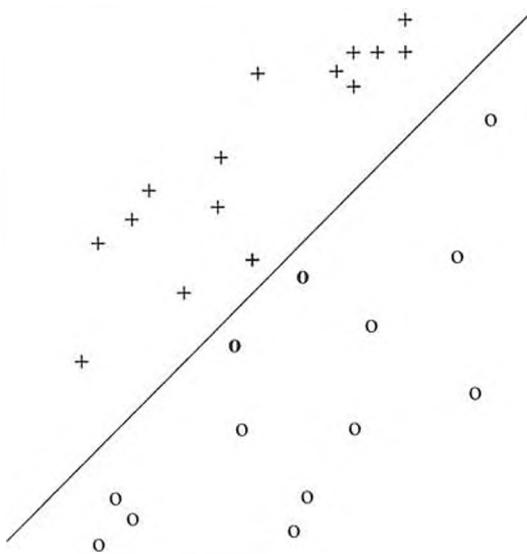
$$(w \cdot u) - b = 0$$

Optimalna hiperravnina mora zadovoljiti omejitve, ki zahtevajo pravilno klasifikacijo vsakega učnega primera:

$$y_j[(w \cdot u_j) - b] \geq 1, j = 1..n \quad (8.2)$$



Slika 8.1: Za linearni dvorazredni klasifikacijski problem obstaja več hiperravnin (v primeru na sliki so to premice), ki eksaktно ločujejo primere iz prvega razreda (označene s +) od primerov iz drugega razreda (označenih z o).



Slika 8.2: Optimalna hiperravnina (premica) za klasifikacijski problem iz slike 8.1; vsi trije podporni vektorji so poudarjeni.

ter hkrati minimizirati

$$\frac{1}{2}(\mathbf{w} \cdot \mathbf{w})$$

Minimizirati želimo velikost koeficinetov hiperravnine in s tem kompleksnost rešitve. Zgornji problem minimizacije z omejitvami (8.2) se prevede na problem maksimizacije funkcionala [3]:

$$W(\alpha) = \sum_{j=1}^n \alpha_j - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{u}_i \cdot \mathbf{u}_j) \quad (8.3)$$

pri omejitvah

$$\alpha_j \geq 0, \quad j = 1..n \quad (8.4)$$

in

$$\sum_{j=1}^n \alpha_j y_j = 0 \quad (8.5)$$

Gre za kvadratni optimizacijski problem, za katerega lahko uporabimo obstoječe (hitre) algoritme [1].

Vsek  $\alpha_j$  ustreza enemu učnemu primeru. Večina je enaka 0, tisti ki niso, pa določajo podporne vektorje. Naj bo  $\alpha_0 = (\alpha_1^0, \dots, \alpha_n^0)$  rešitev tega kvadratnega optimizacijskega problema. Klasifikacijsko pravilo na osnovi optimalne hiperravnine je podano z:

$$Y(\mathbf{u}) = \text{sign}(\mathbf{w}_0 \cdot \mathbf{u} - b_0) = \text{sign} \left( \sum_{\text{podporni vektorji } \mathbf{u}_j} y_j \alpha_j^0 (\mathbf{u}_j \cdot \mathbf{u}) - b_0 \right) \quad (8.6)$$

in je prag  $b_0$  podan z:

$$b_0 = \frac{1}{2}[(\mathbf{w}_0 \cdot \mathbf{u}_*(1)) + (\mathbf{w}_0 \cdot \mathbf{u}_*(-1))]$$

kjer smo z  $\mathbf{u}_*(1)$  označili poljubni podporni vektor iz prvega razreda in z  $\mathbf{u}_*(-1)$  poljubni podporni vektor iz drugega razreda. Vektor koeficientov optimalne hiperravnine  $\mathbf{w}_0$  je podan s podpornimi vektorji:

$$\mathbf{w}_0 = \sum_{\text{podporni vektorji } \mathbf{u}_j} y_j \alpha_j^0 \mathbf{u}_j$$

Kompleksnost rešitve optimalne hiperravnine je podana s številom podpornih vektorjev. Ker je število podpornih vektorjev relativno majhno v primerjavi z vsemu učnimi primeri (v praksi se izkaže, da jih je od 3% do 5%), je kompleksnost rešitve majhna. Dejansko je rešitev odvisna samo od podpornih vektorjev, kar pomeni, da bi enako rešitev dobili, če bi iz učne množice odstranili vse vektorje, ki niso podporni vektorji. Optimizacijski kriterij je minimiziral tudi velikost  $\frac{1}{2}(\mathbf{w} \cdot \mathbf{w})$ , kar pomeni, da so koeficienti pri linearni kombinaciji atributov minimizirani.

### Razširitev na neeksaktno klasifikacijo

Pri neeksaktni klasifikaciji omejitve (8.2), ki zahtevajo pravilno klasifikacijo vsakega učnega primera, sprememimo tako, da dovolimo napako pri klasifikaciji  $j$ -tega učnega primera za  $\xi_j$ :

$$y_j[(\mathbf{w} \cdot \mathbf{u}_j) - b] \geq 1 - \xi_j, \quad j = 1..n \quad (8.7)$$

Optimizacijski problem se spremeni v minimizacijo:

$$\frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + C \sum_{j=1}^n \xi_j$$

pri čemer je  $C$  vnaprej podana vrednost. S parametrom  $C$  uravnavamo razmerje med kompleksnostjo rešitve (prvi sumand) in napako rešitve (drugi sumand). Problem se prevede na maksimizacijo funkcionala (8.3), kot pri eksaktnem primeru, le omejitve (8.4) se spreminja v:

$$C \geq \alpha_j \geq 0, \quad j = 1..n \quad (8.8)$$

medtem ko omejitve (8.5) ostanejo nespremenjene.

Tudi pri neeksaktni rešitvi velja, da so samo koeficienti  $\alpha_j$ , ki določajo podporne vektorje, različni od nič.

## Metoda SVM

Bistvena ideja metode podpornih vektorjev je poleg uporabe podpornih vektorjev uporaba implicitne transformacije atributnega prostora v kompleksnejši atributni prostor. V originalnem prostoru pogosto linearne hiperravnine ne zadošča za sprejemljivo klasifikacijsko točnost. Z nelinearno transformacijo lahko postane prostor primeren za linearno ločitveno hiperravnino. Uporabnik vnaprej izbere transformacijo. Na ta način lahko z različnimi transformacijami rešujemo različne nelinearne probleme.

Prava moč metod SVM je v tem, da transformacij ni potrebno narediti eksplisitno, kot je to potrebno pri ostalih metodah strojnega učenja. Z eksplisitno transformacijo dobimo veliko število atributov (npr. pri produktih osnovnih atributov število atributov naraste polinomsko, stopnja polinoma pa je odvisna od dolžine produktov). Problemu prevelikega števila atributov pravimo tudi *prekletstvo dimenzionalnosti (curse of dimensionality)*.

Pri metodi podpornih vektorjev zadošča, da izračunamo skalarne produkte podpornih vektorjev z novim (testnim) primerom v transformiranem atributnem prostoru. Če je  $\mathbf{u}_j$   $j$ -ti učni primer (vektor) v originalnem atributnem prostoru in  $\mathbf{z}_j$  ta isti primer v transformiranem atributnem prostoru, potem moramo zagotoviti izračun skalarnega produkta s pomočjo jedne funkcije  $K$ , ki omogoča implicitno transformacijo:

$$\mathbf{z}_j \cdot \mathbf{z} = K(\mathbf{u}, \mathbf{u}_j)$$

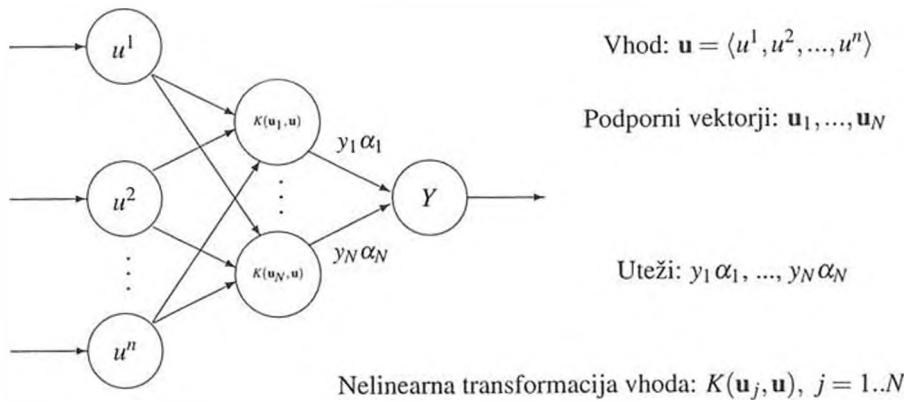
Namesto skalarnih produktov uporabljamo konvolucijo skalarnih produktov. Funkcional za maksimizacijo je torej:

$$W(\alpha) = \sum_{j=1}^n \alpha_j - \frac{1}{2} \sum_{i,j}^n \alpha_i \alpha_j y_i y_j K(\mathbf{u}_i, \mathbf{u}_j) \quad (8.9)$$

pri enakih omejitvah za koeficiente  $\alpha_j$ , kot prej. Odločitvena funkcija za klasifikacijo novih primerov  $\mathbf{u}$  je torej:

$$Y(\mathbf{u}) = \text{sign} \left( \sum_{\text{podporni vektorji } \mathbf{u}_j} y_j \alpha_j^0 K(\mathbf{u}_j, \mathbf{u}) - b_0 \right) \quad (8.10)$$

za dane  $\alpha_j^0$ , ki predstavljajo rešitev optimizacijskega problema. Slika 8.3 ponazarja odločitveno pravilo metode SVM.



Slika 8.3: Odločitveno pravilo metode SVM.

Z različnimi jedrnimi funkcijami dobimo različne transformacije prostora atributov in s tem različne variante metode SVM. Primeri jedrnih funkcij so [3]:

**Linearna:** v tem primeru ohranimo originalni atributni prostor

$$K(\mathbf{u}_j, \mathbf{u}) = \mathbf{u} \cdot \mathbf{u}_j$$

**Polinomska:** za dano stopnjo polinoma  $d$  lahko uporabimo naslednjo funkcijo za konvolucijo skalarnega produkta

$$K(\mathbf{u}_j, \mathbf{u}) = [(\mathbf{u} \cdot \mathbf{u}_j) + 1]^d$$

**Radialna:** za izbrano vrednost parametra  $\gamma$  je konvolucija podana z

$$K(\mathbf{u}_j, \mathbf{u}) = e^{-\gamma|\mathbf{u} - \mathbf{u}_j|^2}$$

**Sigmoidna:** pri dani sigmoidni funkciji  $S$  (npr.  $S = \tanh$ ) dobimo jedrno funkcijo za določene vrednosti parametrov  $v$  in  $c$

$$K(\mathbf{u}_j, \mathbf{u}) = S[v(\mathbf{u} \cdot \mathbf{u}_j) + c]$$

Če izberemo sigmoidno funkcijo, dejansko dobimo arhitekturo trinivojskih nevronskih mrež. Za razliko od trinivojskega perceptronja, opisanega v razdelku 9.3, SVM sam določi optimalno arhitekturo (število skritih nevronov je enako številu podpornih vektorjev). Uteži prvega nivoja ustrezajo vrednostim atributov podpornih vektorjev, uteži drugega nivoja pa izračunanim koeficientom  $\alpha_j^0$ .

V praksi se izkaže, da različne jedrne funkcije pri izračunu optimalne hiperravnine v dani problemski domeni izberejo v veliki meri (okoli 80%) iste podporne vektorje [3].

## Literatura

- [1] N. Cristianini, J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.
- [2] A. C. Rencher. *Methods of Multivariate Analysis*. John Wiley & Sons, 1995.
- [3] V. Vapnik. *The nature of Statistical Learning*. Springer Verlag, 2nd edition, 2000.

## Poglavlje 9

# Umetne nevronske mreže

*Inteligenca nastane iz interakcije velikega števila preprostih procesnih enot.*

*David Rumelhart in John McClelland*

### 9.1 Uvod

*Da bi živel čisto nesobično življenje, ne smeš sredi izobilja ničesar štetiti za svoje.*

*Buda*

#### Preprost primer nevronske mreže

Oglejmo si zgled matričnega računa, ki ga je zmožna opraviti nevronska mreža. Njena naloga naj bo naučiti se razpoznavati naslednja dva vzorca:

$$X_1 = (1, 1, 1)^T$$

in

$$X_2 = (1, -1, -1)^T$$

Oglejmo si najprej, kako to zmore matrični račun. Sestavimo vsoto matrik, ki jih dobimo kot produkt vektorja s transponiranim vektorjem samim:

$$M = X_1 X_1^T + X_2 X_2^T = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 2 \\ 0 & 2 & 2 \end{bmatrix}$$

Definirajmo še odločitveno funkcijo (pragovni element):

$$f(X) = \begin{cases} 1, & X > 0 \\ 0, & X = 0 \\ -1, & X < 0 \end{cases}$$

Če posplošimo odločitveno funkcijo tudi na vektorje, lahko zapišemo naslednje:

$$f(MX_1) = f((2, 4, 4)^T) = (1, 1, 1)^T = X_1$$

$$f(MX_2) = f((2, -4, -4)^T) = (1, -1, -1)^T = X_2$$

Vidimo, da produkt matrike z danim vektorjem obdrži približno smer danega vektorja, ki jo z odločitveno funkcijo še popravimo, in dobimo enak vektor.

Poskusimo uporabiti isti postopek, ko prvotne vektorje poznamo le delno. Z 0 označimo neznano vrednost komponente:

$$X'_1 = (1, 1, 0)^T$$

in

$$X'_2 = (1, 0, -1)^T$$

$$f(MX'_1) = f((2, 2, 2)^T) = (1, 1, 1)^T = X_1$$

$$f(MX'_2) = f((2, -2, -2)^T) = (1, -1, -1)^T = X_2$$

Prodoti matrike z delno poznanim vektorjem nadomesti manjkajočo vrednost. Poskusimo sedaj še z napačnimi vrednostmi:

$$X''_1 = (1, 1, -1)^T = X''_2$$

$$f(MX''_1) = f((2, 0, 0)^T) = (1, 0, 0)^T$$

V tem primeru je odgovor neodločen. Dejansko je v našem primeru nemogoče ugotoviti, ali je napačna druga ali tretja komponenta.

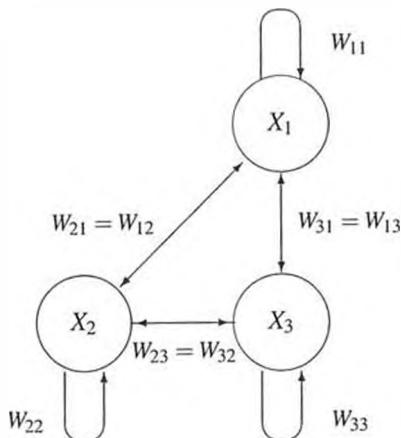
Razmišljanje ne velja vedno oziroma velja le pod določenimi pogoji (potrebna je ortogonalnost vektorjev), vendar to presega okvir tega uvoda. Zanimivo je, da ima vsak shranjeni vzorec shranjen tudi komplement (če npr. shranimo vektor  $(1, -1, -1)$ , se avtomatsko shrani tudi vektor  $(-1, 1, 1)$ ).

Oglejmo si, kako gornji račun realiziramo z nevronsko mrežo. Vsak nevron ima dve različni stanji. Za naš primer je stanje nevrona lahko 1 ali -1. Na sliki 9.1 je prikazana nevronska mreža, sestavljena iz treh nevronov, tako da je vsak povezan z vsakim. Vezi med nevroni ustrezajo elementom matrike. Vezem pravimo tudi *sinapse* po analogiji z biološkimi nevronske mrežami. Vezi so dvosmerne, zato je matrika simetrična. Diagonalni elementi matrike ustrezajo vezem nevronov samih s seboj. Vsaka vez ima pridruženo realno vrednost (pozitivno ali negativno), ki ustrezava povezavi med nevronoma, ki ju vez povezuje. Tem vrednostim pravimo uteži.

Pred učenjem so vse uteži enake 0. Učenje poteka tako, da se vsakemu nevronu vsili vrednost (stanje) ustrezne komponente iz vhodnega vektorja. Če imata nevrona, ki ju vez povezuje, enake vrednosti, se utež vezi poveča za 1, sicer se zmanjša za 1 (to je posplošeno Hebbovo pravilo učenja, ki je opisano pozneje). Opisani postopek ponovimo za vsak vhodni vzorec. Ko je mreža naučena, se uteži ne spreminja več.

Preverjanje novega primera poteka tako, da se vrednosti ustreznih komponent (delnega) vzorca posredujejo nevronom in vsak nevron vzoredno izračuna svoje stanje, ki je hkrati njegov novi izhod, po pravilu:

$$Y_i = f\left(\sum_j W_{ji}X_j\right)$$



Slika 9.1: Primer avtoasociativne umetne nevronske mreže.

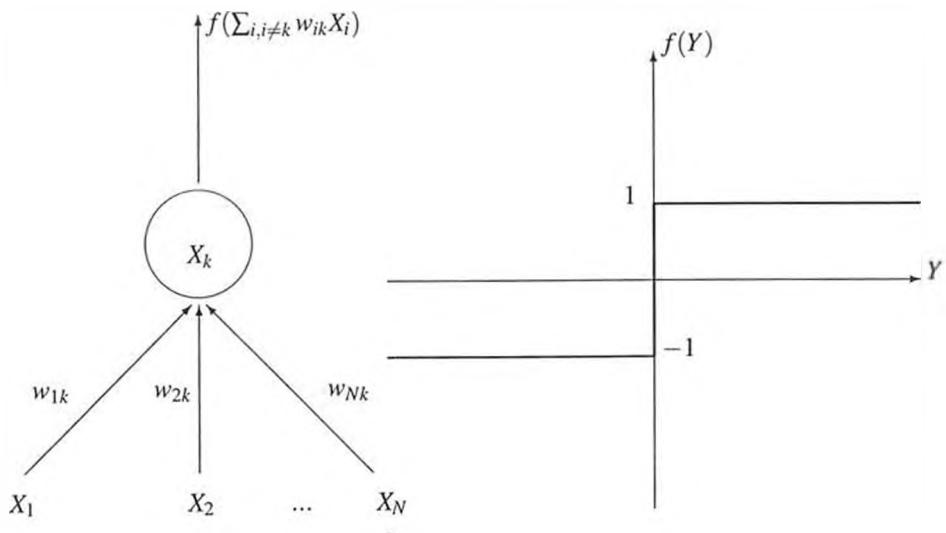
kjer je  $X_j$  stanje  $j$ -tega nevrona,  $W_{ji}$  utež sinapse med  $j$ -tim in  $i$ -tim nevronom in  $Y_i$  novo stanje  $i$ -tega nevrona. Nevrni so procesni elementi, ki znajo izračunati uteženo vsoto vhodov in z odločitveno funkcijo preslikati dobljeno vsoto v eno izmed dveh izhodnih vrednosti (slika 9.2). Uteženi vsoti pravimo *funkcija aktivacije*, odločitveni funkciji  $f$  pa *izhodna funkcija*. Pozneje bomo videli, da lahko izhodna funkcija vrne tudi več kot dve različni vrednosti.

Slika 9.2 ponazarja abstrakten nevron in tipično kombinacijsko funkcijo, ki jo izračunava. Tak nevron ima dve možni stanji: 0 in 1 (neaktiv in aktiv), včasih tudi -1 in 1. Kombinacijsko funkcijo lahko posplošimo, da dobimo zvezna stanja nevrona, ki zavzemajo vrednosti na določenem intervalu (npr. [0,1]).

V našem primeru so vsi nevrni vzporedno in *sinhrono* (naenkrat) izračunali svoj izhod. Ponovitev iste operacije ne bi spremeniла stanj nevronov. Pravimo, da je izvajanje nevronske mreže prišlo v *fiksno točko* (*equilibrium*). V splošnem nevronska mreža iteracije ponavlja, dokler se stanja nevronov ne ustalijo. Če se to zgodi v končnem času, pravimo, da mreža *konvergira*. Izvajanje nevronske mreže je lahko tudi *asinhrono*, tako da nevrni spremimnjo stanja ob različnih časih.

Če je utež na sinapsi pozitivna, pravimo, da je sinapsa *vzbujevalna*. Če je utež negativna, je sinapsa *zaviralna*. Manjkajoči sinapsi ustreza utež 0. Ponavadi so uteži na sinapsah predstavljene z matriko. Dokazano je, da pri simetrični matriki asinhroni model vedno konvergira [17], sinhroni pa ne vedno [18]. Za asinhroni model se zahteva, da samo en nevron naenkrat spremeni svoje stanje. Obstajajo tudi modeli, ki uporabljajo asimetrične matrike, za katere je empirično pokazana konvergenca [31].

Če uporabljamo nevronske mreže kot asociativni pomnilnik, mreža konvergira k fiksni točki, ki ustreza najbližnjemu (najbolj podobnemu) shranjenemu vzorcu glede na dani vhodni vzorec. Fiksna točka ni nujno eden od shranjenih vektorjev in shranjeni vektor ni nujno najbolj podoben vhodnemu vektorju [18]. Fiksne točke so linearne neodvisni vektorji, katerih



Slika 9.2: Umetni nevron in kombinacijska funkcija.

podmnožica so tudi lastni vektorji matrike.

### Porazdeljeni pomnilnik

Osnovni princip nevronske mreže je porazdeljeni (distribuirani) pomnilnik. Pomnenje je v povezavah med nevroni. Vsak nevron ima dostop do pomnilnika na povezavah med seboj in drugimi nevroni (ni nujno, da je vsak nevron povezan z vsakim). To interpretiramo kot veliko število omejitev, ki se hkrati upoštevajo pri generiranju izhoda. Porazdeljeni pomnilnik ne vrne vedno eksaktnega podatka. Ker je podatek shranjen v množici povezav, skupaj z drugimi podatki, medsebojni vpliv lahko delno spremeni rezultat.

Lastnosti porazdeljene predstavitve pomnilnika so:

- avtomatska pospolitev: vsaka pomnilniška celica povezuje aktivnosti dveh nevronov (povprečno glede na vse situacije, v katerih se je mreža nahajala oziroma glede na do sedaj prikazane učne primere); to znanje se uporablja za aproksimacijo manjkajočih vrednosti ali za popravljanje napačnih vrednosti; tako znanje je t.i. *površinsko znanje*. Mreža ne pozna *globokega znanja*, ki obsega relacije in zakone, ki veljajo v dani problemski domen;
- prilagodljivost: s spremenjanjem okolja se mreža inkrementalno uči in s tem prilagaja na okolje; pozabljjanje že naučenega kontroliramo s parametri;

- čim večja je porazdeljenost, tem natančneje lahko shranimo podatke z enakim številom nevronov [8]; porazdeljenost povečamo z drugačnim kodiranjem, npr. namesto da nevron predstavlja eno komponento, lahko več nevronov zakodira vrednosti več komponent vhodnih vektorjev; na ta način je komponenta shranjena v povezavah z več komponentami, kar omogoča njeno natančnejšo rekonstrukcijo;
- konstruktivnost: vsak procesni element (nevron) predstavlja (mikro) atribut, ki kombinira več osnovnih atributov; vsak mikroatribut opisuje dano domeno iz svojega zornega kota, ki pa ga je težko interpretirati;
- ker je podatek v nevronske mreže predstavljen z več elementi in nevron vključen v predstavitev več podatkov, odstranitev enega nevrona povzroči zmanjšanje natančnosti za več podatkov, ne pa popolne izgube posameznega podatka [8];
- spontano spominjanje pozabljenega med ponovnim učenjem drugih podatkov, brez ponovnega učenja istih podatkov [8, 9].

Ko je mreža naučena (uteži na vezeh fiksirane), deluje tako, da nevroni dobijo začetne vrednosti (vhodni vzorec) in spreminjajo svoja stanja glede na uteženc vhodne signale. Delovanje je vzporedno in asinhrono. Delovanje se ustavi, ko ni več sprememb stanj. Pravimo, da je nevronska mreža prišla v stabilno (fiksno) točko.

Za razliko od lokalne predstavitve v računalnikih, kjer podatek naslovimo z naslovom, je v nevronske mreže naslov podatek sam (to velja za avtoasociativni pomnilnik). Če je naslov točen, dobimo identičen podatek, sicer dobimo po določenih kriterijih podoben podatek. Temu pravimo *vsebinsko naslavljjanje*. V nevronske mreže torej ni razlike med podatkom in njegovim naslovom.

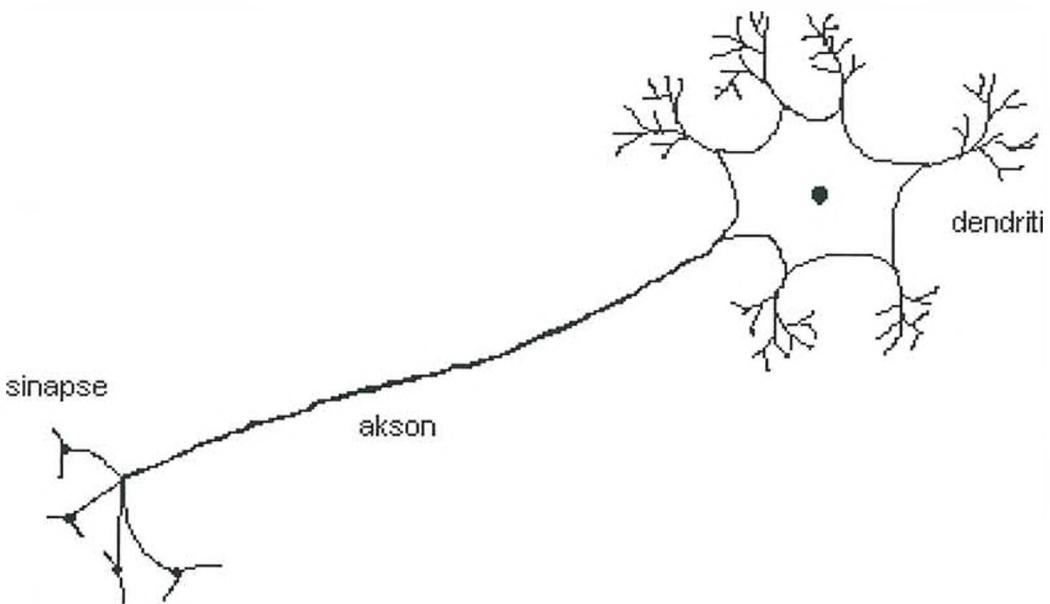
## Lastnosti umetnih nevronske mreže

Lastnosti umetnih nevronske mreže so podobnost biološkim mrežam, visoka stopnja vzporednosti, asinhrono izvajanje, večsmerno izvajanje, prilagodljivost, robustnost na okvare in na manjkajoče podatke, sposobnost učenja in avtomatska generalizacija. Mreža ne potrebuje programske opreme in eksplicitne konfiguracije, ni razlike med podatki in naslovi. Področje nevronske mreže ima tudi dobro matematično podlago. Njihova glavna pomanjkljivost je nezmožnost razložiti svojo odločitev. Prav tako ne obstaja nevronska mreža, ki bi imela vse naštete lastnosti.

**Biološka podobnost** Reševanje problemov z umetnimi nevronske mrežami skuša oponašati delovanje možganov in doseči večjo učinkovitost pri reševanju zahtevnih problemov. Nevronske mreže so abstrakcija in poenostavitev možganskih celic. Nevroni v različnih delih možganov se med seboj močno razlikujejo po obliki, vendar je njihova osnovna struktura enaka.

Primer možganskega nevrona je shematično prikazan na sliki 9.3. Akson prenaša dražljaje iz celičnega telesa, dendriti pa vodijo vhodne dražljaje v celico. Akson se na koncu razveji in se preko povezav, imenovanih sinapse, povezuje z dendriti ostalih nevronov. Aproximacija z modeli umetnih nevronske mreže kljub poenostavitev

zadostuje za učinkovito modeliranje in analizo. Te raziskave vodijo tudi do boljšega razumevanja procesov v možganih.



Slika 9.3: Shematični prikaz možganskega nevrona.

**Vzporednost** Visoka stopnja vzporednosti v nevronske mrežah je posledica dejstva, da vsak nevron deluje neodvisno od ostalih. To omogoča izredno hitro izvajanje. Zato so nevronske mreže zmožne prilagajanja zapletenemu okolju v realnem času. Večina današnjih aplikacij je simulirana s klasičnimi računalniki, deluje pa nekaj prototipov specifičizirane strojne opreme, ki oponaša delovanje nevronskega mrež.

**Večsmerno izvajanje** Pri nevronske mreži, kjer je vsak nevron povezan z vsakim, so vsi nevroni hkrati vhodni in izhodni. Mreža v tem primeru ne dela razlike med vhodom in izhodom. Poljubna podmnožica nevronov je lahko vhodna. Delovanje mreže bo pri danem vhodu aproksimiralo neznane vrednosti, ki bodo v tem primeru izhod. Primer take mreže je na sliki 9.1.

**Robustnost** Nevronska mreža je robustna na okvaro posameznih nevronov in sinaps. Robustnost je posledica porazdeljene predstavitev. Točnost mreže pada sorazmerno s številom uničenih nevronov oziroma povezav (podobno velja za biološke nevronske mreže).

Mreža je robustna tudi na pomanjkljive vhodne podatke. Ta lastnost izvira iz načina njenega delovanja. V mreži, kjer je vsak nevron povezan z vsakim, se bodo manjajoči vhodni podatki aproksimirali že v prvi iteraciji iz razpoložljivih vhodnih podatkov in naučenega. Primer take aproksimacije smo si ogledali na začetku tega razdelka. Aproksimacija bo tem slabša, čim pomanjkljivejši so vhodni podatki.

**Učenje** Učenje v nevronskeih mrežah poteka s spreminjanjem utiči na sinapsah. Ena pomnilniška celica (sinapsa) povezuje aktivnosti dveh nevronov. Ker se utež spreminja glede na vhodne podatke (okolje, učne primere), predstavlja povezanost glede na vse dosevanje situacije, v katerih se je mreža nahajala. Ko je mreža naučena, lahko vsak nevron napove svoje stanje v odvisnosti od stanj z njim povezanih nevronov.

**Relacija med strojno in programsko opremo** V nevronske mreže ni programa v klasičnem pomenu besede, temveč vsebuje le algoritem, po katerem deluje posamezni nevron. Vse ostalo delovanje je posledica tega. Mreža teži k stabilni točki z vzporednim in asinhronim delovanjem nevronov. Besedi računalnik ali izpeljevalnik (*inference machine*) nista primerni za opis delovanja takega stroja. Primernejše ime bi bilo "relaksator". Strojna oprema je v nevronske mreže fiksna, ali pa se spreminja (sinapse propadejo ali se pojavijo nove). Za posamezno izvajanje konfiguracija ni enolično določena. Za nekatere funkcionalnosti ni nujno, da je vsak nevron povezan z vsakim. Zadostuje, da je število povezav med nevroni mnogo večje od števila nevronov [13]. Konfiguracija pa je lahko naključna, kar je analogno možganom.

**Matematična podlaga** Področje teoretično temelji na linearni algebri [12, 13]. Pojmi, kot so linearne transformacije, inverzne transformacije, linearnost, ortogonalnost vektorjev, lastna vrednost, lastni vektor, fiksna točka in konvergenca, so formalno definirani in dobro obdelani.

**Razlaga odločitve** Največja slabost nevronske mreže je težavnost razlage njihovih odločitev. Z gledišča raziskovalcev umetne inteligence sistem, ki svoje odločitve ne more razložiti, ni boljši od statističnega sistema za razpoznavanje vzorcev, ki dobro deluje, vendar interpretacija njegovih rezultatov ni mogoča brez veliko znanja in izkušenj iz matematike in statistike.

## Primeri uporabe

Obstaja več komercialno dosegljivih paketov za razvoj nevronske mreže. Omogočajo hitro definicijo nevronske mreže in uporabo standardnih modelov nevronske mreže, kot so Hopfieldov, Boltzmannov, Grossbergov, večnivojski perceptron, Kohonenov itd. Primeri aplikacij so:

- kontrola kvalitete izdelave trdih diskov, ki vnaprej napove možna odstopanja od zahtevane natančnosti, tako da jih lahko pravočasno odpravimo;
- diagnosticiranje pacientov na osnovi signala EKG;

- razpoznavanje ročno zapisanih znakov; sistemi umetnih nevronske mreže dosegajo višoke točnosti, če sistem lahko izloči znake, ki jih ne more zanesljivo klasificirati;
- preverjanje podpisov na čekih; nevronske mreže so nepravilno klasificirale 4% podpisov, kar presega rezultate strokovnjakov, ki naredijo več napak;
- identifikacija objektov z radarjem.

Mnogo aplikacij je v računalniškem razpoznavanju govora in računalniškem vidu. Kohenen [14, 15] navaja od 92 do 97% točnost pri razpoznavanju govora (pisanja po nareku brez razumevanja) in 96 do 98% točnost pri razpoznavanju govorjenih besed iz slovarja, ki vsebuje 1000 besed. Uporabljena metoda učenja je samoorganiziranje, kjer se vsak nevron specializira za določen vhod.

Želeli bi, da bi bil računalniški vid invarianten glede na rotacijo, translacijo in velikost opazovanih objektov. Hinton [7] je pokazal, kako sistem razpozna vhodni vzorec, čeprav vrste vzorca ne pozna vnaprej in ne ve, katera transformacija je potrebna. Ideja je v tem, da poskuša vzporedno z vsemi transformacijami in s primerjavo z vsemi shranjenimi vzorci (kar je naravni princip v nevronske mrežah). S ponavljanjem se bosta ojačala transformacija in vzorec, ki najbolj ustreza vhodnemu vzorcu. Fukushima [2] je s posebno topologijo nevronske mreže dosegel razpoznavanje cifar invariantno glede na velikost in translacijo in delno invariantno glede na deformacijo. Widrow in Winter [29] sta predlagala razpoznavanje v dveh korakih z dvema nevronskima mrežama. Prva mreža bi spremenjala vhodne vzorce v invariantne glede rotacije, translacije in velikosti. Druga mreža bi se naučila iz vmesnih vzorcev generirati originalne v standardni poziciji, orientaciji in velikosti.

Grabec in Sachse [3] sta uporabila nevronske mreže za analizo in procesiranje signalov akustične emisije. Pokazala sta, da se z nevronske mreže učinkovito reši inverzni problem akustične emisije, t.j. odkrivanje izvora zvoka na netrivialnem problemu, ki je eksaktno praktično nerešljiv. Nevronska mreža se brez uporabe enačb iz elastodinamične teorije iz pravrov nauči odkrivati karakteristike izvora akustičnih signalov. Ta aplikacija napoveduje novo generacijo merilnih instrumentov, ki naj bi temeljila na principu nevronske mreže.

Nevronske mreže hitro najdejo približne rečitve NP-polnih problemov (v linearinem ali celo konstantnem času). Pri tem mora število nevronov ustrezi velikosti problema. Hopfield in Tank [11] sta demonstrirala hitro reševanje problema trgovskega potnika (kjer je treba poiskati najkrajšo pot skozi vsa dana mesta, tako da vsako mesto obiščemo natanko enkrat).

Barto in sod. [1] zagovarjajo kompleksnejše modele nevronov. Dva ustreznega povezana nevrona sta se naučila balansirati palico na premičnem vozičku mnogo bolje kot originalni sistem BOXES, ki sta ga razvila Michie in Chambers [19]. En nevron se je učil kontrolirati voziček iz danega opisa stanja sistema. Drugi nevron se je naučil iz danega stanja sistema napovedovati napake (odkriaval je kritična stanja sistema).

Popolnoma drugačno področje uporabe nevronske mreže je kognitivno modeliranje. Nevronske mreže so dovolj močne za simulacijo nekaterih kognitivnih procesov, po drugi strani pa dovolj preproste in formalno definirane, da jih je razmeroma preprosto modelirati in analizirati. Npr. Hood [10] je uporabljal nevronske mreže za raziskovanje učenja nižjih organizmov. Pazzani in Dyer [21] sta primerjala človekovo učenje konceptov in učenje nevronske mreže.

## Analogija z možgani

*Možgani so čudovit organ. Začnejo delovati v trenutku, ko vstaneš, in se ne ustavijo, dokler ne prideš do pisarne.*

*Robert Frost*

Možgani so vsebinsko naslovljeni. Mentalni procesi uporabljajo možgane v celoti. Pri njih ni mogoče ločiti funkcije pomnilnika od procesorjev, kot je to mogoče v računalnikih. Pomnilnik v možganski skorji je porazdeljen in ne lokalen. Pommembeni delež pomnjenja v možganih prispevajo sinaptične povezave [13].

Hitrost nevronov v možganih je reda milisekunde. Percepcija, procesiranje jezika, intuitivno razmišljanje in dostop do spomina zahtevajo v možganih približno desetinko sekunde, torej okoli 100 korakov. Zato je Feldman definiral "100-koračni program" kot omejitev za izvajanje elementarnih operacij programa, ki bi obrazložil mentalne procese. Realizacija tega programa je mogoča samo z uporabo visoke stopnje vzporednosti.

Tako v možganih kot v nevronske mreži učinkovitost počasi pada s številom uničenih nevronov [26]. Ni ključnega nevrona, katerega okvara bi "porušila" celoten sistem. Tudi v možganski skorji ni dela, od katerega bi bili odvisni vsi ostali deli [23]. Noben del možganov ni nadomestljiv. Pokazano je, da pri otrocih, rojenih samo z eno možgansko poloblo, le-ta prevzame funkcije manjkajoče in otroci odrastejo brez okvarjenih funkcij [26].

Človeško zaznavanje je do določene mere invariantno na velikost, translacijo in rotacijo [13]. Različna področja v možganski skorji so se specializirala za različne signale tako, da ohranljajo topologijo prostorskih senzorjev. Ta princip simuliramo z nevronske mreže. Kohonen [13] navaja, da je precej anatomskih in psiholoških potrditev, da obstaja v možganski skorji samoorganiziranje povezav kot pri umetnih nevronskeh mrežah.

Za učinkovito delovanje nevronske mreže ni potrebno medsebojno povezati vseh nevronov. Zadošča, da je število povezav, če so te porazdeljene po mreži, mnogo večje od števila nevronov. Če bi nevroni v možganih z  $10^{10}$  nevroni in s povprečno  $10^4$  povezavami na nevron delovali kot preprost pragovni element, bi imeli zadostno kapaciteto, da shranijo vse, kar človek doživi v sto letih [13]. Tudi psihologi navajajo, da je precej empiričnih potrditev, da si lahko človek zapomni prav vse, kar se mu pripeti v življenju [26].

Geni ne določajo vseh povezav v možganih, določene povezave so omejene zaradi prostorske lege, in verjetno je del povezav naključen [23]. V možganih ni niti "strojne opreme" v strogem pomenu besede niti "programske opreme" v običajnem pomenu besede.

Rekurzija ni domača človekovemu načinu razmišljanja. Rumelhart in McClelland [23] navajata stavko "The man the boy the girl hit kissed move" kot preprost primer rekurzivnega stavka, ki je težko razumljiv. Tudi umetne nevronske mreže ne znajo procesirati rekurzivnih struktur.

Modeli nevronske mreže ne sledijo popolnoma analogiji z možgani. Večina nevronov v možganski skorji ima ali vzbujevalne ali zaviralne vezi, kar je v nasprotju z umetno nevronske mreže, ki pri enem nevronu dopušča obe vrsti vezi. Signali v možganih so podani s frekvenco impulzov in ne z jakostjo signalov kot v modelih umetnih nevronske mreže [13]. Današnji modeli nevronske mreže ne upoštevajo globalne komunikacije, ki v možganih poteka s pomočjo

umetna inteligencia	umetne nevronske mreže
simbolični nivo	subsimbolični nivo
razlaga odločitve	včasih
eksplicitno shranjena pravila	dinamično kreirana pravila
zaporedno procesiranje	vzporedno procesiranje
logični, zavestni nivo	intuitivni, podzavestni nivo
psihološka analogija	tudi
ni podobnosti z biološkimi sistemi	podobnost z biološkimi sistemi
leva možganska polovica	desna možganska polovica

Tabela 9.1: Primerjava metod umetne inteligence in umetnih nevronske mrež.

kemičnih snovi, ki se pretakajo po krvi med različnimi predeli možganov.

### Relacija z metodami umetne inteligence

Zastarel očitek nevronskim mrežam je, da niso zmožne vsega izračunati. Že Minsky in Papert [20] sta pokazala, da lahko univerzalni računski stroj simuliramo z nevronsko mrežo, vendar ta rezultat ni praktično uporaben. Bolj pereč problem je težavnost razlage odločitev. Tabela 9.1 povzema primerjavo metod umetne inteligence in umetnih nevronske mrež.

Današnje raziskave umetne inteligence so usmerjene v razvoj metod in orodij, ki bi omogočile računalnikom intelligentnejše obnašanje. Pri tem je temeljno, da zna sistem svojo odločitev obrazložiti in argumentirati. Navdih za razvoj novih metod pogosto pride iz analogije s človekovim načinom razmišljanja, vendar samo na visokem, simboličnem in logičnem nivoju. Raziskovalci upoštevajo, kaj so možgani zmožni napraviti, ne pa, kako to dejansko napravijo.

Z nevronskimi mrežami iščemo analogijo s človeškimi možgani na nizkem, "subsimboličnem" nivoju. Pri tem raziskovalcev ne zanima le, kaj možgani zmorejo, ampak tudi, kako delujejo. Opis človekovega učenja, ki ga navajajo raziskovalci umetne inteligence, je naslednji [27]. Neizkušeni strokovnjaki pri delu uporabljajo splošna pravila, veljavna v dani domeni. Ker so pravila splošna, je delo z njimi počasno. Sledenje pravilom je zavestno. Sčasoma si strokovnjak ob reševanju problemov ustvarja svoja bolj specifična pravila, ki jih lahko uporabi brez poglobljanja v teorijo, zato postane njegovo delo hitrejše. Sledenje tem pravilom je podzavestno.

Večina modelov v umetni inteligenci nima biološke podobnosti in analogij. Simbolično učenje generira eksplicitna specifična pravila na osnovi že rešenih problemov, kar zahteva velike računalniške zmogljivosti. Podobno velja pri nevronski mreži, le da specifična pravila niso eksplicitno shranjena, ampak se kreirajo sproti, po potrebi. Vzorci, pridobljeni z izkušnjami, se kombinirajo dinamično na nove načine. To omogoča generiranje pravil, ki jih strokovnjak ni nikoli videl v eksplicitni obliki, oziroma hitro rešitev problema, s katerim se je strokovnjak prvič srečal. Mreža, ki deluje kot klasifikator, se obnaša, kot da pozna pravila.

Za izvajanje zadoščajo preproste procesne enote, ki delujejo asinhrono in potrebujejo samo lokalno informacijo.

Lahko bi rekli, da nevronske mreže simulirajo funkcijo desne možganske polovice, ki deluje bolj vzporedno in podzavestno (intuitivno), medtem ko metode umetne inteligence simulirajo funkcije leve možganske polovice, katere delovanje je bolj zavestno, logično in zaporedno. Z združitvijo obeh metod lahko pričakujemo velik skok zmožnosti in učinkovitosti računalniškega reševanja problemov.

Rumelhart in McClelland [23] poudarjata, da bomo ob razumevanju mikronivoja mogoče formulirali popolnoma drugačne makronivojske modele. Navajata namišljeni računalnik, ki bi imel osnovne ukaze, kot so: "sprosti se v stanje, ki optimalno interpretira trenutni vhod", "pridobi iz pomnilnika predstavitev, ki maksimalno ustrezata trenutnemu vhodu in dodaj manjkajoče podrobnosti k shranjeni predstavitvi" ter "konstruiraj dinamično konfiguracijo struktur znanja, ki ustrezata dani situaciji z ustrezno opredeljenimi spremenljivkami". Takemu sistemu bi bolj ustrezalo ime "relaksator" kot računalnik.

## 9.2 Vrste nevronskeih mrež

*Kdor ceni mir uma in zdravje duše, bo živel najboljše od vseh možnih življenj.*

*Mark Avrelij*

Nevronske mreže delimo po kriterijih:

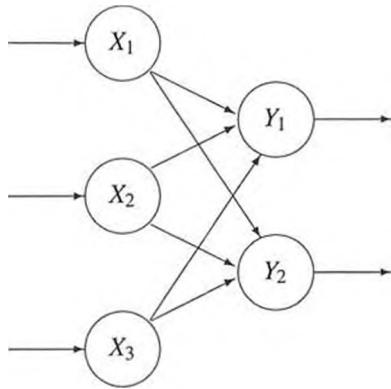
- topologija nevronske mreže,
- namen nevronske mreže,
- pravilo učenja in
- funkcija kombiniranja vhodov nevrona v izhod.

### Topologije nevronske mreže

**Brez nivojev** Pri najsplošnejši obliki nevronske mreže je vsak nevron hkrati vhoden in izhoden in dvosmerno povezan z vsemi nevroni. Primer je podan na sliki 9.1. Mreža deluje tako, da na začetku nevronom vsili vrednosti vhodnega vzorca, zatem pa nevroni spreminjajo svoja stanja in izhod toliko časa, da se izhod mreže ustali.

**Enonivojske usmerjene nevronske mreže** Enonivojska nevronska mreža je sestavljena iz skupine vhodov in iz skupine (izhodnih) nevronov. Vsak vhod je z enosmerno vezjo povezan z vsakim izhodnim nevronom (glej sliko 9.4). Nekateri avtorji tudi vhode označujejo kot nevrone, čeprav ničesar ne izračunavajo, in zato takim mrežam pravijo *dvonivojske*. Izračun poteka tako, da na vhodu dobimo vrednosti komponent vhodnega vzorca in zatem v enem koraku vsi izhodni nevroni vzporedno izračunajo izhodne vrednosti. Primer take mreže je perceptron [20]. Enonivojske usmerjene nevronske mreže

lahko rešijo samo linearne probleme (npr. "ekskluzivni ali" ni rešljiv z enonivojsko mrežo).

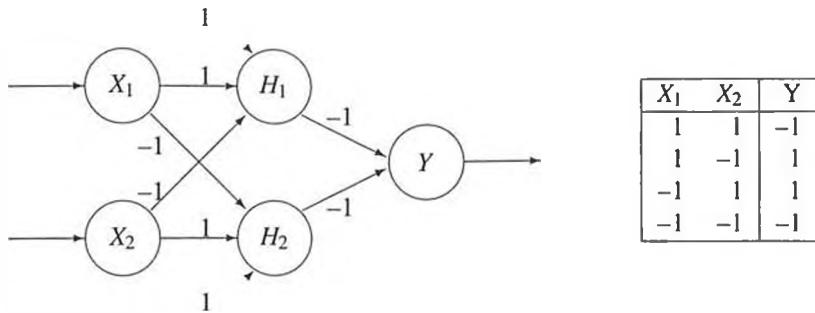


Slika 9.4: Enonivojska usmerjena nevronska mreža s tremi vhodi in dvema (izhodnima) nevronoma.

**Večnivojske usmerjene nevronske mreže** Če enonivojski nevroni med vhodom in izhodnim nivojem dodamo še enega ali več skritih nivojev, dobimo večnivojsko nevronske mrežo. Te mreže delujejo podobno kot enonivojske, le število korakov pri izračunu je enako številu skritih nivojev plus ena. Z večnivojskimi mrežami lahko rešimo tudi nelinearne probleme.

Primer večnivojske mreže, ki izračunava "ekskluzivni ali", je na sliki 9.5. Mreža ima dva vhoda, dva skrita nevona in en izhodni. Skrita nevona imata tudi poseben konstanten vhod z ustrežno 1. Mreža je simetrična glede na vhod. Če sta oba vhoda enaka (oba 1 ali oba -1), sta skrita nevona aktivna (imata izhod 1) in izhodni nevron postane pasiven (izhodna vrednost je -1). Če sta vhoda različna (eden 1 in drugi -1), sta stanji skritih nevronov različni in je izhodni nevron aktivni (vrednost 1).

**Dvosmerni asociativni pomnilnik** Sestavljen je iz dveh nivojev nevronov, ki sta povezana z dvosmernimi vezmi[16, 17]. Izračun poteka tako, da nevronom prvega nivoja vsilimo vhodne vrednosti. Nevroni drugega nivoja izračunajo svoje izhode, zatem nevroni prvega nivoja izračunajo svoje izhode in tako naprej, dokler se izhodi obeh nivojev ne ustalijo. Delovanje je lahko sihrono ali asihrono. Že ime pove, da se tako mreža uporablja kot asociativni pomnilnik, ki izračuna vzorec, ki je bil shranjen kot povezava za vhodni vzorec. Dokazano je, da se pri asinhronem izvajanjtu dvosmerni asociativni pomnilnik vedno ustavi po končnem številu korakov (pri poljubnih utežeh na povezavah med nevroni) [17].



Slika 9.5: Večnivojska nevronska mreža, ki izračunava “ekskluzivni ali”.

### Namen nevronske mreže

**Avtoasociativni pomnilnik** Deluje tako, da so vsi nevroni hkrati vhodni in izhodni. Na vhod damo vzorec ali del vzorca (pomanjkljiv vzorec). Mreža iterativno zgradi (aproksimira) manjkajoči del vzorca in popravlja napake (šum) v vhodnem vzorcu. Ko se izvajanje ustali, dobimo na izhodu cel, dopolnjen in popravljen vzorec. Če je vhodni vzorec preveč pomanjkljiv ali napačen, se lahko zgodi, da je končni vzorec napačen. McEliece in sod. [18] so dokazali, da z  $N$  nevroni lahko eksaktно shranimo  $N/(4 \log N)$   $N$  dimenzionalnih vektorjev. Wong [31] pa je empirično pokazal, da mreža z  $N$  nevroni lahko shrani  $N$  vektorjev iste dimenzije.

**Heteroasociativni pomnilnik** Je varianca avtoasociativnega pomnilnika, pri kateri je vzorec sestavljen iz več podvzorcev. Na vhod damo enega ali več podvzorcev, na izhodu pa dobimo preostali del vzorca.

**Časovni asociativni pomnilnik** Deluje kot heteroasociativni pomnilnik, le da je na vhodu vedno časovno opredeljen vzorec, izhodni vzorec pa je napoved za naslednji časovni interval [17]. Časovni asociativni pomnilnik deluje kot (naivni) kvalitativni simulator, ki na osnovi situacije generira naslednjo situacijo. Učenje poteka na osnovi primerov. Simulator upošteva samo površinsko znanje brez poznavanja zakonov in relacij, ki veljajo v dani domeni. Zato je napovedovanje aproksimacija, ki upošteva predstavljene primere in ne izpeljava iz znanih zakonov.

Časovni asociativni pomnilnik lahko realiziramo tudi tako, da upošteva kontekst in predznanje, kjer opis situacije dopušča več različnih nadaljevanj. V tem primeru je vhod razdeljen na podvzorec situacije in podvzorec konteksta.

**Klasifikacija** Je poseben primer heteroasociativnega pomnilnika z določenimi vhodnimi nevroni in fiksni številom izhodnih nevronov, ki določajo razred. Učenje poteka na

osnovi primerov z znanimi vhodnimi podatki in znanimi razredi. Primer, za katerega so znani samo vhodni podatki ali samo njihov del, se klasificira v enega od razredov.

**Razvrščanje** Tu je problem podan tako, da mora sistem sam razvrstiti vhodne primere v določeno število razredov. Število razredov je lahko podano vnaprej ali ne. Opišimo primer nevronske mreže za razvrščanje [25].

Mreža je sestavljena iz vhodnih in izhodnih nevronov. Vsak nevron je aktiven ali neaktivnen. Vhodni nevroni so povezani z vzbujevalnimi vezmi z izhodnimi nevroni. Število izhodnih nevronov določa število končnih razredov. Izhodni nevroni so povezani z zaviralnimi vezmi z vsemi ostalimi izhodnimi nevroni, tako da je lahko naenkrat aktiven samo en izhodni nevron.

Pred učenjem imajo izhodni nevroni na vezeh z vhodnimi nevroni enake uteži tako, da je vsota vseh uteži enaka 1. Med učenjem izhodni nevron premika uteži od neaktivnih vhodov k aktivnim tako, da vzdržuje vsoto uteži. Sprememba uteži je največja pri izhodnem nevronu, ki postane aktiven (zmaga). Takemu učenju pravimo *tekmovalno učenje* (*competitive learning*). Ščasoma se mreža nauči, da določen izhodni nevron zmaga pri po določenih lastnostih podobnih vzorcih.

Če povežemo več izhodnih skupin nevronov vzporedno brez medsebojnih povezav, dobimo hkrati več različnih razvrstitev. Vsaka izhodna skupina nevronov se nauči razvrstiti vhode glede na določene lastnosti. Rečemo, da se vsak nevron nauči zaznati določen (nov) atribut vhodnega vzorca. Zato takemu učenju pravimo tudi *odkrivanje atributov* (*feature discovery*). Več nivojev detektorjev atributov lahko zaporedoma povezemo. Vsak naslednji nivo dobi vhodne primere, opisane z novimi atributimi.

**Samoorganizacija in urejanje** Včasih je za učinkovito predstavitev informacije potrebno nevrone urediti tako, da vsak odgovarja na določeno vrednost vhoda (npr. frekvence zvoka). Z ustrezno topologijo nevronov dosežemo, da se nevroni sami uredijo tako, da sosedno ležiči nevroni odgovarjajo na podobne vhodne signale. Primer je dvonivojska mreža, kjer je vsak izhodni nevron povezan z bližnjimi izhodnimi nevroni s povratnimi vezmi. Z najbližjimi nevroni je povezan z vzbujevalnimi vezmi, z manj bližnjimi pa z zaviralnimi vezmi [13]. Vsi nevroni sprejemajo na vhodu signal, ki ustreza vrednosti danega vhodnega parametra. Vsak nevron ima spremenljivo utež za vhodni signal, ki jo primerja z vhodnim signalom. Čim manjša je razlika uteži in vhodnega signala, tem močnejši bo nevronov odgovor.

Učenje poteka tako, da se spreminja samo uteži zmagovalnih (aktivnih) nevronov. Uteži se spreminja v smeri vhodnega signala. Ščasoma so v mreži vhodne uteži urejene, torej lega nevrona, ki reagira na dani signal, odgovarja vhodnemu signalu. Kohonen [13] je pod določenimi pogoji pokazal konvergenco tega procesa.

Kohonen navaja dva procesa. Prvič uteži skušajo aproksimirati trenutni signal. Drugič lokalne interakcije med nevroni skušajo ohranjati kontinuiteto. Analogijo lahko najdemo v algoritmu urejanja po metodi mehurčkov. Dva sosednja elementa se podpirata, če sta v pravilnem vrstnem redu. To ustreza drugemu principu. Če nista v pravilnem vrstnem redu, prvi princip prevlada in sosednja elementa se zamenjata.

## Pravilo učenja

**Hebbovo pravilo** Osnovno pravilo učenja, ki ga v več inačicah uporablja nevronske mreže, je definiral že Hebb [6]. Pravilo pravi, da se vez med aktivnima nevronoma ojača (utež se poveča). To zadostuje, da si mreža zapomni frekvenco (in s tem verjetnost), s katero sta sosednja nevrona hkrati aktivna. Biološka verjetnost tega pravila (verjetnost, da se po podobnem pravilu učijo možgani) je v uporabi informacije, ki je nevronom lokalno dosegljiva.

Posplošeno Hebbovo pravilo [25] pravi, da se vez med nevronoma ojača, če sta aktivna oba hkrati ali če sta pasivna oba hkrati. Na ta način si mreža zapomni frekvenco istočasne aktivnosti.

**Pravilo delta** Preprosto pravilo, ki omogoča učenje v enonivojski mreži, sta definirala že Minsky in Papert [20]. V enonivojski mreži so vhodi povezani z izhodni mi nevroni preko uteži. Pred učenjem so uteži naključne. Učenje poteka na podlagi znanih parov vhodnih in izhodnih vzorcev. Mreža na vhodu dobi vzorec in v enem koraku izračuna izhod. Uteži se spremenijo sorazmerno razlike med dejanskim in želenim izhodom. Z iterativno uporabo znanih vhodnih in izhodnih vzorcev se mreža nauči pravilno odgovarjati na vse vhode. Minsky in Papert [20] sta dokazala, da algoritem konvergira k eksaktni rešitvi, če je funkcija, ki se jo mora mreža naučiti, linearja. Enonivojska mreža ne more rešiti nelinearnih problemov (npr. "ekskluzivni ali").

Rumelhart je s sodelavci [24] razvil posplošeno pravilo delta, ki mu pravimo tudi pravilo vzvratnega razširjanja napake (*backpropagation of error*). Le-to omogoča učenje mreže, sestavljene iz poljubnega števila nivojcov. Večnivojska mreža reši tudi nelinearne probleme (npr. "ekskluzivni ali", glej sliko 9.5). Osnovni princip posplošenega pravila delta je enak kot pri navadnem pravilu delta. Na začetku so uteži naključne. Na vhodu mreža dobi vhodni vzorec in po nivojih izračuna izhod. Izračuna se razlika med dejanskim in želenim izhodom. Najprej se spremenijo uteži med zadnjim in predzadnjim nivojem kot pri osnovnemu pravilu delta. Zatem se izračunajo želene vrednosti nevronov na predzadnjem nivoju. Izračuna se razlika med želenimi in dejanskimi vrednostmi nevronov na predzadnjem nivoju in spremjanje uteži se rekurzivno nadaljuje vse do vhoda.

Za posplošeno pravilo delta velja, da ne konvergira vedno k najboljši rešitvi (lahko obtiči v lokalnem minimumu). Zahteva zelo veliko prehodov preko učnih primerov. Mreži moramo velikokrat pokazati iste učne primere, tipično nekaj tisočkrat do nekaj stotisočkrat). Nevronska mreža na sliki 9.5 bi potrebovala okoli 500 prehodov preko štirih učnih primerov za rešitev problema "ekskluzivni ali" [24]. Pravilo delta in posplošeno pravilo delta nimata biološke analogije z možgani [28].

**Tekmovalno pravilo** Zahtevamo, da je v skupini nevronov aktiven vedno natanko eden. To dosežemo z ustrezno topologijo, kjer so v skupini vsi nevroni povezani med seboj z zaviralnimi vezmi. Ko postane nevron aktiven, z zaviralnimi vezmi zmanjša vrednosti ostalim nevronom, da ne morejo postati aktivni. Po tem pravilu se vsakič uči samo zmagovalni nevron tako, da poveča uteži vezem, ki so mu pomagale, da je "zmagal", in

zmanjša uteži ostalim vezem. Primer tekmovalnega pravila je zgoraj opisana metoda razvrščanja Rumelharta in Zipserja [25]. Tudi Kohonenov primer samoorganizacije vključuje tekmovalno pravilo.

**Pozabljanje** Nekatere metode učenja vključujejo tudi pozabljanje tako, da že naučenega ne upoštevajo enakovredno z novim. Na ta način je novo naučeno vedno "najmočnejše". Včasih pozabljanje realizirajo tako, da uteži med nevroni s časom zvezno znižujejo sorazmerno njihovim velikostim. To preprečuje, da bi se uteži preveč povečale.

### Funkcija kombiniranja vhodov nevrona v izhod

Funkcija kombiniranja je sestavljena iz dveh delov (glej sliko 9.2): funkcije aktivacije ( $A$ ) in izhodne funkcije ( $f$ ). Funkcija aktivacije združi vhode v vmesno vrednost. Izhodna funkcija, ki je največkrat pragovna, preslika rezultat aktivacije v izhod nevrona.

**Funkcija aktivacije** Najpogostejsa funkcija aktivacije je utežena vsota:

$$A(X_j) = \sum_i W_{ij} X_i + C_j$$

kjer je  $W_{ij}$  utež vezi med  $i$ -tim in  $j$ -tim nevronom,  $X_i$  stanje  $i$ -tega nevrona in  $C_j$  konstantna aktivacija  $j$ -tega nevrona (prag). Če opišemo stanja  $N$  nevronov z vektorjem

$$X = (X_1, X_2, \dots, X_N)^T$$

in uteži vezi  $W_{ij}$  med nevronoma  $i$  in  $j$  z matriko

$$M = \begin{bmatrix} W_{11} & W_{12} & \dots & W_{1N} \\ W_{21} & W_{22} & \dots & W_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ W_{N1} & W_{N2} & \dots & W_{NN} \end{bmatrix}$$

dobimo stanje  $Y$  nevronske mreže po eni iteraciji z

$$Y = MX$$

Ker matrika  $M$  predstavlja linearno transformacijo, je funkcija aktivacije  $Y = A(X)$  linearна.

Splošnejša je t.i. funkcija *sigma-pi*:

$$sp(X_1, \dots, X_n) = \sum_{S_i \in P} W_i \prod_{k \in S_i} X_k$$

kjer je  $P$  potenčna množica množice indeksov vseh nevronov.  $S_i$  je množica indeksov nevronov, katerih stanja se med seboj zmnožijo in se zmnožek "uteži" z  $W_i$ .

Williams [30] je pokazal, da lahko s funkcijo sigma-pi in s pragovnimi elementi realiziramo poljubno monotono Boolovo funkcijo.

**Izhodna funkcija** Preslika aktivacijo v izhod nevrona. Lahko je deterministična binarna, deterministična zvezna ali stohastična, ki je tudi ponavadi binarna. Pri deterministični funkciji je izhod nevrona enolično določen z vhodom. Pri stohastični funkciji pa je izhod določen samo verjetnostno.

Binarna deterministična funkcija je pragovni element (glej sliko 9.2). Izhodne vrednosti so 0 in 1 ali -1 in 1. Prag je fiksen ali pa se z učenjem spreminja. Zvezna deterministična funkcija je sigmoidne oblike, ki preslika poljubno realno vrednost na interval  $[0, 1]$ . Izhod nevrona je torej realno število med 0 in 1. Zvezna funkcija občutno poveča zmožnost pomnenja nevronske mreže. Guez in sod. [4] so pokazali, da ima mreža z  $N$  nevroni  $3^N$  fiksnih točk. Primer zvezne izhodne funkcije je

$$f(X) = \frac{1}{1 + e^{-X}}$$

V izogib lokalnim ekstremom uporabljajo v nevronske mrežah tudi stohastične funkcije kombiniranja. Te vrnejo verjetnost za eno od vrednosti. Nevron nato izbere vrednost z dano verjetnostjo. Proces konvergence je zato počasnejši. Na ta način se izognemo večini lokalnih ekstremov in sistem z veliko verjetnostjo poišče globalni optimum. Možna je vpeljava temperature sistema po analogiji iz termodinamike. Čim višja je temperatura, tem večja je entropija sistema. Z zniževanjem temperature sistem postaja bolj determinističen. Pri temperaturi 0 so vse odločitve deterministične. Začnemo pri visoki temperaturi, ki jo počasi znižujemo. Čim počasneje se temperatura znižuje, tem večja je verjetnost, da bo sistem pristal v globalnem optimumu. Primer stohastične funkcije je Smolenskyeva teorija harmonije [27]. Harmonija je obratno sorazmerna Hopfieldovi energiji sistema. Čim večja je harmonija, tem nižja je energija in tem bliže je sistem rešitvi. V globalnem optimumu je harmonija največja. Stohastično funkcijo uporablja tudi Hinton in Sejnowski [8] v Boltzmannovih strojih (funkcija sledi Boltzmannovi porazdelitvi).

## 9.3 Perceptron

*Biti globoko ljubljen od nekoga ti daje moč; globoko ljubiti nekoga ti daje pogum.*

*Lao Ce*

### Enonivojski perceptron

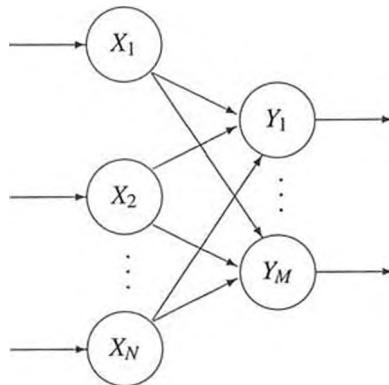
Enonivojski usmerjeni nevronski mreži pravimo tudi *enonivojski perceptron*. Mreža je sestavljena iz skupine vhodov  $X_1, \dots, X_N$  in skupine izhodnih nevronov  $Y_1, \dots, Y_M$  (glej sliko 9.6). Izračun poteka tako, da na vhode postavimo vhodni vzorec  $X = (x_1, \dots, x_N, 1)$ . Zadnja komponenta je konstanta, katere utež predstavlja prag  $\theta$  izhodnega nevrona. Zatem v enem koraku vsi izhodni nevroni vzporedno in neodvisno eden od drugega izračunajo svoje izhodne vrednosti:

$$Y = WX \tag{9.1}$$

kjer je  $W$  matrika uteži. Torej vsak nevron izračunava funkcijo:

$$Y_i = \sum_{j=1}^N W_{ji} X_j + \theta_i \quad (9.2)$$

in je  $W_{ji}$  utež na povezavi med  $j$ -tim vhodnim in  $i$ -tim izhodnim nevronom in  $\theta_i$  prag  $i$ -tega izhodnega nevrona.



Slika 9.6: Enonivojska usmerjena umetna nevronska mreža.

Ker izhodni nevroni izračunavajo svoj izhod neodvisno od drugih izhodnih nevronov, se lahko pri obravnavanju dvonivojske mreže omejimo na samo en izhodni nevron. Zato bo od sedaj naprej  $W$  vektor in ne matrika uteži. Izhodni nevron izračuna svoj izhod po pravilu:

$$Y = W^T X \quad (9.3)$$

Če je vrednost nevrona  $Y \geq 0$ , mreža klasificira primer v prvi razred, oziroma če je  $Y < 0$ , ga klasificira v drugi razred.

Učenje enonivojske mreže pomeni ustrezno nastavitev vektorja uteži  $W$  tako, da mreža pravilno klasificira učne primere. Preprosto pravilo učenja enonivojske mreže je sledeče [22]. Pred začetkom učenja so uteži izbrane naključno (lahko so tudi vse enake 0). Učenje poteka z znanimi pari vhodnih in izhodnih vzorcev. Vhodni nevroni dobijo vhodni vzorec. V enem koraku izračunamo izhod po enačbi (9.3). Če je  $n$ -ti vhodni vzorec  $X(n)$  pravilno klasificiran, se vrednosti uteži ne spremenijo. Če je vzorec napačno klasificiran, se uteži spremenijo po pravilu:

$$W(n+1) = W(n) + \eta X(n) \quad (9.4)$$

če je pravilni razred prvi in je  $W^T(n)X(n) < 0$ , oziroma po pravilu:

$$W(n+1) = W(n) - \eta X(n) \quad (9.5)$$

če je pravilni razred  $n$ -tega vzorca drugi in je  $W^T(n)X(n) \geq 0$ .

$\eta$  je stopnja učenja (learning rate), ki podaja hitrost učnega algoritma. Z iterativnim prikazovanjem vzorcev dosežemo, da se mreža nauči pravilno odgovarjati na vse vhode. Rosenblatt [22, 5] je dokazal, da algoritem konvergira, če je funkcija, ki se jo mora mreža naučiti, linearna.

Enačbi (9.4) in (9.5) lahko nadomestimo z enačbo:

$$W(n+1) = W(n) + \eta \left( d(n) - \text{sgn}(Y(n)) \right) X(n) \quad (9.6)$$

kjer je  $d(n)$  želeni izhod, podan z:

$$d(n) = \begin{cases} +1 & \text{če } X(n) \text{ pripada prvemu razredu} \\ -1 & \text{če } X(n) \text{ pripada drugemu razredu} \end{cases}$$

in  $\text{sgn}(.)$  funkcija predznaka, podana z:

$$\text{sgn}(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Inačica učnega pravila (9.6) je pravilo delta, ki upošteva dejansko razliko med izhodom in želenim izhodom:

$$\begin{aligned} W(n+1) &= W(n) + \eta \left( d(n) - Y(n) \right) X(n) \\ W(n+1) &= W(n) + \eta \left( d(n) - W^T(n)X(n) \right) X(n) \end{aligned} \quad (9.7)$$

Temu pravilu pravimo tudi gradientno pravilo, saj se vektor uteži spremeni v smeri manjše napake oz. v smeri gradijeta funkcije. Kvadrat napake  $E(n)$  je namreč podan z:

$$E(n) = \left( d(n) - W^T(n)X(n) \right)^2$$

in je odvod napake enak

$$\frac{dE(n)}{dW(n)} = -2 \left( d(n) - W^T(n)X(n) \right) X(n)$$

Pravilo (9.7) imenujemo tudi učenje po principu najmanjše srednje kvadratne napake (least-mean-square), ker v limiti (po neskončno iteracijah) in pri zadosti majhni stopnji učenja  $\eta$  konvergira v optimalno mrežo po kriteriju srednje kvadratne napake [5] (prevelik  $\eta$  lahko povzroči preskok optimalne rešitve oziroma lahko učenje oscilira okoli optimuma).

Zanimiva je paketna različica pravila delta. Za mrežo pri trenutnem vektorju uteži  $W(n)$  izračunamo razliko med želenim in dejanskim izhodom za vse učne primere,  $X(1), \dots, X(m)$ , (ne samo za enega kot prej) in šele zatem spremenimo uteži v smeri negativnega odvoda. Napaka je podana z:

$$E(n) = \sum_{i=1}^m \left( d(i) - W^T(n)X(i) \right)^2$$

odvod napake pa z:

$$\frac{dE(n)}{dW(n)} = -2 \sum_{i=1}^m \left( d(i) - W^T(n)X(i) \right) X(i)$$

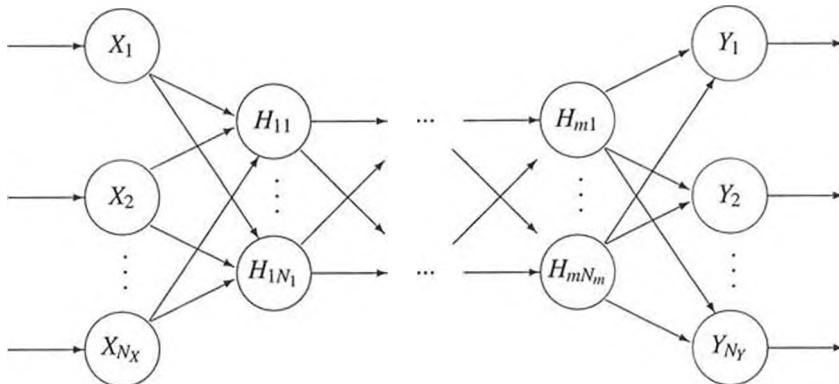
Paketno pravilo delta je torej:

$$W(n+1) = W(n) - \eta \frac{dE(n)}{dW(n)} = W(n) + \eta \sum_{i=1}^m \left( d(i) - W^T(n)X(i) \right) X(i) \quad (9.8)$$

Paketno pravilo upošteva več informacije naenkrat in zato ponavadi hitreje konvergira. Ker pa zahteva učni algoritem izračun napake za vse učne primere, je to pravilo neprimerno za implementacijo z nevronskimi mrežami.

### Večnivojski perceptron

Večnivojske nevronske mreže delujejo podobno kot enonivojske, le da je število korakov pri izračunu enako številu skritih nivojev plus ena (glej sliko 9.7). Z večnivojskimi mrežami rešimo tudi nelinearne probleme. Rumelhart je s sodelavci [24] razvil *posplošeno pravilo*



Slika 9.7: Usmerjena večnivojska umetna nevronska mreža.

*delta*, ki mu pravimo tudi *pravilo vzvratnega razširjanja napake* (backpropagation of error). To omogoča učenje mreže, sestavljene iz poljubnega števila nivojev.

Osnovni princip posplošenega pravila delta je enak kot pri navadnem pravilu delta. Ker moramo izračunavati odvod napake tudi pri skritih nivojih in ne samo pri izhodnih nevronih, je potrebno, da je izhodna funkcija zvezna in zvezno odvedljiva. Uporablja se sigmoidna funkcija:

$$f(X) = \frac{1}{1 + e^{-X}}$$

ki je predstavljena na sliki 4.2. Odvod funkcije  $f$  je:

$$f'(X) = \frac{e^{-X}}{(1 + e^{-X})^2} = f(X)(1 - f(X))$$

Naj bo  $m > 0$  število skritih nivojev, vsak z  $N_k$  nevroni,  $1 \leq k \leq m$ . Naj bo  $N_X$  velikost vhodnega vzorca,  $N_Y$  pa število izhodnih nevronov. Zaradi preprostosti je v nadaljevanju izpuščen konstanten vhod (prag).

Nevron na prvem skritem nivoju izračunava svoj izhod pri učnem primeru  $X(n)$  po enačbi:

$$H_{1i}(n) = f\left(\sum_{j=1}^{N_X} W_{ji}^{(1)}(n)X_j(n)\right) \quad (9.9)$$

kjer oznaka  $W^{(k)}$  pomeni matriko uteži sinaps, ki vodijo v  $k$ -ti skriti nivo nevronov. Definirajmo še

$$A_{1i}(n) = \sum_{j=1}^{N_X} W_{ji}^{(1)}(n)X_j(n)$$

Na  $k$ -tem skritem nivoju,  $k > 1$ , nevroni izračunavajo svoj izhod po enačbi

$$H_{ki}(n) = f(A_{ki}(n)) \quad (9.10)$$

kjer so aktivacijski nivoji izračunani iz izhodov  $H_{k-1,j}$  predhodnega nivoja:

$$A_{ki}(n) = \sum_{j=1}^{N_k} W_{ji}^{(k)}(n)H_{k-1,j}(n)$$

Izhodni nevroni pa izračunavajo svoj izhod po enačbi:

$$Y_i(n) = f(A_i(n)) \quad (9.11)$$

kjer je

$$A_i(n) = \sum_{j=1}^{N_m} W_{ji}^{(m+1)}(n)H_{m,j}(n)$$

Da izpeljemo posplošeno pravilo delta, moramo izračunati napake izhodnih nevronov. Napaka  $i$ -tega izhodnega nevrone za dani  $n$ -ti učni primer  $X(n)$  je podana z:

$$e_i(n) = d(n) - Y_i(n)$$

Napako za celotno mrežo izračunamo kot polovico vsote kvadratov napak po vseh izhodnih nevronih:

$$E(n) = \frac{1}{2} \sum_{i=1}^{N_Y} e_i^2(n)$$

Uteži moramo spremeniti v smeri, ki minimizira napako. To smer določa gradient funkcije. V ta namen se uporablja odvod sigmoidne funkcije. Celotna izpeljava je podana v zaključnem razdelku tega poglavja na str. 203. Tu podajmo le končne formule.

Po pravilu delta je popravek uteži  $W^{(m+1)}$  pri izhodnih nevronih podan z:

$$W_{ji}^{(m+1)}(n+1) = W_{ji}^{(m+1)}(n) + \eta e_i(n) f'(A_i(n)) H_{m,j}(n)$$

Za zadnji skriti nivo je popravek:

$$W_{ji}^{(m)}(n+1) = W_{ji}^{(m)}(n) - \eta f'(A_{mi}(n)) \left( \sum_{l=1}^{N_y} \frac{\partial E(n)}{\partial A_l(n)} W_{il}^{(m+1)}(n) \right) H_{m-1,j}(n)$$

Za  $k$ -ti skriti nivo,  $1 < k < m$  je popravek:

$$W_{ji}^{(k)}(n+1) = W_{ji}^{(k)}(n) - \eta f'(A_{ki}(n)) \left( \sum_{l=1}^{N_{k+1}} \frac{\partial E(n)}{\partial A_{k+1,l}(n)} W_{il}^{(k+1)}(n) \right) H_{k-1,j}(n)$$

in za prvi skriti nivo ( $k = 1$ ) je popravek:

$$W_{ji}^{(1)}(n+1) = W_{ji}^{(1)}(n) - \eta f'(A_{1i}(n)) \left( \sum_{l=1}^{N_2} \frac{\partial E(n)}{\partial A_{2,l}(n)} W_{il}^{(2)}(n) \right) X_j(n)$$

### Slabosti posplošenega pravila delta

Osnovna oblika posplošenega pravila delta ima več slabosti:

1. Velja, da uteži ne konvergirajo vedno k optimalni mreži (lahko obtičimo v lokalnem minimumu).
2. Problematična je izbira topologije mreže, saj je treba empirično določiti ustrezno število skritih nivojev in število skritih nevronov na vsakem nivoju.
3. Zaradi prevelikega prilagajanja učni množici je potrebno učenje ustaviti, ko začne napaka na neodvisni testni množici naraščati, kar se zgodi prej, kot preneha padanje napake na učni množici. Določiti pravi trenutek za ustavitev učenja zahteva časovno zahtevno empirično testiranje.
4. Nastavitev parametra  $\eta$  vpliva na stabilnost (konvergenco) učenja in ga je potrebno določiti empirično.
5. Zahteva zelo veliko število prehodov preko učnih primerov, kar pomeni, da moramo mreži pokazati iste učne primere nekaj tisoč do nekaj stotisočkrat).
6. Nekateri štejejo kot slabost tudi to, da pravilo delta in posplošeno pravilo delta nimata biološke analogije z možgani [28]. To je slabost samo, če želimo dokaj verno oponašati biološke sisteme, kar v tehniki večinoma ne drži.

Prve tri probleme omilimo z ustreznimi izboljšavami posplošenega pravila delta. Zadnjim trem težavam pa se ni moč izogniti. Če postanejo nesprejemljivi, je potrebno izbrati drugačno vrsto nevronske mreže ali drugo metodo strojnega učenja.

Problem lokalnega optimuma omilimo, če vpeljemo v pravilo učenja:

$$W_{ji}(n+1) = W_{ji}(n) - \Delta W_{ji}(n+1)$$

*momentni člen*, tako da pri novi spremembi uteži upoštevamo tudi prejšnjo spremembo:

$$W_{ji}(n+1) = W_{ji}(n) - \Delta W_{ji}(n+1) - \alpha \Delta W_{ji}(n)$$

Parameter  $\alpha$  določa vpliv prejšnje spremembe na novo spremembo. Na ta način učno pravilo pri manjših lokalnih optimumih vztraja vsaj še nekaj časa v smeri, ki so jo določile prejšnje iteracije in se morda reši iz lokalnega optimuma.

Izbire topologije in preprečevanja prevelikega prilagajanja učni množici se lotimo na različne načine. Zanimiva je metoda eliminacije uteži, ki v funkcijo napake doda člen, ki kaznuje velike uteži. Na ta način učenje teži k manjšim utežem. Ko utež na sinapsi postane zadost majhna, sinapso odstranimo. Na ta način odstranimo tudi odvečne skrite nevrone, lahko pa tudi odvečne vhodne nevrone in s tem nepotrebne atribute, čeprav to ni osnovni namen metode. Z eliminacijo uteži rešimo oba problema hkrati. Začnemo s preveliko mrežo, med učenjem pa se odstranijo odvečni nevroni. Zaradi avtomatske nastavitev optimalne velikosti mreže se izognemo tudi prevelikemu prilagajanju učni množici. Slabost metode so dodatni parametri za kontrolo učenja, ki jih ni enostavnono nastaviti (glej npr. [5]).

## 9.4 Izpeljave in dokazi

**Izpeljava vzvratnega razširjanja napake** Parcialni odvod napake celotne mreže po uteži  $W_{ji}^{(m+1)}$  zapišemo po verižnem pravilu:

$$\frac{\partial E(n)}{\partial W_{ji}^{(m+1)}(n)} = \frac{\partial E(n)}{\partial e_i(n)} \frac{\partial e_i(n)}{\partial Y_i(n)} \frac{\partial Y_i(n)}{\partial A_i(n)} \frac{\partial A_i(n)}{\partial W_{ji}^{(m+1)}(n)}$$

Posamezni odvodi so:

$$\frac{\partial E(n)}{\partial e_i(n)} = e_i(n)$$

$$\frac{\partial e_i(n)}{\partial Y_i(n)} = -1$$

$$\frac{\partial Y_i(n)}{\partial A_i(n)} = f'(A_i(n))$$

$$\frac{\partial A_i(n)}{\partial W_{ji}^{(m+1)}(n)} = H_{mj}(n)$$

Po pravilu delta je popravek uteži  $W^{(m+1)}$  pri izhodnih nevronih podan z:

$$W_{ji}^{(m+1)}(n+1) = W_{ji}^{(m+1)}(n) - \eta \frac{\partial E(n)}{\partial W_{ji}^{(m+1)}(n)}$$

$$W_{ji}^{(m+1)}(n+1) = W_{ji}^{(m+1)}(n) - \eta \frac{\partial E(n)}{\partial A_i(n)} H_{mj}(n)$$

$$W_{ji}^{(m+1)}(n+1) = W_{ji}^{(m+1)}(n) + \eta e_i(n) f'(A_i(n)) H_{mj}(n)$$

Za popravek uteži pri skritih nevronih napake  $e_i$  ne moremo direktno izračunati. Pomačemo si z napakami izhodnih nevronov, ki jih propagiramo do skritih nevronov tako, da izračunamo parcialne odvode napake po izhodnih vrednostih skritega nevrona.

$$\frac{\partial E(n)}{\partial A_{mi}(n)} = \frac{\partial E(n)}{\partial H_{mi}(n)} \frac{\partial H_{mi}(n)}{\partial A_{mi}(n)}$$

Drugi faktor je enak:

$$\frac{\partial H_{mi}(n)}{\partial A_{mi}(n)} = f'(A_{mi}(n))$$

Prvi faktor pa izračunamo po verižnem pravilu:

$$\begin{aligned} \frac{\partial E(n)}{\partial H_{mi}(n)} &= \sum_{j=1}^{N_Y} \frac{\partial E(n)}{\partial A_j(n)} \frac{\partial A_j(n)}{\partial H_{mi}(n)} \\ \frac{\partial E(n)}{\partial H_{mi}(n)} &= \sum_{j=1}^{N_Y} \frac{\partial E(n)}{\partial A_j(n)} \frac{\partial \left( \sum_{l=1}^{N_m} W_{lj}^{(m+1)}(n) H_{ml}(n) \right)}{\partial H_{mi}(n)} \\ \frac{\partial E(n)}{\partial H_{mi}(n)} &= \sum_{j=1}^{N_Y} \frac{\partial E(n)}{\partial A_j(n)} W_{ij}^{(m+1)}(n) \end{aligned}$$

Torej za zadnji ( $m$ -ti) skriti nivo dobimo:

$$\frac{\partial E(n)}{\partial A_{mi}(n)} = f'(A_{mi}(n)) \sum_{j=1}^{N_Y} \frac{\partial E(n)}{\partial A_j(n)} W_{ij}^{(m+1)}(n)$$

Ker je za zadnji skriti nivo odvod enak

$$\frac{\partial A_{mi}(n)}{\partial W_{jl}^{(m)}(n)} = H_{m-1,j}(n)$$

je pravilo delta zanj:

$$\begin{aligned} W_{ji}^{(m)}(n+1) &= W_{ji}^{(m)}(n) - \eta \frac{\partial E(n)}{\partial W_{ji}^{(m)}(n)} \\ W_{ji}^{(m)}(n+1) &= W_{ji}^{(m)}(n) - \eta \frac{\partial E(n)}{\partial A_{mi}(n)} H_{m-1,j}(n) \\ W_{ji}^{(m)}(n+1) &= W_{ji}^{(m)}(n) - \eta f'(A_{mi}(n)) \left( \sum_{l=1}^{N_Y} \frac{\partial E(n)}{\partial A_l(n)} W_{il}^{(m+1)}(n) \right) H_{m-1,j}(n) \end{aligned}$$

Za  $k$ -ti skriti nivo,  $1 < k < m$ , velja:

$$\frac{\partial A_{ki}(n)}{\partial W_{ji}^{(k)}(n)} = H_{k-1,j}(n)$$

in je torej pravilo delta zanj:

$$W_{ji}^{(k)}(n+1) = W_{ji}^{(k)}(n) - \eta f'(A_{ki}(n)) \left( \sum_{l=1}^{N_{k+1}} \frac{\partial E(n)}{\partial A_{k+1,l}(n)} W_{il}^{(k+1)}(n) \right) H_{k-1,j}(n)$$

Za prvi skriti nivo ( $k = 1$ ) velja:

$$\frac{\partial A_{1i}(n)}{\partial W_{ji}^{(1)}(n)} = X_j(n)$$

in je zanj pravilo delta:

$$W_{ji}^{(1)}(n+1) = W_{ji}^{(1)}(n) - \eta f'(A_{1i}(n)) \left( \sum_{l=1}^{N_2} \frac{\partial E(n)}{\partial A_{2,l}(n)} W_{il}^{(2)}(n) \right) X_j(n)$$

□

## Literatura

- [1] A. G. Barto, R. S. Sutton, C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.
- [2] K. Fukushima. A neural network for visual pattern recognition. *IEEE Computer*, str. 65–75, 3 1988.
- [3] I. Grabec, W. Sachse. Application of an intelligent signal processing system to acoustic emission analysis. Report #6428. Dep. of Theoretical and Applied Mechanics, Cornell University, Ithaca, New York, 1988.
- [4] A. Guez, V. Protopopescu, J. Barhen. On the stability, storage capacity and design of nonlinear continuous neural networks. *IEEE Trans. on Systems, Man, and Cybernetics*, 18(1):80–87, 1988.
- [5] S. Haykin. *Neural Networks: A Comprehensive Foundation*. New York: Macmillian College Publ. Comp, 1994.
- [6] D. O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.
- [7] G. E. Hinton. A parallel computation that assigns canonical object based frames of reference. V Proc. 7th Int. Joint Conf. on AI IJCAI-81, 1981.

- [8] G. E. Hinton, J. L. McClelland, D. E. Rumelhart. Distributed representations. V D.E. Rumelhart, J.L. McClelland, (ur.), *Parallel Distributed Processing, Foundations*, volume 1. Cambridge: MIT Press, 1986.
- [9] G. E. Hinton, T. J. Sejnowski. Learning and relearning in Boltzmann machines. V D.E. Rumelhart, J.L. McClelland, (ur.), *Parallel Distributed Processing, Foundations*, volume 1. Cambridge: MIT Press, 1986.
- [10] G. Hood. Neural modeling as one approach to machine learning. V T.M. Mitchell, J.G. Carbonell, R.S. Michalski, (ur.), *Machine Learning – A Guide to Current Research*. Kluwer Academic Publishers, 1986.
- [11] J. J. Hopfield, D. W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.
- [12] M. I. Jordan. An introduction to linear algebra in parallel distributed processing. *Parallel Distributed Processing. Foundations*, 1, 1986.
- [13] T. Kohonen. *Self-Organization and Associative Memory*. Berlin: Springer-Verlag, 1984.
- [14] T. Kohonen. An introduction to neural computing. *Neural Networks*, 1:3–16, 1988.
- [15] T. Kohonen. The neural phonetic typewriter. *IEEE Computer*, str. 11–22, 3 1988.
- [16] B. Kosko. Constructing an associative memory. *Byte*, str. 137–144, 9 1987.
- [17] B. Kosko. Bidirectional associative memories. *IEEE Trans. on Systems, Man, and Cybernetics*, 18(1):49–50, 1988.
- [18] R. J. McEliece, E. C. Posner, E. R. Rodemich, S. S. Venkatesh. The capacity of the Hopfield assiciative memory. *IEEE Trans. on Information Theory*, IT-33(4):461–482, 1987.
- [19] D. Michie, R. A. Chambers. BOXES: An experiment in adaptive control. V E. Dale, D. Michie, (ur.), *Machine Intelligence 2*. Edinburgh: Oliver and Boyd, 1968.
- [20] M. Minsky, S. Papert. *Perceptrons*. Cambridge, MA: MIT Press, 1969.
- [21] M. Pazzani, M. Dyer. A comparison of concept identification in human learning and network learning with generalized delta rule. V *10th Int. Conf. on Artificial Intelligence*, str. 147–150, 1987. Milano, August 23–28.
- [22] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, Washington, DC, 1962.
- [23] D. E. Rumelhart, G. E. Hinton, J. L. McClelland. A general framework for parallel distributed processing. V D. E. Rumelhart, J. L. McClelland, (ur.), *Parallel Distributed Processing: Foundations*, volume 1. Cambridge: MIT Press, 1986.
- [24] D. E. Rumelhart, G. E. Hinton, R. J. Williams. Learning internal representations by error propagation. V D. E. Rumelhart, J. L. McClelland, (ur.), *Parallel Distributed Processing: Foundations*, volume 1. Cambridge: MIT Press, 1986.

- [25] D. E. Rumelhart, J. L. McClelland, (ur.). *Parallel Distributed Processing: Foundations*, volume 1. Cambridge: MIT Press, 1986.
- [26] P. Russell. *The Brain Book: Know Your Own Mind and How to Use It*. Routledge and Kegan Paul, London & Hawthorne, New York, 1979.
- [27] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. V D.E. Rumelhart, J.L. McClelland, (ur.), *Parallel Distributed Processing: Foundations*, volume 1. Cambridge: MIT Press, 1986.
- [28] P. D. Wasserman, T. Schwartz. Neural networks: What are they and why are everybody so interested in them now. Part 1: Winter 1987, pp. 10–12, part 2: Spring 1988, pp. 10–15. *IEEE Expert*, 1987, 1988.
- [29] B. Widrow, R. Winter. Neural nets for adaptive filtering and adaptive pattern recognition. *IEEE Computer*, str. 25–39, 3 1988.
- [30] R. J. Williams. The logic of activation functions. V D.E. Rumelhart, J.L. McClelland, (ur.), *Parallel Distributed Processing: Foundations*, volume 1. Cambridge: MIT Press, 1986.
- [31] A. J. W. Wong. Recognition of general patterns using neural networks. *Biological Cybernetics*, 58(6):361–372, 1988.

## Poglavlje 10

# Verjetnostno modeliranje

*Je nesmiselno  
pravi pamet  
Nemogoče je  
pravi izkušnja  
Je kar je  
pravi ljubezen*

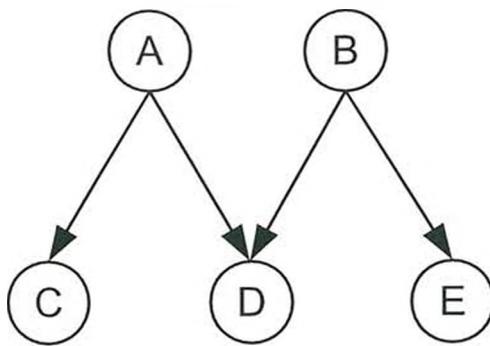
Erich Fried

V tem poglavju predstavljamo verjetnostno modeliranje, ki je danes razširjeno daleč prek meja, ki jih pokriva področje umetne inteligence. Verjetnostni račun in statistika se uporablja na številnih področjih umetne inteligence in računalništva nasploh ter mu marsikdaj dajeta potreben trdni temelj. V dodatku A podajamo kratek povzetek pojmov in rezultatov, pomembnih za naše nadaljnje izvajanje. V tem poglavju se osredotočamo predvsem na grafične modele, oziroma bolj specifično na Bayesove mreže in skrite markovske modele. Podamo pravzaprav le uvod v to področje in preskočimo večino težij in zahtevnejših tematik. Kljub temu proti koncu poglavja omenimo tudi avtomatsko učenje Bayesovih modelov.

Več na splošno o verjetnostnem modeliranju najde bralec v splošnih delih s področja umetne inteligence npr. [4, 6], za poglobljen študij pa so primerna dela [3, 1, 5].

### 10.1 Bayesove mreže

Bayesove mreže (*BBN - Bayesian Belief Networks*) so verjetnostni grafični model, s katerim lahko predstavimo odvisnosti med slučajnimi spremenljivkami. Odvisnosti so predstavljene z usmerjenim acikličnim grafom. V vozliščih so slučajne spremenljivke, ki predstavljajo bodisi dejstvo ali hipotezo, povezave pa predstavljajo odvisnosti med spremenljivkami. Pogosta je na primer raba v medicini, kjer Bayesova mreža predstavlja povezave med boleznjijo in simptomi. Glede na opažene simptome mreža pomaga izračunati verjetnosti posameznih bolezni.



Slika 10.1: Primer Bayesove mreže.

Preprost primer Bayesove mreže prikazuje slika 10.1, kjer je vozlišče  $C$  odvisno od  $A$ , vozlišče  $D$  od  $A$  in  $B$ , vozlišče  $E$  pa od  $B$ . Nepovezana vozlišča niso odvisna, npr.  $A$  ni odvisno od  $B$  pa tudi  $A$  in  $E$  sta neodvisna.

Vozliščem pripisemo verjetnosti glede na odvisnosti. Zaradi enostavnnejšega zapisa bomo v nadaljevanju namesto  $P(X = \text{true})$  in  $P(X = \text{false})$  pisali  $P(X)$  in  $P(\neg X)$ , podobno pa bomo okrajšali tudi pogojne verjetnosti. Za naš preprost primer zadošča, da podamo naslednje verjetnosti:

$$\begin{array}{ll}
 P(A) = 0.1 & P(D|A \wedge B) = 0.5 \\
 P(B) = 0.7 & P(D|A \wedge \neg B) = 0.4 \\
 P(C|A) = 0.2 & P(D|\neg A \wedge B) = 0.2 \\
 P(C|\neg A) = 0.4 & P(D|\neg A \wedge \neg B) = 0.0001 \\
 P(E|B) = 0.2 & \\
 P(E|\neg B) = 0.1 &
 \end{array}$$

Enostavneje to predstavimo s tabelami pogojnih verjetnosti, kot to prikazuje tabela 10.1.

$P(A)$		$P(B)$		$P(D)$	
				true	false
0.1		0.7		0.5	
				0.4	
				0.2	
				0.0001	

$A$	$P(C)$	$B$	$P(E)$
true	0.2	true	0.2
false	0.4	false	0.1

Tabela 10.1: Tabele (pogojnih) verjetnosti za Bayesovo mrežo s slike 10.1.

Iz tabel pogojnih verjetnosti lahko razberemo tudi verjetnostni značaj modeliranja dogodkov z Bayesovimi mrežami. Na primer, vsota obeh pogojnih verjetnosti v tabeli  $P(C|A)$  ni

enaka 1 ( $0.2 + 0.4 = 0.6 < 1$ ), kar pomeni da obstajajo za  $C$  tudi drugi, neopredeljeni vzroki, ki niso zajeti v mreži. Seveda pa velja  $P(C|A) + P(\neg C|A) = 0.2 + 0.8 = 1$  in  $P(C|\neg A) + P(\neg C|\neg A) = 0.4 + 0.6 = 1$ .

Opozorimo še, da se v tem poglavju v zapisu verjetnosti več spremenljivk večinoma uporabljam vejico, na primer  $P(C|(A \cap B))$  ali  $P(C|(AB))$  zapišemo kot  $P(C|A, B)$ .

### Skupna verjetnostna porazdelitev

Poznavanje skupne verjetnostne porazdelitve vseh  $N$  slučajnih spremenljivk v Bayesovi mreži nam omogoči, da sklepamo o vseh dogodkih v dani domeni. Da bi to lahko dosegli, je potrebno poznati in v tabeli hraniti  $2^N$  verjetnosti, sklepanje pa vzame  $O(2^N)$  časa. Za malce večje mreže bi bil zato direkten izračun s tabelo skupne verjetnostne porazdelitve vsaj nepraktičen, če že ne nemogoč.

Stvar pa ni tako črna, kot smo pravkar prikazali. Z uporabo Bayesovega pravila,  $P(A|B) = P(A \cap B)/P(B)$ , lahko izraz razbijemo, npr.  $P(A, B, C, D, E) = P(E|A, B, C, D) \cdot P(A, B, C, D)$ .

Ker mreža hrani pogojne odvisnosti, lahko za mrežo s slike 10.1 z zaporedno uporabo Bayesovega pravila dobimo

$$P(A, B, C, D, E) = P(E|A, B, C, D) \cdot P(D|A, B, C) \cdot P(C|A, B) \cdot P(B|A) \cdot P(A).$$

Mreža določa tudi odvisnosti, npr. spremenljivka  $E$  ni odvisna od  $A, C$  in  $D$ . Na tej podlagi izraz poenostavimo, denimo  $P(E|A, B, C, D) = P(E|B)$ . Upoštevaje vse odvisnosti dobimo  $P(A, B, C, D, E) = P(E|B) \cdot P(D|A, B) \cdot P(C|A) \cdot P(B) \cdot P(A)$ . Opredeljene vrednosti lahko tudi izračunamo s pomočjo verjetnosti iz tabel, npr.  $P(A, \neg B, \neg C, D, E) = P(E|\neg B) \cdot P(D|A, \neg B) \cdot P(\neg C|A) \cdot P(\neg B) \cdot P(A) = 0.1 \cdot 0.4 \cdot (1 - 0.2) \cdot (1 - 0.7) \cdot 0.1 = 0.00096$ .

Upoštevanje odvisnosti iz mreže bistveno poenostavi izračun in nam omogoči, da v tabelah pogojnih verjetnosti hranimo le res potrebne verjetnosti. Na podlagi mreže skupno porazdelitev izračunamo kot

$$P(x_1, x_2, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{predhodniki}(X_i)).$$

### Urejenost vozlišč v mreži

Prepričajmo se, da je zahtevnost izračuna odvisna od vrstnega reda pogojev. V prejšnjem primeru bi lahko vrstni red pogojev zapisali tudi drugače, saj sta unija in presek dogodkov komutativna. Denimo, da spremenljivke preuredimo in poskušamo izračunati  $P(E, D, C, B, A)$ . Po poenostavitvi z Bayesovim pravilom dobimo

$$P(E, D, C, B, A) = P(A|E, D, C, B) \cdot P(B|E, D, C) \cdot P(C|E, D) \cdot P(D|E) \cdot P(E).$$

Izračun ni mogoč, npr.  $D$  je odvisno od  $E$ , med našimi podatki pa manjka  $P(D|E)$ . Podobno velja tudi za druge odvisnosti, npr.  $C$  od  $E$  in  $D, \dots$

Za uspešen izračun skupne verjetnosti morajo biti vozlišča urejena glede na odvisnost: če je  $X$  odvisen od  $Y$ , mora  $X$  v izračunu nastopati pred  $Y$ .

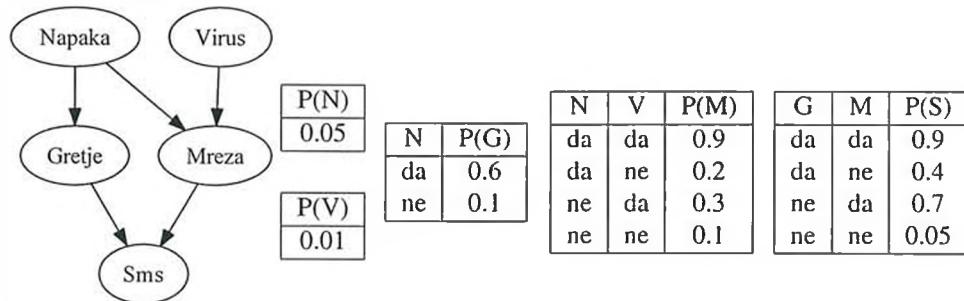
V primeru mreže s slike 10.1 lahko uporabimo poljuben vrstni red, kjer A in B nastopata pred C, D in E, npr. B, A, E, D, C.

Mreža in odvisnosti v njej nam omogočajo, da učinkovito hranim skupno verjetnostno porazdelitev. Tako ne hranim nepotrebnih pogojnih verjetnosti, denimo na primeru s slike 10.1 ni potrebno hraniti  $P(E|A)$ , saj E ni odvisna od A in zato ta pogojna verjetnost ne bo nastopala v nobenem izračunu. Na ta način se z mrežo izognemo eksponentni prostorski in časovni zahtevnosti, ko bi bilo potrebno hraniti tabelo z  $2^N$  vrsticami in sklepati z njo.

Podoben razmislek kot pri izračunu upoštevamo že pri konstrukciji mreže. Upoštevamo smislen vrstni red ter vozlišča v mrežo postavimo tako, da vzrokom sledijo posledice. V realnih primerih je možnih soodvisnosti in vzrokov mnogo, zato dobro pretehtamo, kaj se splača vključiti v mrežo. Poglejmo si primer izračuna in premisleka pri sestavi mreže.

Denimo, da sistemski operater v računalniškem centru dobi na mobilni telefon SMS sporočilo o nepravilnem delovanju, če se pojavi previelik promet na mreži ali če se kateri od računalnikov pregrevata. Preobremenitev mreže lahko zakrvi računalniški virus ali napaka pri delovanju katerega od programov. Napake pri delovanju programov pa lahko povzročijo tudi pregrevanje računalnikov. Za te dogodke so možni tudi drugi vzroki, ki pa jih je težko določiti.

Sistemec se vseh teh vzrokov zaveda, a ima že nekaj izkušenj z Bayesovimi mrežami. Ve, da se težko opredeljivih vzrokov ne izplača vključevati v mrežo, saj model ni eksakten. Ker gre za verjetnostno opredeljen model, jih lahko implicitno vključi. Po premislu si je zato sestavil Bayesovo mrežo in pripadajoče tabele pogojnih verjetnosti na sliki 10.2. Dogodke je okrajšal kar z začetnicami (N - napaka pri delovanju programov, V - virus, G - gretje računalnikov, M - mrežna napaka, S - SMS sporočilo).



Slika 10.2: Bayesova mreža za primer sistemskega operaterja.

Kako so vključeni neopredeljeni vzroki, lahko vidimo iz tabele, ki podaja pogojne verjetnosti  $P(G|N)$ . Vsota stolpičev pogojnih verjetnosti  $P(G|N)$ ,  $P(M|N,V)$  in  $P(S|G,M)$  ni nikjer enaka 1. Vključimo le dogodke, ki jih lahko ustrezno opredelimo glede na znanje, izkušnje in/ali podatke.

Izračunajmo verjetnost dogodka, da dobi operater SMS sporočilo in je tega kriv prevelik

promet na mreži, ki ga je zakrivil virus, medtem ko vsi programi delujejo pravilno in se računalniki ne pregrevajo.

Izračunati želimo  $P(S, M, \neg G, V, \neg N)$ . Z zaporedno uporabo Bayesovega pravila dobimo  $P(S, M, \neg G, V, \neg N) = P(S|M, \neg G, V, \neg N) \cdot P(M|\neg G, V, \neg N) \cdot P(\neg G|V, \neg N) \cdot P(V|\neg N) \cdot P(\neg N)$ .

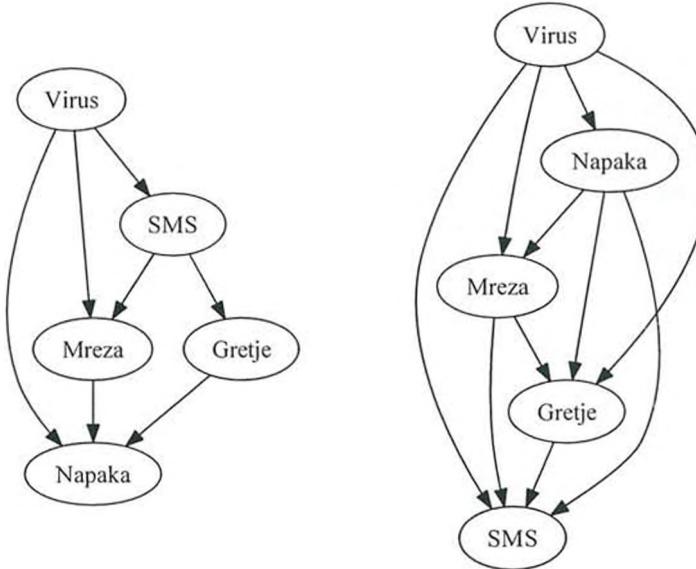
Spremenljivke smo že razvrstili tako, da se predhodniki v mreži pojavljajo kasneje, zato lahko upoštevamo odvisnosti in izraz poenostavimo. Dobimo

$$P(S, M, \neg G, V, \neg N) = P(S|M, \neg G) \cdot P(M|V, \neg N) \cdot P(\neg G|\neg N) \cdot P(V) \cdot P(\neg N).$$

Ker lahko vse verjetnosti preberemo iz tabel, zdaj izračunamo

$$P(S, M, \neg G, V, \neg N) = 0.7 \cdot 0.3 \cdot (1 - 0.1) \cdot 0.01 \cdot (1 - 0.05) \approx 0.0018.$$

Z drugačnim razmislekom bi lahko sestavili tudi drugačno Bayesovo mrežo. Pri avtomatskem dodajanju je mreža odvisna od vrstnega reda dodajanja. Dobili bi npr. mreži na sliki 10.3.



Slika 10.3: Dve mogoči Bayesovi mreži za primer sistemskoga operaterja.

Obe mreži bi zahtevali specifikacijo precej več pogojnih verjetnosti. Sestava dobre mreže zaenkrat še ni zadovoljivo avtomatizirana, čeprav obstaja precej poskusov. Kot kriterij za kvaliteto mreže se večinoma uporablja posteriorna verjetnost mreže pri danih podatkih. Če naj bi mreža pomagala tudi pri razumevanju problema in določala vzročne odvisnosti, pa se zdi, da je nepogrešljivo tudi sodelovanje človeških ekspertov za dano področje.

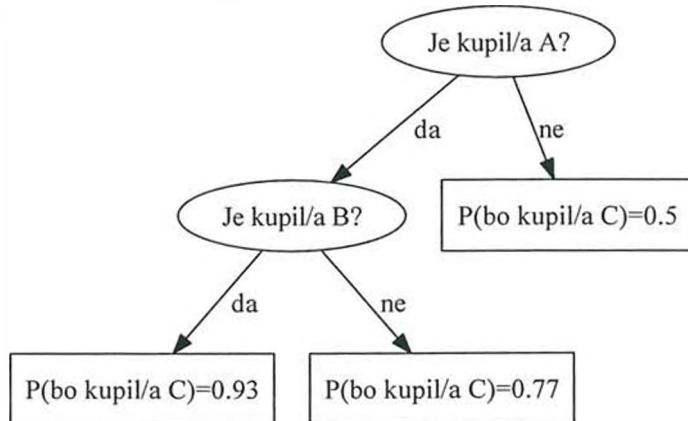
### Skupinsko filtriranje - bayesovski pristop

Zanimiva uporaba bayesovskega sklepanja je v problemu skupinskega filtriranja (*collaborative filtering*). Pri tem gre za izločanje koristnih vzorcev s sodelovanjem večih virov informacij (npr. ljudi, agentov, podatkovnih virov). Primer takšne aplikacije je generiranje priporočil za nakup v spletni trgovini glede na uporabnikove prejšnje nakupe.

Sklepamo takole: če imata Janez in Micka oba rada izdelke A, B in C, Janez pa tudi D, je smiselno predpostaviti, da bo tudi Micki všeč D. Izračunamo pogojne verjetnosti

$$P(\text{Janez ima rad } Z | \text{Janez ima rad } A, \text{Janez ima rad } B, \dots, \text{Janez ima rad } Y).$$

Za zanesljivo oceno pogojnih verjetnosti je potrebno zbrati zadosti podatkov. Različni spletni trgovci uporabljajo različne pristope. Kot izraženo preferenco štejejo nakup, uporabniki si na svojem računu lahko oblikujejo seznam želja, spodbujajo jih k ocenjevanju produktov, glasovanju za naj produkt, itd. Iz zbranih podatkov zgradimo za vsak izdelek odločitveno drevo, kot je tisto na sliki 10.4. Iz vsch dreves zgradimo Bayesovo mrežo, ki za vsakega kupca izračuna najboljša (najverjetnejša) priporočila.



Slika 10.4: Odločitveno drevo za posamezen izdelek.

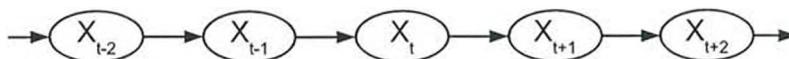
Za skupinsko filtriranje to ni edini mogoč pristop, je pa precej uporabljan.

## 10.2 Sklepanje v času

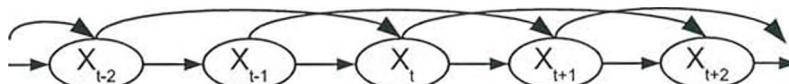
Pri številnih pomembnih problemih se vrednosti stanj sistema, podane s slučajnimi spremenljivkami, spreminja s časom. Tako denimo pri diagnostiki diabetesa poznamo zgodovino dosedanjega krvnega tlaka in krvnega sladkorja ter imamo podatke o zaužiti hrani in prejeti dozi insulina, napovedati pa želimo trenutno stanje krvnega sladkorja. Podobne primere lahko najdemo pri napovedih ekonomskih aktivnosti ali pri razpoznavanju govora, kjer na podlagi že razpoznanih fonemov želimo določiti naslednjega.

Pomagamo si z Bayesovim modelom in stanje sveta v vsakem časovnem trenutku  $t$  predstavimo z več spremenljivkami. Ločimo opazovane spremenljivke  $E_t$  in skrite spremenljivke  $X_t$ , do katerih nimamo dostopa, vemo pa, da vplivajo na model. Pri diabetesu so opazovane spremenljivke npr. merjeni sladkor in pulz, skrite pa krvni sladkor in vsebina želodca. Predpostavimo, da je čas diskreten (spremembe računamo po korakih) in da so spremembe stacionarne, kar preprosto povedano pomeni, da isti zakoni spremenjanja veljajo ves čas. Opozorimo, da stacionarnost ni enaka statičnosti, ki bi pomenila, da se svet ne spreminja.

Pod temi pogoji lahko naredimo markovsko predpostavko, namreč da je trenutno časovno stanje odvisno le od končnega števila predhodnih stanj. Markovska veriga prvega reda predpostavlja, da je trenutno stanje odvisno le od enega predhodnega. Slika 10.5 grafično prikazuje Bayesov model z markovsko predpostavko prvega reda, slika 10.6 pa Bayesov model z markovsko predpostavko drugega reda.



Slika 10.5: Bayesova mreža za markovski model prvega reda.



Slika 10.6: Bayesova mreža za markovski model drugega reda.

Za prehode skritih stanj (tranzicijski model) to formalno zapišemo tako, da je verjetnost trenutnega stanja glede na prejšnje stanje  $P(X_t|X_{t-1})$  enaka verjetnosti trenutnega stanja glede na vsa prejšnja stanja  $P(X_t|X_{0:t-1})$ :

$$P(X_t|X_{0:t-1}) = P(X_t|X_{t-1}).$$

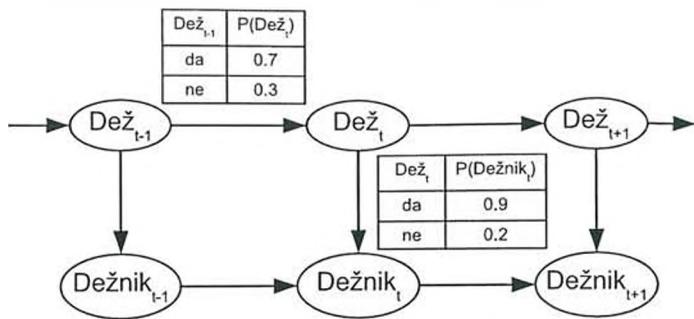
Smiselno je omejiti tudi opazovane spremenljivke in predpostaviti, da so odvisne le od trenutnega stanja skritih spremenljivk (senzorski model):

$$P(E_t|X_{0:t}, E_{0:t-1}) = P(E_t|X_t).$$

Poglejmo si preprost primer, ki ga povzemamo po [6]. Bayesova mreža je prikazana na sliki 10.7.

Denimo, da varnostnik v podzemnem prostoru brez oken skuša določiti vreme na podlagi dežnikov, ki jih obiskovalci nosijo v roki. Varnostnik opazuje obiskovalce vsako polno uro (diskretni časovni korak). Dežnik v roki obiskovalca pomeni veliko verjetnost, da zunaj dežuje, čeprav se lahko zgodi, da kdo prinese dežnik za vsak primer. Obenem varnostnik ve, da v njegovem kraju ponavadi ne dežuje le kratek čas in iz dežja pri prejšnjem opazovanju sklepa, da s precejšnjo verjetnostjo dežuje tudi pri naslednjem časovnem koraku.

Opozorimo na vzročnost (kavzalnost) zajeto v mreži, ko stanje sveta (dež) povzroča stanje senzorja (dežnika), medtem ko gre sklepanje v drugo smer.



Slika 10.7: Bayesova mreža za primer sklepanja o vremenu.

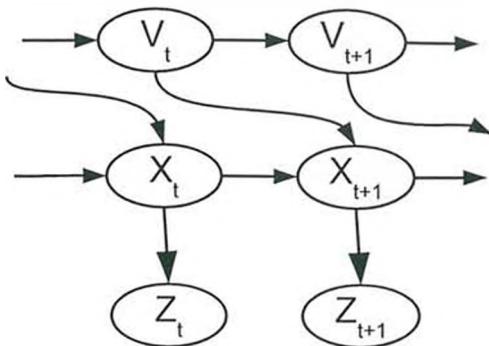
Skupno verjetnostno distribucijo za markovsko verigo prvega reda izračunamo na podlagi verjetnosti začetnega stanja  $P(X_0)$  in zbranih opazovanj:

$$P(X_0, X_1, \dots, X_t, E_1, \dots, E_t) = P(X_0) \prod_{i=1}^t P(X_i | X_{i-1}) P(E_i | X_i).$$

Z markovsko verigo lahko izvajamo naslednje operacije.

- Opazovanje: na podlagi dosedanjih opažanj napovemo trenutno stanje, torej izračunamo  $P(X_t | e_1, e_2, \dots, e_t)$ . Na primer, napovemo dež na podlagi dosedanjih opažanj dežnika.
- Izračunamo lahko verjetnost določenega zaporedja opažanj  $P(e_{1:t})$ .
- Za napoved prihodnjega stanja izračunamo  $P(X_{t+k} | e_{1:t})$  za  $k > 0$ .
- Popravimo prejšnje napovedi z več kasneje zbranimi podatki  $P(X_k | e_{1:t})$  za  $0 \leq k < t$ .
- Izračunamo najverjetnejšo razlago, oz. zaporedje stanj, ki je generiralo zaporedje opažanj,  $\arg \max_{X_{1:t}} P(X_{1:t} | e_{1:t})$ . V prepoznavanju govora bi to lahko pomenilo, da določimo najverjetnejše besede, ki so generirale zabeležene zvoke.
- Učimo se prehodov z iterativnim popravljanjem najverjetnejših stanj in prehodov, kar je ideja algoritma EM.

Markovska predpostavka nas vodi do približka. Če ta približek ni zadovoljiv in ga želimo izboljšati, lahko uporabimo markovski model višjega reda in vključimo odvisnosti za več korakov nazaj. V nekaterih aplikacijah je to smiselno (npr. pri prepoznavanju govora), drugje pa manj. Na točnosti napovedi pridobimo tudi z vključitvijo novih relevantnih spremenljivk. Pri napovedovanju dežja bi poleg dežnika varnostnik lahko upošteval še letni čas in morebitni senzor za vlažnost zraka. Takšna razširitev modela z novimi spremenljivkami zahteva, da določimo več pogojnih verjetnosti in napovemo vrednosti več spremenljivk.



Slika 10.8: Bayesova mreža za položaj objekta predstavljenega z zveznimi spremenljivkami. Spremenljivki  $V$  predstavlja hitrost objekta, spremenljivki  $X$  njegovo lokacijo, spremenljivki  $Z$  pa senzor za lokacijo.

Markovski modeli niso omejeni na diskretne spremenljivke. Poglejmo si preprost primer za zvezne spremenljivke, ko poskušamo napovedati položaj objekta iz dosedanjih opažanj. Grafično ga prikazuje slika 10.8.

Predpostavimo preprost linearni model, kjer naslednji položaj določa trenutna pozicija  $X_t$  in vektor hitrosti  $V_t$ . Spremenljivka  $Z_t$  predstavlja merilec pozicije.

$$X_{t+\Delta} = X_t + V_t \Delta.$$

V takšen model lahko dodamo gaussovski šum. Predstavljamo si, da gre za model premikanja robota po ravnini. Ker ni nobenih ovir, zadostuje že poznavanje prejšnje pozicije in hitrosti. Če pa je robot na bateriji, moramo vključiti še dodatno spremenljivko napolnjenost baterije, sicer bo markovska predpostavka kršena (ko se bo baterija izpraznila, bo robot obstal, ne glede na prejšnjo pozicijo in hitrost, torej markovska predpostavka o odvisnosti le od prejšnjega stanja ne velja).

Sklenemo lahko, da idealni model vsebuje le tiste spremenljivke, ki jih dejanski proces res zahteva.

## Skriti markovski model

Skriti markovski model (*HMM - Hidden Markov Model*) je verjetnostni časovni model, kjer je stanje predstavljeno z eno samo diskretno spremenljivko. Za primer napovedovanja dežja bi bila to spremenljivka Dež. Model vseeno ni trivialen, saj lahko odvisnost od več spremenljivk vključimo tako, da povečamo število možnih vrednosti tega edinega stanja, denimo s kartičnim produkтом možnih vrednosti vseh spremenljivk. Prednost preproste predstavitve je v enostavnosti izračuna, saj se poenostavijo operacije na modelu in jih lahko realiziramo z vektorskimi ter matričnimi operacijami.

Vrednosti stanja oštevilčimo z  $1, 2, \dots, S$ . Prehode v času, oziroma tranzicijski model,  $P(X_t | X_{t-1})$ , predstavimo z matriko  $T$  dimenzij  $S \times S$ , kjer je  $T_{i,j} = P(X_t = i | X_{t-1} = j)$ . Za

vremenski primer s slike 10.7 takšno matriko zapišemo kot

$$T = \begin{bmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{bmatrix}.$$

Tudi senzorski model zapišemo kot matriko, pri tem pa vrednost  $E_t$  poznamo (denimo, da gre za vrednost  $e_t$ ), zato za vsak korak  $t$  skonstruiramo diagonalno matriko opazovanj  $O_t$  z vrednostmi  $P(e_t | X_t = i)$ . Za vrednost Dežnik<sub>1</sub> = da, dobimo

$$O_1 = \begin{bmatrix} 0.9 & 0 \\ 0 & 0.2 \end{bmatrix}.$$

Pomembno področje uporabe skritega markovskega modela je prepoznavanje govora in označevanje besedil. Predstavljamo primer označevanja zaporedja besed z njihovo stavčno vlogo. Primer je prirejen po [2].

V zadosti velikem označenem korpusu lahko ocenimo verjetnosti za vlogo posameznih besed v stavku. Uporabljen je korpus univerze Brown z 87 oznakami. Več o obdelavi teksta in označevanju stavkov najdemo v poglavju 11.

Skriti markovski model predstavlja tabeli 10.2 in 10.3. Stavek, ki ga želimo označiti, je *I want to race*.

Tabela 10.2 podaja  $a_{i,j}$ , verjetnosti prehodov med stanji skritega markovskega modela. Stanja modela predstavljajo oznake za vlogo besede v stavku (v angleščini: VB=base form of verb; TO=infinitive marker; NN=singular or mass noun; PPSS=nominative pronoun, <s>=začetek stavka). Tabela 10.3 podaja  $b_i(o_t)$  verjetnosti opažanja besed glede na oznake  $O_t$ .

	VB	TO	NN	PPSS
<start>	0.019	0.0043	0.041	0.067
VB	0.0038	0.035	0.047	0.0070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

Tabela 10.2: Tabela prehodov  $a$  med stanji (oznakami) skritega markovskega modela  $p(t_i | t_{i-1})$ . Vrstice predstavljajo pogojni del, tako je  $P(TO|VB)=0.0043$ .

	I	want	to	race
VB	0	0.093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

Tabela 10.3: Tabela verjetnosti opažanj  $b$  za primer skritega markovskega modela.

```

// vhod: seznam besed o dolžine T
// skriti markovski model predstavljen z matrikama a in b
// vrača seznam kazalcev na najboljše predhodnike
List Viterbi(List o, matrika a, matrika b) {
    kreiraj matriko verjetnosti prehodov V dimenzij [N + 2, T];
    kreiraj matriko najverjetnejših prehodov najPrehod dimenzij [N, T + 1]
    for (s = 1 ; s <= N; s++) { // inicializacija, čez vsa stanja
        V[s, 1] = a[0, s] * b[s](o[1]); // začnemo v stanju 0, torej <s>
        najPrehod[s, 1] = 0; // prvo stanje je vedno <s>
    }
    for (t = 2 ; t <= T; t++) { // obdelamo vso sekvenco besed
        for (s = 1 ; s <= N ; s++) { // za vsa stanja HMM
            V[s, t] = maxNu=1 V[u, t - 1] · a[u, s] · b[s](o[t]); // po vseh predhodnih stanjih u
            najPrehod[s, t] = arg maxNu=1 V[u, t - 1] · a[u, s];
        }
    }
    V[N+1, T] = maxNs=1 V[s, T] · a[s, N+1] // še zadnji korak do stanja N+1, torej <e>
    najPrehod[N+1, T] = arg maxNs=1 V[s, T] · a[s, N+1]

    return seznam, ki ga dobimo, ko sledimo prehodom iz najPrehod[N+1, T] nazaj
}

```

Slika 10.9: Pseudokoda Viterbijevega algoritma za označevanje stavkov.

Za sklepanje s skritim markovskim modelom uporabimo Viterbijev algoritem, ki temelji na dinamičnem programiranju. Pseudokodo algoritma prikazuje slika 10.9.

Poleg  $N$  stanj, ki ustrezajo oznakam, algoritem upošteva še dve posebni stanji  $s$  in  $e$ , ki označujejo začetek in konec stavka. Viterbijev algoritem določi, katera pot skozi graf je najbolj verjetna. Algoritem deluje tako, da naprej poišče najbolj verjetno naslednje stanje ter nastavi kazalec na to stanje. Ko algoritem konča svoje delovanje, za izpis najboljše poti le sledimo kazalcem nazaj skozi graf.

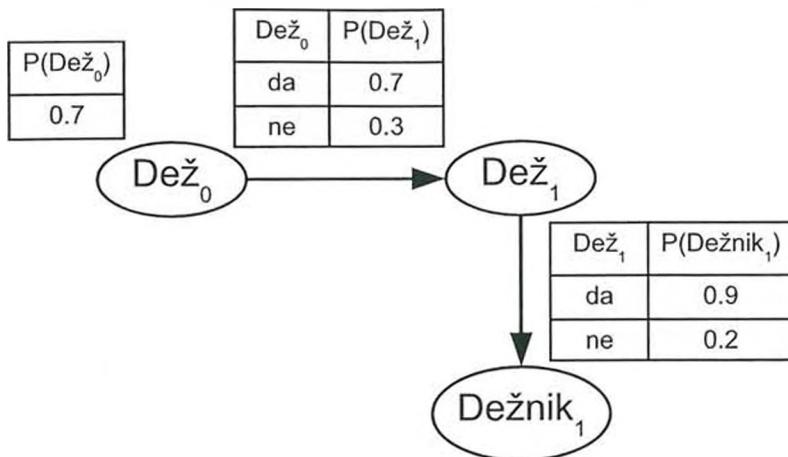
### Dinamična Bayesova mreža

Namesto s skritim markovskim modelom lahko časovne odvisnosti predstavimo z dinamično Bayesovo mrežo (*DBN - Dynamic Bayesian Network*). V Bayesovo mrežo vključimo časovno komponento tako, da imamo za vsak  $t$  spremenljivki stanja  $X_t$  in opažanj  $E_t$ .

Predpostavimo markovsko lastnost prvega reda, torej ima vsaka spremenljivka lahko povezave le v trenutnem in predhodnem času. Za stacionarne procese zadostuje, da podamo mrežo za dve zaporedni stanji, saj stacionarnost ohranja zakonitosti prehajanja.

Dinamična Bayesova mreža v primerjavi s skritim markovskim modelom potrebuje manj prostora in učinkoviteje računa, ker razstavi problem na med seboj neodvisne dele.

Za primer vremena bi lahko časovno pogojen proces zato zapisali s sliko 10.10. Če to mrežo primerjamo z mrežo na sliki 10.7, vidimo, da je predstavitev res bolj kompaktna.



Slika 10.10: Dinamična Bayesova mreža za primer sklepanja o vremenu.

Podobno lahko časovno odvisnost prikažemo za primer robota v ravnini. Denimo, da gre za robota na baterije in dodamo še spremenljivko  $B_1$ , ki vsebuje stanje baterije v času 1, ter senzor napoljenosti baterije Bsenzor. Dinamično Bayesovo mrežo prikazuje slika 10.11, običajno Bayesovo mrežo pa slika 10.8.

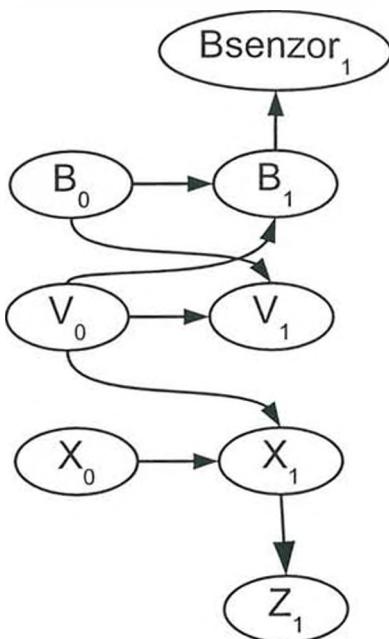
### 10.3 Učenje v grafičnih verjetnostnih modelih

Učenje v grafičnih verjetnostnih modelih bi lahko v grobem razdelili na dva dela, na učenje strukture modela in na učenje parametrov.

Učenje strukture pomeni, da se poskušamo na podlagi podatkov naučiti relacij med spremenljivkami in kavzalnosti. Ker gre za pridobivanje novega znanja in ker je prostor vseh relacij neskončen ali vsaj ogromen, je problem težek. Predstavljamо si, da želimo razložiti vremensko dogajanje v Sahari. Kaj vse vključiti v model? Poleg običajnih vremenskih spremenljivk tudi površino arktičnega ledu, sončne pege in količino proizvedenega  $CO_2$ ? Kaj pa lunine mene in orkane nad Pacifikom?

Preveč odvisnosti poveča zahtevnost sklepanja, težavno pa je tudi sestaviti tabele mnogih pogojnih odvisnosti. Pojavlji se problem pretirane prilagoditve podatkom, ker se z več spremenljivkami veča verjetnost naključnega ujemanja s podatki. Ena od rešitev je uporaba Occamove britve, ki pravi, da je med enakovrednimi modeli najbolj verjeten najenostavnejši. V splošnem velja, da je določanje optimalne strukture grafičnega modela NP-poln problem.

Ker je za človeka težko določiti številne parametre in pogojne verjetnosti, ki se pojavljajo v grafičnih modelih, se jih je smiselno naučiti. Pogoj za to je zadostna količina opazovanj, ki



Slika 10.11: Dinamična Bayesova mreža za primer premikanja baterijskega robota po ravnini. Spremenljivki  $B$  predstavljata napoljenost baterije, spremenljivki  $V$  hitrost robota, spremenljivki  $X$  njegovo lokacijo,  $Z$  senzor za lokacijo in  $B_{senzor}$  merilec napoljenosti baterije.

nam omogoča kalibracijo vrednosti parametrov, npr. verjetnosti prehodov med stanji. Učna naloga je, glede na dani model, izbrati parametre, ki dajo optimalno skupno verjetnostno distribucijo za dano spremenljivko glede na dana opažanja. Učenje parametrov je vključeno tudi v postopek učenja strukture in iterira med določanjem strukture in primernih parametrov za dano strukturo.

Zanimiva in priljubljena metoda za učenje parametrov v modelih je algoritem EM (*Expectation Maximization*). Za skriti markovski model bi algoritem najprej preštel pričakovano število prehodov med stanji in pričakovano število pojavitev stanja glede na dane vrednosti v matriki  $a$  in  $b$ , kar imenujemo E korak (*Expectation*). V naslednjem koraku M (*Maximization*) pa bi izračunana števila uporabil za izračun maksimalno verjetnih parametrov iz matrik  $a$  in  $b$ , ki bi generirala te pojavitve. Celoten postopek iterativno izvaja E in M korak, dokler se vrednosti pričakovanih in dejanskih pojavitev zadosti ne približajo.

## Literatura

- [1] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
- [2] Daniel Jurafsky, James H. Martin. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall, 2008.

- natural language processing, computational linguistics, and speech recognition.* Pearson Education Inc., 2009.
- [3] Daphne Koller, Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques.* The MIT Press, 2009.
  - [4] George F Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6th edition.* Addison-Wesley Pearson Education, Boston, 2009.
  - [5] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann, 1988.
  - [6] Stuart J. Russell, Peter Norvig. *Artificial intelligence: a modern approach, 3rd edition.* Prentice Hall, 2009.

## Poglavlje 11

# Obdelava naravnega jezika

*V tišini je vse.  
A le v besedi se vse  
razpre.*

Barbara Korun

Razumevanje naravnega jezika v pisni ali govorjeni obliki je zahteven problem, ki nemalo krat povzroča težave tudi ljudem. Le spomnimo se težav pri učenju tujih jezikov ali razprav o šibki funkcionalni pismenosti v naši državi.

*kdo lahko razume moj svet?  
mu sam sploh lahko sledim?  
hodim in iščem, sem mar zaklet?  
čakam, da se zbudim?*

Težave pri razumevanju ne veljajo le za poezijo, ampak tudi za bolj vsakodnevne tekste, recimo navodila za uporabo tehničnih izdelkov, časopisne članke, zapise na spletnih forumih, ne nazadnje tudi za seminarske naloge in izpite, ki jih učitelji na fakulteti dobimo od študentov. Poleg poznavanja samih besed, sintakse, semantike in konteksta, v katerem te besede nastopajo, je za razumevanje potrebno sklepati tudi o piščevem znanju in njegovih predpostavkah. Na primer, spodnji stavek s pomenskega vidika na prvi pogled nima nobenega smisla.

*Prijatelja so zaprli, zato sem bil vesel.*

Če pa vemo, da je Prijatelj lahko tudi priimek, lahko o odnosu pisca do omenjene javne osebe sklepamo kaj več. Razumevanje in pravilno reagiranje pa ni odvisno le od zapisa, pač pa tudi od ciljev in pričakovanj, ki jih je pisec imel. Na primer, če v spletni iskalnik vpišemo besedico *jaguar*, želimo odgovore, povezane z velikim plenilcem ali s športnim avtomobilom. Besedica *Paris* ne označuje le glavnega mesta Francije, pač pa še več kot 70 krajev ter številne osebe z istim imenom. Razločevanje med njimi na podlagi samo ene besedice ni mogoče, zato

tehnike razumevanja naravnega jezika poskušajo najti in uporabiti še dodatne informacije. Spletni iskalniki se skušajo približati željam uporabnika tako, da hranijo zgodovino njihovih dosedanjih povpraševanj, izdelajo njihov individualni uporabniški profil in na tej podlagi verjetnostno sklepajo, ali gre za ljubitelja avtomobilov ali živali.

Umetni sistem, ki bi lahko razumel naravni jezik na nivoju človeka, bi moral vsebovati velik del človeškega znanja in bi dosegel skoraj vse cilje umetne inteligeunce, njegovih odgovorov na naša vprašanja pa bi ne mogli ločiti od človekovih. Verjetno je podoben razmislek vodil Alana Turinga pri zasnovi njegovega testa za razpoznavanje inteligentnih strojev: če v pogovoru brez vidnega stika (npr. preko tipkovnice in zaslona) ne moremo ločiti med človekom in strojem/programom, je stroj dejansko inteligenten. Kot je pokazalo tekmovanje računalniških sistemov za Loebnerjevo nagrado, ki vsako leto preizkuša napredok programov pri pogovoru z ljudmi, ima Turingov test precej pomankljivosti, saj se ljudje ne vedemo vedno racionalno in tako programom marsikdaj uspe preliščiti posamezne sodnike, da ne ločijo med njimi in človeškimi sogovorci.

Kakorkoli že, Turingovega testa do danes ni zadovoljivo opravil še noben program, je pa področje obdelave naravnega jezika živahnno raziskovalno in komercialno polje s številnimi pristopi in zanimivimi aplikacijami, npr. sintetizatorji govora, avtomatski odzivniki, avtomatsko prevajanje, povzemanje besedil, vmesniki do baz podatkov, inteligentno iskanje in pridobivanje informacij. V tem poglavju se tega obširnega področja le dotaknemo in predstavimo nekaj uporabnih tehnik in aplikacij s poudarkom na statističnih pristopih, ki so z dostopnostjo velikih jezikovnih podatkovnih baz (korpusov) prevladali. Čeprav se omejimo na analizo besedil, pa iste probleme rešujejo tudi na področjih razumevanja in generiranja govora. Bralec, ki ga zanima več o obdelavi naravnega jezika, lahko poseže po eni od številnih knjig s tega področja, na primer [8, 2, 7, 9, 6].

## 11.1 Mikrosvetovi

Prvi sistemi za razumevanje naravnega jezika so se omejili na mikrosvetove, torej na določena ozka problemska področja, ki so za obvladovanje zahtevala le malo znanja. Eden prvih takšnih sistemov je bil Winogradov SHRDLU (razvit leta 1970), ki se je zmogel pogovarjati o t.i. svetu kock, sestavljenem iz preprostih geometrijskih objektov (kocka, kvader, piramida, valj) različnih barv in roke, ki jih je lahko premikala. Svet ilustrira slika 11.1.

SHRDLU je znal odgovoriti na preprosta angleška vprašanja:

What is sitting on the red block? (*Kaj stoji na rdečem telesu?*)

What shape is the blue block on the table? (*Kakšne oblike je modro telo na mizi?*)

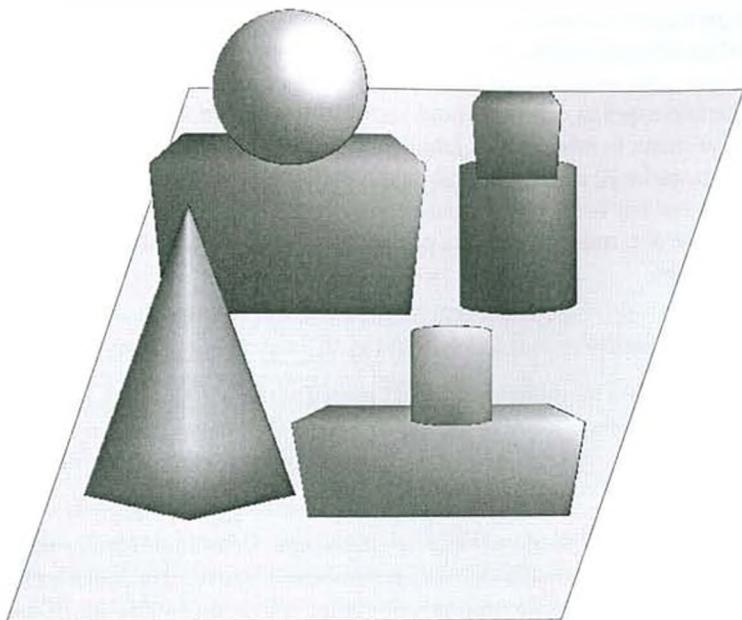
Place the green pyramid on the red brick. (*Postavi zeleno piramido na rdeč kvader?*)

Is there a red block? Pick it up. (*Ali obstaja rdeče telo? Dvigni ga.*)

What color is the block on the blue brick? Shape?

(*Kakšne barve je telo na modrem kvadru? Oblike?*)

Ker je bil svet sistema SHRDLU tako preprost, je bilo vanj mogoče vgraditi znanje o skoraj vsem potrebnem. Čeprav SHRDLU ni potreboval zahtevnejših elementov sklepanja,



Slika 11.1: Primer sveta kock, o katerem se je znal pogovarjati SHRDLU.

kot so vzročnost, obstoj različnih možnosti ali čas, je vseeno združeval sintakso in semantiko jezika in dokazal, da lahko umetni sistem z zadostnim znanjem o dani domeni smiselno komunicira v naravnem jeziku.

Podobni vmesniki v naravnem jeziku za ozka področja se uporabljajo še danes, v splošnem pa je tovrsten pristop za analizo jezika neuporaben, saj je človeško znanje preveč raznoliko, jezik pa preveč bogat in dvoumen, da bi lahko enostavno zajeli njegovo sintakso in semantiko.

## 11.2 Dva pristopa

V nadalnjem razvoju se je najprej uveljavil pristop, ki ga danes imenujemo simboličen, saj temelji na neposredno vgrajenem znanju, ki ga mora človek vnesti v sistem. Znanje o jeziku in svetu je tako lahko predstavljeno v obliki gramatik, okvirov, izraznih dreves in drugih načinov za predstavitev znanja. Tipično za tovrsten pristop je, da od zgoraj navzdol uveljavlja gramatične vzorce in pomen. Nekateri avtorji tovrstni pristop (rahlo posmehljivo) imenujejo "dobra stara umetna inteligenco" (*Good Old-Fashioned AI*).

S pojavom spleta in večanjem procesorske ter pomnilniške zmogljivosti so se odprle nove možnosti in uveljavil se je t.i. empiričen pristop k obdelavi jezika. Njegova značilnost je uporaba statističnih metod analize ter pridobivanje znanja iz velikih zbirk tekstovnih podatkov (korpusov). Ker je na voljo ogromno tekstov in primerov uporabe jezika v najrazličnejših okoliščinah, poteka iskanje vzorcev in povezav iz teksta od spodaj navzgor. To omogoča ve-

čjo robustnost tovrstnim sistemom. Na primer, pri analizi spletnih forumov, kjer uporabniki pišejo v slovnično šibkem jeziku, polnem okrajšav, slenga in drugih nižjejezikovnih oblik, bi klasičen pristop odpovedal, saj ne bi razpoznal "inovativne rabe jezika". Statistične metode so lahko vseeno uspešne, saj se tovrstni vzorci pojavljajo na mnogih mestih, kar sistemu omogoča, da se pomena in rabe nauči s tehnikami strojnega učenja.

Seveda pa najuspešnejši sistemi uporabljajo najboljše iz obeh pristopov in s pridom izkoristijo tako gramatike kot besedilne korpusa, tako eksplisitno predstavljeno znanje kot tisto pridobljeno z tekstovnim rudarjenjem. Na primer, naučijo se lahko tudi gramatik in jih z rabo iterativno izboljšujejo.

### Lingvistična analiza jezika

Pri analizi jezika naletimo na mnogo nalog: od prepoznavanja glasov, črk in sintaktične členitve, do določanja pomena, čustev in drugih višenivojskih oznak. Čeprav se naloge medsebojno prepletajo in je za uspešno analizo mnogokrat potreben črpati informacije iz več faz hkrati, se vseeno uporablja naslednja delitev.

- Prozodija: nauk o dolžini zlogov in o naglaševanju. Določimo ritem in intonacijo, kar nam v naslednji fazi pomaga določiti posamezne glasove. Da posamezne faze med seboj niso neodvisne, lahko vidimo prav tukaj, saj je od intonacije in naglaševanja lahko odvisen tudi pomen, na primer z drugačnim naglasom sporočamo sarkazem ali jezo.
- Fonologija ali glasoslovje: nauk o fonemih oziroma glasovih kot nosilcih pomenskega razlikovanja. Glasoslovje je še posebej pomembno za prepoznavanje in generiranje govora.
- Morfologija ali besedoslovje: nauk o besednih vrstah, oblikah in funkcijah. Ukvvarja se s tvorbo besed in določa pravilno rabo predpon in končnic ter določa vlogo posamezne besede v stavku (na primer, potreben je določiti čas, število in besedno vrsto).
- Sintaksa ali skladnja: zlaganje besed v besedne zvezze in stavke. V tej fazi uporabimo pravila za sintaktično analizo, kar je najbolje definiran in avtomatiziran del celotne analize.
- Semantika ali pomenoslovje: nauk o pomenu besed, fraz in stavkov.
- Pragmatika ali praktična raba: kako uporabljamo jezik in kako jezik vpliva na uporabnika. Na primer, če nas za mizo pri kosilu nekdo vpraša: "Ali mi lahko podate sol?" je sicer pravilno, če odgovorimo z "Lahko." ni pa tak odgovor to, kar je sojedec pričakovaval. Pragmatika je verjetno najtežja faza analize jezika, saj zahteva celovito poznavanje sveta: tako znanje o fizičnem svetu, človeku in družbi, kakor tudi o pomenu ciljev in namenov pri medsebojni komunikaciji, kar je bistveno za uspešno pomensko analizo.

Kot rečeno je navedena lingvistična delitev omejena, saj se različni nivoji delitve medsebojno prepletajo in vplivajo drug na drugega. Kljub temu pa vsega naenkrat ne moremo postoriti in tipično se razumevanja naravnega jezika lotevamo v treh fazah:

1. sintaktična analiza,
2. interpretacija pomena,
3. uporaba znanja o svetu.

Navedene tri faze v nadaljevanju ilustriramo z nekaj primeri.

### Sintaktična analiza

Prva faza pri razumevanju besedila je sintaktična analiza (*parsing*). Ugotoviti poskuša najprej sintaktično strukturo stavka, kar pomeni, da preveri ali je stavek pravilen in kakšna je njegova struktura, nato določi besedno vrsto vsaki od besed, npr. samostalnik, glagol, pridevnik, venzik, ... Za tovrstno analizo se uporablja kratica označevanje POS (*POS (part-of-speech) tagging*). Nazadnje iz strukture stavka in vrste besed poskušamo določiti še njihovo vlogo v stavku: ali gre za osebek, povedek, predmet, itd., kar omogoči kasnejše razumevanje. Večinoma rezultat sintaktične analize predstavimo kot drevo izpeljav (*parse tree*). Analiza zahteva poznavanje sintakse, morfologije in tudi nekaj semantike.

Kot primer navedimo najprej rezultat, ki ga za spodnje slovensko besedilo vrne JOS ToTaLe text analyser [4], ki je prosto dostopni spletni servis. Za vneseno besedilo določi osnovne oblike besed, kar imenujemo lematizacija, in njihove oblikoskladenjske oznake (*morphosyntactical tagging*).

*Nekega dne sem se napotil v naravo. Že spočetka me je žulil čevelj, a sem na to povsem pozabil, ko sem jo zagledal. Bila je prelepa. Povsem nezakrita se je sončila na trati ob poti. Pritisk se mi je dvignil v višave. Popoln primerek kmečke lastovke!*

Analizator vrne kot rezultat po odstavkih (oznaka *<p>*) in stavkih (oznaka *<s>*) ločeno besedilo:

```

<div>
<p>
<s>
analiza prvega stavka
</s>
<s>
analiza drugega stavka
</s>
...
</p>
</div>

```

Za posamezne besede vrača analizator oznake, definirane v specifikaciji Multext-East, npr. za besedico "dne" določi oznako *Somer*, kjer *S* pomeni samostalnik, *o* obče ime, *m* moški spol, *e* ednino in *r* roditeljnik. Kot osnovno obliko besede pravilno določi besedico "dan". Izpis za prvi stavek našega primera prikazuje tabela 11.1.

beseda	oznaka	osnovna oblika
Nekega	Zn-mer	nek
dne	Somer	dan
sem	Gp-spe-n	biti
se	Zp—k	se
napotil	Ggdd-em	napotiti
v	Dt	v
naravo	Sozet	narava
*	*	*

Tabela 11.1: Oblikoskladenske oznake kot jih določi JOS ToTaLe.

Za določitev vloge besed v stavku uporabljam gramatike. Spet so na voljo številna orodja, ki nam olajšajo izdelavo lastnih gramatik, uporabimo pa lahko tudi katero od že izdelanih gramatik za različne potrebe. Primer takšnega orodja je knjižnica NLTK (*Natural Language Toolkit*) v pythonu ali pa kar programski jezik prolog, ki v večini dostopnih implementacij že vsebuje podporo gramatikam. Težava, ki se ji pri gramatični analizi ne moremo izogniti, je dvoumnost naravnega jezika, ki povzroči, da lahko dobimo za isti stavek več dreves izpeljav.

Dvoumnost ilustrirajmo tako, da z uporabo NLTK analiziramo znani primer stavka iz filma Groucho Marxa "Animal Crackers" (1930). Primer povzemamo po [2].

*While hunting in Africa, I shot an elephant in my pajamas.*

Najprej pripravimo preprosto gramatiko, ki jo NLTK predela v interno obliko, potrebno za učinkovito analizo. Uporabimo kar angleške okrajšave za stavčne člene (S=sentence, N=noun, P=preposition, V=verb, NP=noun phrase, VP=verb phrase, PP=propositional phrase, Det=determiner) in v pythonu zapišemo:

```
groucho_grammar = nltk.parse_cfg("""
    ... S -> NP VP
    ... PP -> P NP
    ... NP -> Det N | Det N PP | 'I'
    ... VP -> V NP | VP PP
    ... Det -> 'an' | 'my'
    ... N -> 'elephant' | 'pajamas'
    ... V -> 'shot'
    ... P -> 'in'
    ... """)
```

Zdaj to gramatiko uporabimo za analizo našega stavka. To storimo takole:

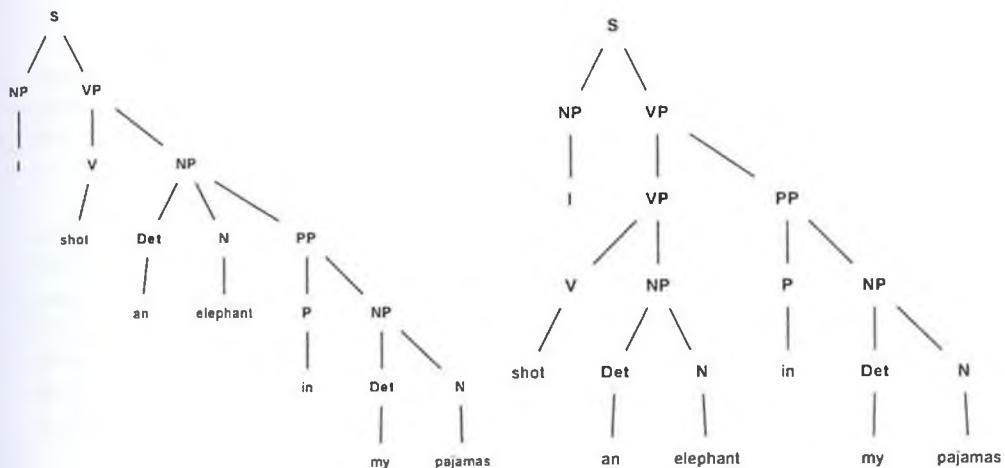
```
>>> sent = ['I', 'shot', 'an', 'elephant', 'in', 'my', 'pajamas']
>>> parser = nltk.ChartParser(groucho_grammar)
>>> trees = parser.nbest_parse(sent)
>>> for tree in trees:
```

```
*** print tree
```

in dobimo dve drevesi izpeljav

```
(S
  (NP I)
  (VP
    (V shot)
    (NP (Det an) (N elephant) (PP (P in) (NP (Det my) (N pajamas))))))
(S
  (NP I)
  (VP
    (VP (V shot) (NP (Det an) (N elephant)))
    (PP (P in) (NP (Det my) (N pajamas))))
```

Drevesi lahko narišemo, kot to prikazuje slika 11.2.



Slika 11.2: Dve drevesi izpeljav za dvoumen stavek.

Dvoumnost ni v različnem razumevanju posameznih besed, čeprav bi lahko glagol *shot* razumeli kot *streljati* ali kot *posneti fotografijo*, pač pa v različnih vlogah, ki jo v stavku igra *my pajamas*. Ta se enkrat nanaša na slona, drugič pa na streljanje. Za kaj je dejansko šlo, nam je v filmu pojasnil presenetljivi stavek: *How an elephant got into my pajamas I'll never know.*

V okviru sintaktične analize se pogosto uporablja tudi označevanje z n-grami, kar pomeni, da pri označevanju ene besede v zaporedju besed upoštevamo  $n-1$  predhodnih besed. To nam omogoči uporabo statističnih tehnik in učenje najverjetnejših stavčnih oblik iz velikih besedilnih korpusov. Za zdaj le omenimo, da se za to najpogosteje uporabljo Markovski modeli,

oziroma skriti Markovski modeli (*HMM - Hidden Markov Model*), ki se jih lahko učinkovito naučimo z algoritmom EM (*Expectation Maximization*). Ko poskušamo najti oznako za besedo, ki je najbolj verjetna na podlagi prejšnjih oznak, maksimiziramo izraz

$$P(\text{beseda}|\text{oznaka}) \times P(\text{oznaka}|\text{prejšnjih } n \text{ oznak}).$$

Najpreprosteje je to storiti le na podlagi ene same predhodne besede. Če za stavčno oznako na  $i$ -tem mestu uporabimo matematičen zapis  $t_i$ , za besedo na  $i$ -tem mestu pa  $w_i$ , lahko to formalno zapišemo kot

$$t_i = \arg \max_j P(t^{(j)}|t_{i-1})P(w_i|t^{(j)}).$$

Pri tem  $t^{(j)}$  označuje vse možne oznake, ki jih preizkusimo, da izberemo najverjetnejšo.

Marsikdaj pravi kontekst ni v predhodnih besedah pač pa v naslednjih. Računsko zahodna poslošitev je uporaba algoritma k-najbližjih sosedov, ki pri danem besedilu upošteva tudi bližnje sledeče besede.

### Interpretacija pomena besedila in uporaba znanja o svetu

Ko smo določili sintakso in označili dele stavka, je za interpretacijo pomena potrebno uporabiti znanje o pomenu besed in njihovi jezikovni vlogi. Rezultat je ena od predstavitev znanja, npr. konceptualni graf, okvirji ali pa v prologu kar logični program. V tej fazi preverimo tudi semantično smiselnost stavka in se po potrebi vrnemo na sintaktično analizo (izberemo na primer drugo drevo izpeljav).

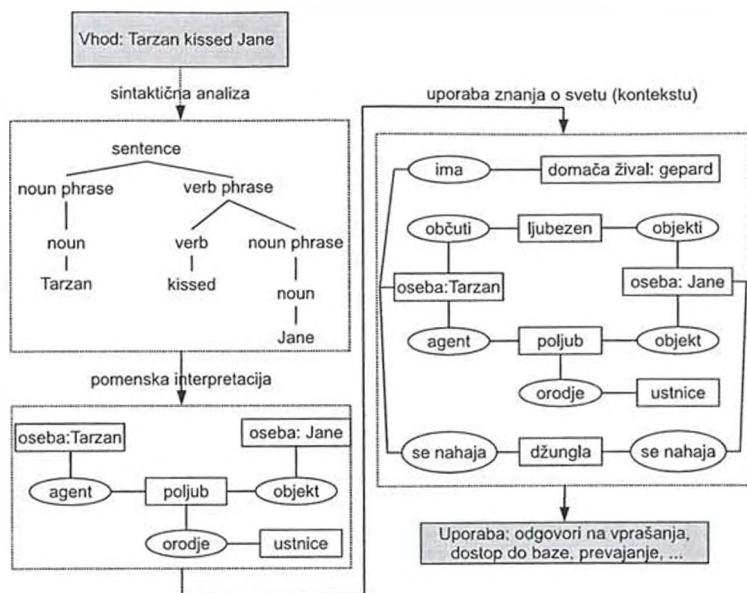
Če je mogoče, predstavitev pomena razširimo s predznanjem. Ker je celotno človeško predznanje praktično nemogoče zajeti (poskus v to smer je bil projekt Cyc, začet leta 1984, danes deloma dosegeljiv kot OpenCyc [5, 3]), v praksi sistemi uporabljujo le ozko podmnožico predznanja glede na namen uporabe (npr. povzemanje tehničnih besedil, vmesnik do baze podatkov o finančnem poslovanju podjetja, itd.) Ponavadi se uporablja inkrementalna analiza, kjer analizirane stavke in njihov pomen sproti dodajamo v predstavitev znanja in jih takoj uporabimo pri analizi nadaljnjih stavkov - to ustrezza tudi zgradbi večine besedil, kjer je v uvodu pojasnjeno področje in poglaviti pojmi, ki se uporabljajo v nadaljevanju.

Slika 11.3 predstavlja tovrstno analizo za znani primer stavka *Tarzan kissed Jane*.

## 11.3 Empirični pristop

V novejšem času se je s pojavom velikih korpusov besedil, vse večje računske moči in uveljavljivijo strojnega učenja in podatkovnega rendarjenja tudi pri analizi besedil področje korenito spremenilo. Skoraj povsem je prevladal empiričen pristop, kar pomeni, da se modelov poskušamo naučiti iz informacij, ki so na razpolago na spletu ali v besedilnih zbirkah. V tem razdelku poskušamo predstaviti nekaj poglavitnih idej, ki se pojavljajo pri najpogostejših nalogah.

**Pridobivanje dokumentov** (*document retrieval*): za učinkovito pridobivanje relevantnih dokumentov je potrebno indeksirati vso zbirko dokumentov. Boljši uspeh dosežemo, če o



Slika 11.3: Primer pomenske analize stavka.

posameznih uporabnikih hranimo specifične informacije, si zapomnimo, kaj ponavadi iščejo, in si s tem profiliranjem pomagamo pri naslednjih iskanjih. Za merjenje uspeha pridobivanja dokumentov uporabljamo specifične mere relevantnosti.

**Pridobivanje informacij** (*information extraction*): s pojavom spleta je iskanje specifičnih informacij prešlo iz ozkih strokovnih krogov v splošno rabo in slabosti ter prednosti iskalnikov so postale splošno znane. Pridobivanje informacij se ukvarja s tovrstnim iskanjem bodisi iz vsega spleteta, bodisi iz posameznih večjih zbirk. Najpogosteje so tovrstne aplikacije namenjene iskanju poslovnih novic, političnih dogodkov ali varnostno tveganih dogodkov.

**Klasifikacija dokumentov** (*document classification*): dokumente pogosto uvrščamo glede na vsebino v več razredov ali v neko hierarhijo oz. taksonomijo. Tako bi agencijске novice klasificirali v hierarhijo razredov po posameznih področjih, npr. po lokaciji (celine, regije, države, pokrajine) ali področju (politika, posel, tehnologija, zabava, šport,...). Podobno bi uvrstili prispevo elektronsko pošto v skupino koristnih in nekoristnih sporočil (*spam*).

**Povzemanje dokumentov** (*document summarization*): posamezne dokumente ali pa cele zbirke dokumentov poskušamo povzeti in jih človeku predstaviti v skrajšani obliki, ki še vedno vsebuje vse bistvene informacije.

**Tekstovno rudarjenje** (*text mining*): iz dokumentov ali zbirk želimo pridobiti nove informacije, jih povezati na nov način ali predstaviti zanimive povezave med njimi.

### Korpusi

Večkrat smo že omenili, da je za empiričen pristop k razumevanju teksta potreben dostop do velikih zbirk označenih tekstovnih podatkov. Svetovni splet je že sam po sebi velikanska zbirka podatkov, ki pa žal ni vedno dovolj zanesljiva in predvsem ni ustrezno označena in prečiščena za potrebe razumevanja naravnega jezika. Sčasoma so se pojavili veliki besedilni korpusi, ki te pomankljivosti odpravljajo in so koristni za zgoraj omenjene naloge. Naštajmo nekaj važnejših korpusov ter spletnih mest, ki vsebujejo zbirke jezikovnih tehnik.

- Na univerzi v Stanfordu vodijo zbirko virov za empirično analizo jezika *Statistical natural language processing list of resources* [10].
- Vire o jezikovnih tehnikah v Sloveniji vzdržuje *Slovensko društvo za jezikovne tehnologije* [12].
- Za slovenščino je najpomembnejši prostost dostopni korpus slovenskega jezika FidaPLUS [1], ki obsega več kot 600.000 besed.
- Večjezični viri so pomembni predvsem za prevajalce. Eden največjih korpusov, ki je sicer specializiran za pravno izrazoslovje, je prevod skupne zakonodaje Evropske unije v vseh 22 uradnih jezikov *JRC-Acqui multilingual parallel corpus* [13].

### Predobdelava besedila

Tudi pri empirični analizi potrebujemo osnovna lingvistična orodja. Tako je vsak dokument potrebno najprej razbiti na odstavke in še naprej na stavke ter posamezne besede. Besede in stavke je potrebno ustrezno označiti glede na njihovo vlogo v besedilu in potem nad označenimi stavki izvedemo sintaktično in gramatično analizo.

Že razbitje besedila na stavke ni tako preprosto, kot bi na prvi pogled pričakovali. Zgolj ločila in velike začetnice niso zadostni. Poglejmo stavek

*Npr. ostanke 1. Timbuktuja iz 5. st. pr. n.š. je odkril dr. Barth.*

Pika ne označuje le konca stavka pač pa tudi okrajšave in vrstilne števниke. Da je naloga še težja, lahko v teh vlogah nastopa tudi pred veliko začetnico, ki ni začetek stavka (*dr. Barth*). Za korektno razbitje potrebujemo regularne izraze, odločitvena pravila ali ročno segmentirane korpusa, iz katerih se lahko naučimo ustreznih pravil.

Podobno velja tudi za leksikalno analizo oz. za določitev besed (*tokenization, word segmentation*). Tudi tukaj uporaba zgolj presledkov in ločil ne zadošča. Recimo denarni znesek lahko v besedilu najdemo zapisan kot "1.999,00€!" ali "1,999.00€". Edino smiselno je, da celoten znesek obravnavamo kot eno entiteto in se zavedamo, da gre za denar. Treba je poznati različne vloge, ki jo igrajo vejice, pike in posebni znaki. Podobno velja za večbesedna lastna imena, recimo *Ravne na Koroškem* ali *Port-au-prince*, ki jih tudi po razbitju na besede želimo ohraniti kot enotne entitete. Pravilno moramo torej upoštevati vlogo presledka in pomišljaja, pa tudi drugih znakov. Nasproten primer, ko je eno besedo potreben razbiti na več entitet, najdemo recimo v nemščini. Beseda *Lebensversicherungsgesellschaft* pomeni

*družba za življenjsko zavarovanje* in jo bomo pri razumevanju besedila verjetno potrebovali ločeno na njene sestavne dele. Tudi razbitja na besede se zato lotimo z regularnimi izrazi oziroma končnimi avtomati, uporabljamo pravila, statistične modele in si pomagamo s slovarji, predvsem slovarji lastnih imen.

Naslednji korak v predobdelavi besedila sta korenjenje in/ali lematizacija, ko za besede poiščemo korene in/ali generične oblike. Za določitev korenov besed je potrebna morfološka analiza, kar je enostavno v nekaterih jezikih, v drugih pa precej težko. Tako je angleščina relativno preprosta in ima le nekaj možnih končnic. Obstajajo sicer izjeme, recimo glagol biti: *go, goes, going, gone, went*. Slovensčina in drugi pregibni jeziki so tukaj precej bolj zapleteni, saj težave povzroča sklanjanje in spreganje besed, denimo: *jaz, mene, meni, mano*. Potrebna je uporaba pravil in slovarja, različne oblike besed pa lahko privedejo do dodatnih dvoumnosti. Recimo beseda *mano* lahko predstavlja 6. sklon osebnega zaimka *jaz*, ali pa 4. sklon samostalnika *mana*.

Za nekatere namene poglobljena in natančna analiza niti ni potrebna, takrat si lahko pomagamo s približnimi reštvami in hevristikami, npr. v angleščini samo odstranimo končnice *-ing, -ation, -ed*.

Preden začnemo iz besedila pridobivati informacije, ga še označimo ter določimo besedne vrste in vlogo besed v stavku. Potrebno je prepoznati fraze in lastna imena ter enolično prepoznati, za katero lastno ime gre. Znani primeri tovrstnih težav so blagovne znamke, kjer proizvajalci nalač izbirajo znane besede, ki gredo v uho. Težavna so tudi lastna imena in geografske oznake. Na primer *London* je britanska prestolnica pa tudi eno od ljubljanskih predmestij, *Dunaj* pa poleg v avstrijski prestolnici najdemo tudi na Dolenjskem. Za pravilno in enolično določitev entitete moramo poznati kontekst besedila, kjer se besedica pojavlja. Uporabljamo pravila in modele strojnega učenja.

V nadaljevanju si poglejmo nekaj pomembnih nalog, ki zahtevajo razumevanje naravnega jezika.

### Iskanje dokumentov in pridobivanje informacij

Iskanje dokumentov je bila ena najpomembnejših nalog, še preden se je pojavil svetovni splet in se je uveljavil empiričen pristop k analizi besedil. Ker iskanje po celotnem besedilu še ni bilo mogoče, se je zgodovinsko uporabljalo predvsem ključne besede. S pojavom spletnih iskalnikov in njihovo enostavno splošno rabo, so se tovrstne tehnologije preselile celo v knjižničarstvo, kjer so se ti pristopi najdlje obdržali.

Za uspešno iskanje relevantnih dokumentov je potrebno organizirati ustrezno specializirano bazo podatkov, ki bo hrnila informacije, ki jih bomo pridobili z indeksiranjem. Pri tem se je treba zavedati, da je velikost indeksa lahko tudi reda velikosti vseh indeksiranih dokumentov. Le na tako pripravljenem indeksu bodo iskalni algoritmi dali zadovoljive odgovore. Kakovost odgovorov pa ni pogojena le s pripravo indeksa, upoštevati je treba, da je vhod v iskalni algoritem tipično le nekaj besed, ki jih vnese uporabnik. Če so te vprašljive kvalitete, preveč splošne ali dvoumne, je uporabnost odgovora ponavadi majhna.

Poglejmo si najprej indeksiranje dokumentov, ki omogoča kvalitetno iskanje. Indeks vsebuje vse besede iz vseh dokumentov (imenujemo ga slovar oziroma angl. *inverted file*). Besede pretvorimo v korensko ali v osnovno obliko (npr. I. sklon ednine). Za vsako besedo shranimo:

- število dokumentov, kjer nastopa, kar omogoča izračun ustreznih statistik za pomembnost besed,
- skupno število pojavitev besede v vseh dokumentih.

Za vsak dokument v bazo zapišemo:

- število pojavitev besede v dokumentu, kar kaže na pomembnost neke besede v dokumentu,
- mesto pojavitve besede v dokumentu, kar pri odgovorih omogoči prikaz bližnjih besed, oziroma konteksta, kjer beseda nastopa.

Kot rezultat dobimo indeks, katerega delček prikazuje slika 11.4.

osnovna oblika	število dokumentov	število pojavitev	kazalec
abeceda	40	45	•
abranek	2	2	•
absolvirati	356	389	...
abstinent	72	102	...
...	...	...	...

zap. št. dokumenta	frekvenca	pozicije besede	kazalec naprej
7	2	356, 711	•

zap. št. dokumenta	frekvenca	pozicije besede	kazalec naprej
365	1	49	•

zap. št. dokumenta	frekvenca	pozicije besede	kazalec naprej
...	...	...	...

zap. št. dokumenta	frekvenca	pozicije besede	kazalec naprej
21	1	238	...

Slika 11.4: Izsek indeksa za iskanje dokumentov.

### Iskanje z logičnimi operatorji

Danes že tehnološko preseženo, vendar še vedno v uporabi pri starejših sistemih in v knjiznicah, je iskanje z logičnimi operatorji. Praktično vsi tovrstni sistemi podpirajo operatorje konjunkcije, disjunkcije in negacije (AND, OR, NOT) in niso občutljivi na veliko in malo zacetnico. Kot rezultat dobimo množico dokumentov, ki ustrezajo podanim logičnim pogojem. Na primer pozvedba "jaguar AND car" bi nam vrnila množico dokumentov, ki vsebujejo

obe besedi, *jaguar* in *car*. Podobno bi dvoumnost besede *jaguar* odpravila tudi poizvedba "jaguar NOT animal". Nekateri tovrstni sistemi podpirajo tudi sosednost v tekstu (operator NEAR) in celo lematizacijo (operator !), ki upošteva vse oblike dane besede. Tako bi poizvedba "president NEAR(10) bush" vrnila dokumente, kjer beseda *president* nastopa največ deset besed oddaljena od besede *bush*. Poizvedba "pariz! NEAR(3) francij!" pa bi poiskala vse dokumente, kjer *pariz* v vseh možnih oblikah nastopa največ tri besede oddaljen od vseh možnih oblik besed s korenom *francij*.

Sistemi z logičnimi operatorji imajo žal številne pomankljivosti. Kot rezultat pogosto dobimo (pre)veliko množico rezultatov, ki jo poskušamo zmanjšati z dodajanjem novih členov v logični izraz, kot končni rezultat pa dobimo zapletene in težko razumljive izraze. Preveč množica rezultatov morda niti ne bi bila težava, vendar dobljeni rezultati niso urejeni po pomembnosti ampak npr. po datumu ali nekem drugem za uporabnika nesmiselnem kriteriju (npr. poziciji v bazi podatkov). Strogost ujemanja z logičnim izrazom, kjer ni delnega ujemanja (dokument bodisi zadošča kriteriju ali pa ne) pomeni, da po nepotrebnem izgubljamo delno koristne rezultate. Za posamezne besede, ki jih želimo poiskati, lahko dokumenti uporabljajo sinonime (npr. osčeno vozilo namesto avto) in uporabnik se bo le težko spomnil vse možnosti. To težavo delno rešimo z uporabo besednjakov, slovarjev, leksikonov in tezavrov. V logičnem izrazu imajo vsi členi enako težo, kar onemogoča, da bi izkoristili morebitno dodatno informacijo, ki bi jo dodali tako, da bi najprej navedli najpomembnejše člene. Naštete slabosti so iskanje z logičnimi operatorji porinile v pozabo, nadomestilo pa ga je rangirano iskanje.

### Rangirano iskanje

Uveljavilo se je svetovnim spletom in smo ga najprej uporabljali za spletno iskanje, kasneje pa se je preselilo tudi v vse druge sisteme, ki omogočajo pridobivanje dokumentov. Glavna značilnost tega iskanja je, da kot rezultat dobimo množico dokumentov sortirano glede na pomembnost. Osnovna ideja rangiranja je, da so redki členi pomembnejši, saj vsebujejo več informacije o dokumentu, v katerem nastopajo. Vhod je lahko kar v naravnem jeziku, saj sistem sam odstrani t.i. nepotrebne besede (*stop words*), dokumente in vprašanja pa lematizira (predstavi besede v osnovni obliki).

Dokumenti in vprašanja so v takšnih sistemih predstavljeni kot vektorji: vsaka osnovna oblika besede je ena dimenzija, frekvanca besede v dokumentu pa je razsežnost v tej dimenziji. Takšno predstavitev imenujemo tudi vreča besed (*bag-of-words*). Poglejmo naslednji sestavek.

*Slon je sesalec. Sesalci so živali. Tudi človek je sesalec. Sloni in ljudje živijo tudi v Afriki.*

Zanj bi sistem zgradil tabelo 11.5, izpustil predlog *v* in dokument predstavil z 8 dimenzionalnim vektorjem (1, 3, 2, 3, 2, 2, 1, 1).

Dobra lastnost vektorske predstavitve je, da lahko podobnost med dokumenti in povpraševanji izrazimo kot razdaljo v vektorskem prostoru. Če nekoliko nerealistično predpostavimo, da so besede nekorelirane, so si posamezne dimenzijske med seboj ortogonalne in lahko podobnost vektorjev izračunamo kot kosinus kota med njimi. Med vektorji dokumentov in

Afrika	biti	človek	sesalec	slon	tudi	v	žival	živeti
1	3	2	3	2	2	1	1	1

Slika 11.5: Predstavitev sestavka z vektorjem besed.

vektorjem povpraševanj izračunamo skalarni produkt. Ker je kosinus na odseku med 0 in  $\frac{\pi}{2}$  monotono naraščajoč, pomeni manjša vrednost kosinusa manjši kot in večjo podobnost med dokumentoma. Za dokumenta oziroma vektorja  $A$  in  $B$  tako velja mera podobnosti

$$\cos(\Theta) = \frac{A \cdot B}{|A||B|}$$

Kot smo že omenili, vse besede v iskalnem nizu in dokumentu niso enako pomembne. Pomembnejši so dokumenti, v katerih so povpraševane besede pogoste, manj pomembni pa so dokumenti, v katerih so povpraševane besede redke ali ne nastopajo. Dodatno velja, da so za ločevanje med dokumenti pomembnejše redke besede, saj te glede na teorijo informacij vsebujejo več informacije. Zanima nas relativna redkost besed (*idf - inverse document frequency*), zato definirajmo  $N$  kot število dokumentov v zbirk,  $n_b$  pa kot število dokumentov z besedo  $b$ . Redkost besede  $b$  potem izračunamo kot

$$\text{idf}_b = \log \frac{N}{n_b}.$$

S pomočjo redkosti besed lahko dimenzijsko oziroma besede utežimo tako, da bolj redke besede dobijo večjo učinkovitost. Pri tem je potrebno upoštevati tudi frekvenco besede znotraj danega dokumenta. Če je  $f_{b,d}$  frekvanca besede  $b$  v dokumentu  $d$ , lahko utež besede  $b$  v vektorju dokumenta  $d$  zapišemo kot produkt

$$w_{b,d} = f_{b,d} \cdot \text{idf}_b.$$

Na podlagi tega definiramo uteženo podobnost med povpraševanjem  $q$  in dokumentom  $d$ , ki upošteva uteži vseh besed v vektorju in izračuna kosinus kota med uteženim vektorjem povpraševanja in dokumenta

$$\text{podobnost}(q, d) = \frac{\sum_b w_{b,d} \cdot w_{b,q}}{\sqrt{\sum_b w_{b,d}^2} \sqrt{\sum_b w_{b,q}^2}}$$

Pri izpisu nato dokumente rangiramo glede na padajočo podobnost.

### Uspešnost iskanja

Če želimo med seboj primerjati različne pristope k iskanju in različne iskalnike, potrebujemo merljive ocene uspešnosti iskanja. Za to tipično uporabljamo statistične mere ter (subjektivne) ocene zadovoljstva uporabnikov. Omenimo najpogostejši statistični oceni: točnost in priklic. Za splošno rabo sta definirani v poglavju 3, tukaj pa si ju oglejmo v kontekstu iskanja

dokumentov. Naj bo  $N$  število vseh dokumentov v zbirki in  $n$  število pomembnih dokumentov za dano povpraševanje  $q$ . S povpraševanjem pridobimo  $m$  dokumentov od tega  $a$  relevantnih. Točnost (*precision*) je definirana kot

$$P = \frac{a}{m}. \quad (11.1)$$

Vidimo, da gre za delež pravih dokumentov med vsemi pridobljenimi. Priklic (*recall*) je definiran kot

$$R = \frac{a}{n} \quad (11.2)$$

in sporoča delež pridobljenih pravih elementov med vsemi obstoječimi pravimi dokumenti. Obe meri ponavadi predstavimo z grafom, kjer za različne stopnje priklica predstavimo točnost. Uporabnik se lahko tako lažje odloči, koliko pridobljenih dokumentov bo obdelal.

Za enostavnejšo primerjavo med iskalniki, pogosto želimo namesto grafa eno samo številko. V takih primerih lahko uporabimo mero  $F$ , ki združuje in upošteva tako točnost  $P$  kot priklic  $R$

$$F_\beta = \frac{(1 + \beta^2)PR}{\beta^2 P + R} \text{ za } \beta > 0$$

Izraz daje različno težo točnosti in priklicu glede na uporabljeni parameter  $\beta$ . Pogosto se uporablja vrednosti  $\beta = 2$  in  $\beta = \frac{1}{2}$ , za  $\beta = 1$  pa dobimo uteženo harmonično sredino.

Točnost, priklic in mera  $F$  ne upoštevajo kvalitete rangiranja dokumentov, kar je za uporabnika tipično še bolj pomembno. V ta namen se uporablja prilagojene mere, izpeljane iz osnovne točnosti in priklica. Naj bo  $r_i$  rang, ki ga je iskalnik pripisal  $i$ -temu najpomembnejšemu dokumentu. Logaritmična točnost in rangirani priklic upoštevata razliko med vsoto vrnjenih rangov in vsoto rangov, ki bi jo dobili v idealnem primeru, ko bi iskalnik  $i$ -temu najpomembnejšemu dokumentu pripisal rang  $i$ . Definirana sta kot

$$\text{LogP} = \frac{\sum_{i=1}^n \log i}{\sum_{i=1}^n \log r_i}$$

$$\text{RangR} = \frac{\sum_{i=1}^n i}{\sum_{i=1}^n r_i}$$

### Izboljšave iskanja

Čeprav podjetja, ki se ukvarjajo s spletnim preiskovanjem zlepa ne razkrijejo skrivnosti svojih algoritmov, vendarle omemimo nekaj možnih izboljšav. Smiselno je na primer razširiti osnovno vprašanja z uporabo besednjaka (npr. Wordnet [11]). Sopomenk in morebitne hierarhije nadpomenk se lahko naučimo tudi iz korpusa in tako za dano vprašanje dobimo širši nabor odgovorov. Tudi ta pristop s seboj prinaša težave, recimo z večpomenkami lahko dobimo odgovore, ki so nepovezani z osnovnim vprašanjem, kar bo povzročilo nezadovoljstvo uporabnikov.

Nekateri iskalniki so poskušali razširiti vprašanja z uporabnikovo informacijo o pomembnosti posameznih vprašalnih izrazov, drugi pa pričakujejo povratno informacijo uporabnika,

ki naj bi označil zanj pomembne dokumente. Vse to vodi v profiliranje uporabnikov, saj lahko iskalnik na podlagi zgodovine iskanj natančneje določi zanimanja specifičnega uporabnika in pridobi zanj bolj relevantne odgovore. Na tem mestu je potrebno omeniti tudi skrb za zasebnost uporabnikov, saj ne želimo, da bi si iskalniki zapomnili preveč osebnih informacij, ki bi jih lahko izluščil iz povpraševanj (npr. zdravstveno stanje in druge občutljive podatke). Drug mogoč pristop je predpostavka o pomembnosti najboljših vrnjenih dokumentov in razširitev ali omejitve povpraševanja z dodatnimi značilnimi besedami iz teh dokumentov.

### Spletne iskanje

Spletne iskanje je poleg široke in trenutne dostopnosti velike količine informacij s seboj prineslo tudi nekaj novih težav. Uporabniki tako nimajo kontrole nad vsebino vrnjenih dokumentov in ne morejo popraviti niti očitnih napak ali manipulacij. Najdeni dokumenti so zelo različne kvalitete, pa tudi ažurnost nekaterih dokumentov je lahko vprašljiva. Pojavlja se problem neveljavnih povezav, ki jih uporabniki zaprtih sistemov nimajo. In nazadnje, zaradi ekonomske pomembnosti iskanja in želje podjetij, da bi se znašle na vrhu seznama zadetkov priljubljenih iskalnikov, se je pojavilo prirejanje rezultatov in poskus manipulacije iskalnikov. Upravljalci iskalnikov se temu upirajo in poskušajo osnovne iskalne principe prilagoditi na načine, ki to onemogočajo. Podrobnosti tega mnogoboja pa so že izven konteksta tega učbenika.

Posebnost spletnega iskanja, ki ga loči od običajnega pridobivanja dokumentov, je uveljavitev ideje, da ni vsebina dokumenta tista, ki odloča o njegovi pomembnosti, pač pa so to povezave, ki kažejo nanj. V grobem orišimo algoritem za rangiranje dokumentov PageRank, ki se v osnovni inačici sicer ne uporablja več, je pa poučna njegova ideja, saj je služil kot osnova najuspešnejšemu današnjemu iskalniku.

Zamisel algoritma je, da je za oceno kvalitet dolожene spletne strani potrebno upoštevati kvaliteto spletnih strani, ki kažejo na to stran. Naj bo  $p$  spletна stran,  $O(p)$  množica strani, na katere kaže  $p$ ,  $I(p) = \{i_1, i_2, \dots, i_n\}$  pa množica strani, ki kažejo na  $p$ . Definirajmo še parameter  $d$ , ki predstavlja faktor dušenja in je vrednost med 0 in 1 (tipično 0.85 ali 0.9). Potem je  $\pi(p)$ , kvaliteta strani  $p$ , definirana kot

$$\pi(p) = (1 - d) + d \frac{\pi(i_1)}{|O(i_1)|} + \dots + d \frac{\pi(i_n)}{|O(i_n)|}.$$

Kvaliteta strani  $\pi(p)$  je tako vsota prispevkov vseh strani, ki kažejo nanj, s tem, da so ti prispevki normalizirani s številom njihovih izhodnih povezav. Normalizacija preprečuje, da bi prevelik vpliv dobole strani z veliko množico izhodnih povezav (npr. spletni indeksi). Seveda se pojavi vprašanje, kako določiti kvaliteto strani, ki kažejo na dano stran. Odgovor je v iterativnem algoritmu, ki vsem  $N$  stranem na začetku določi vrednost  $\frac{1}{N}$ , nato pa kvaliteto strani iterativno popravlja, vsota kvalitet vseh strani pa ostane omejena na vrednost 1. Tovrstni izračun lahko vse na spletu prosto dosegljive strani obdela v nekaj urah.

Analiza pokaže, da algoritem PageRank sledi modelu naključnega spletnega deskarja, torej uporabnika, ki naključno prehaja iz strani na stran. Kvaliteta strani, ki jo določa PageRank konvergira k verjetnosti, da bo naključni deskar obiskal določeno stran. Ker v resnici

uporabniki strani ne obiskujejo naključno, pač pa sledijo lastnim interesom, se model naključnega deskarja danes poskuša nadomestiti z modelom namenskega deskarja. Kako pridobiti informacije o namenih posameznega spletnega deskarja? Spletne podjetja odgovor iščejo v profiliraju uporabnikov in različne orodne vrstice, ki jih ponujajo uporabnikom, beležijo strani, ki jih le-ta obišče in podatke o tem posredujejo podjetju. Na ta način lahko iz velikega števila uporabnikov podjetje dokaj zanesljivo izdelo model obnašanja in izdela graf interesov tako za posameznega uporabnika kot za ves splet.

PageRank ni imun na manipulacije. Najbolj znane so tako imenovane farme povezav (*link farms*), ki so v bistvu skupine nekaj (deset) tisoč navideznih spletnih mest, ki vsebujejo povezave na izbrane spletne strani in jim na ta način umetno dvignejo njihovo oceno kvalitete, za kar dobijo plačilo. Kolikor je znano, poskušajo upravniki spletnih iskalnikov identificirati tovrstne farme in jih ne upoštevati pri rangiranju rezultatov.

V zadnjem času spletni iskalnik izboljšuje svoje rezultate na podlagi algoritmov za analizo grafov. Celoten splet na podlagi medsebojnih povezav predstavlja kot usmerjen graf, ki ga potem analizirajo z različnimi metodami teorije grafov. Na ta način pridobijo znanje o posebnostih posameznih delov grafa, ki jih lahko upoštevajo pri rangiranju rezultatov.

### Klasifikacija dokumentov in tekstovno rudarjenje

Klasifikacija dokumentov je ena najpogosteje uporabljenih aplikacij s področja razumevanja naravnega jezika. Ker smo o klasifikaciji na splošno že obširno govorili v poglavjih o strojnem učenju, jo tukaj zgolj omenjamamo.

Dejansko je na področju obdelave teksta mnogo različnih aplikacij, ki uporabljajo različne klasifikacijske metode. Pogosto se uporabljajo naivni Bayesov klasifikator, odločitvena pravila in odločitvena drevesa, boosting, in naključni gozdovi. Dokumente je potrebno klasificirati med drugim pri iskanju, izbiri relevantnih novic (novica je relevantna ali ne za dano rabo), sortirjanju sporočil v kategorije (npr. agencisce ali poslovne novice sortiramo v hierarhijo vnaprej pripravljenih kategorij). Priljubljena je tudi klasifikacija elektronske pošte v mape, ki jih definirajo uporabniki, ter razločevanje med slamo (*spam*) in koristnimi sporočili. Tudi organizacije uporabljajo klasifikacijo za označevanje dokumentov na svojem intranetu (npr. pravo, borza, znanstveni članki, tehnično navodilo, ...)

Raziskovalno bolj zanimivo kot klasifikacija je tekstovno rudarjenje (*text mining*), katerega namen je pridobivanje novega znanja iz tekstov. Že dosedaj smo v tem poglavju opazili, da številne naloge pri razumevanju teksta zahtevajo učno komponento.

Omenimo tipične aplikacije in naloge. Povzemanje besedil poskuša iz daljšega teksta ali več tekstov avtomatsko povleči bistvene informacije in jih predstaviti v človeku berljivi obliki. Za številne namene je zanimivo ugotavljanje relacije med dokumenti in grupiranje dokumentov. Za agencisce hiše, varnostne analitike in še za koga je pomembna detekcija novih tematik in iskanje povezanih novic. Tekstovno rudarjenje se uporablja tudi pri kreiranju katalogov pomembnih oseb/podjetij in izgradnji taksonomij.

Omenimo določanje, prepoznavanje in odpravo dvoumnosti pri entitetah, kar potrebujemo že pri sami sintaktični in semantični analizi teksta. Treba je razpozнатi ljudi, kraje,

podjetja, organizacije, izdelke, datume, števila, procente, itd. Pri tem si pomagamo s katalogi, slovarji in hevristikami. Marsikdaj je potrebno uporabiti celo iterativno določanje, saj pri prvem prehodu skozi tekst še ni mogoče vsega razbrati.

Podatkovno rudarjenje je koristno tudi za pravilno določanje referenc in koreferenc, npr. za prepoznavanje oseb. V nekem tekstu se besede *president*, *George Bush*, *Mr. Bush*, *g. Bush head of state*, *he*, *bushism* nanašajo na isto osebo. Če želimo kvalitetno opraviti zgoraj naštete naloge, je potrebno ta nanašanja zaznati in jih pripisati isti osebi.

### Povzemanje teksta

Zaradi vse večjega števila dokumentov, ki so na razpolago za določeno področje, in zamudnosti ročnega pisanja povzetkov, se je pojavila ideja, da bi povzemanje avtomatizirali z orodji, razvitimi za razumevanje naravnega jezika. Naloga je torej iz danega dokumenta oziroma dokumentov izdelati povzetek z najpomembnejšo vsebino.

Kot vemo, pomembnost vsebine ni enolično določena, ampak jo lahko definiramo glede na temo, ki nas zanima (določen tekst je lahko zanimiv z naravoslovnega ali sociološkega vidika), glede na uporabnikov namen (novica je lahko za nekoga poslovno dogajanje, za varnostno agencijo pa je zanimiv drugačen vidik). Ne nazadnje je za pomembnost povzemanja treba upoštevati tudi povpraševanje, ki ga je definiral uporabnik.

Kot rezultat povzemanja dobimo izvleček ali povzetek, ki je ponavadi izbira pomembnih stavkov. V ta namen je potrebno rangiranje stavkov, odstavkov in posameznih fraz. Učenje poteka iz korpusov besedil z obstoječimi povzetki, razvite pa so bile številne hevristike.

Še težavnejše je povzemanje iz več dokumentov. Obstojecici pristopi se problema lotujejo z iskanjem skupnih tem in grupirajo odstavke v vseh besedilih z metodami razvrščanja vzorcev (*clustering*). Drugi pristopi izdelajo povzetke za posamezne dokumente in jih potem združujejo v en sam povzetek.

Uspešnost povzemanja merimo s primerjavo z ročno napisanimi povzetki (trenutno okoli 50% ujemanje n-gramov) ali pa z anketami zajamemo subjektivne ocene uporabnikov.

## 11.4 Trendi v razumevanju naravnega jezika

Področje razumevanja naravnega jezika je velik napredek doživel vo množico označenih korpusov. Za nadaljnji napredek bo potrebna pomoč novih (spletih) tehologij, denimo strukturiranje spletja s pomočjo schem XML in tehnologije RDF (*resource description framework*), kjer so posamezni viri že označeni - tipično s trojkami *osebek-povedek-predmet* (*subject-predicate-object*). Če bi takšno označevanje zaživel vo na dobršnem delu spletja, bi se splet spremenil v velikanski označen korpus, kar bi omogočilo številne nove rabe. Raziskovalci in podjetja mnogo pričakujejo tudi od semantičnega spletja (*semantic web*), ki bo za najrazličnejše vire prinesel uporabo metapodatakov, kar bi lahko bistveno izboljšalo uspešnost nekaterih pristopov.

Kot posledica napredka jezikovnih tehologij se danes uveljavljajo (pol)inteligentni pomočniki in agenti, ki na podlagi poznavanja posameznega uporabnika zanj izvajajo nekatere naloge (npr. iskanje, pomoč pri pisanku, priporočila glede nakupov, ...). V uporabi pa so tudi sistemi za avtomatsko prevajanje, ki pa je v trenutni fazi kvečemu polavtomatsko.

## Literatura

- [1] Špela Arhar, Vojko Gorjanc, Simon Krek. FidaPLUS corpus of Slovenian. The New Generation of the Slovenian Reference Corpus: Its Design and Tools. V *Proceedings of the Corpus Linguistics Conference*, 2007. URL <http://www.fidaplus.net/Index.htm>.
- [2] Steven Bird, Ewan Klein, Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 2009. ISBN 0596516495, 9780596516499.
- [3] Cyc Foundation. OpenCyc. URL <http://www.opencyc.org>.
- [4] Tomaž Erjavec, Camelia Ignat, Bruno Pouliquen, Ralf Steinberger. JOS ToTaLc text analyser, 2010. URL <http://n12.ijs.si/analyze/>.
- [5] R.V. Guha, Douglas B. Lenat. Cyc: a mid-term report. *Applied Artificial Intelligence*, 5(1):45–86, 1991.
- [6] P. Jackson, I. Moulinier. *Natural language processing for online applications: Text retrieval, extraction and categorization*. John Benjamins Pub Co, 2007.
- [7] Daniel Jurafsky, James H. Martin. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Pearson Education Inc., 2009.
- [8] George F Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6th edition*. Addison-Wesley Pearson Education, Boston, 2009.
- [9] C.D. Manning, H. Schütze. *Foundations of statistical natural language processing*. MIT Press, 2000.
- [10] Christopher Manning. Statistical natural language processing list of resources. Stanford University, 2010. URL <http://nlp.stanford.edu/links/statnlp.html>.
- [11] G.A. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.
- [12] Slovensko društvo za jezikovne tehnologije. Jezikovne tehnologije v Sloveniji, 2010. URL <http://www.sdjt.si/sdjt-www.html>.
- [13] Ralf Steinberger, Bruno Pouliquen, Anna Widiger, Camelia Ignat, Tomaž Erjavec, Dan Tufis, Dániel Varga. The JRC-Acquis: A multilingual aligned parallel corpus with 20+ languages. V *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC'2006)*, 2006. URL <http://langtech.jrc.it/JRC-Acquis.html>.

## Poglavlje 12

# Evolucijsko računanje

*Čas sega v večnost kot v človeka meč.  
Nastaja strah, preprosta srčnost zginja,  
kar sva bila, ne bova nikdar več,  
predobro veva: nekaj se spreminja.*

Edvard Kocbek

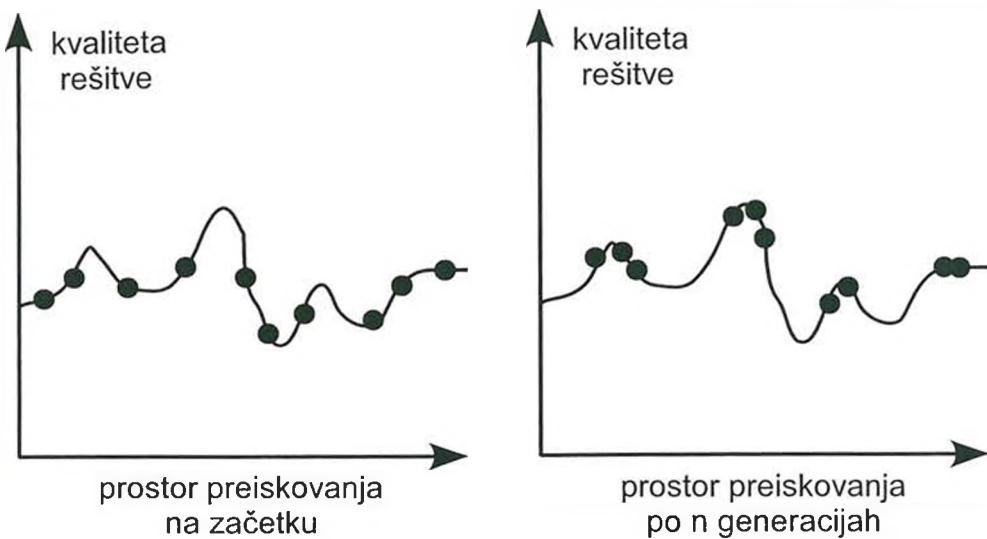
Pojem biološko navdahnjenih računskih metod zajema širok spekter računalniških algoritmov in metod, ki jim je skupno to, da prevzemajo idejo za svoje delovanje od v naravi skozi evolucijo dokazano uspešnih načinov delovanja in preživetja.

Najpomembnejše in najbolj razširjene med njimi so metode evolucijskega računanja, s katerimi se bomo pretežno ukvarjali v tem poglavju. Zanje je značilno, da na različne načine uporabljajo teorijo evolucije kot algoritem. Širok spekter metod s tega področja gleda na napredok, ki je bodisi učenje, bodisi optimizacija, kot na tekmovanje razvijajočih se struktur, rešitev in/ali programov. Te metode delujejo na populacijah podobnih, med seboj neodvisnih osebkov, kar pomeni, da jih je mogoče enostavno paralelizirati. Med njihove najpomembnejše prednosti štejemo tudi prilagodljivost in robustnost. Zelo splošno lahko njihovo delovanje opišemo s psevdokodo na sliki 12.1.

```
generiraj populacijo struktur ;  
do {  
    izračunaj kvaliteto struktur ;  
    izberi strukture za reprodukcijo glede na kvaliteto ;  
    ustvari nove inačice struktur ;  
    zamenjaj stare strukture z novimi ;  
} while (nisi zadovoljen) ;
```

Slika 12.1: Zasnova splošne metode za evolucijsko računanje.

Splošni pojmi, ki smo jih uporabili v psevdokodi, nakazujejo, da jih je mogoče nadomestiti z najrazličnejšimi konkretnimi primeri. Najprej izberemo primerno podatkovno strukturo, kamor na nek način inicializiramo začetno populacijo osebkov. Zanje zahtevamo le to, da jim je mogoče določiti kvaliteto, saj glede na izračunano kakovost na nek način izberemo najboljše med osebki in iz njih ustvarimo nove strukture. Vsaj nekaj na novo ustvarjenih struktur nadomesti stare in celoten postopek ponavljamo, dokler ne dosegнемo cilja, ki smo si ga zadali z našim računanjem.



Slika 12.2: Kvaliteta populacije v prostoru rešitev na začetku evolucijskega računanja (levo) in po nekaj generacijah (desno). Točke predstavljajo posamezne osebke.

Primer, kaj lahko pričakujemo od uspešnega evolucijskega računanja, prikazuje slika 12.2. V začetku je populacija dokaj enakomerno razporejena po vsem prostoru in imamo tako le malo dobrih rešitev. Po nekaj generacijah pa se dobre rešitve zgostijo okoli vrhov kriterijske funkcije.

Dobre lastnosti evolucijskih pristopov so:

- prilagodljivost in splošnost: lotimo se lahko skoraj vsakega optimizacijskega problema, kar potrjuje ogromna množica aplikacij na najrazličnejših področjih,
- potrebno je le šibko znanje o problemu: zadošča ustrezeno predstaviti osebke in izračunati njihovo kvaliteto,
- robustnost: evolucijski pristop dobro deluje s celo vrsto začetnih pogojev in večinoma ni pretirano občutljiv na nastavitev sicer številnih parametrov,
- dobimo več alternativnih rešitev, kar je lahko koristno, ko ena od rešitev ne ustreza ali v primeru večkriterijske optimizacije,

- možna je hibridizacija: evolucijski pristopi se enostavno kombinirajo s številnimi drugimi pristopi,
- paralelizacija: ker imamo opravka z neodvisnimi osebki, jih dokaj enostavno vzporedno obdelamo na paralelnih računalniških arhitekturah.

Seveda pa ima metodologija tudi slabe strani. Naštejmo jih nekaj:

- neoptimalnost rešitev: nimamo zagotovil, kako dobre so dobljene rešitve,
- številni parametri: pri evolucijskih algoritmih je potrebno nastaviti precej parametrov (kar zmede začetnike) in pri nekaterih problemih je potrebno vsaj nekaj (sistematicnega) poizkušanja, da metoda začne vračati dobre rešitve,
- računska zahtevnost: ker imamo opravka s celo populacijo osebkov, je računska zahtevnost velika in za uspešne rešitve je včasih potrebno precej generacij.

Sicer pa je pri izbiri različnih optimizacijskih metod potrebno omeniti tudi teorem zastonjskega kosila (*no free lunch theorem*), ki sta ga razvila Wolpert in Macready. Povedano s preprostimi besedami, teorem trdi, da sta poljubna dva algoritma ekvivalentna, če njuno uspešnost povprečimo čez vse možne probleme. Drugače povedano, bolj se izplača za problem poiskati pravi algoritem, kot uporabiti en sam fiksen algoritem na vseh problemih.

Poglavitne tehnikе evolucijskega računanja so:

- genetski algoritmi, kjer je za osebke večinoma značilna nespremenljiva dolžina genetskega zapisa,
- genetsko programiranje, kjer je namen evolucije razviti računalniški program; osebki so večinoma zakodirani z drevesnimi strukturami,
- metode roja, kjer so osebki stalni in ne sledijo naravni selekciji, pač pa obstaja med njimi komunikacija in si izmenjujejo informacije, kje v problemskem prostoru se nahajajo dobre rešitve; predstavnika te metodologije sta metodi roja delcev in kolonije mravelj.

Pri evolucijskem računanju uporabljamо celo vrsto pojmov iz biologije. Treba se je zavestati, da gre le za približne vzporednice, katerih natančen biološki pomen je lahko povsem drugačen. Recimo, gen opisuje eno od lastnosti osebka, alel je vrednost ali pomen lastnosti, npr. barva oči. Evolucija v naši rabi je zgolj variacija frekvence alelov v populaciji skozi čas. Samo šibka podobnost z biologijo velja tudi za pojme reprodukcija, variacija, mutacija, križanje in selekcija.

Evolucijsko računanje je opisano v številnih učbenikih in priročnikih, npr. [1, 2, 3, 4], kot tudi v člankih in na specializiranih spletnih straneh. Precej o drugih biološko navdahnjenih metodah najdemo v [5].

V nadaljevanju poglavja najprej predstavimo nekaj ključnih konceptov iz evolucijskega računanja, predvsem se osredotočimo na genetske algoritme. Začnemo s predstavitvijo osebkov, se pravi s potrebnimi podatkovnimi strukturami in operacijami na njih. Povemo nekaj o funkcijah uspešnosti in o potrebnih spremenljivostih populacije. V zvezi s kvaliteto rešitve omenimo lokalne in globalne ekstreme ter pojasnimo pojma adaptivno križanje in koevolucija. V drugem delu poglavja predstavimo še genetsko programiranje in metode inteligence roja. Kot predstavnika slednjih si ogledamo optimizacijo z rojem delcev in kolonijo mravelj.

## 12.1 Genetski algoritmi

Najpogosteje uporabljana metoda evolucijskega računanja so genetski algoritmi. Predstavimo njihove sestavne dele, ki so razvidni že iz psevdokode na sliki 12.1.

### Predstavitev osebkov

Čeprav lahko principe evolucije uveljavljamo na poljubni podatkovni strukturi, se je v praksi vendarle uveljavilo nekaj uspešnih struktur, ki jih lahko uporabimo za najrazličnejše namene. Naštejmo jih ter jih na kratko predstavimo.

**Bitna** predstavitev zapiše osebek kot zaporedje bitov. Koliko bitov potrebujemo, je odvisno od števila parametrov, ki jih želimo zakodirati. Na primer, dva osebka, predstavljena z desetimi biti, bi lahko zapisali takole: 1101011100 in 0111000101.

**Vektorska** predstavitev zapiše nabor parametrov, ki predstavljajo osebek, kot vektor realnih ali celih števil, npr.: (6.13, 4.89, 17.6, 8.2). Obstaja dilema med vektorsko in bitno predstavitvijo: ali števila zapisati kot bite, ali jih predstaviti direktno? Enoličnega odgovora na to vprašanje teorija evolucijskega računanja ne pozna. Obstajajo pa mnogi namigi in primeri uspešne rabe različnih predstavitev.

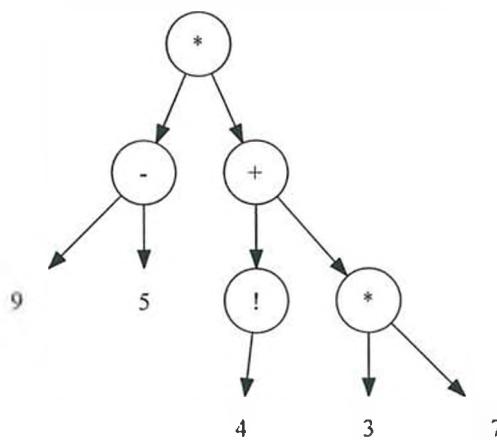
**Nizovna** predstavitev obravnava osebek kot zaporedje znakov. Zaporedje znakov, ki kodirajo najkrajšo pot skozi labirint, bi lahko zapisali kot  $SSVSV SZZJJZJV SZZJJJJVSJ$ , kjer posamezni znaki  $S, J, V$  in  $Z$  označujejo smeri neba, v katere se je potrebno premakniti.

**Permutacije** so naravna predstavitev osebkov za mnoge kombinatorične probleme. Pri problemu trgovskega potnika lahko mesta oštevilčimo in en obhod skozi npr. osem mest zapišemo kot permutacijo (4, 7, 1, 2, 6, 3, 5, 8).

**Drevesna** predstavitev se uporablja za predstavitev zapletenejših osebkov, na primer funkcij, izrazov in pri genetskem programiranju tudi programov. Slika 12.3 prikazuje drevesno predstavitev aritmetičnega izraza  $(9 - 5) * (4! + 3 * 7)$ .

### Grayeve kodiranje binarnih števil

Zaželena lastnost kodiranja je, da so podobni osebki predstavljeni s podobnimi kodami, kar je pomembno za križanje in mutacijo. Običajna predstavitev naravnih števil v binarnem pozicijskem zapisu tej lastnosti ne zadošča. Na primer, števili 7 in 8 se razlikujeta le za ena,

Slika 12.3: Drevesna predstavitev izraza  $(9 - 5) * (4! + 3 * 7)$ .

v dvojiškem pozicijskem zapisu pa imata kodi 0111 in 1000 ter se razlikujeta štirih mestih. Kodiranje, ki problem poskuša rešiti tako, da zagotavlja, da se dve zaporedni števili razlikujeta le na enem mestu, se po svojem odkritelju imenuje Grayevo kodiranje. Kode lahko izračunamo rekurzivno. Tabela 12.1 predstavlja primer tovrstnih 4 bitnih kod.

desetiško	binarno	Grayevo
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Tabela 12.1: Grayevo štiribitno kodiranje naravnih števil.

Kodo za  $n$  bitov izračunamo rekurzivno iz kode za  $(n - 1)$  bitov. Poglejmo si kako zgradimo 3-bitne kode iz 2-bitnih. Začnemo z vsemi dvobitnimi kodami: 00, 01, 11, 10. Postopek

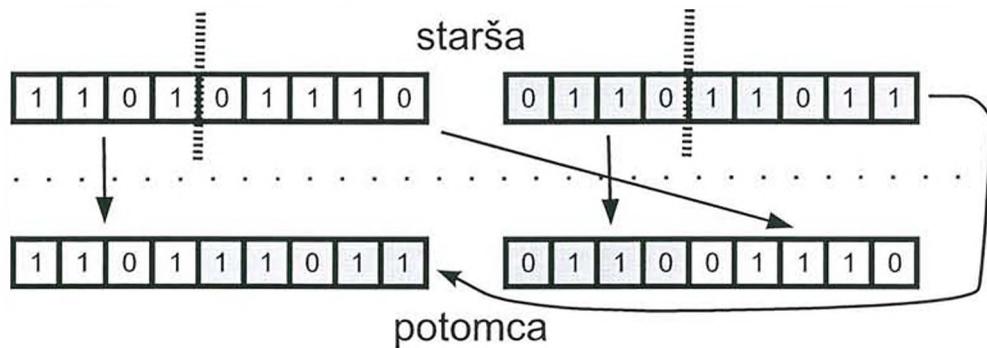
je naslednji:

1.  $(n - 1)$  bitne kode zapišemo v obratnem vrstnem redu: 10, 11, 01, 00
2. originalnim  $(n - 1)$  bitnim kodam dodamo na začetku 0: 000, 001, 011, 010
3. obrnjениm  $(n - 1)$  bitnim kodam dodamo na začetku 1: 110, 111, 101, 100
4. združimo originalne in obrnjene kode z dodanimi 0 in 1 ter dobimo  $n$  bitne kode: 000, 001, 011, 010, 110, 111, 101, 100.

Kot vidimo iz primera na sliki 12.1, se na enem mestu razlikujeta sosednji števili, za dve mesti se razlikujeta števili na oddaljenosti dva, za večjo oddaljenost pa ta lastnost ne velja več, saj koda enostavno ni zadosti dolga, da bi to zagotovljala. Prav tako vidimo, da se samo na enem mestu razlikujeta denimo števili 0 in 15, 1 in 14 ter 2 in 13. Grayeve kodiranje torej le deloma rešuje problem predstavitev bližnjih števil v binarnem zapisu.

## Križanje

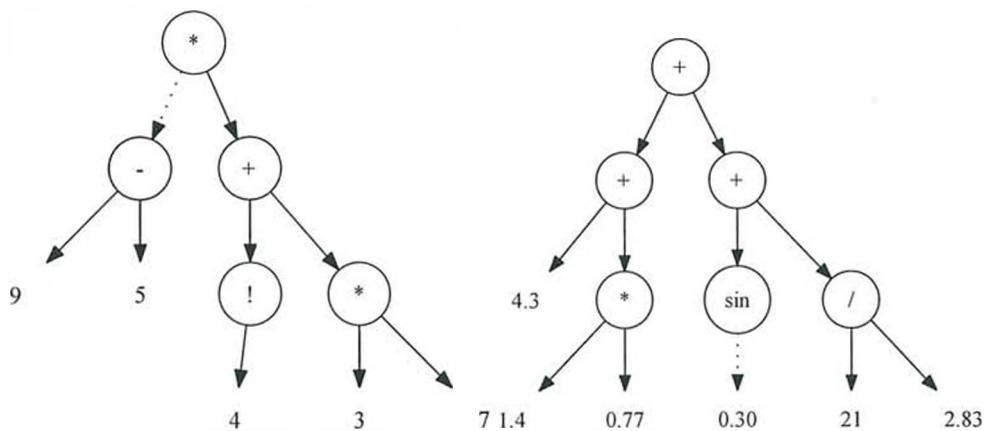
Križanje (*crossover*) je najpomembnejša tehnika izmenjave genetskega materiala med osebkami. Uporablja se eno, dvo in večmestno križanje. Enomestno križanje poteka tako, da naključno izberemo točko križanja in preko te točke izmenjamo genetski material. Slika 12.4 prikazuje, kako iz dveh bitno zapisanih osebkov s križanjem nastaneta dva nova osebka. Naključno izbrano mesto križanja je označeno s črtico. Podobno poteka tudi dvo- in večmestno križanje,



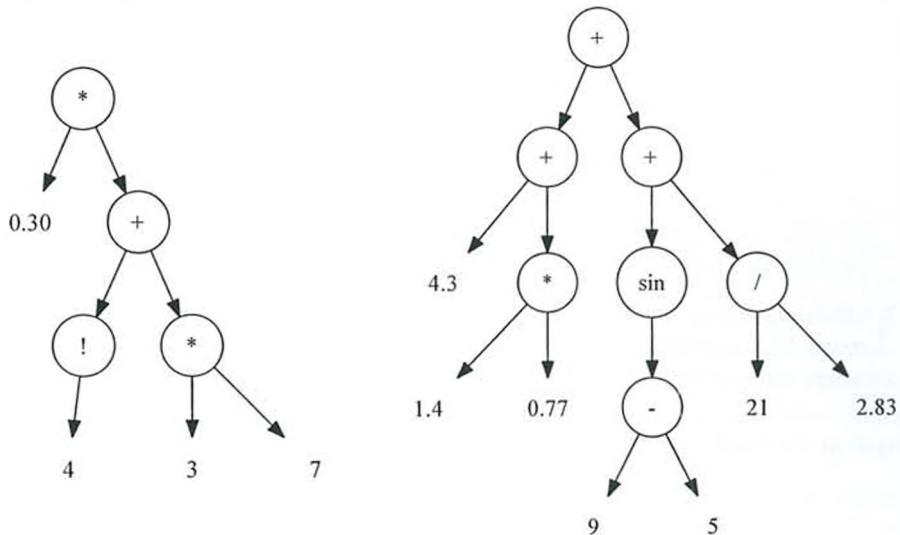
Slika 12.4: Križanje dveh osebkov.

le da izberemo več točk in genetski material izmenjamo preko teh točk.

Nekoliko drugačno je križanje dreves. V vsakem od staršev naključno izberemo poddrevo in poddrevesi izmenjamo. Pri tem moramo paziti, da se ohrani smiselnost genetskega zapisa. Na primer, binarni operator mora tudi po križanju ohraniti dva naslednika. Slika 12.5 prikazuje starša, slika 12.6 pa njuna naslednika nastala s križanjem. Mesto križanja je pri starših označeno s črtkano črto.



Slika 12.5: Drevesi z označenima točkama križanja.



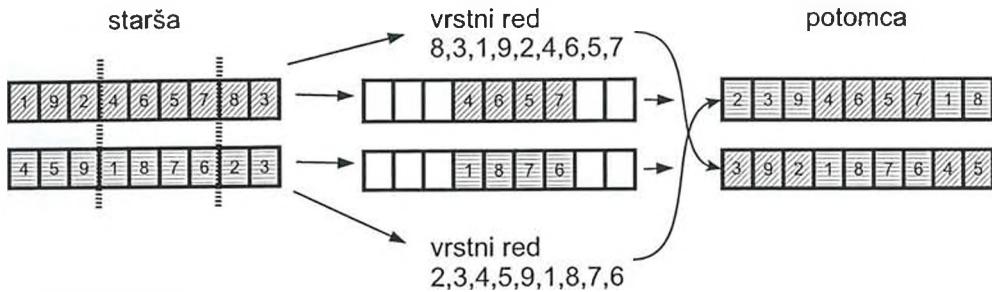
Slika 12.6: Potomca pridobljenega s križanjem dreves na sliki 12.5.

Križanje naj bi generiralo veljavne osebke, saj z neveljavnimi v nadaljevanju nimamo kaj početi. Da je dobro izbrana predstavitev tesno povezana z možnostjo realizacije operatorjev križanja in mutacije, nam pokaže problem trgovskega potnika (*TSP - traveling salesman problem*).

Denimo, da imamo 9 mest, označimo jih s števili 1 ... 9. Vsako mesto predstavimo s štirimi biti in dobimo devet veljavnih kod: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001. Celoten sprehod predstavimo z  $9 \cdot 4 = 36$  biti. Zastavi se vprašanje, kako realizirati križanje. Če dovolimo križanje na poljubnih mestih, lahko dobimo za nekatera mesta

neveljavne kode (1010, 1011, 1100, 1101, 1110 ali 1111), ali pa so nekatera mesta izpuščena, druga pa podvojena, česar ne želimo. Ugotovimo, da bitna predstavitev za problem trgovskega potnika ni dobra. Praksa je pokazala, da je mnogo boljša predstavitev s permutacijami. Za devet mest en obhod napišemo npr. (7, 2, 3, 1, 4, 5, 9, 8, 6).

Pomembno za uspešen operator križanja je tudi, da ohranja strukturo osebka. Vsaj del kode kvalitetne rešitve naj bi se ohranil. Pri trgovskem potniku dosežemo to zahtevo s posebnim urejenim križanjem, ki ohrani dele celih sekvenč. Postopek je prikazan na sliki 12.7 in poteka takole:



Slika 12.7: Urejeno križanje permutacij.

- naključno izberemo dve točki križanja (dvomestno križanje). Pri potomcih mesta med izbranimi točkama ohranimo nespremenjena. Zdaj moramo prevzeti del zapisa od drugega starša, pri tem pa mesta, ki smo jih ohranili med obema točkama, ne smemo ponovno uporabiti.
- Od drugega reza naprej izpišemo vrstni red permutacije (krožno) in iz tega vrstnega reda odstranimo že obstoječa mesta. Nova osebka ohranjata del kode vsakega od staršev in sta zagotovo veljavna obhoda trgovskega potnika.

### Adaptivno križanje

Obstajajo številne inačice križanja. Med zanimivejšimi idejami omenjamo adaptivno križanje, ki je poskus prilagoditve križanja različnim fazam evolucije. Tako kot v naravi tudi pri reševanju problemov določena lastnost ni vedno enako pomembna in v različnih fazah preiskovanja dobro delujejo različni operatorji križanja.

Način, kako lahko to dosežemo pri genetskem algoritmu, je z uporabo nastavkov, ki določajo, katera mesta so podvržena spremjanju. Tabela 12.2 podaja primer križanja dveh osebkov, predstavljenih z vektorjem naravnih števil. Za križanje uporabimo nastavek prvega izbranega starša. Vrednost 0 v nastavku pomeni, da izberemo istega starša, vrednost 1 pa da izberemo drugega starša. S tem nastavkom kreiramo oba naslednika. Tudi nastavki so podvrženi evoluciji, uporabljam pa lahko povsem drugačne operatorje. Tabela prikazuje njihovo enomestno križanje na izbranem mestu 3.

Na ta način se evolucija lažje prilagaja specifičnim zahtevam okolja in dinamiki reševanja problema. Uspešni nastavki vsebujejo koristno informacijo o problemu, ki pa je za človeka

	genetski zapis	nastavek
1. starš	2, 5, 8, 9, 1, 7	010101
2. starš	4, 2, 3, 0, 6, 3	011010
1. potomec	2, 2, 8, 0, 1, 3	010010
2. potomec	4, 5, 3, 9, 6, 7	011101

Tabela 12.2: Adaptivno križanje z nastavki za vektorje naravnih števil. V nastavku 0 pomeni, da se ohrani isti starš, 1 da vzamemo drugega. Oba potomca uporabljata nastavek prvega starša, pri križanju nastavkov pa je uporabljeno enomestno križanje na poziciji 3.

težko razumljiva. Če je potrebno isti problem večkrat reševati, se morda izplača uspešne nastavke shraniti in jih ponovno uporabiti. Opozorimo tudi na različno časovno dinamiko, ki jo ponavadi imajo osebki in njihovi nastavki. Pričakovali bi, da je evolucija nastavkov počasnejša in temu lahko prilagodimo tudi računanje.

Pri osebkih in nastavkih lahko govorimo o koevoluciji, ko dve različni vrsti sodelujeta (lahko tudi tekmujeta) pri dosegbi boljših rešitev in se v tem procesu vzajemno prilagajata.

## Mutacija

Mutacija je postopek, ko na nekem naključno izbranem mestu spremenimo genetski zapis izbranega osebka. Na primer, z naključno izbiro se spremeni bit na drugem mestu osebka 0111001100 → 0011001100.

Brez mutacij bi križanje lahko izmenjavalo samo genetski material vsebovan v začetni slučajno generirani generaciji osebkov. Mutacija doda potencialno novo informacijo. Verjetnost mutacije kljub njeni koristnosti ne sme biti prevelika, saj se sicer pokvari preveč koristnih genov v osebkih in postane evolucijsko računanje podobno naključnemu preiskovanju.

Obstaja več načinov, kako lahko izvedemo mutacijo, npr. enotočkovno, večtočkovno,... Povečini se uporablja uniformna verjetnostna porazdelitev za izbiro točk, ni pa to edina močna izbira. Varianta, ki vključi v genetski algoritem element požrešnega preiskovanja, je Lamarckovska mutacija, kjer izberemo najboljšo mutacijo izmed vseh možnih.

## Evolucijski model

Z besedo evolucijski model imenujemo izbiro osebkov za razmnoževanje. Pri tem je treba odgovoriti na več vprašanj, ki so si marsikdaj protislovna: kako ohraniti dobre osebke, kako zagotoviti raznolikost populacije in kako preprečiti prezgodnjo konvergenco.

Dobri osebki vsebujejo kakovosten genetski material, ki ga želimo ohraniti v populaciji. Zato je smiselno, da dobri osebki tvorijo več potomcev in da se morda celo nespremenjeni prenesejo v naslednjo generacijo. Slednje imenujemo elitizem, ko se določen delež najuspešnejših osebkov (elita) ohrani v populaciji. Nevarnost, ki jo pri tem tvegamo, je prezgodnja konvergenco, saj lahko večina populacije prekmalu postane preveč podobna enemu ali več najboljšim osebkom. Nujno je zato zagotoviti dovolj raznoliko populacijo, ki zagotavlja, da se preišče dovolj velik del problemskega prostora in da ne končamo prekmalu v lokalnih optimumih.

Nekaj uveljavljenih evolucijskih modelov predstavljamo v nadaljevanju.

### Proporcionalna izbira

Smiselno je, da je verjetnost izbire posameznega osebka za razmnoževanje proporcionalna njegovi uspešnosti. To dosežemo tako, da na podlagi funkcije uspešnosti  $f$ , ustvarimo verjetnostno distribucijo, kjer je verjetnost osebka  $i$  enaka

$$p_i = \frac{f_i}{\sum_{j=1}^n f_j}, \quad (12.1)$$

nato pa vzorčimo na podlagi te porazdelitve.

i	$f_i$	proporcionalno		rangovno	
		$p_i$	$r_i$	$p_i$	$r_i$
1	135	0.12	2	0.13	
2	213	0.18	3	0.20	
3	413	0.35	5	0.33	
4	87	0.07	1	0.07	
5	333	0.28	4	0.27	

Tabela 12.3: Tabela osebkov s funkcijo uspešnosti in distribucijo verjetnosti na podlagi proporcionalnosti in rangov.

Implementacija tega vzorčenja si lahko predstavljamo kot ruletno kolo, kjer vsak osebek zaseda širino (kot) kolesa, sorazmerno njegovi verjetnosti. Primer tabelirane funkcije uspešnosti in verjetnostne porazdelitve najdemo v treh levih stolpičih tabele 12.3, slika 12.8 pa ilustrira postopek izbire. Osebek na delu kolesa, ki se ustavi pri oznaki, je izbran za razmnoževanje.

Proporcionalno izbiro je mogoče implementirati tudi tako, da za vsakega od osebkov vnaprej izberemo število potomcev  $n_i$ , ki je sorazmerno njihovi funkciji kvalitete. Denimo, da želimo izbrati  $n$  osebkov, potem osebek  $i$  izberemo  $(np_i)$ -krat. Še vedno pa potrebujemo element naključnosti, ki bo združil pare izbranih osebkov za razmnoževanje.

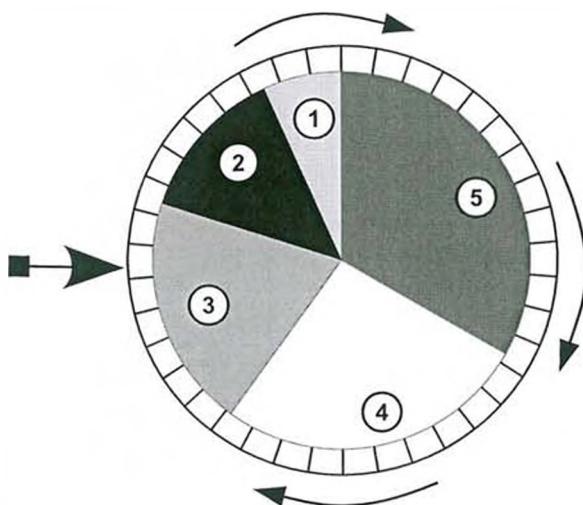
Slabost proporcionalne izbire je, da ni robustna. V primeru, ko ima eden ali več osebkov bistveno večjo funkcijo kvalitete kot vsi ostali, lahko ti osebki (in njihovi potomci) povsem prevladajo in iskanje prezgodaj konvergira.

### Rangovna izbira

Z rangovno izbiro dosežemo večjo robustnost kot pri proporcionalni izbiri. Osebke sortiramo glede na kvaliteto od najslabšega do najboljšega in jim priredimo rang  $r_i$  glede na vrstni red v tej razvrstitvi. Na podlagi rangov nato ustvarimo verjetnostno distribucijo

$$p_i = \frac{r_i}{\sum_{j=1}^n r_j}.$$

Vzorčenje osebkov nato poteka na podlagi te distribucije. Primer rangov in distribucijo na tej podlagi najdemo v desnih dveh stolpičih tabele 12.3.



Slika 12.8: Izbira osebkov z ruletnim kolesom. Delež kolesa, ki ga osebek zaseda, je sorazmeren njegovi uspešnosti iz tabele 12.3.

### Turnirska izbira

Osebke lahko za reprodukcijo izbiramo tudi s simulacijo turnirjev. Izvedemo več turnirjev in zmagovalci so udeleženi v razmnoževanju.

Najprej določimo velikost turnirja  $t$  in verjetnost izbire  $p$ . Za posamezen turnir izberemo kot udeležence  $t$  osebkov iz populacije. Najboljši glede na kriterijsko funkcijo je med zmagovalci z verjetnostjo  $p$ , drugi najboljši z verjetnostjo  $p(1 - p)$ , tretji najboljši z verjetnostjo  $p(1 - p)^2, \dots$

Turnirska izbira je precej fleksibilna. Večji turnirji favorizirajo najboljše glede na funkcijo kakovosti. S  $p = 1$  dobimo dobimo na vsakem turnirju deterministično izbiro. Velikost turnirja  $t = 1$  ustreza naključnemu vzorčenju. Dobra stran tega načina izbire je tudi enostavna paralelizacija.

### Stohastično univerzalno vzorčenje

Slabost dosedaj opisanih metod izbire je velika varianca glede na funkcijo kvalitete. Lahko se zgodi, da najboljši osebek sploh nima nobenega potomca. Manjšo varianco zagotavlja stohastično univerzalno vzorčenje, ki ga podaja psevdokoda na sliki 12.9 in ilustrira slika 12.10.

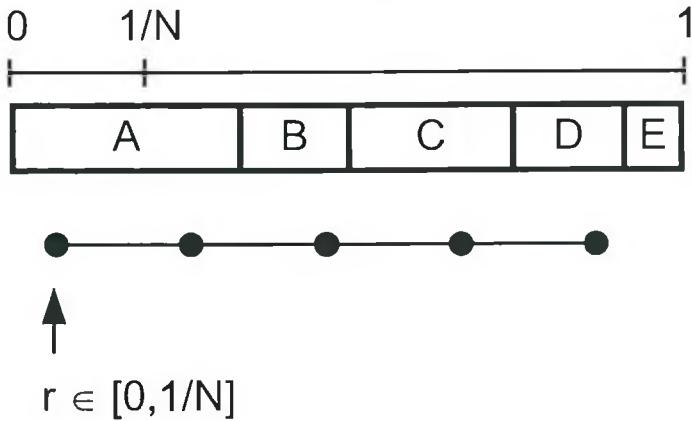
Na podlagi kvalitete osebkov  $f_i$  tvorimo verjetnostno distribucijo kot pri proporcionalni izbiri  $p_i = \frac{f_i}{\sum_{j=1}^n f_j}$ . Osebke naključno razvrstimo na številski trak na interval med 0 in 1 in vsak dobi dolžino traku sorazmerno  $p_i$ . Določimo število osebkov  $N$ , ki jih želimo generirati in naključno izberemo število z intervala  $[0, 1/N]$ . Za razmnoževanje izberemo osebke, ki pokrivajo točke  $r + iN, i \in [0, 1, \dots, N - 1]$ .

```

// Vhod: verjetnostna distribucija osebkov ( $p_1, p_2, \dots, p_m$ ), število naslednikov  $N$ 
// Vrača vektor ( $c_1, c_2, \dots, c_m$ ), kjer  $c_i$  predstavlja število naslednikov osebka  $i$ .  $\sum_{i=1}^m c_i = N$ 
int[] SUS(double[] p, int N) {
    naključno izberi  $r \in [0, 1/N]$ 
    sum = 0.0;
    for ( i = 1 ; i <= m ; i++ ) {
         $c_i = 0$  ;
        sum +=  $p_i$  ;
        while (r < sum) {
             $c_i ++$  ;
            r += 1/N ;
        }
    }
    return c ;
}

```

Slika 12.9: Algoritem stohastičnega univerzalnega vzorčenja.

Slika 12.10: Ilustracija stohastičnega univerzalnega vzorčenja. Osebke razvrstimo na interval  $[0, 1]$  proporcionalno njihovi kvaliteti. Izbiramo  $N$  osebkov, zato začetno točko  $r$  naključno izberemo na intervalu  $[0, \frac{1}{N}]$ . Osebke izberemo na mestih  $r + \frac{i}{N}, i \in [0, 1, \dots, N - 1]$ .

### Nadomeščanje osebkov

Nadomeščanje osebkov iz trenutne generacije z novo generacijo je marsikdaj tesno povezano z izbiro osebkov za razmnoževanje, čeprav gre v principu za dve ločeni nalogi. Imamo precej različnih možnosti.

**Zamenjaj vse** zamenja vse osebke iz trenutne generacije s potomci, pri tem se upošteva kvaliteta osebkov. Na voljo imamo številne možnosti, kako to storiti: proporcionalna

izbira, rangovna izbira, uporabimo lahko ruletno kolo, organiziramo turnirje, ...

**Elitizem** je strategija, ki ohrani določen del najboljših osebkov za naslednjo generacijo.

Dobra stran tega je, da imamo v vsaki generaciji na voljo najboljše osebke in jih ni potrebno posebej hraniti, poleg tega pa zaradi variance pri izbiri ne izgubljamo dobrega genetskega materiala. Slabost je v mogoči prezgodnji konvergenci, če elita prevlada.

**Lokalni elitizem** je inačica, kjer otroci zamenjajo svoje starše, če so boljši. Pri tem je treba paziti, da se genetsko preiskovanje ne izrodi v požrešno iskanje.

### Enoturnirska izbira

Gre za priljubljen in preprost način hkratne izbire osebkov za razmnoževanje in istočasno nadomeščanja osebkov. Poteka takole:

1. naključno razbij populacijo na majhne skupine velikosti  $g$ ,
2. dva najboljša osebka iz vsake skupine se križata in njuna potomca nadomestita najslabša osebka v skupini.

Prednost enoturnirske izbire je, da preživi v naslednjo generacijo ( $g - 2$ ) najboljših osebkov in tako ne izgubljamo prehitro dobrih osebkov pa tudi maksimalna kvaliteta populacije ne pada. Po drugi strani strategija zagotavlja, da še tako dober osebek nima več kot dveh potomcev, in se tako ohranja raznolikost populacije.

### Ustavitev pogoj

Jasno je, da želimo čim boljše rešitve, vendar pa nas pragmatičnost sili, da si zastavimo dosegljiv cilj in ustavimo preiskovanje, ko ga dosežemo ali ko se kvalitete rešitve ne izboljšuje več (dovolj). V praksi se uporablja različni kriteriji za ustavitev preiskovanja. Definiramo jih glede na kvaliteto rešitve, ki jo želimo doseči, ali pa evoluciji vnaprej določimo fiksno število generacij. Predvsem na velikih sistemih se kot ustavitev kriterij uporablja izraba proračuna oziroma razpoložljivega procesorskega časa. Smiselno je preiskovanje ustaviti tudi, kadar dosežemo plato, oziroma kadar se najboljši osebek več generacij ne izboljša. Uporabljamo tudi kombinacijo več kriterijev.

### Pomembne podrobnosti

Kot smo videli, pri evolucijskem računanju le težko damo napotke, ki bi veljali za vse primere. Čeprav v splošnem velja, da je pristop robusten in široko uporaben, je vendar treba paziti na številne podrobnosti in parametre. Nekatere omenjamo v tem razdelku.

### Velikost populacije

Premajhna populacija nima zadosti različnih osebkov, zato rekombiniranje ne more dati zadowljivih rezultatov. Po drugi strani pa je problematična tudi prevelika populacija, saj je računsko zahtevna, poleg tega pa v začetni naključni generaciji vsebuje mnogo šibkih osebkov, ti pa za svoje izboljšanje potrebujejo čas (ozioroma mnogo generacij).

### Nišna specializacija

Večkrat smo že omenili težavo prezgodnje konvergencije, ko si osebki postanejo preveč podobni in se izboljšujejo le v določeni okolini, čemur bi z biološko vzporednico rekli evolucijska niša. Če lahko definiramo podobnost osebkov (npr. med vektorji v  $\mathbb{R}^n$ ), se lahko nišni specializaciji izognemo. Definiramo parameter  $r$ , ki predstavlja radij podobnosti osebkov. Z  $\text{radij}(i, r)$  označimo množico osebkov, ki so v radiju  $r$  osebka  $i$ . Posameznemu osebku funkcijo kvalitete  $f$  zmanjšamo glede na število osebkov zunaj radija podobnosti. Ena od možnosti, kako to storimo, je

$$f_i = \frac{f_i}{q(r, i)}, \text{ kjer } q(r, i) = \begin{cases} 1, & |\text{radij}(i, r)| \leq 4; \\ |\text{radij}(i, r)|/4, & \text{sicer.} \end{cases}$$

Na ta način postanejo rešitve, ki jih najde mnogo osebkov, manj privlačne in preiskovanje se usmeri v druge dele problemskega prostora.

### Zakaj genetski algoritmi delujejo?

Obstaja več poskusov razlage delovanja evolucijskih pristopov, vendar trenutno še nobena od razlag ni v celoti zadovoljiva.

Trenutno je najbolj uveljavljena razlaga z zidaki (*building blocks hypothesis*). Ta razlaga definira shemo ozioroma zidak kot nizkonivojsko predstavitev, ki prispeva k nadpovprečni kvaliteti osebka. Hipoteza trdi, da genetski pristopi implicitno in učinkovito identificirajo in sestavljajo zidake ter na ta način sestavljajo vse boljše rešitve iz delnih nizkonivojskih podrešitev. Boljše sheme imajo skozi selekcijo in rekombinacijo osebkov večjo verjetnost preživetja, se večkrat razmnožijo kot slabše oz. naključne sheme in sčasoma prevladajo v populaciji.

Ceprav precej primerov in variant genetskih algoritmov pritrjuje tej hipotezi, ostaja v splošnem nepotrjena in močno kritizirana. Predvsem ji nasprotujejo nekateri poskusi z mutacijami, ki namenoma razbijajo potencialno koristne zidake, kljub temu pa dosežejo enako kvaliteto rešitev kot metode, ki tovrstnih mutacij ne uporabljajo.

### Večkriterijska optimizacija

Pomembna vrsta problemov, ki se rešuje z evolucijskimi pristopi pa tudi drugimi biološko navdahnjenimi metodami, je večkriterijska optimizacija. Zanjo je značilno, da ima kriterijska funkcija kvalitete rešitve več komponent, ki jih vse želimo optimirati. Na primer, pri izdelku želimo optimirati ceno, kvaliteto, vpliv na okolje, ...

Formalno to zapišemo

$$\min F(x) = \min(f_1(x), f_2(x), \dots, f_k(x))$$

Vsaka od komponent ima svoje ekstreme in prav mogoče je, da ne moremo doseči vseh naenkrat. Uvedemo pojem Pareto-optimalne rešitve, ki preprosto povedano pomeni rešitev, ko ne moremo izboljšati ene od komponent, ne da bi pri tem poslabšali vsaj eno od ostalih komponent.

Pri večkriterijski optimizaciji iščemo nabor Pareto-optimalnih rešitev, med katerimi se na koncu odločimo z nekim zunanjim kriterijem. Takšen pristop je pisan na kožo genetskim algoritmom, kjer so v populaciji osebkov lahko zastopani dobri geni vseh komponent. Pri reprodukciji je potrebno upoštevati raznolikost rešitev po vseh kriterijih.

### Tipične aplikacije

Genetski algoritmi zajemajo izredno širok spekter aplikacij. Uporabljajo se predvsem v težavnih problemih, kjer želimo globalno optimizacijo in je to skorajda edina možnost. Priljubljeni so za reševanje problemov, za katere ne obstajajo specializirane metode za reševanje. To so ponavadi problemi, kjer je na voljo le funkcija kvalitete, ne pa tudi njen odvod. Povečini gre za zelo težke probleme. Genetski algoritmi so se izkazali za robustne in omogočajo kombiniranje z drugimi pristopi, predvsem z lokalno optimizacijo, ki lahko pospeši evolucijo v zadnji fazi približevanja ekstremu kriterijske funkcije.

Čeprav je naš spisek pomankljiv, omenimo nekaj najpogostejših aplikacij:

- razporejanje opravil,
- sestavljanje urnikov,
- optimizacija parametrov v problemih z mnogimi lokalnimi ekstremi oziroma multimedialno kriterijsko funkcijo, kjer iščemo globalni optimum,
- bioinformatika,
- načrtovanje,
- večkriterijsko optimiranje,
- ...

### Orodjarne in knjižnice

Zaradi uspešnosti evolucijskih pristopov so te metode vključene v številne splošnonamenske in specializirane knjižnice in orodja, katerih namen je optimizacija in reševanje problemov. Nekaj najbolj znanih je

- CIIlib – computational intelligence library,
- EO (C++) - evolutionary computation library,
- ECJ (Java),
- EvA2 (Java),
- JAGA (Java),
- ECF- Evolutionary Computation Framework (C++),
- Matlab,
- ...

## 12.2 Genetsko programiranje

Z imenom genetsko programiranje označujemo evolucijske pristope na zahtevnejših podatkovnih strukturah, npr. funkcijah, izraznih drevesih in predvsem programih. Problemski prostor je za takšne primere ogromen in tako je glavna težava zagotoviti učinkovitost računanja. Najpogostejša področja uporabe genetskega programiranja so učenje struktur pri načrtovanju izdelkov, učenje topologije nevronskih mrež in učenje gramatik.

Da zagotovimo računsko vzdržnost je osnovna zahteva genetskim operatorjem, kot sta križanje in mutacija, da ostane struktura osebka veljavna tudi po njihovi uporabi. Ker lahko vse te kompleksne strukture predstavimo z drevesi, navedimo primer, kako lahko tovrstno obnašanje zagotavljamo pri križanju dreves.

Sintaktične omejitve zahtevajo pravilne vrednosti v listih in samo funkcijski operatorje v notranjih vozliščih. Vsak funkcijski operator zahteva pravo število argumentov, ki so v drevesni predstavitvi predstavljeni s poddrevesi. Pogosto se zato uporablja poenostavitev, ki jo imenujemo zaprtje (*closure*), namreč vse funkcije vračajo isti podatkovni tip, kar zmanjša število sintaktičnih omejitev. Pri tem pogoju je mogoče križanje preprosto realizirati z zamenjavo dveh naključno izbranih poddreves. Tovrstni primer križanja funkcij prikazuje sliki 12.5 in 12.6. Naključno izbrano mesto križanja je pri starših na sliki 12.5 označeno s črtkano črto. Križanje je zato kljub kompleksnosti strukture računsko nezahtevno, saj je potrebna le prevezava kazalcev, kot to prikazuje rezultat na sliki 12.6.

Velikanski problemski prostor pa kljub temu zahteva posebne ukrepe. Velikost dreves skozi generacije lahko zelo naraste in nujna je dodatna omejitve glede globine dreves ali števila vozlišč v drevesu.

Drug način, da zagotovimo veljavnost dreves, je tipizirano križanje, kjer šele po naključni izbiri prvega poddrevesa izberemo vozlišče ustreznega tipa v drugem drevesu.

## 12.3 Inteligenca rojev

Med ostalimi biološko navdahnjenimi metodami omenimo le dve, roj delcev (*particle swarm optimization*) in kolonijo mravelj (*ant colony*). Od genetskih algoritmov se obe razlikujeta po fiksni populaciji preprostih osebkov, ki se usmerjeno spreminja. V nasprotju z genetskimi algoritmi, kjer se informacija iz generacije v generacijo prenaša preko genetskega zpisa, osebki pa med sabo ne komunicirajo, je tukaj poudarek na medsebojnem sodelovanju osebkov pri doseganju skupnega cilja. Če potegnemo biološko vzporednico, tovrstno kolektivno obnašanje najdemo pri rojih čebel in drugih žuželk ter pri jatah rib in ptic. Za skupinsko dinamiko je neobhodno potrebno, da osebki znotraj populacije med seboj komunicirajo. Vsak posameznik v skupini ima določeno avtonomijo, na primer vsak zase išče hrano, ko pa jo najde, o tem obvesti ostale osebke in nato odkritje izkoristijo vsi skupaj.

### Roj delcev

Optimizacija z rojem delcev je najpreprostejša in verjetno najbolj razširjena metoda, ki izkoristi moč kolektivnega obnašanja. Iskanje maksimuma s to metodo si lahko predstavljamo,

kot da roj delcev išče najvišjo točko v nekem prostoru in si člani roja medsebojno obveščajo o najboljših točkah, ki so jih odkrili, zato se ves roj usmerja v najobetavnejšo smer. V prvi vrsti je metoda namenjena preiskovanju v zveznem prostoru, čeprav obstajajo tudi prilagoditve za diskretne prostore. Vsak osebek (delec) predstavlja dva vektorja.

1. Lokacijski vektor  $\mathbf{x} = (x_1, x_2, \dots)$  predstavlja trenutno lokacijo osebka in ustreza genetskemu zapisu pri genetskih algoritmih.
2. Vektor hitrosti  $\mathbf{v} = (v_1, v_2, \dots)$  predstavlja hitrost in smer potovanja delca v časovnem koraku. Če sta  $\mathbf{x}(t-1)$  in  $\mathbf{x}(t)$  lokaciji delca v času  $(t-1)$  in  $t$  velja  $\mathbf{v} = \mathbf{x}(t) - \mathbf{x}(t-1)$ .

Vsek delec na začetku inicializiramo z naključno lokacijo in naključnim, a majhnim vektorjem hitrosti. Možnosti, ki se uporabljajo, so polovica vektorja razdalje do naključne druge točke, naključna majhna vrednost ali pa kar vektor **0**.

Za to, da je delovanje delcev v roju več kot le vsota posameznih obnašanj, je v roju potrebna izmenjava informacij. Delec ne izmenjuje informacij z vsemi osebki v roju, pač pa le z manjšo skupino t.i. informatorjev, npr. z nekaj delci iz sosedine ali pa z nekaj naključno izbranimi delci. Pri metodi roja delcev vsak delec pozna podatke o svoji dosedanji uspešnosti. V ta namen hrani tri lokacijske vektorje:

- svojo najboljšo lokacijo  $\mathbf{x}^*$ ,
- najboljšo lokacijo  $\mathbf{x}^+$  svojih informatorjev,
- skupno najboljšo lokacijo  $\mathbf{x}^!$ .

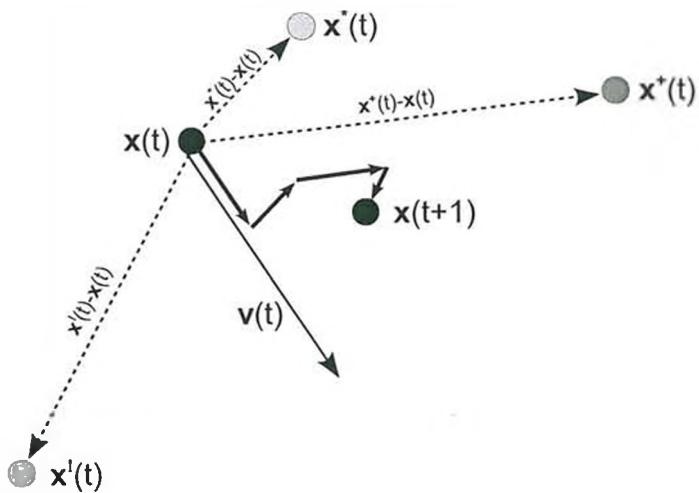
Namesto naslednje generacije, kot pri genetskih algoritmih, imamo pri roju delcev naslednji časovni korak, v katerem se delci premaknejo. Pri premikanju delci upoštevajo vse tri vektorje dosedanje uspešnosti  $\mathbf{x}^*$ ,  $\mathbf{x}^+$  in  $\mathbf{x}^!$ , vektor hitrosti v ter svojo trenutno lokacijo  $\mathbf{x}$  kot to ilustrira slika 12.11. Koliko se pri premikanju upošteva posamezna smer, določajo parametri.

V vsakem časovnem koraku se izvedejo naslednje operacije:

1. izračunamo kvaliteto vsakega delčka in po potrebi popravimo  $\mathbf{x}^*$ ,  $\mathbf{x}^+$  in  $\mathbf{x}^!$ ,
2. za vsak delček določimo spremembe:
  - (a) vektor hitrosti spremenimo tako, da v njem upoštevamo smeri proti  $\mathbf{x}^*$ ,  $\mathbf{x}^+$  in  $\mathbf{x}^!$
  - (b) smeri dodamo tudi nekaj naključnega šuma (vsaki dimenziji prištejemo različne majhne naključne vrednosti) in na ta način zagotovimo, da se izognemo prehitri konvergenci.
3. izračunamo novo lokacijo delčka v smeri vektorja hitrosti

Koliko pri izračunu nove lokacije upoštevamo posamezno smer, je odvisno od vrednosti parametra za vsako smer posebej. Uporabljajo se naslednje oznake in imena parametrov.

- $\alpha$  - delež ohranitve dosedanjega vektorja hitrosti  $\mathbf{v}$ .



Slika 12.11: Premikanje pri optimizaciji z metodo roja delcev.  $x(t)$  je trenutna pozicija delca,  $v(t)$  njegova hitrost,  $x^*$  njegova najboljša lokacija,  $x^+$  najboljša lokacija njegovih informatorjev in  $x^l$  skupno najboljša lokacija. Utežen seštevek teh vektorjev da novo pozicijo  $x(t+1)$ .

- $\beta$  - delčč najboljšč lastne vrednosti  $x^*$ , ki ga upoštevamo. Prevelika vrednost usmerja delček k svojemu maksimumu, ne pa k odkrivanju globalnega optimuma in dobimo skupino požrešnih individualnih iskalcev, ne pa skupinskega iskanja.
- $\gamma$  - delež najboljše vrednosti informatorjev  $x^+$ , ki ga upoštevamo. Ta parameter ima učinek med  $\alpha$  in  $\delta$ , kar pa je odvisno tudi od števila informatorjev, če so ti izbrani naključno. Več informatorjev daje večjo težo globalnemu maksimumu, manj pa lokalnemu.
- $\delta$  - delež najboljše globalne vrednosti  $x^l$ , ki ga upoštevamo. Če je vrednost velika, se delci pomikajo proti globalnemu maksimumu in dobimo eno samo požrešno iskanje, namesto več ločenih iskanj. Da se temu izognemo, marsikdaj postavimo to vrednost na 0.
- $\varepsilon$  - hitrost premikanja delcev. Velika vrednost povzroči, da delci napredujejo v velikih skokih, in se hitro usmerijo proti najboljšim področjem, obenem pa obstaja nevarnost da dobro rešitev kar preskočimo. Z veliko vrednostjo  $\varepsilon$  težko optimiziramo vrednost do natančnega maksimuma, zato se lahko ta vrednost med iskanjem dinamično spreminja.
- swarmsize – velikost roja, določa zahtevnost računanja.

Celoten algoritem prikazuje psevdokoda na sliki 12.12.

```

// vrača najboljši osebek, ki ga najde
Osebek PSO( $\alpha, \beta, \gamma, \delta, \varepsilon$ , swarmsize){
    // vhodni parametri:
    //  $\alpha$  – utež vektorja hitrosti  $v$ 
    //  $\beta$  – utež najboljše lastne vrednosti  $x^*$ 
    //  $\gamma$  – utež najboljše vrednosti informatorjev  $x^+$ 
    //  $\delta$  – utež najboljše globalne vrednosti  $x^!$ 
    //  $\varepsilon$  – korak premikanja delcev
    // swarmsize – velikost roja

    P = [] ; // vektor delcev
    for (i=0 ; i < swarmsize ; i++)
         $P_i$  = nov delec z naključno pozicijo  $x_i$  in naključno hitrostjo  $v_i$  ;
    best = null ;
    do {
        for (i=0 ; i < swarmsize ; i++) {
            določi kvaliteto( $P_i$ )
            if ( kvaliteta( $P_i$ ) > kvaliteta(best) )
                best =  $P_i$  ;
        }
        for (i=0 ; i < swarmsize ; i++) {
             $x^*$  = lokacija  $x_i$  z dosedaj največjo kvaliteto ;
             $x^+$  = lokacija  $x$  informatorjev delca  $P_i$  z dosedaj največjo kvaliteto ;
             $x^!$  = lokacija  $x$  z dosedaj največjo kvaliteto med vsemi delci ;
            for (j=0; j < steviloDimenzij; j++) { // izračun po komponentah vektorjev
                b = naključno število med 0 in  $\beta$  ;
                c = naključno število med 0 in  $\gamma$  ;
                d = naključno število med 0 in  $\delta$  ;
                 $v_{i,j} = \alpha v_{i,j} + b(x_j^* - x_{i,j}) + c(x_j^+ - x_{i,j}) + d(x_j^! - x_{i,j})$  ;
            }
             $x_i = x_i + \varepsilon v_i$  ;
        }
    } while (best ni optimalen AND čas ni iztekel) ;
    return best ;
}

```

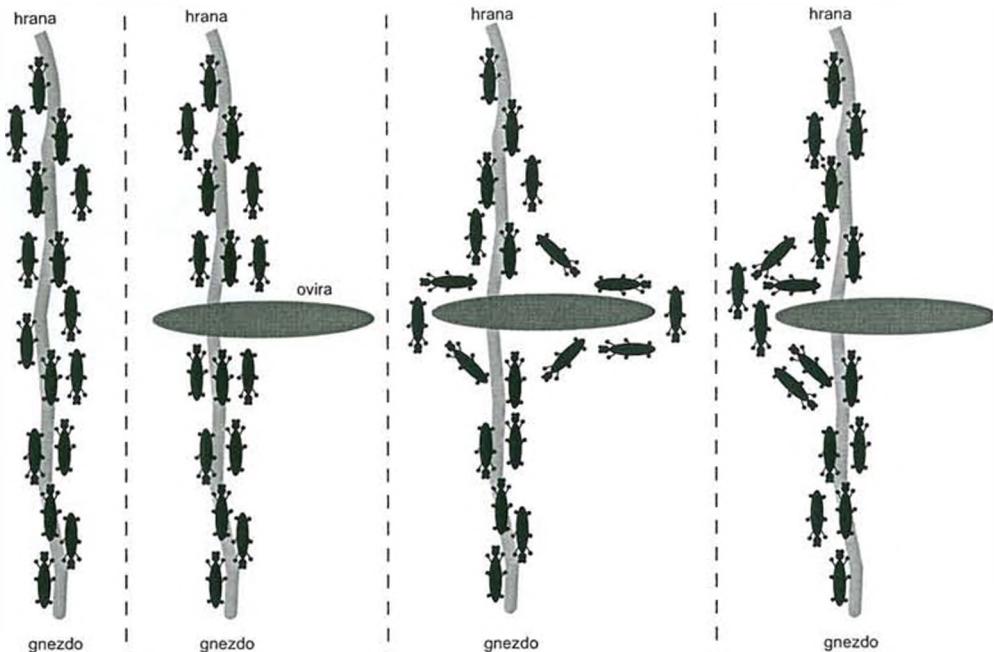
Slika 12.12: Psevdokoda optimizacije z rojem delcev.

### Kolonija mravelj

Biološko navdahnjen je tudi algoritem za optimizacijo s kolonijo mravelj. Znano je, da mravljje poti k hrani označijo s posebno snovjo - feromoni, ki jo druge mravljje iz gnezda zaznajo in ji sledijo na poti do hrane. Algoritem je uspešen na problemih kombinatorične optimizacije, še posebej pri iskanje dobrih poti skozi graf, kot to zahteva npr. problem trgovskega

potnika.

Idejo algoritma lahko povzamemo takole. Na začetku mravlje iščejo hrano naključno. Mravlja, ki najde hrano, med vračanjem v gnezdo označi svojo pot s feromoni. Druge mravlje, ki naletijo na feromonsko sled, ji sledijo in jo ojačajo, če tudi najdejo hrano. S časom sled slabi, kar pomeni, da na slabih oziroma daljših poteh izhlapi več feromonov. Idejo ilustrira slika 12.13, kjer je časovni potek delovanja algoritma ilustriran s slikami od leve proti desni.



Slika 12.13: Potek preiskovanja s kolonijo mravelj ilustrira zaporedje slik od leve proti desni.

Izhlapevanje poskrbi, da iskanje ne konvergira k enemu samemu cilju. Metoda kolonije mravelj je zato primerna tudi za probleme, ki se s časom in dinamično spreminjajo, npr. poti se ukinjajo, spreminja, nastajajo nove, pojavljajo se ovire, itd. Osnutek algoritma prikazuje slika 12.14.

Splošnost psevdokode dopušča številne inačice in dejansko se v praksi izvajajo številne različice tega postopka. Opišimo eno od pogostejših variant.

Med konstruiranjem rešitve osebek – mravlja prehode izbira stohastično. Poleg cene prehoda upošteva tudi feromone in verjetnost prehoda med stanji je odvisna tako od feromonov kot od cene prehoda, razmerje pa kontrolirajo dodatni parametri. Pomemben element algoritma je ažuriranje količine feromonov na sledi (povezavi). Z  $\tau_{i,j}$  označimo količino feromonov na povezavi med vozliščema  $i$  in  $j$ ,  $\rho$  pa je parameter, ki kontrolira hitrost izhlapevanja. Na vsakem časovnem koraku ažuriramo količino feromonov kot

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} + \Delta\tau_{i,j}.$$

```
// vrača najboljšo najdeno rešitev
Rešitev ACO() {
    inicializiraj feromonske sledi ;
    do {
        for (vsaka mravlja) {
            sestavi rešitev z uporabo feromonskih sledi ;
            popravi sledove: izhlapevanje, ojačanje ;
        }
    } while (! zadovoljen) ;
    return najboljša najdena rešitev ;
}
```

Slika 12.14: Psevdokoda optimizacije s kolonijo mravelj.

Količino novih feromonov na posamezni povezavi smo označili z  $\Delta\tau_{i,j}$ . Če je  $c_{i,j}$  cena povezave med  $i$  in  $j$ , je v najpreprostejši različici dodatek feromonov kar

$$\Delta\tau_{i,j} = \begin{cases} \frac{1}{c_{i,j}}, & \text{če mravlja potuje po povezavi } (i, j); \\ 0, & \text{sicer.} \end{cases}$$

Za to, kdaj obnavljamo feromonske sledi, imamo na voljo številne možnosti. Najpogosteje se sledi obnavljajo takrat, ko pridejo na cilj vse mravlje, mogoče pa je sledi obnavljati tudi med konstrukcijo na vsakem koraku, ali ko pride na cilj posamezna mravlja.

Količina novih feromonov je lahko odvisna tudi od kvalitete najdene rešitve. Ko pridejo na cilj vse mravlje, se rešitve sortirajo in feromoni se obnovijo sorazmerno s kvaliteto rešitve ali proporcionalno vrstnemu redu rešitve med sortiranimi rešitvami. Elitistično obnavljanje feromonov favorizira najboljšo najdeno rešitev, slabe rešitve pa lahko celo povzročijo negativne količine novih feromonov.

Poglejmo delovanje kolonije mravelj na primeru trgovskega potnika. Mesta (vozlišča) označimo zaporedno  $1, 2, \dots, n$ . Razdaljo oz. ceno prehoda med vozliščema  $i$  in  $j$  zapišemo kot  $c_{i,j}$ . Za privlačnost prehoda med  $i$  in  $j$  je smiselno uporabiti inverz cene  $\eta_{i,j} = 1/c_{i,j}$ . Začetno mesto lahko mravlja izbere naključno, v nadaljevanju pa izračunamo verjetnosti prehodov in pot izberemo uteženo glede na to verjetnost. Verjetnost, da mravlja izbere pot med  $i$  in  $j$ , izračunamo kot

$$p_{i,j} = \frac{\tau_{i,j}^\alpha \eta_{i,j}^\beta}{\sum_{k \in S} \tau_{i,k}^\alpha \eta_{i,k}^\beta} \text{ za } \forall j \in S,$$

kjer množica  $S$  vsebuje vsa vozlišča, ki še niso vsebovana v trenutni poti, parametra  $\alpha$  in  $\beta$  pa kontrolirata vpliv feromonov in privlačnosti povezav. Psevdokoda na sliki 12.15 podaja podrobnosti algoritma. V predstavljeni inačici smo ojačali le najboljšo pot.

```

// vrača najboljšo pot
Pot ACO_TSP( $\alpha, \beta, \rho, \Delta$ ) {
    //  $\alpha$  – utež vpliva feromonov
    //  $\beta$  – utež vpliva privlačnosti povezav
    //  $\rho$  – hitrost izhlapevanja feromonov
    //  $\Delta$  – količina novih feromonov

    inicializiraj feromonske sledi ;
    do {
        for (vsaka mravlja) {
            // sestavljanje rešitev z uporabo feromonskih sledi
             $S = \{1, 2, \dots, n\}$ ; // množica še ne izbranih mest
            i = naključno izbrano začetno mesto iz  $S$  ;
             $S = S - \{i\}$  ;
            do {
                izberi novo mesto  $j$  z verjetnostjo  $p_{i,j} = \frac{\tau_{i,j}^\alpha \eta_{i,j}^\beta}{\sum_{k \in S} \tau_{i,k}^\alpha \eta_{i,k}^\beta}$  ;
                 $S = S - \{j\}$ ;
                i = j ;
            } while ( $S \neq \emptyset$ ) ;
        }
         $\pi =$  najboljša rešitev med vsemi mravljam ;
        // popravi sledove
        for (  $i, j \in \{1, 2, \dots, n\}$  )
             $\tau_{i,j} = (1 - \rho)\tau_{i,j} + \rho \tau_{j,i}$  ; // izhlapevanje
        for ( povezava  $(i, j) \in \pi$  )
             $\tau_{i,j} = \tau_{i,j} + \Delta$  ; // ojačanje samo najboljše poti
    }
    } while (! zadovoljen)
    return najboljša najdena rešitev
}

```

Slika 12.15: Psevdokoda optimizacije s kolonijo mravelj.

## Literatura

- [1] Daniel Ashlock. *Evolutionary computation for modeling and optimization*. Springer-Verlag, 2006.
- [2] Tomas Bäck, David B. Fogel, Zbigniew Michalewicz, (ur.). *Handbook of evolutionary computation*. Taylor & Francis, 1997.
- [3] Tomas Bäck, David B. Fogel, Zbigniew Michalewicz, (ur.). *Evolutionary computation I*:

- Basic algorithms and operators.* Taylor & Francis, 2000.
- [4] David B. Fogel, Tomas Bäck, Zbigniew Michalewicz, (ur.). *Evolutionary computation 2: advanced algorithms and operators.* Taylor & Francis, 2000.
- [5] Talbi, El-Ghazali . *Metaheuristics, From Design To Implementation.* Wiley, 2009.

## Poglavlje 13

# Hevristično preiskovanje

*Kdor išče modrost je moder, kdor misli, da jo je našel, je bedak.*

Abu Nasr al-Farabi

Preiskovanje je ena od osnovnih tehnik umetne inteligence in je opisano v številnih učbenikih umetne inteligence, npr. [6, 3, 1].

V grobem ga lahko razdelimo na izčrpno (tudi slepo ali neinformirano) preiskovanje in na hevristično ali informirano preiskovanje. Skupna lastnost vseh preiskovalnih algoritmov je predstavitev problema s t.i. prostorom stanj in s tem pretvorbo problema v usmerjeni graf, ponavadi drevo. Vsako stanje problema ustrezava enemu vozlišču v grafu. Usmerjene povezave med vozlišči ustrezajo možnim prehodom med posameznimi stanji. Preiskovalni algoritem potrebuje generator naslednikov, ki za dano vozlišče vrne množico z njim povezanih stanj. Definiramo tudi eno začetno stanje in eno ali več končnih stanj, ki ustrezajo rešenemu problemu. Končno stanje definiramo glede na pot, ki nas je privedla do stanja ali glede na neko merljivo lastnost stanja. Včasih povezavam ozioroma prehodom med stanji priredimo tudi ceno. Ceno vozlišča  $n$  označimo z  $g(n)$  in ga definiramo kot vsoto cen povezav na poti od začetnega stanja do  $n$ . Množico vseh stanj danega problema imenujemo prostor stanj.

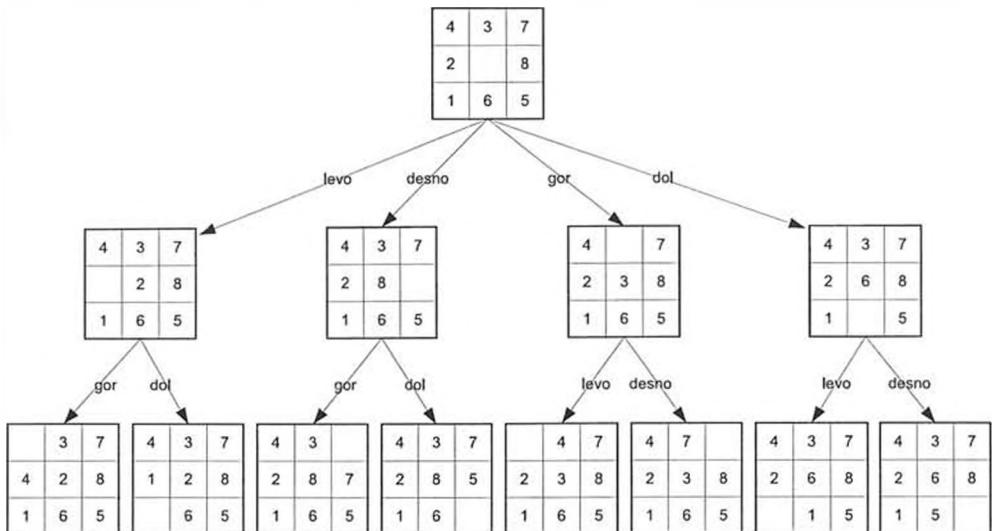
Kot enostaven primer predstavitve problema si poglejmo igro drsečih ploščic. V kvadrat velikosti  $3 \times 3$  razvrstimo osem ploščic označenih s števili od 1 do 8, kar pomeni, da je eno polje prosto. V to prazno polje lahko zdrsne katerakoli sosednja ploščica. Cilj igre je, da iz poljubne začetne konfiguracije ploščic sestavimo neko točno določeno razvrstitev števil, npr. tisto na sliki 13.1.

Vsaka možna konfiguracija ploščic predstavlja eno stanje. Veljavni premiki iz danega stanja generirajo naslednja stanja. Slika 13.2 prikazuje del prostora stanj in možne prehode med stanji za dano začetno konfiguracijo, ki se nahaja v korenju drevesa.

Drugačen primer prostora stanj, kjer je vsako stanje tudi ciljno stanje in iščemo najboljšo (optimalno, najcenejšo) rešitev, je problem trgovskega potnika (*TSP - traveling salesman problem*). Naloga je v grafu poiskati Hamiltonov cikel, torej najkrajšo krožno pot, ki povezuje vse točke grafa in gre skozi vsako točko le enkrat. Primer tovrstnega grafa predstavlja slika

1	2	3
8		4
7	6	5

Slika 13.1: Ciljno stanje igre drsečih ploščic.



Slika 13.2: Del prostora stanj igre drsečih ploščic.

13.3. Prostor stanj tukaj predstavljajo vsi možni obhodi skozi graf. Za poln graf bi bile to kar vse možne permutacije vozlišč. Sosedna stanja lahko dobimo z neko perturbacijo, na primer z zamenjavo dveh vozlišč.

Omenimo še, da smo s predstavitvijo problema v obliki grafa pridobili analitične zmožnosti in algoritme iz teorije grafov, kar lahko s pridom izkoristimo.

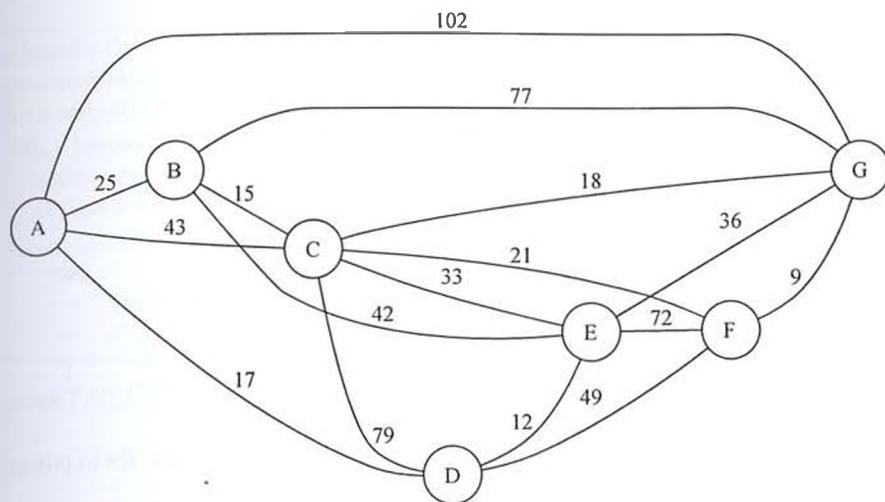
Preden se lotimo preiskovalnih algoritmov, si oglejmo, kako meriti uspeh preiskovalnih algoritmov. Uporabljajo se štirje kriteriji:

**popolnost** – ali algoritem zagotovo najde rešitev, če ta obstaja,

**optimalnost** – ali algoritem najde najboljšo možno (optimalno) rešitev,

**časovna zahtevnost** – koliko časa porabi algoritem, da najde rešitev,

**prostorska zahtevnost** – koliko pomnilnika je potrebno za preiskovanje.



Slika 13.3: Primer grafa, na katerem lahko rešujemo problem trgovskega potnika.

Ker preiskovalni algoritmi nemalokrat preiskujejo neskončne prostore stanj, je za izračun časovne in prostorske kompleksnosti smiselno definirati  $b$  kot vejivteni faktor oziroma maksimalno število naslednikov poljubnega vozlišča,  $d$  kot globino najplitvejšega ciljnega vozlišča in  $m$  - najdaljšo pot v prostoru stanj. Časovno zahtevnost nato izražamo kot število generiranih stanj, prostorsko pa kot število stanj, ki jih je potrebno naenkrat hraniti v pomnilniku. Obe zahtevnosti podajamo za preiskovanje drevesa, saj v nadaljevanju opisani algoritmi graf razvijajo kot drevo.

## 13.1 Neinformirano preiskovanje

Preiskovanje, ki ne uporablja dodatne informacije o problemskem prostoru, imenujemo neinformirano preiskovanje. Uporablja se predvsem tri strategije preiskovanja: v širino, v globino in iterativno poglavljanje. Na kratko si jih poglejmo, saj nam bodo služile kot izhodišče za primerjavo z bolj naprednimi metodami.

### V globino

Ideja algoritma je med generiranimi nasledniki izbrati prvega in se vračati šele, ko naslednikov zmanjka. Ker algoritem preiskuje graf, kot bi bil drevo (kar pomeni, da lahko nekatera stanja preiše tudi večkrat), si preiskovanje lahko predstavljamo, kot da gre od leve proti desni strani drevesa. Podrobnosti ilustrira psevdokoda algoritma na sliki 13.4, ki jo poklicemo z `depth(start_state)`. Metoda vrača zastavico tipa status, ki sporoča uspešnost iskanja: uspeh (`SUCCESS`) ali neuspeh (`FAILURE`).

Predstavljena je le zelo osnovna varianta algoritma, ki ne preverja, ali je naslednik morda že del trenutno preiskane poti, zato bi preiskovanje v takem primeru zašlo v neskončno zanko.

```

status depth (state current ) {
    if (goal (state))
        return SUCCESS;
    suc = successors (state);
    foreach ( s in suc )
        if ( depth(s) == SUCCESS )
            return SUCCESS ;
    return FAILURE ;
}

```

Slika 13.4: Algoritem preiskovanja v globino.

Pri tem algoritmu je smiselno omejiti tudi maksimalno globino preiskovanja, kot to prikazuje koda na sliki 13.5.

```

status depthLimited (state current, int toDepth) {
    if (goal (state))
        return SUCCESS;
    if ( toDepth>0 ) {
        suc = successors (state);
        foreach ( s in suc )
            if ( depthLimited(s, toDepth-1) == SUCCESS )
                return SUCCESS ;
    }
    return FAILURE ;
}

```

Slika 13.5: Algoritem preiskovanja v globino z omejeno globino.

Algoritem preišče  $O(b^m)$  vozlišč, kar je lahko mnogo več kot  $O(b^d)$ . Za prostore stanj, kjer globina ni omejena, je to število lahko kar neskončno. Dobra stran algoritma je, da porabi zelo malo pomnilnika, saj v vsakem trenutku hrani na skladu le vozlišča, preko katerih je prispel v dano vozlišče, torej je prostorska zahtevnost reda  $O(bm)$ .

### V širino

Preiskovanje v širino pregleduje graf po nivojih: najprej začetno vozlišče, nato njegove naslednike, naslednike naslednikov,... Ta strategija zagotavlja, da bomo najprej našli najplitvejše ciljno vozlišče. Pri implementaciji uporabimo vrsto (*queue*). Podrobnosti so razvidne iz psevdokode na sliki 13.6.

Ker pregledujemo po nivojih, pregledamo na prvem nivoju 1 vozlišče, na drugem  $b$  vozlišč, na tretjem  $b^2$ , skupaj torej  $1+b+b^2+\dots+b^{d-1} = O(b^d)$ , kar ustreza časovni zahtevnost

```

status breadth () {
    queue.makenull(); // inicializacija
    queue.enqueue(start_state); // začetno stanje v vrsto
    while ( !queue.isEmpty() ) {
        state = queue.dequeue(); // trenutno stanje je prvo v vrsti
        if ( goal(state) )
            return SUCCESS;
        else
            queue.enqueue(successors(state)); // na konec vrste
            // closed.enqueue(state); // če bi hranili vsa vozlišča
    }
    return FAILURE;
}

```

Slika 13.6: Algoritem preiskovanja v širino.

te metode. Tudi prostorska zahtevnost je reda  $O(b^d)$  - ko najdemo ciljno vozlišče na nivoju  $d$ , v vrsti že hranimo vozlišča za naslednji nivo.

### Iterativno poglabljanje

Da se izognemo prostorski zahtevnosti preiskovanja v širino in ohranimo preiskovanje po nivojih, uvedemo strategijo z imenom iterativno poglabljanje. Uporabimo preiskovanje v globino z omejeno globino, ki pa jo postopoma zvišujemo. Nova vozlišča tako odpiramo po nivojih, seveda pa moramo vsa vozlišča do mejne globine preiskati vsakič znova. Podrobnosti podaja psevdokoda na sliki 13.7.

```

status deepening () {
    int depth = 1 ;
    while (depth < MAX_DEPTH) {
        state = depthLimited(start_state, depth) ;
        if (state == SUCCESS)
            return SUCCESS ;
        else depth++ ;
    }
    return FAILURE ;
}

```

Slika 13.7: Algoritem iterativnega poglabljanja.

Prostorska zahtevnost algoritma je  $O(bd)$ . Za določitev časovne zahtevnosti pa upoštevamo, da vozlišča na največji globini  $d$  razvijemo enkrat, na globini  $d - 1$  dvakrat, in tako

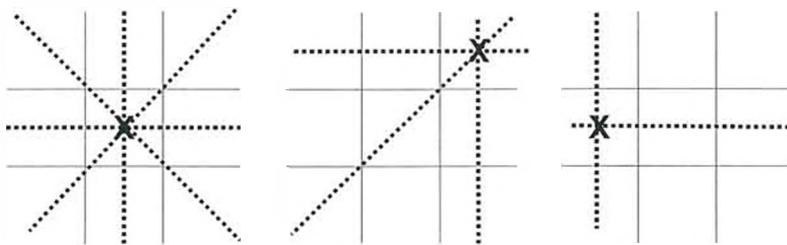
naprej do naslednikov začetnega vozlišča, ki jih razvijemo  $d$ -krat. Skupaj torej razvijemo  $bd + b^2(d - 1) + \dots + b^d 1 = O(b^d)$  vozlišč. Dobimo enako asymptotično časovno zahtevnost kot pri iskanju v širino, seveda pa večjo multiplikativno konstanto, kar pomeni, da manjši prostor zamenjamo z večjim časom.

Za zelo velike prostore obstaja tudi kombinirana strategija, kjer preiskujemo v širino skorajda do meje razpoložljivega prostora, nato pa od vseh vozlišč v vrsti dalje izvajamo iterativno poglabljanje.

## 13.2 Informirano preiskovanje

Hevristično preiskovanje imenujemo tudi informirano preiskovanje, saj uporabljamo dodatno informacijo o problemskem prostoru za usmerjanje preiskovanja. Beseda hevristika označuje (približne) metode in pravila za odkrivanje novega. Beseda izhaja iz znanega Arhimedovega vzklika med kopanjem v kadi: "Eureka!" kar pomeni: "Našel sem!" Pristop uporabljamo, kadar je iskanje optimalne rešitve računsko prezahtevno. V našem izvajanju hevristika označuje neko uporabno informacijo, ki usmerja preiskovanje v obetavno smer.

Kot preprost primer si poglejmo znano igro križcev in krožcev, kjer nasprotnika izmenoma na polje  $3 \times 3$  postavlja križce in krožce, zmaga pa tisti, ki ima tri svoje znake v ravni črti navpično, vodoravno ali diagonalno. Denimo, da želimo v dani poziciji poiskati dobro potezo in ne moremo preiskati vseh možnih nadaljevanj. Smiselna hevristika bi bilo lahko število načinov, kako lahko zmagamo v nadaljevanju. Izberemo torej potezo, ki nam daje največ možnosti za zmago. Primer za prvo potezo ilustrira slika 13.8. Zaradi simetrije prikazani načini pokrivajo vse možnosti, zato je glede na to hevristiko najbolj obetavna postavitev v središče polja, ki ima štiri možne načine za zmago. Hevristično preiskovanje bi za začetek izbral to potezo.



Slika 13.8: Hevristika za križce in krožce: na koliko načinov lahko zmagam?

### Požrešno preiskovanje

Najpreprostejša strategija, ki izkorišča hevristično informacijo, je požrešno preiskovanje (*greedy search, hill-climbing*), ki na vsakem koraku izbere najboljšega naslednika glede na hevristično oceno kvalitete stanja. Algoritem predstavi psevdokoda na sliki 13.9.

```

status hillClimbing () {
    stack.makenull(); // inicializacija
    stack.push(start_state); // začemo na vrh
    while ( !stack.isEmpty() ) {
        state = stack.pop(); // state = na vrhu sklada
        if ( goal(state) )
            return SUCCESS;
        else {
            suc = successors(state);
            sort(suc); // uredi glede na hevristično oceno
            stack.push(suc);
        }
    }
    return FAILURE ;
}

```

Slika 13.9: Algoritem požrešnega preiskovanja.

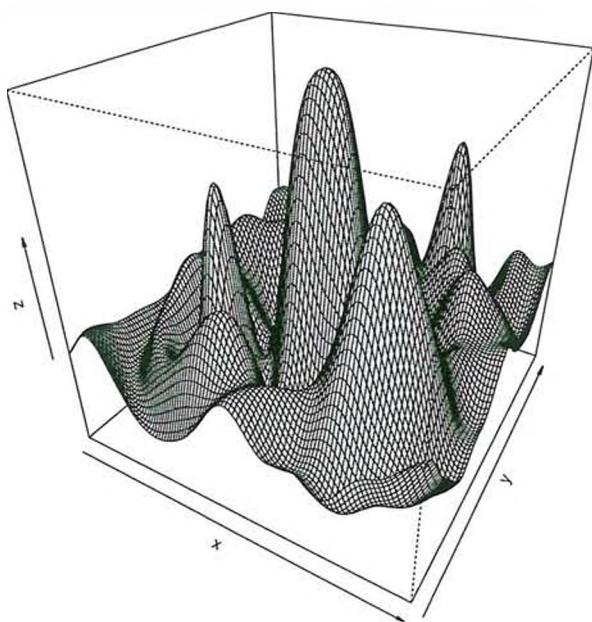
Težava požrešnega preiskovanja je prav v njegovi požrešnosti: ker vedno izbere najboljšo ocenjeno nadaljevanje, pogosto pristane v lokalnem ekstremu, kjer so vsa nadaljevanja manj obetavna od trenutno najboljšega, zato ne najde globalno najboljših rešitev. Takšen primer ilustrira slika 13.10, kjer vidimo funkcijo z več lokalnimi maksimumi, ki lahko zavedejo požrešno preiskovanje, da ne najde globalno optimalne rešitve.

Drugo možno težavo, ki pa ni omejena le na požrešne algoritme, ampak povzroča težave prav vsem preiskovalnim strategijam, imenujemo plato in je prikazan na sliki 13.11. Velik del prostora ocen kvalitete ima skoraj enako vrednost in predstavlja plato (označen s križcem), dobre rešitve pa so locirane drugje. Preiskovalni algoritmi na podlagi podatkov iz sosedstva ne znajo izbrati obetavne poti.

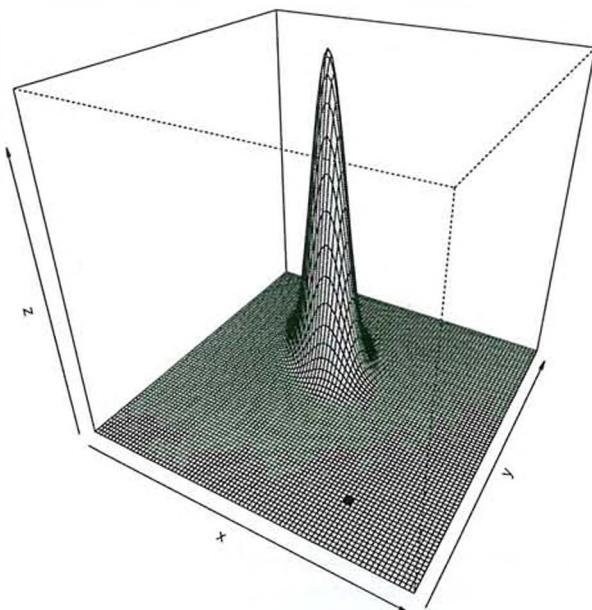
Za iskanje v zveznem prostoru so problematični tudi grebeni, kot ga prikazuje slika 13.12. V s piko označeni točki na vrhu grebena je ocena vozlišč skoraj v vseh smereh manjša, le v eni sami ozki smeri narašča, zato imajo algoritmi težave tudi z grebeni.

Nekatere od opisanih težav požrešnih algoritmov rešujemo s pogledom naprej (*look-ahead*), ki na vsakem koraku razvije nekaj nivoje vozlišč naprej in se potem odloči za smer, ki izgleda najbolj obetavna. Zavedati se je potrebno, da lahko to težave le omili, saj dlje od nekaj korakov naprej tudi ta metoda ne vidi. To kratkovidnost imenujemo problem horizonta in jo karakterizira slika 13.13.

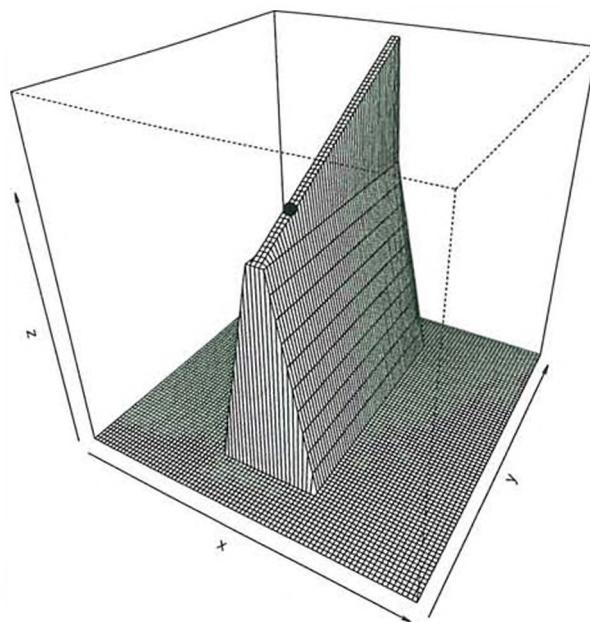
Denimo, da smo se iz začetnega vozlišča premaknili v točko A. Števila poleg vozlišč pomenijo njihovo hevristično oceno. Če izberemo pogled za 1 naprej, bomo izbrali srednjo pot. Pri pogledu za dva nivoja naprej, bi izbrali desno vejo, pri pogledu za 3 nivoje naprej, pa bi odšli po levi veji. Horizont je torej omejitev, ki je ne moremo preseči, saj razen v posebnih primerih nimamo zagotovila, kaj lahko pričakujemo že na za ena večji globini.



Slika 13.10: Prikaz lokalnih in globalnega maksimuma.



Slika 13.11: Prikaz platoja.



Slika 13.12: Prikaz grebena.

### Najprej najboljši

Bolje kot požrešno preiskovanje informacijo hevristične ocene izkorišča metoda najprej najboljši (*best first search*), ki na vsakem koraku razvije najbolj obetavno vozlišče med vsemi že razvitimi, a še ne pregledanimi vozlišči. Vozlišča hranimo v prioritetni vrsti, prioriteto pa določa hevristična ocena. Podrobnosti so razvidne iz psevdokode na sliki 13.14.

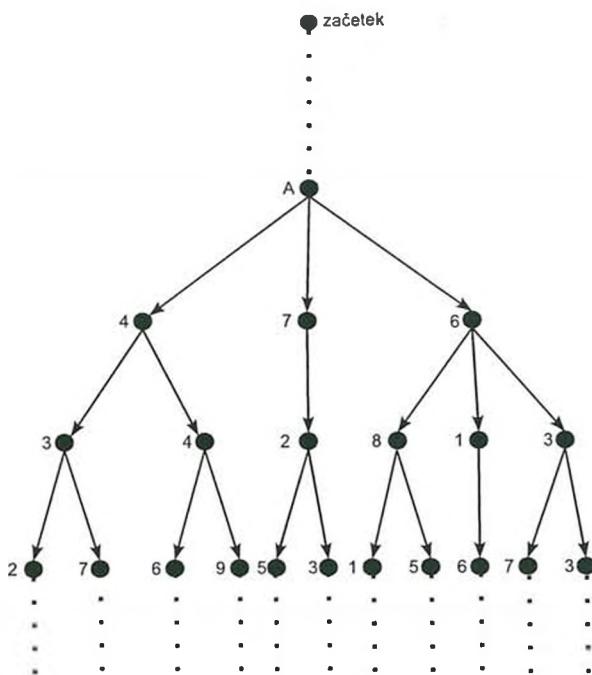
### Algoritem A

Poseben primer iskanja najprej najboljši sta algoritma A in  $A^*$ . Predpostavimo, da lahko hevristično oceno vozlišča  $n$  razbijemo na vsoto dveh delov

$$f(n) = g(n) + h(n),$$

pri tem  $g(n)$  predstavlja vsoto cen že prehodenih povezav od začetnega vozlišča do  $n$ ,  $h(n)$  pa je ocena za naprej, do najbližjega ciljnega vozlišča, kot to prikazuje slika 13.15. Preiskovalni algoritem "najprej najboljši", ki uporablja hevristiko  $f(n)$ , imenujemo algoritem A. Priporočimo, da iskanje "najprej najboljši" vozlišča preiskuje urejeno glede na  $h(n)$ .

Poglejmo si za primer igro drsečih ploščic, ki smo jo predstavili na sliki 13.2. Naloga je od danega začetnega stanja s kar najmanj potezami priti do končnega stanja podanega na sliki 13.1. Kot oceno razdalje od poljubnega stanja do ciljnega vozlišča lahko sestavimo tri hevristike:



Slika 13.13: Problem horizonta.

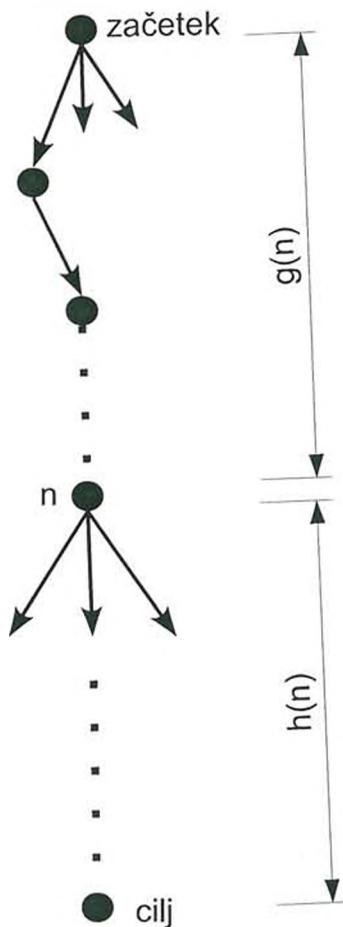
```

status bestFirst () {
    priorityQueue.makenull(); // inicializacija
    priorityqueue.enqueue(start_state); // začetno stanje
    while ( !priorityqueue.isEmpty() ) {
        state = priorityQueue.deleteMin(); // prvi po prioriteti
        if ( goal(state) )
            return SUCCESS;
        else
            priorityQueue.addSorted(successors(state)); // urejeno dodajanje
    }
    return FAILURE ;
}

```

Slika 13.14: Algoritem najprej najboljši.

1. preštejemo ploščice, ki niso na mestu; pri tem uporabljamo le del informacije, saj ne upoštevamo, razdalje ploščic do pravega mesta,
2. izračunamo vsoto razdalj (premikov) vseh ploščic do pravega mesta; ta hevristica ne



Slika 13.15: Hevristična funkcija  $f(n) = g(n) + h(n)$ .

upošteva težavnosti zamenjav, saj je za zamenjavo dveh sosednjih ploščic potrebno premakniti več kot dve potezi,

3. dvakratno število direktnih zamenjav; upoštevamo pomankljivost iz prejšnje točke.

Izračun vseh treh hevristik za začetno in končno stanje, ki sta podani levo od tabele, prikazuje slika 13.16.

Preiskovanje algoritma A s hevristiko "vsota razdalj do pravega mesta" prikazuje slika 13.17.

Primer povzemamo po [4, 3]. Vrstni red razvijanja vozlišč označujejo številke v oseenčenih krožcih. Za razumevanje primera je potrebno upoštevati, da poskušamo minimizirati oceno  $f$  in da te vrednosti hrаниmo v prioriteten vrst.

začetna pozicija		število ploščic, ki niso na mestu	5
ciljno stanje		vsota razdalj do mesta	6
		2 * število direktnih menjav	0

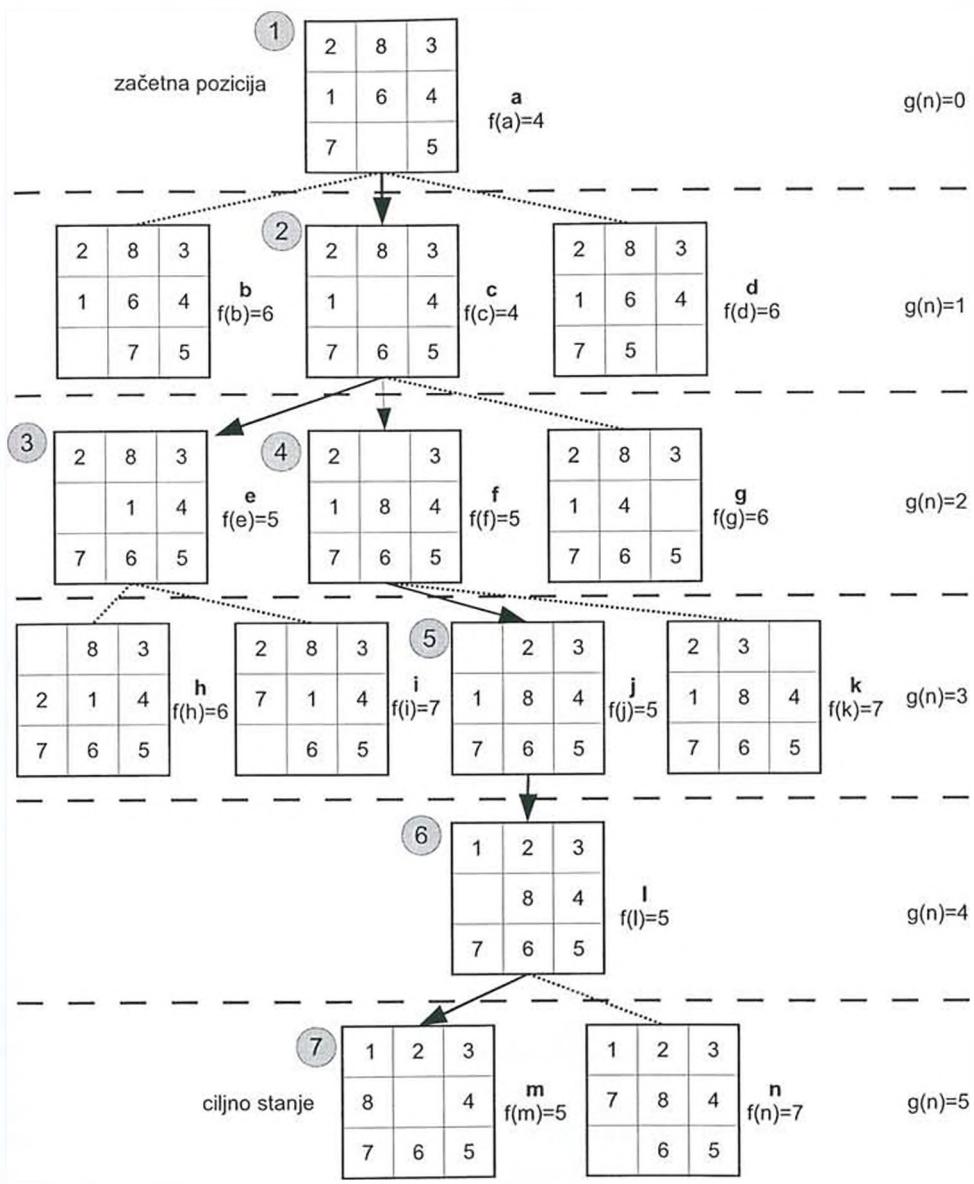
Slika 13.16: Izračun treh hevristik za igro drsečih ploščic.

### Algoritem $A^*$

Marsikdaj želimo s hevriстиčnim preiskovanjem zagotovo najti najboljšo (optimalno) rešitev, če ta obstaja. V ta namen vpeljemo pojem dopustnosti. Rečemo, da je algoritem je doposten (*admissible*), če zanesljivo najde najkrajšo pot, če ta obstaja.

Za lažjo analizo predpostavimo, da je  $f^*(n) = g^*(n) + h^*(n)$ , kjer je  $g^*(n)$  dolžina najkrajše poti do  $n$ ,  $h^*(n)$  pa dolžina najkrajše poti od  $n$  do cilja. Algoritem, ki bi uporabljal hevristiko  $f^*(n)$  bi bil zanesljivo doposten, saj bi zaradi uporabe prioritetne vrste najbližje ciljno vozlišča zanesljivo razvil pred vsemi drugimi, dražjimi ciljnimi vozlišči. Žal za večino problemov funkcije  $f^*(n)$  ne znamo določiti.

Ocena  $g(n)$  je približek za  $g^*(n)$  in velja  $g(n) \geq g^*(n)$ . Enakost velja le, če smo do vozlišča  $n$  že odkrili optimalno pot, v tem primeru bi bila dopustna tudi  $f(n) = g(n) + h(n)$ . Kot približek za  $h^*(n)$  uporabimo kar  $h(n)$ . Če je  $h(n)$  dopustna, kar pomeni, da podcenjuje

globina preiskovanja  $g(n)$ 

Slika 13.17: Delovanje algoritma A v igri drsečih ploščic s hevristiko "vsota razdalj do pravega mesta".

razdaljo do cilja  $h(n) \leq h^*(n)$ , je doposten tudi algoritom, ki uporablja  $f(n) = g(n) + h(n)$  in tak algoritmom imenujemo  $A^*$ . Razlika med algoritmoma  $A$  in  $A^*$  je zgolj v uporabi dopustne hevristike. Trivialna dopustna hevristika je  $h(n) = 0$  za vsa vozlišča, vendar ne prinaša hevristične informacije.

S protislovjem dokažimo dopustnost algoritma  $A^*$ . Algoritom  $A^*$  uporabimo na grafu z dvema ciljnima vozliščema  $G_1$  in  $G_2$ . Naj bo cena poti do  $G_1$  enaka  $f_1$ , cena poti do  $G_2$  pa  $f_2$  in naj velja  $f_2 > f_1$ . Denimo, da algoritom najde  $G_2$  pred  $G_1$  in torej ni našel optimalne rešitve. Poglejmo vozlišče  $n$  na optimalni poti od začetka do  $G_1$ . Ker je  $h$  dopustna hevristika velja  $f_1 \geq f(n)$ . Algoritom bi preiskal  $G_2$  pred  $n$  le v primeru, če je  $f(n) \geq f(G_2)$ . Neenakosti združimo in dobimo  $f_1 \geq f(n) \geq f(G_2)$ , torej  $f_1 \geq f(G_2)$ . Ker je  $G_2$  ciljno vozlišče zanj velja  $h(G_2) = 0$  in torej  $f(G_2) = g(G_2)$ . Tako dobimo  $f_1 \geq g(G_2) = f_2$ , kar je v nasprotju z začetno domnevo, da je  $G_2$  bolj oddaljen kot  $G_1$ .

Dokaz potrjuje, da lahko  $A^*$  najde le najcenejšo pot do cilja.

### Hevristike

Razvoj dobre hevristike ni trivialen. Omenimo nekaj pojmov, ki nam lahko pomagajo pri razvoju informativnih hevristik.

Ker pri dopustnosti ne zahtevamo  $g(n) = g^*(n)$ , lahko spočetka do nekaterih neciljnih vozlišč pridemo po daljši (neoptimalni) poti. Monotonost oziroma lokalna dopustnost pomeni, da do vsakega vozlišča pridemo po najkrajši poti. Za to morata biti izpolnjena pogoja:

1. za vsa stanja  $n_i$  in  $n_j$ , kjer je  $n_j$  naslednik  $n_i$ , mora veljati  $h(n_i) - h(n_j) \leq cost(n_i, n_j)$ , kjer je  $cost(n_i, n_j)$  dejanska razdalja med  $n_i$  in  $n_j$ .
2. za vsa ciljna vozlišča  $c$  velja  $h(c) = 0$

Če je hevristika monotona, pri vsakem vozlišču že pri prvem obisku vemo, da smo do njega našli najkrajšo pot. Ko zamenjamo oceno z dejansko vrednostjo, se  $f(n)$  ne zmanjša in je monotono nepadajoča. Velja, da je vsaka monotona hevristika tudi dopustna.

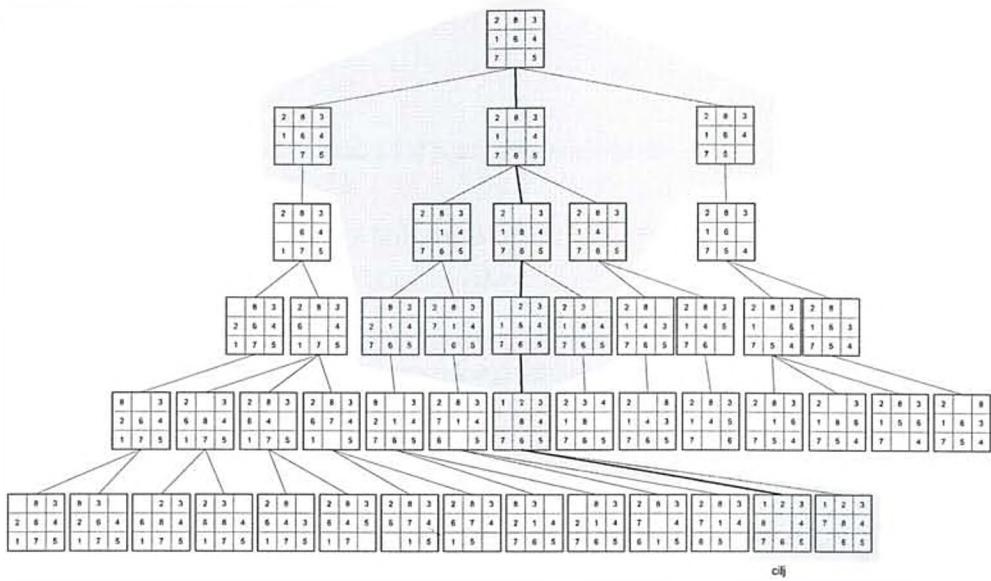
Poseben primer algoritma  $A^*$  predstavlja  $f(n) = g(n)$  oziroma  $h(n) = 0$ . V tem primeru algoritom vozlišča razvija v enakem vrstnem redu kot Dijkstrov algoritom za iskanje najkrajših poti v usmerjenem grafu, saj na vsakem koraku razvije tisto vozlišče, ki je najbliže začetnemu. Delovanje algoritma je optimalno za monotono naraščajoč  $g(n)$ , kar pomeni, da se razdalja od začetnega vozlišča za nobenega od naslednikov ne sme zmanjšati. Podobno pri Dijkstrovem algoritmu ne dopuščamo negativnih cen povezav, če želimo optimalno rešitev.

Če so dodatno vse cene povezav enake, je preiskovanje enako preiskovanju v širino. V primeru, da uporabljamo  $f(n) = h(n)$  pa gre za iskanje najprej najboljši, ki pa nima zagotovljene optimalnosti.

Če za dve dopustni hevristiki  $h_1$  in  $h_2$  velja, da za vsa stanja  $n$  drži neenakost  $h_1(n) \leq h_2(n)$ , pravimo, da je  $h_2(n)$  bolje informirana. Intuitivno si predstavljamo, de je dopustna  $h_2(n)$  bližje  $h^*(n)$ . Pri izbiri in snovanju hevristik poleg kvalitete presojamo tudi časovno zahtevnost hevristike.

Razliko v kvaliteti dveh hevristik ilustrirajmo spet na problemu drsečih ploščic. Primerjajmo hevristiko  $h_1(n) = 0$  za vsa vozlišča in hevristiko  $h_2(n)$  = "vsota razdalj do prvega mesta", kot to ilustrira slika 13.18.

Vsa prikazana vozlišča ustrezajo uporabi  $A^*$  s  $h_1(n) = 0$ , osečeni del pa razvijemo s  $h_2$ . Odebeljena črta prikazuje pot do ciljnega vozlišča.



Slika 13.18: Delovanje  $A^*$  z dvema hevristikama.

### Razveji in omeji

Prostorsko varčnost preiskovanja v globino in hevristično informacijo združuje tudi preiskovanje po načelu razveji in omeji (*branch and bound*). Ta algoritem je še posebej uporaben, kadar do končnega vozlišča vodi mnogo poti, najti pa želimo optimalno. Podrobnosti predstavlja koda na sliki 13.19.

Predpostavimo, da je ocena  $h(n)$  dopustna hevristika in torej podcenjuje dejansko razdaljo do najbližjega končnega vozlišča. Ideja algoritma je, da si zapomnimo najboljšo že najdeno pot (best) in njeno dolžino ( $d$ ). Če pri nadaljnjem preiskovanju razvijemo vozlišče  $s$ , za katerega velja  $g(s) + h(s) \geq d$ , lahko izpustimo vozlišče  $s$  in vse njegove naslednike, saj bo najdeno končno vozlišče kvečjemu enako oddaljeno kot to, ki ga že imamo.

Na začetku kličemo proceduro z zgornjo mejo enako  $\infty$ , vendar lahko namesto tega uporabimo vsako drugo vrednost, ki precenjuje optimalno rešitev. Če bi začetna vrednost omejitve le malenkost presegala optimalno vrednost najkrajše poti, bi algoritem razvil enako število vozlišč kot algoritem  $A^*$ . Ko bi algoritem enkrat našel ciljno vozlišče, bi namreč preiskoval le še vozlišča z manjšo vrednostjo  $f(n)$ , kot jo ima najdeno vozlišče - natanko ta vozlišča pa

```

// vrača zgornjo mejo, do koder je še smiselno preiskovati
// kličemo z: best = null ; branchAndBound(start_state, ∞)
int branchAndBound( state n, int d ) {
    // n – trenutno stanje
    // d – zgornja meja iskanja
    if ( g(n) + h(n) ≥ d ) // omeji iskanje
        return ∞ ;
    if ( goal( n ) ) {
        best = n ; // nova najboljša rešitev
        return g(n) ; // nova meja
    }
    suc = successors (n);
    foreach ( s ∈ suc ) // razveji
        d = min(d, branchAndBound(s, d) ) ; // išči in ažuriraj mejo
    return d ;
}

```

Slika 13.19: Preiskovanje po načelu razveji in omeji.

razvije tudi  $A^*$ .

### Izboljšave algoritma $A^*$

Težava algoritma  $A^*$  je velika poraba pomnilnika, saj je potrebno hraniti vsa vozlišča z vrednostjo  $f(n)$  manjšo od najcenejšega ciljnega vozlišča. Zato se je pojavilo nekaj algoritmov, ki porabijo manj pomnilnika, a še vedno zagotavljajo optimalnost iskanja.

### Iterativno poglabljanje z $A^*$

Prva ideja, ki jo predstavljam, je združitev iterativnega poglabljanja z  $A^*$ . Tako dobimo algoritem *IDA\** (*Iterative-Deepening A\**). Namesto poglabljanja globine iskanja, poglabljamo vrednost hevristične ocene  $f(n)$ , tako da pri vsaki novi, večji vrednosti  $f(n)$ , na katero naletimo pri razvijanju vozlišč, začnemo preiskovati od začetka v globino. Nova vozlišča tako res preiskujemo tako kot  $A^*$ . Jasno je, da takšna strategija lahko deluje dobro le pri problemih z malo različnimi vrednostmi  $f(n)$  in zelo slabo pri npr. zveznih vrednostih  $f(n)$ .

### RBFS in MRBFS

Korf [2] je predlagal algoritem RBFS (*Recursive Best First Search*), ki shrani vrednosti  $f$  vseh direktnih naslednikov vozlišč na trenutni preiskovalni poti. Iskanje lahko zato omeji do meje, ki jo predstavljajo vrednosti shranjenih sovozlišč. Pri vračanju si zapomni vrednost  $f$  najboljšega lista in s to vrednostjo ažurira vse predhodnike, preko katerih se mora vračati.

```

double rbfs(searchNode node, int bound) {
    if ( node.f > bound )
        return node.f ;
    if ( goal(node) )
        terminate_search(node) ;

    children = getChildrenSorted(node) ;
    if ( children.length == 0 )
        return Infinite ;

    while ( children[0].f <= bound ) { // preverimo najboljšega
        if (children.length == 1)
            fSiebling = Infinite ;
        else
            fSiebling = children[1].f ; // omejitev

        children[0].f = rbfs(children[0], min(bound, fSiebling)) ; // iščemo do meje
        children.sort() ; // najboljši naj bo vedno na začetku
    }
    return children[0].f
}

```

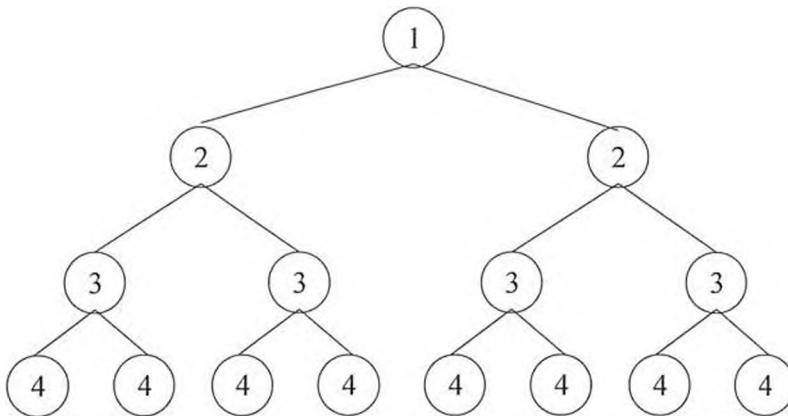
Slika 13.20: Algoritem RBFS.

Zato, ko je potrebno neko vejo ponovno razviti, ve, katere veje se izplača ponovno generirati. Podrobnosti najde bralec v psevdokodi algoritma na sliki 13.20.

Kako algoritem deluje, demonstriramo na problemskem prostoru, kjer so vrednosti  $f$  vozlišč enake njihovi globini, kot to ilustrira slika 13.21. Zaporedje razvitih vozlišč in omejitev, ki jih algoritem upošteva, prikazuje slika 13.22.

RBFS doseže prostorsko zahtevnost  $O(bd)$  in preiskuje nova vozlišča v enakem vrstnem redu kot  $A^*$ . Glede na to, da je potrebno ponovno generiranje vozlišč, je uspešnost algoritma odvisna od hevristične funkcije in potrebe po regeneriranju, ki se pojavlja v danem problemu. Preprosto povedano, algoritem je uspešen, če ne preskakuje preveč z enega na drug konec drevesa preiskovanja, saj mora pri tem vedno znova generirati velike dele vmesnega prostora.

Izboljšava tega principa je algoritem MRBFS (*Memory-aware RBFS*) [5], ki deluje podobno kot RBFS, le da hrani v pomnilniku toliko vozlišč, kot mu jih dovoljuje razpoložljiv pomnilnik. S tem odpade ponovno generiranje mnogih vozlišč. Z brisanjem vozlišč in ažuriranjem vrednosti staršev začne šele, ko pomnilnika zmanjka. Takrat začne izbirati vozlišča, ki jih lahko zavrne, in za to izbiro uporablja različne strategije. Nasprotno kot  $IDA^*$  opisani postopek deluje dobro, če je v problemu mnogo različnih vrednosti  $f(n)$ , saj se v tem primeru



Slika 13.21: Drevo, kjer so vrednosti  $f$  vozlišč enake njihovi globini.

hrani zadosti dobrih vozlišč. Če pa je različnih vrednosti  $f(n)$  malo, je postopek v zadregi, saj ne ve, katere med enakimi vrednostmi se splača zavreči.

#### *SMA\** in *MA\**

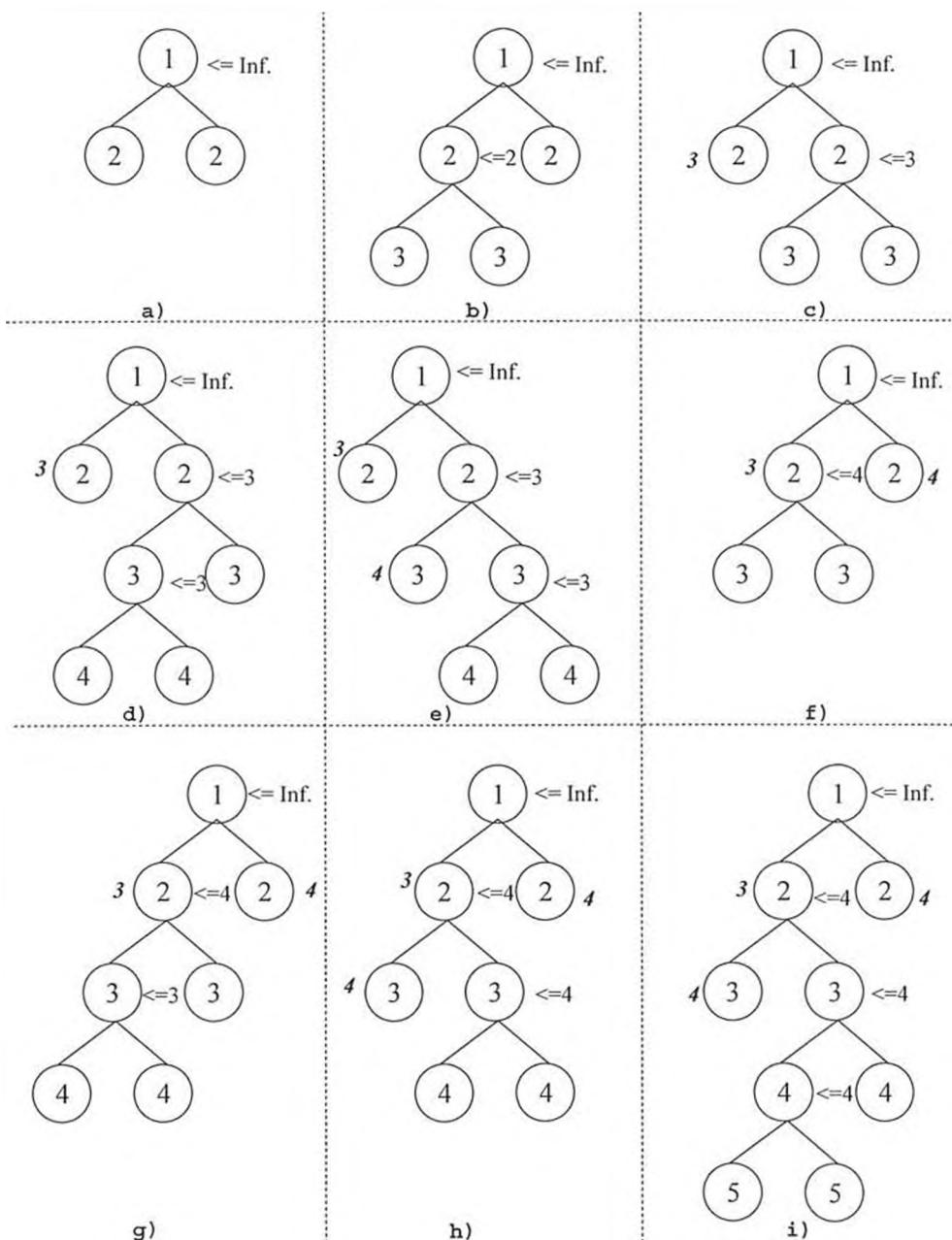
Idejo o pametnejši izrabi pomnilnika uporabimo tudi v algoritmu  $A^*$  in dobimo pomnilniško omejen algoritem *MA\** (*Memory-bounded A\**) ter njegovo poenostavljenico inačico *SMA\**. *SMA\** deluje kot  $A^*$ , dokler ne zmanjka pomnilnika, nato pa odstrani list z najslabšo vrednostjo  $f$  in podobno kot RBFS ažurira njegovo vrednost v predhodniku. Če ima več listov enako vrednost  $f$ , odstrani najprej najstarejšega, med vozlišči z enako vrednostjo pa razširi najprej najmlajšega. Tako se poskuša izogniti pretiranemu regeneriraju vozlišč. Tudi *SMA\** in *MA\** najdetra optimalno rešitev.

### 13.3 Preiskovanje po principu minimaks

Predvsem za reševanju nekaterih iger se je razvilo posebno hevristično preiskovanje. Princip minimaksa uporabimo pri reševanju iger dveh nasprotnikov, saj poskuša igralec na potezi maksimizirati svojo korist in minimizirati možnosti nasprotnika.

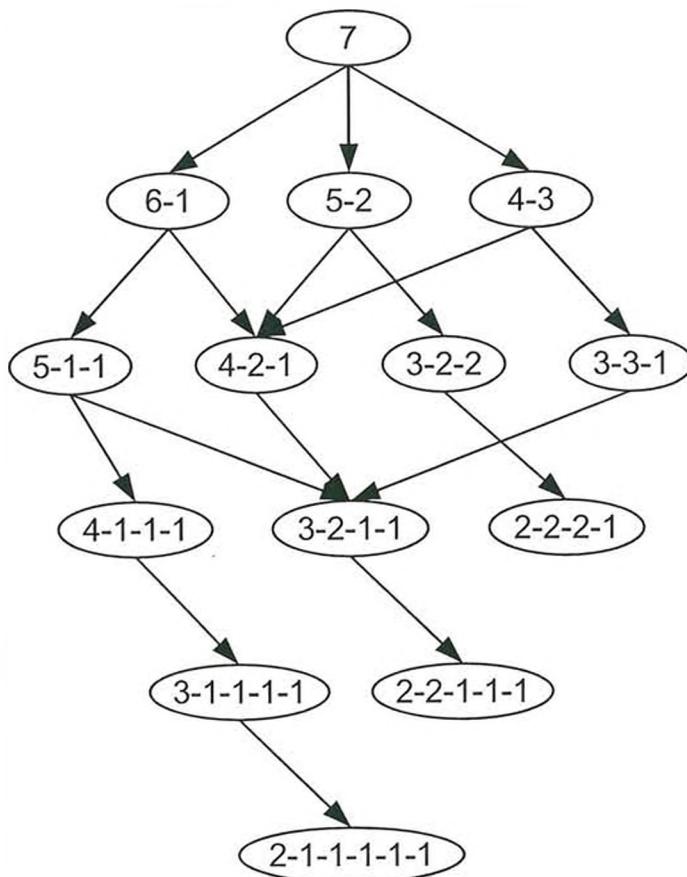
Kot preprost primer predstavljamo igro nim, kjer pred nasprotnika položimo neko število lesenih ploščic (npr. 7). Prvi igralec začetni kup ploščic razdeli na dva neenaka dela (npr. 5 in 2). Na potezi je drugi igralec, ki spet na neenaka dela razdeli enega od nastalih kupčkov (npr. 5 razdeli na 3 in 2). Spet je na potezi prvi igralec, ki ima zdaj pred sabo tri kupčke ploščic (velikosti 3, 2 in 2) in na neenaka dela razdeli enega od njih. Tako nadaljujeta in vsakokrat igralec na potezi na dva neenaka dela razdeli enega od kupčkov na mizi. Igralec, ki nobenega od kupčkov ne more razdeliti na neenaka dela, izgubi igro.

Prostor stanj pri malo začetnih ploščic je dovolj majhen, da ga lahko uporabimo za predstavitev preiskovalnih principov, saj lahko po potrebi pregledamo vse stanja. Prostor stanj za



Slika 13.22: Zaporedje razvitetih vozlišč in omejitev, ki jih upošteva RBFS.

nim s sedmimi začetnimi ploščicami prikazujemo na sliki 13.23.

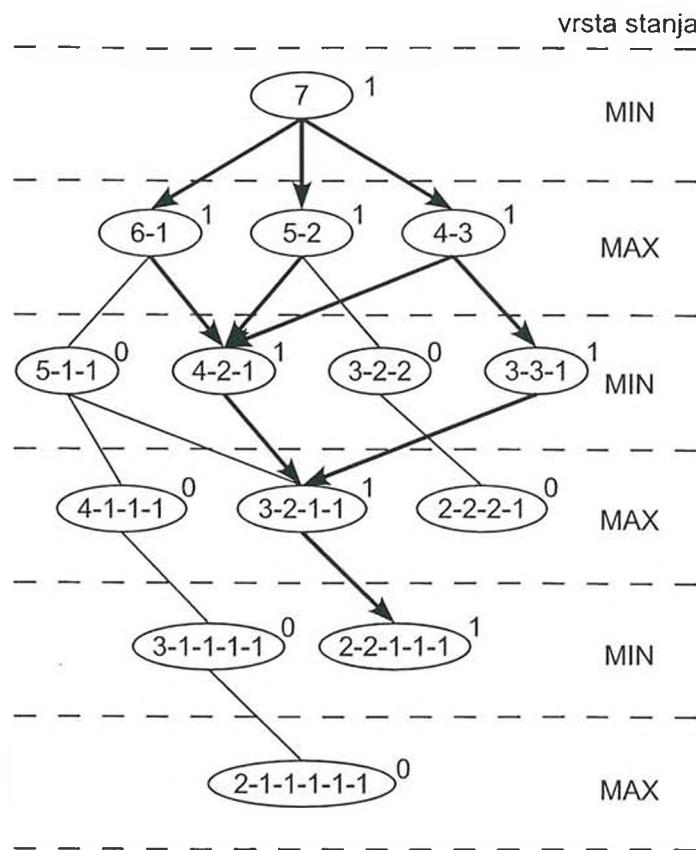


Slika 13.23: Prostor stanj za igro nim s sedmimi začetnimi ploščicami.

Preiskovanje po principu minimaksa deluje po predpostavki, da imata oba nasprotnika, MIN in MAX, na voljo enake informacije in da oba poskušata zmagati. V nadaljevanju poskušamo najti optimalne poteze za igralca MAX. Igralec MAX poskuša maksimizirati svoj rezultat proti igralcu MIN, ki poskuša minimizirati MAX-ov rezultat.

Vozlišča označimo z oznakami MIN in MAX po nivojih glede na na to, kdo je na potezi. Za igro nim 7, kjer začne MIN, je to prikazano na desni strani slike 13.24. Vsem vozliščem priredimo vrednosti. Začnemo pri listih. Ti dobijo vrednost 1, kadar zmaga MAX in vrednost 0, kadar zmaga MIN. Algoritem minimax te vrednosti propagira navzgor:

- če je predhodnik tipa MAX, dobi maksimum svojih naslednikov, kar ustreza temu, da izbere najboljšo možno poteko zase (tisto, ki ga bo privrdla do zmage);
- če je predhodnik vrste MIN, dobi minimum svojih naslednikov, kar ustreza temu, da

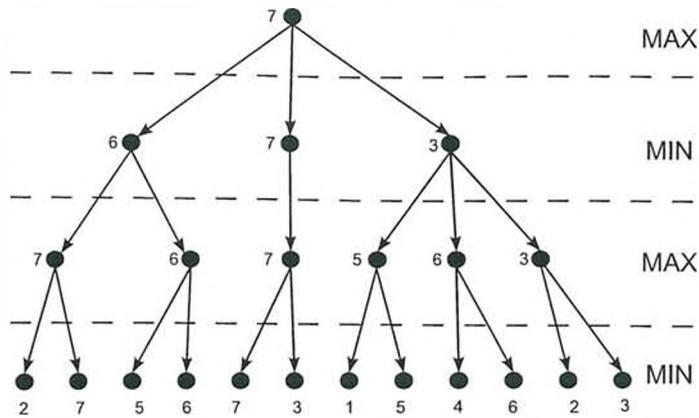


Slika 13.24: Vrednosti stanj, ki jih v igri nim 7 dodeli algoritmu minimax.

izbere najslabšo možno potezo za nasprotnika, se pravi najboljšo zase (tisto, ko MAX v nadaljevanju izgubi in je označena z 0).

Povratno ažurirane vrednosti povedo, kaj največ lahko igralec pričakuje v trenutnem stanju, in poteze izbiramo glede na te vrednosti. Ažurirane vrednosti za ves prostor stanj igre nim 7 najdemo na sliki 13.24, kjer so vrednosti zapisane desno poleg vozlišč. Z odbeljenimi puščicami so prikazane možnosti, ki na tej podlagi vodijo do zmage igralca MAX. Vidimo, da lahko igralec MAX zmaga, ne glede na začetno potezo igralca MIN.

V zanimivih ighah prostora stanj ne moremo pregledati do listov, zato stanja pregledamo naprej do neke (fiksne) globine glede na čas ter računske in pomnilniške zmogljivosti, ki jih imamo na voljo. Gre torej spet za pogled naprej globine  $n$ . Na globini  $n$  vozliščem določimo hevristično oceno kvalitete in jo po načelu minimaksa propagiramo navzgor. Primer ažuriranja za hipotetičen prostor globine 4 prikazuje slika 13.25. Listi vsebujejo hevristične ocene stanj, vrednosti v notranjih vozliščih pa so ažurirane.



Slika 13.25: Primer ažuriranja vrednosti stanj z minimaksom globine 4.

Na primer za šah lahko kot hevristično oceno kvalitete stanja uporabimo ovrednotenje stanja na šahovnici po  $n$  potezah, npr. število in moč figur, njihovo postavitev, itd.

Tako izračunamo oceno najboljšega stanja do globine  $n$ , kaj bo naprej pa ne vemo (spet horizont). Psevdokoda algoritma minimax na sliki 13.26 predstavlja podrobnosti implementacije.

```
double minimax(current_node) {
    if ( is_leaf(current_node) || depth(current_node) == MaxDepth )
        return heuristic_evaluation (current_node);
    if ( is_min_node (current_node) )
        return min(minimax(children_of(current_node)));
    if ( is_max_node (current_node) )
        return max(minimax(children_of(current_node)));
}
```

Slika 13.26: Psevdokoda principa minimaks.

Slabost minimaksa je kratkovidnost, saj ne vidi dlje kot fiksno število korakov naprej. Navidez dobro stanje na vidni globini lahko zapelje igralca, da izbere, globalno gledano, slabo potezo. Ena od rešitev je, da preiščemo še nekaj nivojev naprej le od najboljših stanj. Pri minimaksu govorimo tudi o anomalijah, ko princip deluje slabše od pričakovanj. Pri nekaterih igrah so kršene osnovne predpostavke, na primer pri šahu se zgodi, da nasprotnik ne želi zmagati, ampak doseči remi. Nekatere pozicije so težje za oceno, ali pa so tako zapletene, da v njih (človeški) nasprotnik ne vidi dobrih nadaljevanj. Načelo minimaksa tega ne zmore upoštevati in zaradi napačnih predpostavk sistem ne odigra potez, ki bi bile lahko uspešne.

Osnovni algoritem minimaks je računsko zahteven in po nepotrebnem preišče ves prostor

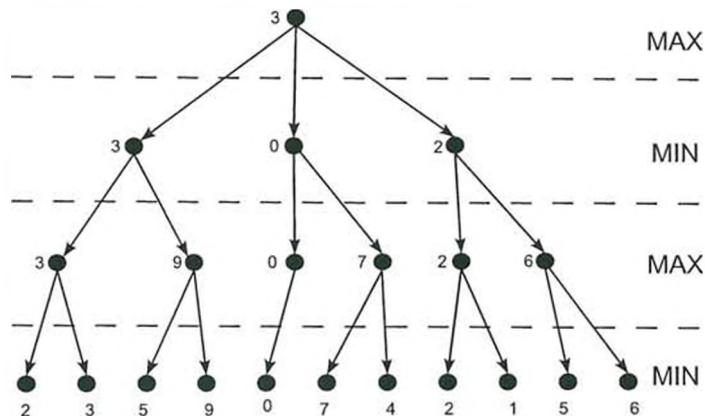
do globine  $n$ , čeprav za nekatere veje že vnaprej ve, da niso perspektivne. To slabost odpravlja rezanje alfa-beta.

### Rezanje alfa-beta

Minimaks preišče vsa vozlišča do globine  $n$  in ocene propagira navzgor. Ker so številna poddrevesa lahko neobetavna, jih ne bi bilo treba preiskati. Ideja za izboljšavo, ki jo imenujemo rezanje  $\alpha\beta$  ( $\alpha\beta$ -pruning), je naslednja:

- preiskujmo v globino,
- za vozlišča MAX naj  $\alpha$  predstavlja najboljšo doslej najdeno vrednost,
- za vozlišča MIN naj  $\beta$  predstavlja najslabšo vrednost doslej,
- zagotovo lahko v vozliščih MAX zavrhemo vrednosti manjše od  $\alpha$  in v vozliščih MIN vrednosti večje od  $\beta$ .

Delovanje algoritma na hipotetičnem prostoru s slike 13.27 prikazuje slika 13.28.

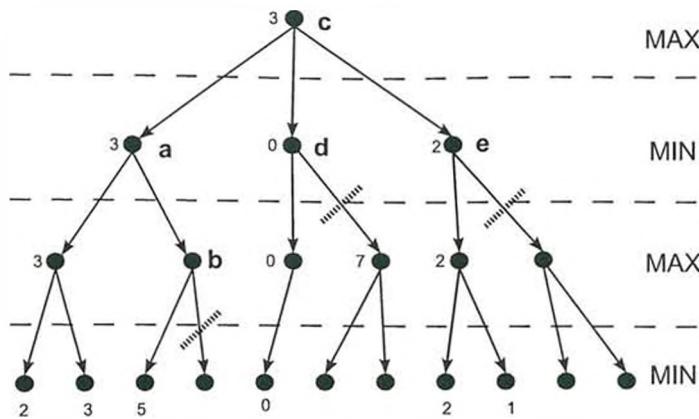


Slika 13.27: Prostor stanj in z minimaksom ažurirane vrednosti.

S poševno črto so označene točke, kjer nam zaradi rezanja ni potrebno pregledovati naprej. Vozlišče A ima vrednost  $\beta = 3$ , in ker je vozlišče MIN, to pomeni, da več kot 3 ne bo. Zato lahko  $\beta$ -odrežemo desnega sina vozlišča B, katerega levi sin ima vrednost 5 in  $5 > 3$ . V točki C je vrednost  $\alpha = 3$  in ker je to vozlišče MAX, c ne bo manjši od 3. Ko v točki D, ki je vozlišče MIN, od levega sina dobimo vrednost 0, lahko desnega sina  $\alpha$ -odrežemo, saj je  $0 < 4$ . Podobno lahko  $\alpha$ -odrežemo tudi desnega sina vozlišča E, saj ima desni sin vrednost 2 in je  $2 < 3$ . Opazimo, da rezanje ne spremeni rezultata povratnega ažuriranja vozlišč in v korenu dobimo enako vrednost 3.

Pravila  $\alpha\beta$ -rezanja zapišimo natančneje:

1. odrežemo naslednike vozlišča MIN z vrednostjo  $\beta$  manjšo ali enako vrednosti  $\alpha$  njegovega predhodnika MAX,

Slika 13.28: Prostor stanj porezan z  $\alpha\beta$ -rezanjem.

2. odrežemo naslednike vozlišča MAX z vrednostjo  $\alpha$  večjo ali enako vrednosti  $\beta$  njegova predhodnika MIN.

Za rezanje na nivoju  $m$  tako primerjamo vrednosti z nivoja  $(m - 1)$  in  $(m + 1)$ . V izogib doseganju vrednosti  $\alpha$  in  $\beta$  nazaj pri predhodnikih, učinkovita implementacija te vrednosti pošlje kot parametre. Za vozlišče MAX tako shranimo minimum vrednosti  $\beta$  njegovih naslednikov MIN kot vrednost  $\beta$ . Za vozlišče MIN shranimo maksimum vrednosti  $\alpha$  njegovih naslednikov MAX kot  $\alpha$ . Vsako notranje vozlišče bo tako hranilo vrednosti  $\alpha$  in  $\beta$ . Koren drevesa inicializiramo z  $\alpha = -\infty$ ,  $\beta = +\infty$ . Podrobnosti so razvidne iz psevdokode na sliki 13.29.

```
// klic: alpha_beta (start_node, -infinity, infinity);
double alphaBeta (node, alpha, beta) {
    if ( leaf(node) )
        return heuristic_evaluation(node);
    if ( max_node(node) ) {
        alpha = max(alpha, alphaBeta(children, alpha, beta));
        if ( alpha >= beta )
            cut_off_search_below(node);
    }
    if ( min_node(node) ) {
        beta = min(beta, alphaBeta(children, alpha, beta));
        if ( beta <= alpha )
            cut_off_search_below (node);
    }
}
```

Slika 13.29: Uporabna implementacija  $\alpha\beta$ -rezanja.

### Uporaba preiskovanja v igrah

Na kratko omenimo še nekaj programov, ki so preiskovanje uporabili za igranje znanih iger. Več podrobnosti lahko zainteresirani bralec izve na (angleški) Wikipediji ali na za te igre specializiranih spletnih mestih.

#### Dama

Damo so raziskovalci že zelo zgodaj uporabili kot testni poligon metod umetne inteligence. Povprečni vejitveni faktor pri tej igri je  $b = 8$ , celoten prostor stanj obsega okoli  $10^{20}$  stanj, pri preiskovanju z minimaksom pa bi drevo imelo okoli  $10^{31}$  listov. To mero imenujemo kompleksnost igrальнega drevesa (*game-tree complexity*).

Samuel je že leta 1959 izdelal samoučeč se, uporaben program, ki je igral na nivoju srednje dobrih igralcev. Uporabljal je princip minimaksa,  $\alpha\beta$ -rezanje in oceno pozicij.

Najslavnejši in najboljši umetni igralec dame je gotovo Schaefferjev sistem Chinook, ki ga je predstavil leta 1990 in kasneje večkrat izboljšal. Uporablja princip minimaksa,  $\alpha\beta$ -rezanje in ima zelo dobro oceno pozicij. Izboljšane verzije uporabljajo bazo končnic do 8 figur, bazo otvoritev do globine 20 ter rezanje vnaprej glede na izgubo. Sistem je večkrat premagal svetovnega prvaka. Leta 2007 je ekipa razvijalcev Chinooka objavila, da je rešila damo, in da se igra dobrih igralcev vedno konča neodločeno.

Zanimiv je tudi sistem Blondie24, ki ga je Fogel predstavil leta 2000. Igra na ravni dobrih igralcev dame, strategije pa se uči z nevronskimi mrežami in evolucijskimi pristopi.

#### Šah

Šah je v računalniških krogih mnogo bolj priljubljena igra kot dama, saj sposobnost dobrega igranja šaha že od nekdaj velja za znak inteligence. Prostor stanj ima povprečni vejitveni faktor 38 in med  $10^{43}$  ter  $10^{47}$  legalnih pozicij. Kompleksnost igrальнega drevesa je ocenjena na  $10^{123}$  stanj.

Sistemi za avtomatsko igranje šaha uporabljajo širok repertoar preiskovalnih metod: minimaks,  $\alpha\beta$ -rezanje, oceno pozicij, baze otvoritev in končnic, itd. Trenutno najboljši programi preiskujejo do globine približno 12 potez vnaprej. Uveljavilo se je mnenje, da obstaja linearna povezava med globino preiskovanja in kvaliteto igre.

Leta 1997 je program DeepBlue na specializiranem paralelnem računalniku premagal svetovnega prvaka. Trenutno najboljši računalniški šahisti na osebnih računalnikih, kot sta Rybka in DeepFritz, igrajo na nivoju velemojsrov.

#### Nekaj drugih iger

Pozornost raziskovalcev so pritegnile tudi druge igre, ki se igrajo na igralnih ploščah.

Starodavna igra Go se igra na plošči dimenzij  $19 \times 19$  in ima vejitveni faktor okoli 360. Trenutno programi igrajo na nivoju povprečnih igralcev. Poenostavljena varianta Go-Moku se igra na plošči  $15 \times 15$  in je dejansko varianta igre "pet v vrsto". Igra je s stališča odločljivosti dokončno rešena.

Pri igri Othello (nekateri jo imenujejo tudi *reversi*) preiskujejo programi do globine 50 in premagajo svetovnega prvaka.

Kot odziv na uspešnost šahovskih programov v boju s svetovnim prvakom je nastala igra Arimaa, ki se igra na šahovski plošči z enakimi, vendar drugače poimenovanimi figurami in drugačnimi pravili. Igra je v izviv avtomatskemu preiskovanju z grobo procesorsko močjo uvedla več preprek in ima povprečni vejitveni faktor ocenjen na 17281.

Precej pozornosti so deležne tudi igre z elementi naključja, kot so backgammon, bridge, tarok, poker, itd. Pri njih se uporablja prilagojena verjetnostna inačica principa minimaks, ki jo imenujejo expectiminimax [6].

## Literatura

- [1] Ivan Bratko. *Prolog programming for artificial intelligence, 3rd edition.* Addison-Wesley, 2000.
- [2] R.E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.
- [3] George F Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6th edition.* Addison-Wesley Pearson Education, Boston, 2009.
- [4] Nils J. Nilsson. *Principles of artificial intelligence.* Morgan Kaufman, 1980.
- [5] Marko Robnik-Šikonja. Effective use of memory in linear space best first search. Tehnično poročilo, University of Ljubljana, Faculty of Computer and Information Science, 1996. URL <http://lkm.fri.uni-lj.si/rmarko/papers>.
- [6] Stuart J. Russell, Peter Norvig. *Artificial intelligence: a modern approach, 3rd edition.* Prentice Hall, 2009.

## Poglavlje 14

# Inteligentni agenti in roboti

*In ko bi imel dar preroštva in ko bi poznal vse skrivnosti in imel vse spoznanje in ko bi imel vso vero, da bi gore premikal, ljubezni pa bi ne imel, nisem nič.*

Pavel iz Tarza

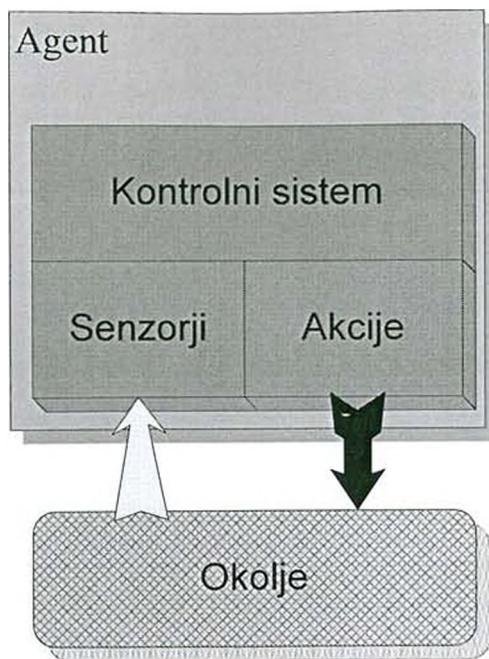
V okviru umetne inteligence obstaja mnogo različnih definicij agenta. Zelo splošno lahko agenta definiramo kot entiteto, ki lahko izvrši različne akcije za rešitev določenega problema. Nekateri avtorji smatrajo agente za enega temeljnih konceptov umetne inteligence [3, 4] in na njem utemeljijo vse ostale koncepte, od preiskovanja, učenja in sklepanja do evolucijskega računanja in robotike.

Agente delimo na biološke, kot so ljudje in živali, ter na umetne, kot so roboti in programi. Slednje, ki za uporabnika izvršijo določeno nalogu, imenujemo tudi programski agenti.

Arhitekturo agenta, ki ustreza naši splošni definiciji, prikazuje slika 14.1. Kot vidimo, mora agent delovati v okolju, zato potrebujemo senzorje, ki iz okolja sprejemajo različne informacije, in aktuatorje, ki omogočajo agentu, da v okolju izvaja različne akcije. Notranja stanja sistema hrani nadzorni sistem, ki procesira informacije pridobljene od senzorjev ter inteligentno usmerja akcije, ki naj se izvedejo.

Naštejmo še nekaj lastnosti, ki so značilne za večino inteligentnih agentov.

- Delovanje v interaktivnem okolju: agenti sprejemajo informacije iz okolja ter na podlagi teh in notranjega stanja izvajajo akcije, ki spreminjajo okolje. Okolje je lahko zelo različno, od interneta, kjer delujejo agenti, ki iščejo informacije, navideznih svetov računalniških iger, do prilagojenega nogometnega igrišča za robotski nogomet.
- Avtonomnost in fleksibilnost: agenti delujejo samostojno, brez eksplicitnih ukazov in imajo kontrolo nad svojimi akcijami. Vzdržujejo svoje notranje stanje ter se marsik-daj učijo iz izkušenj. Odzivanje glede na dano situacijo in samostojna aktivnost sta posledica sledenja nekemu vnaprejšnjemu cilju in/ali načrtovanju, ki ga izvaja agent sam.



Slika 14.1: Arhitektura agenta.

- Skupinsko delovanje: agenti delujejo v interakciji z drugimi entitetami, ki so lahko prav tako agenti ali ljudje. Pri doseganju nekaterih ciljev je potrebno sodelovanje z drugimi entitetami, zato mora tovrsten agent razviti ustrezne strategije in se pogajati. V okolju, kjer deluje več agentov, je smiselna specializacija agentov za določene naloge in delegiranje nalog specializiranim agentom. Posledica avtonomnosti agentov je tudi distribuirano in asinhrono reševanje problemov, ko več agentov istočasno rešuje nalogo, a vsak svoj del.

Omenimo še podobnost med objekti in agenti. Številne lastnosti agentov so lastne tudi programskim objektom, vendar so objekti omejeni na ožji okvir programskega jezika in računalnika, v katerem se izvajajo. Podobnost pa vendarle izkoriščajo številna agentska okolja, ki so povečini napisana v objektnih programskih jezikih (denimo v javi). Tako s pridom izkoriščajo podobnosti za lažjo realizacijo in enostavnje programiranje agentov.

Agenti se pojavljajo v številnih aplikacijah. Navedimo nekaj primerov:

- proizvodnja: agenti so del številnih proizvodnih procesov. Na primer, lahko modelirajo procese, optimirajo vire, skrbijo za izredne dogodke, ...
- avtomatska kontrola procesov: kot primer navedimo transportne sisteme in zračni promet, kjer agenti nadzorujejo in optimirajo poti,

- telekomunikacije: za to področje so značilni porazdeljeni sistemi sodelujočih komponent, kar je pisano na kožo agentnim tehnologijam, ki lahko skrbijo za nadzor, upravljanje, sodelovanje, ...
- upravljanje z informacijami: agenti skrbijo za zbiranje informacij, njihovo filtriranje, preverjanje, posodabljanje, ...
- e-poslovanje: številni agenti delujejo na borzah, kjer skrbijo za hitre odzive na nepredvidene situacije. Avtomatizacija ter informatizacija elektronskega poslovanja je omogočila agentom, da enakopravno sodelujejo z ljudmi in pri nekaterih vrstah trgovanj je danes večina udeležencev agentov. Agenti so se uveljavili tudi za iskanje nekaterih produktov, npr. najcenejših knjig, hotelskih in letalskih rezervacij, itd.
- interaktivne igre: virtualno okolje teh iger je idealen poligon za preizkušanje agentnih tehnologij in agenti v njih nastopajo enakopravno z ljudmi in sledijo lastnim ciljem ali za ljudi opravljajo določene naloge.

V nadaljevanju poglavja najprej agente razdelimo na nekaj kategorij in za vsako opišemo nekaj poglavitnih značilnosti. Opišemo nekaj najpogosteje uporabljenih agentnih arhitektur. Ker so roboti v bistvu le posebna kategorija agentov, na kratko opišemo njihove značilnosti z vidika umetne inteligence. V zadnjem delu poglavja po zgledu [5] podamo uvod v distribuirano računanje z agenti.

## 14.1 Vrste agentov

Navedimo najpogostejše kategorije agentov, ki se pojavljajo v literaturi, ter njihove najpomembnejše značilnosti. Delitev ni enolična in nekatere agente bi lahko razvrstili tudi v več skupin.

**Odzivni agenti** (*reactive agents*) se na okolje odzivajo glede na vnaprej definirana pravila.

Teh pravil se lahko tudi naučijo in jih popravijo, ko opazijo, da se je okolje spremenilo in da njihovo delovanje ni več ustrezno. Primeri takšnih agentov so razvrščevalnik elektronske pošte, ki sporočila za uporabnika razvršča v mape, filter neželene elektronske pošte (*spam*), ki odloča o koristnosti prispetih sporočil, in vpisovalnik v koledar, ki zna z drugimi agenti uskladiti urnik in rezervirati ustrezni termin.

**Ciljno usmerjeni agenti** (*goal oriented agents*) so nadgradnja odzivnih agentov. Na okolje se ne le odzivajo, pač pa zasledujejo določen cilj. Za to vrsto agentov je značilna uporaba iskanja ali načrtovanje poti do cilja. Primer je iskanje specifičnih strani na internetu, najcenejše letalske karte itd.

**Sodelujoči agenti** (*collaborative agents*): tudi zelo šibki agenti lahko v skupini dosežejo dober rezultat. Primeri tovrstnih agentov so algoritmi za optimizacijo po principu kolonije mravelj in genetski algoritmi. Prednost tega pristopa je redundanca, saj dobimo več rešitev, pa tudi večja robustnost, ker uspeh ni odvisen le od posameznega agenta. Tovrstne pristope tudi zlahka paraleliziramo.

**Uporabnostni agenti** (*utility based agents*) poleg doseganja ciljev maksimirajo še druge kriterije, vsebovane v funkciji uporabnosti. Denimo, poleg relevantnosti strani, agent upošteva še čas iskanja in kvaliteto strani.

Funkcija uporabnosti, ki bi jo lahko imenovali tudi funkcija koristnosti ali zadovoljstva, lahko vsebuje najrazličnejše objektivne in subjektivne kriterije uporabnika. V ekonomski teoriji se ta funkcija uporablja za pojasnitve racionalnega obnašanja. Pri agentih bi to lahko pomenilo, da agent izgubi igro, da doseže druge cilje, recimo pripravi nasprotnika, da poveča stavo.

Kaj pomeni racionalnost odločanja, lahko pogledamo na znanem primeru iz [1]. Anketirancem so ponudili, da izberejo med naslednjima možnostma:

- A: s 100% verjetnosti dobiš 3000\$ in
- B: z 80% verjetnosti dobiš 4000\$.

Večina je izbrala možnost A, čeprav bi s strogo racionalno izbiro izbrali B, kjer je pričakovani dobitek večji:  $0.8 \cdot 4000\$ + 0.2 \cdot 0\$ = 3200\$$ . Naslednjo izbiro so anketiranci imeli med

- C: z 20% verjetnosti dobiš 4000\$ in
- D: s 25% verjetnosti dobiš 3000\$

Večina je tokrat izbrala možnost C, kar je racionalna izbira. To si potrdimo z izračunom pričakovanih vrednosti  $E(C) = 0.2 \cdot 4000\$ = 800\text{\$}$  in  $E(D) = 0.25 \cdot 3000\$ = 750\text{\$}$ . Poskus potrjuje, da uporabljamo ljudje zahtevnejše funkcije uporabnosti, kot je preprosta pričakovanata vrednost (v danem primeru verjetno upoštevamo tudi tveganje, svoje denarno stanje, ...).

**Vmesniški agenti** (*interface agents*) so neke vrste osebni pomočniki, ki za uporabnika opravljajo različne naloge. Sodelujejo z uporabnikom in so sposobni učenja. Učijo se od uporabnika, lahko pa tudi od drugih agentov. Primer tovrstnih agentov so pomočniki pri učenju programa, inteligentni osebni iskalci novic, kjer uporabnik označi, katere spletnne novice oz. strani so mu všeč, itd.

**Mobilni agenti** (*mobile agents*) se lahko premikajo iz kraja v kraj. Gre lahko za fizično premikanje (roboti) ali virtualno premikanje (po internetu ali med računalniki v lokalni mreži). Primer tovrstnih agentov so računalniški virusi, nadzorni programi in nekateri programi za porazdeljeno računanje, ki se lahko avtonomno in asinhrono premikajo med računalniki in izvajajo različne računske naloge. Za mobilnost je potrebna ustrezna infrastruktura, ki jo omogoča ustrezna programska knjižnica, ali okolje postavljeno na vseh sodelujočih računalnikih.

**Informacijski agenti** (*information-gathering agents*) za uporabnika zbirajo, filtrirajo in klasificirajo različne informacije, največkrat na internetu ali intranetu. Njihovo uspešnost merimo z merami, kot sta priklic in natančnost (glej izraza (11.2) in (11.1) v poglavju 11). Večinoma so sposobni učenja. Težava pri tovrstnih agentih je šumnost, ažurnost in verodostojnost podatkov, ki jih zberejo, kar je posledica virov, ki jih uporabljam.

**Učeči se agenti** (*learning agents*) se z učenjem prilagajajo okolju. Ker je učenje eden od temeljev inteligence, je učna komponenta v taki ali drugačni obliki vsebovana v mnogih, če ne vseh, drugih vrstah agentov.

**Večagentni sistem** (*multiagent system*) je način, kako združimo moč več posameznih avtonomnih agentov, kar je za marsikatero nalogu bolj primerno kot posamezni agenti in omogoča porazdeljeno reševanje problemov. Rezultat je program za več agentov, ki rešujejo probleme v interaktivnem okolju in so zmožni avtonomnih, prilagodljivih ter skupinsko organiziranih akcij, ki so usmerjene k doseganju nekega cilja.

Učenje večagentnih sistemov je bodisi centralizirano, kjer se centralni sistem uči od vseh agentov, bodisi decentralizirano, kjer se agenti učijo vsak zase in od drugih agentov. Slednje omogoča večjo robustnost in prilagodljivost sistema.

**Hibridni agenti** vsebujejo lastnosti več predhodno opisanih skupin.

## 14.2 Agentne arhitekture

V praksi se je uveljavilo več načinov, kako je agent zgrajen in kako so posamezni deli povezani med seboj in z okoljem. To s skupnim imenom imenujemo arhitektura agenta. Omenimo nekaj primerov.

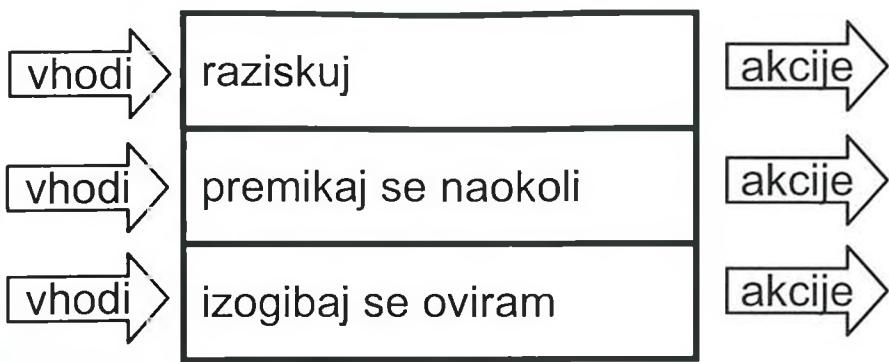
**Arhitektura vsebovanosti** (*subsumption architecture*) je večnivojska arhitektura, namenjena primitivnim robotom.

Arhitektura vsebovanosti ne predvideva centralne inteligence, pač pa poleg vhoda in izhoda vsebuje več plasti, kjer vsaka zasleduje svoje cilje oz. nadzira določeno agentovo obnašanje. Višji nivoji imajo dostop do akcij, ki jih generirajo nižji nivoji in jih lahko po potrebi blokirajo. Vsak nivo vsebuje svoja pravila (npr. if-then), ki se izvajajo asinhrono. Nivoji so med seboj neodvisni, kar omogoča enostavno dodajanje novih nivojev.

Slabost arhitekture vsebovanosti je težavno odkrivanje napak. V zapletenih situacijah, je težko odkriti, kaj je šlo narobe, kateri nivo je za to odgovoren in kaj bi bilo treba storiti, da do napake ne bi več prihajalo.

Na sliki 14.2 je primer preproste arhitekture robota, ki je namenjen raziskovanju (najvišji nivo). Nižja nivoja skrbita, da se robot premika naokoli in da se izogiba oviram. S tem, ko ima višji nivo raziskovanja dostop do nižjenivojskih akcij, lahko poskrbi, da robot resnično koristno raziskuje in se ne zgolj izogiba preprekam.

**Arhitektura BDI: verjetje, želja, namen** (*BDI - Belief, Desire, Intention*) je sestavljena iz treh komponent. Verjetje je izjava o okolju, za katero agent verjame, da je resnična. Množica takih izjav opisuje agentov pogled na okolje. Želja je ciljno stanje, ki ga želi agent doseči. Namen je načrt, ki ga ima agent za dosego cilja. Agent preveri različne možnosti, izbrana možnost postane njegova želja in obveže se, da bo dosegel cilj. Namenu oz. načrtu sledi, dokler ga ne doseže ali ta postane nemogoč. Cilj se lahko med preiskovanjem tudi spremeni. Kolikokrat se med preiskovanjem spremeni načrt,



Slika 14.2: Arhitektura vsebovanosti za preprostega raziskovalnega robota. Višji nivoji lahko blokirajo akcije nižjih nivojev.

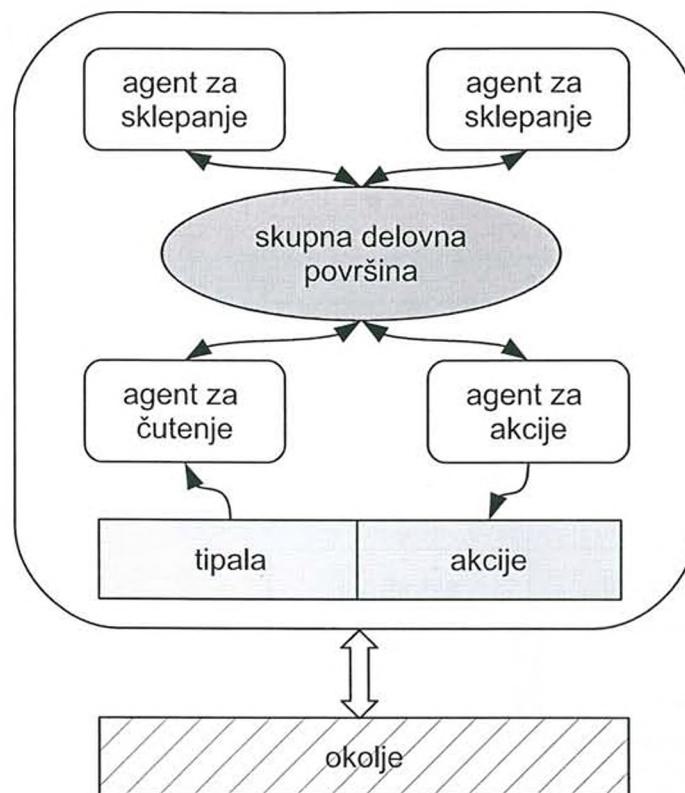
je odvisno od okolja. V hitro spremenjajočem se okolju je potreba po spremembah ciljev in načrtov večja.

**Arhitektura skupne delovne površine** (*blackboard architecture*) vsebuje skupno delovno površino, ki si jo delijo vsi agenti in vsebuje informacije o okolju in vmesne rezultate procesiranja. Predstavljena je na sliki 14.3. Agenti opravljajo naloge, ki jih najdejo na površini, in nanjo zapisujejo svoje rezultate. Vsi agenti imajo lahko enake sposobnosti ali pa so specializirani.

Potrebna je koordinacija dostopa do podatkov za kar se uporabljajo mehanizmi za kontrolo dostopa do skupnih virov. Arhitektura skupne delovne površine je enostavna za implementacijo v večnitem okolju, kjer vsak agent predstavlja svojo nit, niti pa si delijo nekatere vire.

**Mobilne arhitekture** omogočajo agentom, da se premikajo po virtualnem okolju. Še posej so primerni za porazdeljeno računanje, saj s premikanjem iščejo proste vire za izračun naloge, ki so jo prevzeli.

Primer tovrstne arhitekture Aglets, ki jo je 1990 razvil IBM, prikazuje slika 14.4. Arhitektura je realizirana v programskega jezika java in s pridom izkorišča javansko tehnologijo serializacije. Ta omogoča, da objekte zapišemo kot zaporedje bitov, ki jih nato shranimo v pomnilniku, v datoteki ali pa jih prenesemo preko mreže. Bistveno je, da lahko kasneje in na drugem mestu ponovno ustvarimo identičen objekt iz shranjene slike. Zaradi varnosti Aglets deluje v javanskem peskovniku, ki je od siceršnjega sistema ločeno izvajalno okolje z omejenimi funkcijami, kar omogoča zaščito pred zlorabo.



Slika 14.3: Arhitektura skupne delovne površine.

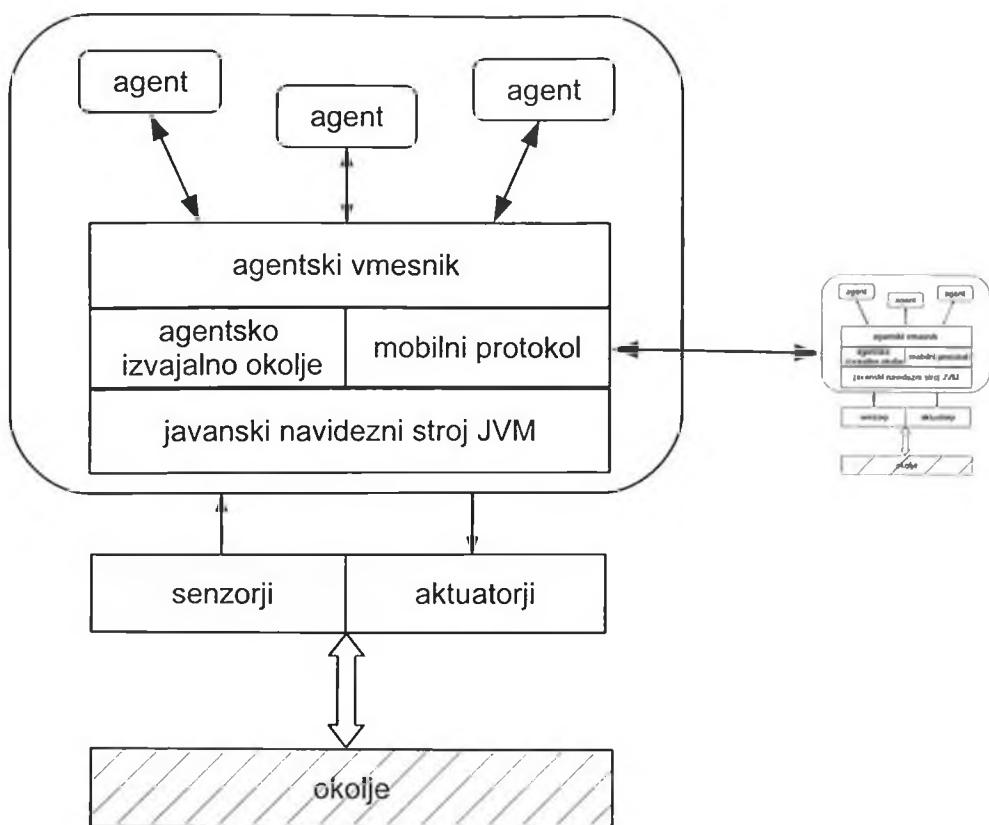
### 14.3 Robotika

Roboti so agenti posebne vrste, ki delujejo v realnem okolju, ki je bolj zahtevno in nepredvidljivo od umetnega. Zaradi številnih izzivov je robotika že od samih začetkov testno okolje za algoritme umetne inteligence. Slika 14.5 prikazuje shemo robota z vidika umetne inteligence.

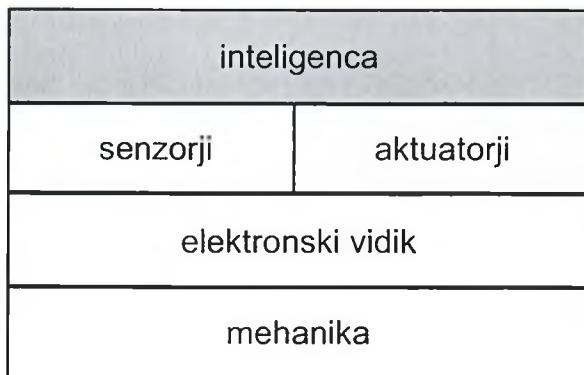
Ker je inteligenco le zgornji majhen del robota in ker so številne tehnologije, potrebne za inteligentno obnašanje, predvsem učenje in planiranje, opisane v drugih delih knjige, v tem razdelku le na kratko predstavljamo nekaj dodatnih vidikov.

Obstaja mnogo različnih vrst in delitev robotov. Naštejmo jih nekaj.

- Fiksni roboti se ne premikajo po prostoru. Takšni so npr. industrijski roboti. Kljub temu nekateri potrebujejo zahtevne algoritme za usmerjanje in izogibanje oviram, recimo robotska roka z več prostorskimi stopnjami.
- Roboti z nogami uporabljajo za premikanje različno število nog. Obstajajo roboti z eno, dvema, štirimi, šestimi in osmimi nogami.



Slika 14.4: Mobilna agentska arhitektura Aglets.



Slika 14.5: Shema robota.

- Roboti na kolesih se premikajo z enim, dvema ali tremi pari koles.
- Podvodni roboti pri premikanju posnemajo različne morske organizme, denimo ribe, rake ali črve.
- Zračni roboti so npr. avtomatska letala in sateliti.
- Polimorfni roboti se lahko preoblikujejo glede na različne naloge, ki jih opravlja. Lahko hodijo, se plazijo, se valijo ali plezajo.
- Porazdeljeni roboti ali robotske jate so skupine robotov, ki določen problem rešujejo skupaj.
- Programski roboti ali softboti v nasprotju s fizičnimi roboti delujejo v nekem programskem okolju, npr. na internetu ali v navideznem svetu npr. v interaktivni računalniški igri.

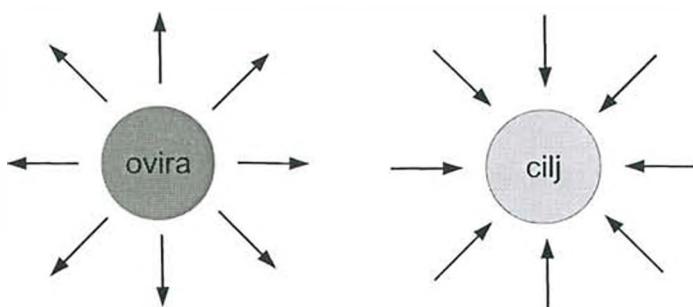
Roboti za svoje delovanje potrebujejo stik z okoljem. Dražljaje iz okolja sprejemajo preko tipal za:

- vid oziroma elektromagnetne valove,
- sluh oziroma zračne valove,
- okus in vonj, kjer dražljaje sprejemajo kemični receptorji,
- tip, kjer senzor zaznava pritisk,
- eholokacijo, kjer senzor podobno kot pri netopirjih zaznava odmev ultrazvoka,
- elektrocepcijo, oziroma zaznavo s senzorji za električno polje,
- magnetocepcijo, kot imenujemo zaznavo sprememb v magnetnem polju,
- ekilibriocepcijo, ki pomeni zaznavanje ravnotežja oziroma pospeška,
- termocepcijo, ki pomeni zaznavo temperature,
- ...

Na okolje delujejo roboti preko aktuatorjev. Za premikanje so to kolesa, noge, plavuti, krila, itd., za delovanje v okolju pa različna prijemala in robotske roke.

Nadzorni sistem robota je organiziran podobno kot pri agentih. Uporabljajo se podobne arhitekture, npr. arhitektura vsebovanosti, pa tudi številni algoritmi strojnega učenja. Pogoste so nevronske mreže, evolucijski pristopi in spodbujevano učenje.

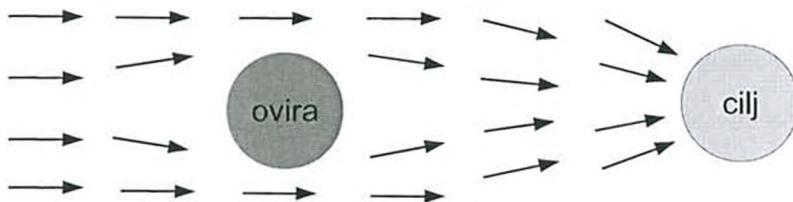
Eden od pomembnih elementov robotike, ki je bistven za inteligenčno obnašanje, je načrtovanje. V dinamičnem okolju že načrtovanje gibanja ni preprosto. Potrebno je neprestano načrtovanje in prilagajanje načrtov spremenjenim okoliščinam (*anytime planning*). Načrt se postopno izboljšuje s časom, a je vedno pripravljen.



Slika 14.6: Odbijajoče in privlačno polje potenciala.

V zapletenem okolju bi bilo obvladovanje celotnega okolja prezapleteno in računsko prezahtevno. Uporablja se dekompozicija okolja na celice, kjer se prostor razbije na povezane celice. Robot izdela načrt premikanja za vsako celico posebej, nato pa poišče prehode med celicami. S tem zmanjša problemski prostor in zahtevnost procesiranja.

Uporabna tehnika je tudi potencialno polje. Okoli ovir se tvori odbijajoče polje, ki robota odvrača, da bi se jim približal, okoli cilja pa privlačno polje, ki robota vleče k sebi, kot to prikazuje slika 14.6. Robot se med premikanjem po terenu, ki ga prikazuje slika 14.7, zaradi negativnega potenciala ovire odmakne od nje in se usmeri proti cilju, ki ga privlači.



Slika 14.7: Premikanje skozi potencialno polje.

Načrtovanje premikanja olajšajo značilne točke, ki si jih robot zapomni in jih uporabi za orientacijo v prostoru.

Da bi olajšali razvoj programske opreme za robotske sisteme, je bilo razvitih več specializiranih programskih jezikov, npr. LAMA, RAPT in Karel. Druga tehnika, ki olajša razvoj, testiranje in validacijo programske opreme za robote, so simulatorji. Z njimi lahko preverimo delovanje algoritmov, preden jih preizkusimo na fizičnih robotih. Zanimivi sta okolji Open Dynamics System, ki omogoča simulacijo gibanja trdnih teles, in Simbad, ki omogoča takojšnjo vizualizacijo sprogramiranega obnašanja v tridimenzionalnem okolju.

Da naša obravnavo agentov in robotov ne bo ostala le na faktografskem naštevanju dejstev, si poglejmo z njimi povezano praktično zanimivo tematiko iz porazdeljenega računanja.

## 14.4 Porazdeljeno računanje z agenti

Problem porazdeljenega računanja z agenti v tem razdelku razumemo kot reševanje problema, ki ga definira center, agenti pa sodelujejo pri njegovem reševanju. Agenti so samostojni, a nimajo lastnih preferenc. Komunicirajo v svoji sosedstvini in poskušajo najti rešitev, ki upošteva globalne omejitve. Naša naloga je razviti algoritem, ki naj ga izvajajo agenti, da bo center dobil globalno informacijo. Gre torej za problem koordinacije agentov.

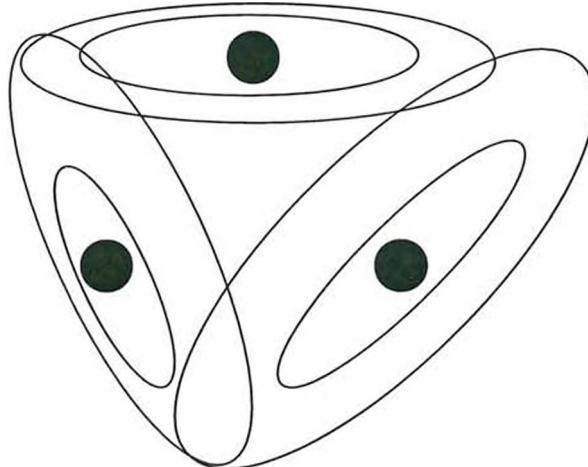
Primer distribuiranega računanja je mreža senzorjev. Denimo, da imamo v nekem prostoru ali zgradbi nameščeno večje število senzorjev (npr. detektorjev dima) z omejenimi zmogočnostmi procesiranja in majhno porabo energije. Senzorji imajo majhen domet in lahko komunicirajo le v sosedstvini, vseeno pa želimo z njimi dobiti globalno sliko.

Poglejmo si, kako bi v takšnem kontekstu rešili problem upoštevanja omejitev. V nadaljevanju se pri predstavitevki zgledujemo po [5].

### Porazdeljeno upoštevanje omejitev

Pri večagentnih sistemih je pogosto potrebno rešiti problem upoštevanja omejitev (*constraint satisfaction problem*). Problem in nekaj načinov njegovega reševanja predstavimo na primerih, ne da bi strogo formalno definirali vse komponente.

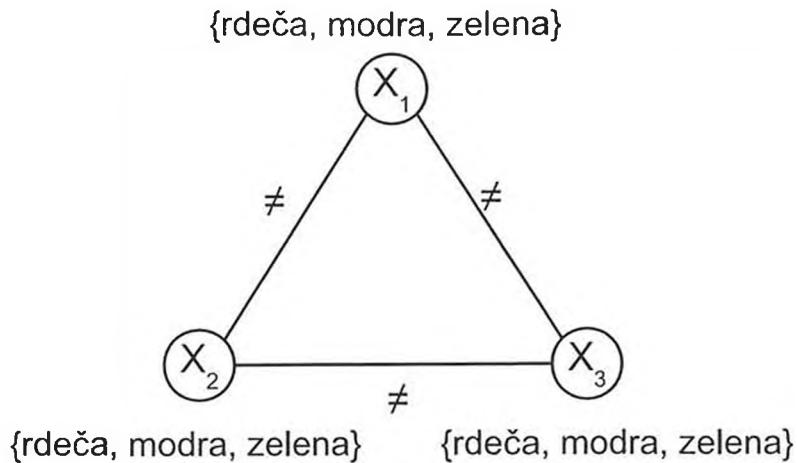
Problem upoštevanja omejitev je definiran z množico spremenljivk, njihovimi domenami in omejitvami glede vrednosti, ki jih lahko spremenljivke zavzamejo. Naša naloga je spremenljivkam prirediti vrednosti tako, da so upoštevane vse omejitve, ali pa ugotoviti, da to ni mogoče. Tovrstne probleme najdemo v računalniškem vidu, pri razumevanju naravnega jezika, planiranju, dokazovanju teoremov, razporejanje opravil itd.



Slika 14.8: Problem prekrivanja komunikacijskih frekvenc treh senzorjev.

Preprost primer s senzorji je predstavljen na sliki 14.8. Senzorji lahko komunicirajo le v svoji bližini. Elipse okoli njih prikazujejo doseg njihovega signala, ki je določen z močjo

oddajanja in dano konfiguracijo prostora, v katerem se nahajajo. Opazimo, da se frekvence prekrivajo. Naloga senzorjev je, da medsebojno uskladijo in si priedijo frekvenco iz nabora treh dovoljenih, tako da se frekvence ne prekrivajo in bodo lahko med seboj komunicirali.



Slika 14.9: Problem prekrivanja komunikacijskih frekvenc treh senzorjev, spremenjen v problem barvanja grafa.

Omejitve frekvenc predstavimo kot problem barvanja grafov, kot to ilustrira slika 14.9. Imamo množico spremenljivk  $X = \{X_1, X_2, X_3\}$ . Zaloga vrednosti (domena)  $D_i$  za vsako spremenljivko je množica barv {rdeča, modra, zelena}. Množico omejitev  $C$  sestavljajo  $\{X_1 \neq X_2, X_1 \neq X_3, X_2 \neq X_3\}$ . V tem kontekstu je rešitev naše naloge legalna prieditev spremenljivk, kar pomeni takšno prieditev vrednosti vsaki od spremenljivk iz njene zaloge vrednosti, ki ne krši nobene od omejitev iz dane množice omejitev. V porazdeljenem upoštevanju omejitev je vsak agent svoja spremenljivka, rešitev pa naj določijo z asinhrono komunikacijo brez centralnega nadzora.

### Algoritmi z rezanjem domen

Prva vrsta algoritmov poskušaj omejitvam zadostiti tako, da odstranijo prepovedane vrednosti iz domen. Če v končni rešitvi ostane ena sama vrednost, agent razglasí, da je našel rešitev.

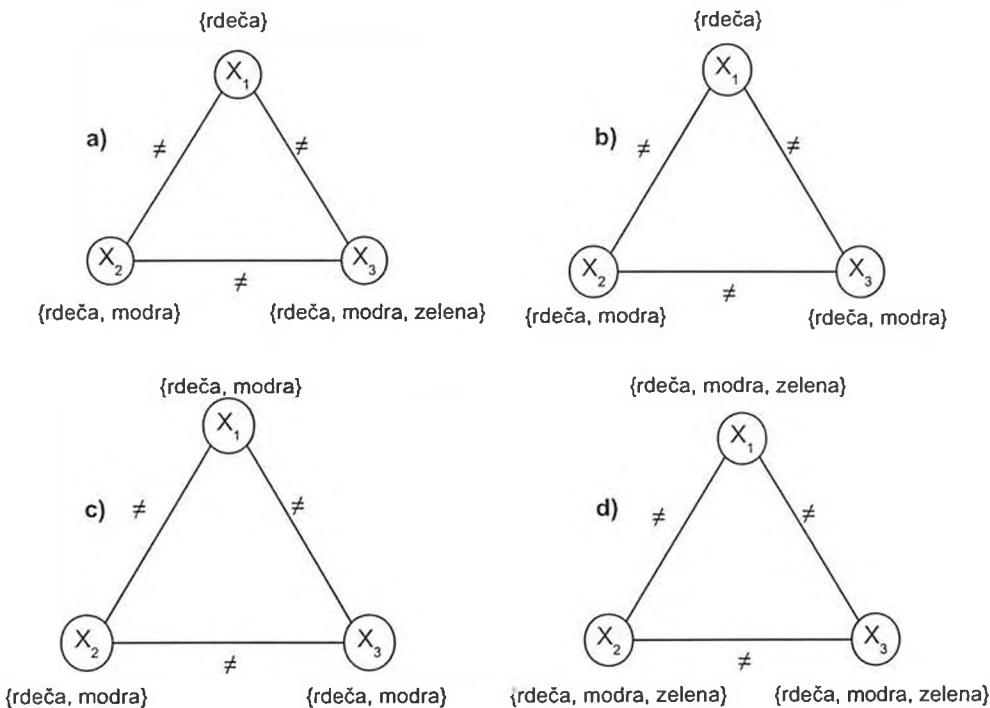
Da bi odstranila prepovedane vrednosti iz svojih domen, vozlišča komunicirajo s sodi. Postopek konsistentnosti povezav (*arc consistency*) prikazuje slika 14.10.

Vsako vozlišče  $X_i$  z domeno  $D_i$  izvaja algoritem za vse svoje sosedje  $X_j$ . Postopek se konča, ko je ena od domen prazna, kar pomeni, da ni rešitve, ali ko ni več izločanja vrednosti. Če ostane v domeni vsaj ena vrednost, imamo rešitev, sicer pa ne vemo, ali rešitev obstaja. Algoritem se konča in je pravilen, ni pa popoln, kar pomeni, da ni zanesljivo, da bomo našli rešitev.

Delovanje algoritma si poglejmo na nekaj primerih. Na začetni konfiguraciji, ki jo prikazuje slika 14.11a, deluje algoritem takole:

```
// postopek izvaja vsako vozlišče  $X_i$  z domeno  $D_i$  vse svoje sosedne  $X_j$ 
void arcConsistency(  $X_i$ ,  $X_j$  ) {
    foreach (  $v_i \in D_i$  )
        if ( ne obstaja  $v_j \in D_j$  konsistentno z  $v_i$  )
             $D_i = D_i - \{v_i\}$  ;
}
```

Slika 14.10: Psevdokoda algoritma za rezanje domen s konsistentnostjo povezav.



Slika 14.11: Nekaj začetnih konfiguracij za porazdeljeno barvanje grafa.

- vozlišča si pošiljajo sporočila,
- učinkovita so samo sporočila vozlišča  $X_1$ , zato
- $X_2$  in  $X_3$  odstranita rdečo iz svoje domene, ostane  $D_1 = \{rdeča\}$ ,  $D_2 = \{modra\}$  in  $D_3 = \{modra, zelena\}$ ,
- ko  $X_3$  dobi sporočilo od  $X_2$ , odstrani modro iz svoje domene in dobimo pravilen rezultat  $D_1 = \{rdeča\}$ ,  $D_2 = \{modra\}$  in  $D_3 = \{zelena\}$ .

Tudi na konfiguraciji s slike 14.11b deluje algoritem pravilno:

- kot v primeru a)  $X_2$  in  $X_3$  odstranita rdečo iz svoje domene, ostane  $D_1 = \{\text{rdeča}\}$ ,  $D_2 = \{\text{modra}\}$  in  $D_3 = \{\text{modra}\}$ ,
- oba,  $X_2$  in  $X_3$ , odstranita modro in ostaneta jima prazni domeni  $D_1 = \{\text{rdeča}\}$ ,  $D_2 = \{\}$  in  $D_3 = \{\}$ , zato pravilno razglasita, da ni rešitve.

Za algoritmom rezanja domen s konsistentnostjo povezav sta težavnejša primera s slike 14.11c in 14.11d. Obakrat nobeno od vozlišč ne more odstraniti nobene vrednosti iz svoje domene in algoritom konča brez zaključka, kar pomeni da v primeru c) ne ugotovi, da ni rešitve, v primeru d) pa ne najde rešitve.

Ugotovimo, da je rezanje domen prešibko, da bi našlo rešitev v splošnem primeru. V praksi se uporablja kot predprocesiranje bolj zahtevnemu algoritmu. Pot k močnejšemu algoritmu opazimo z ugotovitvijo, da je rezanje domen ekvivalentno logični resoluciji v propozicijski logiki. Enotska resolucija je naslednje pravilo sklepanja

$$\frac{\begin{array}{c} A_1 \\ \neg(A_1 \wedge A_2 \wedge \cdots \wedge A_n) \end{array}}{\neg(A_2 \wedge \cdots \wedge A_n)}$$

Velja  $\neg(A_1 \wedge A_2 \wedge \cdots \wedge A_n)$ . Ker istočasno velja  $A_1$ , lahko sklepamo, da  $A_1$  ne spada v negirani del, in dobimo  $\neg(A_2 \wedge \cdots \wedge A_n)$ .

Da vidimo ekvivalenco z rezanjem domen, zapišimo omejitve v obliki logičnih pravil, ki jih imenujemo Nogood in ponazarjajo neveljavne kombinacije vrednosti. Omejitev, da  $X_1$  in  $X_2$  ne moreta hkrati zavzeti vrednosti rdeča v logiki zapišemo  $\neg(X_1 = \text{rdeča} \wedge X_2 = \text{rdeča})$ , v obliki omejitev pa Nogood  $\{X_1 = \text{rdeča}, X_2 = \text{rdeča}\}$ . V primeru s slike 14.11b je agent  $X_2$  osvežil svojo domeno na podlagi sporočila  $X_1 = \text{rdeča}$  in Nogood  $\{X_1 = \text{rdeča}, X_2 = \text{rdeča}\}$ , kar je ekvivalentno logičnemu sklepu

$$\frac{\begin{array}{c} X_1 = \text{rdeča} \\ \neg(X_1 = \text{rdeča} \wedge X_2 = \text{rdeča}) \end{array}}{\neg(X_2 = \text{rdeča})}$$

Enotska resolucija je dokaj šibko pravilo sklepanja, za katerega pa obstaja močnejša poslošitev, ki jo imenujemo hiperresolucija:

$$\frac{\begin{array}{c} A_1 \vee A_2 \vee \cdots \vee A_m \\ \neg(A_1 \wedge A_{1,1} \wedge A_{1,2} \wedge \dots) \\ \neg(A_2 \wedge A_{2,1} \wedge A_{2,2} \wedge \dots) \\ \vdots \\ \neg(A_m \wedge A_{m,1} \wedge A_{m,2} \wedge \dots) \end{array}}{\neg(A_{1,1} \wedge A_{1,2} \wedge \cdots \wedge A_{2,1} \wedge A_{2,2} \wedge \cdots \wedge A_{m,1} \wedge A_{m,2} \wedge \dots)}$$

Hiperresolucija je pravilna in popolna za propozicijsko logiko, zato lahko na njeni osnovi sestavimo popoln porazdeljen program za upoštevanje omejitev. Psevdokodo algoritma prikazuje slika 14.12.

```

// postopek rezanja domen s hiperresolucijo izvaja vsako vozlišče  $X_i$  z domeno  $D_i$ 
void HR(  $NG_i$ ,  $NG_j^*$  ) {
    //  $NG_i$  je množica omejitvev, ki se jih agent i zaveda
    //  $NG_j^*$  je množica novih omejitvev, ki jih agent i dobi od agenta j
    do {
         $NG_i = NG_i \cup NG_j^*$ 
         $NG_i^* = \text{hiperresolucija}(NG_i, D_i)$  ;
        if (  $NG_i^* \neq \{\}$  ) {
             $NG_i = NG_i \cup NG_i^*$  ;
            pošlji  $NG_i^*$  vsem sosedom vozlišča i ;
            if (  $\{\} \in NG_i^*$  )
                stop ;
        }
    } while ( se je spremenila  $NG_i$  );
}

```

Slika 14.12: Algoritma za rezanje domen na podlagi hiperresolucije.

Vsek agent generira nove omejitve za svoje sosede, jih o tem obvešča in reže svojo domeno glede na omejitve, ki jih dobi od sosedov.  $NG_i$  je množica omejitvev (Nogoods), ki se jih agent  $i$  zaveda,  $NG_j^*$  pa množica novih omejitvev, ki jih agent  $i$  dobi od agenta  $j$ . Algoritem se ustavi po končnem številu poslnih sporočil. Če rešitev obstaja, jo zagotovo najde.

Poglejmo si delovanje algoritma s hiperresolucijo na primeru s slike 14.11c, za katerega algoritrom rezanja s konsistentnostjo povezav ni deloval.

- $X_1$  ima v  $NG_1$  naslednje omejitve  $\{X_1 = \text{rdeča}, X_2 = \text{rdeča}\}$ ,  $\{X_1 = \text{rdeča}, X_3 = \text{rdeča}\}$ ,  $\{X_1 = \text{modra}, X_2 = \text{modra}\}$ ,  $\{X_1 = \text{modra}, X_3 = \text{modra}\}$
- velja tudi omejitev glede vrednosti  $X_1 = \text{rdeča} \vee X_1 = \text{modra}$ ,
- s hiperresolucijo lahko  $X_1$  sklepa

$$\begin{array}{c}
 X_1 = \text{rdeča} \vee X_1 = \text{modra} \\
 \neg(X_1 = \text{rdeča} \wedge X_2 = \text{rdeča}) \\
 \neg(X_1 = \text{modra} \wedge X_3 = \text{modra}) \\
 \hline
 \neg(X_2 = \text{rdeča} \wedge X_3 = \text{modra})
 \end{array}$$

in zato doda omejitev  $\{X_2 = \text{rdeča}, X_3 = \text{modra}\}$  k svojemu seznamu  $NG_1$ .

- podobno k  $NG_1$  doda tudi  $\{X_2 = \text{modra}, X_3 = \text{rdeča}\}$ ,
- $X_1$  pošlje obe generirani omejitvi svojima sosedoma  $X_2$  in  $X_3$ ,

- $X_2$  lahko na podlagi svoje domene, svojih Nogood in prejetih omejitv sklepa

$$\begin{array}{c} X_2 = \text{rdeča} \vee X_2 = \text{modra} \\ \neg(X_2 = \text{rdeča} \wedge X_3 = \text{modra}) \\ \neg(X_2 = \text{modra} \wedge X_3 = \text{modra}) \\ \hline \neg(X_3 = \text{modra}) \end{array}$$

- na podlagi druge prejete omejitev  $X_2$  izpelje tudi  $\neg(X_3 = \text{rdeča})$ ,
- $X_2$  pošlje obe generirani omejitvi sosedu  $X_3$ ,
- $X_3$  generira  $\{\}$  in algoritem se ustavi z ugotovitvijo, da rešitve ni.

Hiperresolucija je močnejša kot enotska resolucija, vendar lahko število generiranih omejitev Nogoods zelo naraste. Če bi vsa vozlišča procesirala omejitev vzporedno, bi dobili še precej več sporočil in omejitev, kot smo to prikazali v primeru.

Razlog za nepraktičnost algoritma s hiperresolucijo je njegovo previdno delovanje, saj sklepa vedno samo na to, kar je dokazljivo in nikoli ne zavzame nobene nedokazane vrednosti. Alternativa takšnemu postopanju je hevristično preiskovanje, ko spremenljivke zavzamejo še nedokazane vrednosti in se v primeru protislovja sproži vračanje.

### Hevristično iskanje rešitev z omejitvami

Enostavna rešitev za upoštevanje omejitev je centralno vodeno dogovarjanje in urejanje spremenljivk. Spremenljivke uredimo, npr.  $X_1, X_2, \dots, X_n$ , in poklicemo rekurzivno funkcijo izčrpno( $X_1, \{\}$ ) podano na sliki 14.13.

```
// izčrpno preiskovanje glede na dani vrstni red spremenljivk
// na začetku kličemo z izčrpno(X1, {,})
void izčrpno(Xi, {v1, v2, ..., vi-1}) {
    // postopek določi vrednost spremenljivki Xi,
    // {v1, v2, ..., vi-1} so vrednosti že določene predhodnim spremenljivkam X1, X2, ..., Xi-1
    vi = vrednost v ∈ Di konsistentna z {v1, v2, ..., vi-1} ;
    if ( vi ne obstaja )
        vračanje ;
    else if ( i == n )
        stop
    else
        izčrpno(Xi+1, {v1, v2, ..., vi} );
}
```

Slika 14.13: Upoštevanje omejitev z izčrpnim preiskovanjem kronološko urejenih spremenljivk.

Gre za kronološko vračanje, saj se vračamo po prvotnem vrstnem redu nazaj za en nivo in izberemo naslednjo še ne uporabljenou konsistentno vrednost. Algoritem izvaja izčrpno preiskovanje, agenti pa se kličejo sekvenčno in tako ne izkoriščajo prednosti, ki bi jih dobili s porazdeljenim računanjem.

Povsem drugačne slabosti ima naivna vzporedna asinhrona rešitev, predstavljena s psevdokodo na sliki 14.14. Algoritem izvajajo vsi agenti hkrati.

```
// naivno vzporedno asinhrono upoštevanje omejitev
void naiveParallel() {
    agent X; izbere vrednost iz svoje domene  $v_i \in D_i$  ;
    do {
        if (  $v_i$  je konsistentna z vrednostmi sosedov ali nobena vrednost ni konsistentna )
            ne stori ničesar ;
        else {
            izberi vrednost  $v_i$  iz svoje domene, ki je konsistentna s sosedji ;
            obvesti sosedje o izbrani vrednosti ;
        }
    } while ( so še spremembe vrednosti ) ;
}
```

Slika 14.14: Naivni vzporedni asinhroni postopek za upoštevanje omejitev spremenljivk.

Če se algoritem ustavi, ker ni našel nobenih kršitev, je rešitev pravilna, v ostalih primerih pa je nepopolna, saj se lahko postopek nikoli ne konča, ali sploh ne najde prave rešitve. Na primeru iz slike 14.11d bi lahko agenti ciklično menjavali izbrane barve in algoritem se ne bi nikoli ustavil.

Algoritem, ki si od izčrpnega algoritma izposodi idejo globalno urejenih spremenljivk, od naivnega vzporednega pa asinhrono delovanje in izmenjavo sporočil, imenujemo asinhrono vračanje (*ABT - asynchronous backtracking*) in je prikazan na sliki 14.15.

Agenti imajo vnaprej določeno prioriteto (ideja urejenosti) in sporočila potujejo od agentov z višjo k agentom z nižjo prioriteto. Vsi agenti delujejo vzporedno in lahko vrednosti postavijo hkrati. Vsi tudi čakajo na sporočila in nanje odgovarjajo. Ko agent izbere vrednost ali spremeni svoje stanje, pošlje svojo novo vrednost vsem povezanim agentom. Ko agent prejme sporočilo o vrednosti agenta z višjo prioriteto, poskuša svojo vrednost prilagoditi tako, da ne krši prejetih omejitev.

Komunikacija med agenti poteka s sporočili tipa  $ok?(A_j, v_j)$  in  $NG(Nogood)$ . Ko agent  $A_i$  prejme sporočilo "ok?" agenta  $A_j$ , shrani prejete vrednosti v podatkovno strukturo  $agentView$ . Preveri, če njegova trenutna vrednost  $v_i$  še ustreza trenutnemu stanju v  $agentView$ . Če ustreza, ne stori ničesar, sicer v svoji domeni poišče novo konsistentno vrednost. Če jo najde, jo priredi svoji spremenljivki in pošlje sporočilo  $ok?(A_i, v_i)$  vsem povezanim agentom z nižjo prioriteto, sicer začne vračanje.

```

// upoštevanje omejitev s postopkom asinhronega vračanja
// kodo izvaja vsak agent  $A_i$  vzporedno in asinhrono
void ABT() {
    when sprejeto( Ok?( $A_j, v_j$ ) ) {
        dodaj ( $A_j, v_j$ ) v agentView ;
        check(agentView) ;
    }
    when sprejeto( NG(nogood) ) {
        dodaj nogood v lastni seznam Nogood ;
        foreach ( ( $A_k, v_k$ ) ∈ nogood ) {
            if (  $A_k$  ni sosed vozlišča  $A_i$  ) {
                dodaj ( $A_k, v_k$ ) v agentView ;
                zahtevaj od  $A_k$  da doda  $A_i$  kot sosed
                // soseda moramo dodati, da bomo obveščeni o njegovih spremembah
            }
        }
        check(agentView) ;
    }
}
// procedura za preverjanje konsistentnosti prireditev za agenta  $A_i$ 
void check(agentView) {
// agentView vsebuje vrednosti, ki jih pozna agent
    if ( ! konsistentna(agentView ,  $v_i$ ) ) {
        if ( obstaja v  $D_i$  z agentView konsistentna vrednost ) {
            izberi  $d \in D_i$  konsistentno z agentView ;
             $v_i = d$  ;
            pošlji ok?( $A_i, d$ ) sosedom z nižjo prioriteto
        }
        else
            vračanje ;
    }
}
// vračanje
void backtrack() {
    nogood = nekonsistentna množica, ki smo jo dobili npr. s hiperresolucijo ;
    if ( {} ∈ nogood)
        sporoči vsem agentom, da ni rešitve, in končaj ;
    else {
        izberi ( $A_j, v_j$ ) ∈ nogood &&  $A_j$  ima najnižjo prioriteto v nogood ;
        pošlji NG(nogood) agentu  $A_j$  ;
        odstrani ( $A_j, v_j$ ) iz agentView ;
        check(agentView) ;
    }
}

```

Slika 14.15: Asinhrono vračanje za upoštevanje omejitev spremenljivk.

Vračanje se začne s sporočilom NG. Množica Nogood vsebuje nekonsistentne delne preditve spremenljivk. Za agenta  $A_i$  je Nogood torej kar njegova vrednost agentView. Agent  $A_i$  pošlje Nogood agentu z najnižjo prioriteto med tistimi, ki imajo že prirejene vrednosti. Ko  $A_i$  pošlje Nogood agentu  $A_j$ , predvideva, da bo  $A_j$  spremenil svojo vrednost in zato iz svojega stanja agentView izbriše prireditev za  $A_j$  ter ponovno poskuša najti konsistentno prireditev.

Asinhrono vračanje je požrešna verzija hiperresolucije. Namesto pošiljanja vseh omejitve vsem agentom, agent poskuša zadovoljiti svoje omejitve in v Nogood vključi le vrednosti, ki so jih priredili agenti nad njim. Nove vrednosti sporoči le nekaj agentom, omejitve v Nogood pa le enemu.

Za primer s slike 14.11c deluje asinhrono vračanje takole:

- denimo, da so agenti urejeni po prioriteti  $X_1, X_2, X_3$ ,
- na začetku vsi izberejo naključno vrednost, npr. vsi modro,
- $X_1$  obvesti o svoji izbiri  $X_2$  in  $X_3$ , agent  $X_2$  obvesti  $X_3$ ,
- $X_2$  doda v svoj agentView  $\{X_1 = \text{modra}\}$ ,  $X_3$  doda  $\{X_1 = \text{modra}, X_2 = \text{modra}\}$ ,
- $X_2$  in  $X_3$  morata preveriti konsistentnost z lastno vrednostjo,
- $X_2$  ugotovi konflikt, spremeni svojo vrednost v rdeča in obvesti  $X_3$
- v tem času tudi  $X_3$  ugotovi konflikt, spremeni vrednost na rdeča, a o tem nikogar ne obvesti,
- $X_3$  sprejme drugo sporočilo od  $X_2$  in popravi svoj agentView na  $\{X_1 = \text{modra}, X_2 = \text{rdeča}\}$ ,
- $X_3$  ne more najti konsistentne vrednosti, zato uporabi hiperresolucijo in generira omejitev  $\{X_1 = \text{modra}, X_2 = \text{rdeča}\}$ ,
- to omejitev pošlje  $X_2$ , ker ima ta najnižjo prioriteto v seznamu Nogood,
- $X_2$  prejme sporočilo NG in ne more najti konsistentne vrednosti, zato generira omejitev  $\{X_1 = \text{modra}\}$  ter ga pošlje  $X_1$ ,
- $X_1$  ugotovi nekonsistentnost in spremeni svojo vrednost na rdeča ter pošlje sporočilo o tem  $X_2$  in  $X_3$ ,
- tako kot prej,  $X_2$  spremeni svojo vrednost na modra,  $X_3$  ne najde konsistentne vrednosti in generira omejitev  $\{X_1 = \text{rdeča}, X_2 = \text{modra}\}$ , nakar  $X_2$  generira omejitev  $\{X_1 = \text{rdeča}\}$  in ga pošlje  $X_1$ ,
- v tem trenutku ima  $X_1$  v Nogood  $\{X_1 = \text{modra}\}$  in  $\{X_1 = \text{rdeča}\}$ , uporabi hiperresolucijo in generira omejitev  $\{\}$ . Algoritem se ustavi z ugotovitvijo, da ni rešitve.

Asinhrono vračanje je najpomembnejši algoritem porazdeljenega upoštevanja omejitev. Zanj obstajajo številne razširitve. Tako denimo agentu ni potrebno poslati svojega celotnega stanja agentView. Za iskanje manjše, a še vedno zadovoljive množice Nogood, se uporablajo hevristike, saj je problem v splošnem NP-poln.

Podoben problem upoštevanju omejitev, ki ga je potrebno reševati v svetu agentov, je optimizacija z upoštevanjem omejitev. Poglejmo si enega od pristopov.

## 14.5 Porazdeljena optimizacija

Naloga porazdeljene optimizacije je, da agenti skupaj najdejo optimalno vrednost globalne funkcije. V tem poglavju sta opisana dva algoritma za ta problem, ki ju bomo uporabili na problemu iskanja najkrajših poti skozi graf. Več in obširneje o tej temi najde bralec v [5, 6, 2].

Problem iskanja najcenejših poti definiramo na usmerjenem grafu z  $n$  vozlišči in  $m$  povezavami, kjer je vsaki povezavi  $(a, b)$  pripisana cena  $c(a, b)$ . Naloga je najti pot z najmanjšo vsoto cen povezav od začetnega vozlišča  $s$  do enega od končnih vozlišč  $t \in T$ . Tako definiran problem je dovolj splošen, da ga lahko prilagodimo ne le za telekomunikacijske in transportne probleme, pač pa tudi za planiranje in številne druge probleme.

Za problem iskanja najkrajših poti obstajajo znani algoritmi, npr. Dijkstrov ali Bellman-Fordov, vendar nas v kontekstu agentov zanimajo porazdeljeni pristopi, ko vsak agent izračuna del najboljše rešitve in komunicira le z agenti v svoji okolini.

### Asinhrono dinamično programiranje

Za rešitev problema najprej uporabimo princip dinamičnega programiranja, ki temelji na dejstvu, da če leži vozlišče  $x$  na optimalni poti od  $s$  do  $t$ , potem je del poti od  $s$  do  $x$  tudi optimalen, kakor tudi del poti od  $x$  do  $t$ . To razbitje problema nam omogoča, da optimalno rešitev gradimo postopno od spodaj navzgor.

Najkrajšo razdaljo od poljubnega vozlišča  $i$  do končnega vozlišča  $t$  označimo z  $h^*(i)$ . Najkrajša pot od  $i$  do  $t$ , ki gre preko sosednjega vozlišča  $j$  lahko zapišemo kot  $f^*(i, j) = c(i, j) + h^*(j)$ , najkrajšo pot iz  $i$  preko poljubnega sosednjega vozlišča pa  $h^*(i) = \min_j f^*(i, j)$ . Algoritmom asinhronega dinamičnega programiranja na sliki 14.16 se izvaja na vsakem od vozlišč grafa.

Vsako od vozlišč  $i$  hrani vrednost  $h(i)$ , ki je približek  $h^*(i)$ . Na začetku te vrednosti inicializiramo na  $\infty$ , med izvajanjem pa se te vrednosti zmanjšujejo in konvergirajo k pravi vrednosti  $h^*(i)$ . Konvergenca zahteva en korak za vsako vozlišče na najkrajši poti, kar pomeni, da bo v najslabšem primeru potreben  $n$  korakov.

Postopno delovanje algoritma prikazuje zaporedje grafov na sliki 14.17. Poleg vozlišč so pri vsakem koraku zapisane posodobljene vrednosti ocene  $h(i)$ .

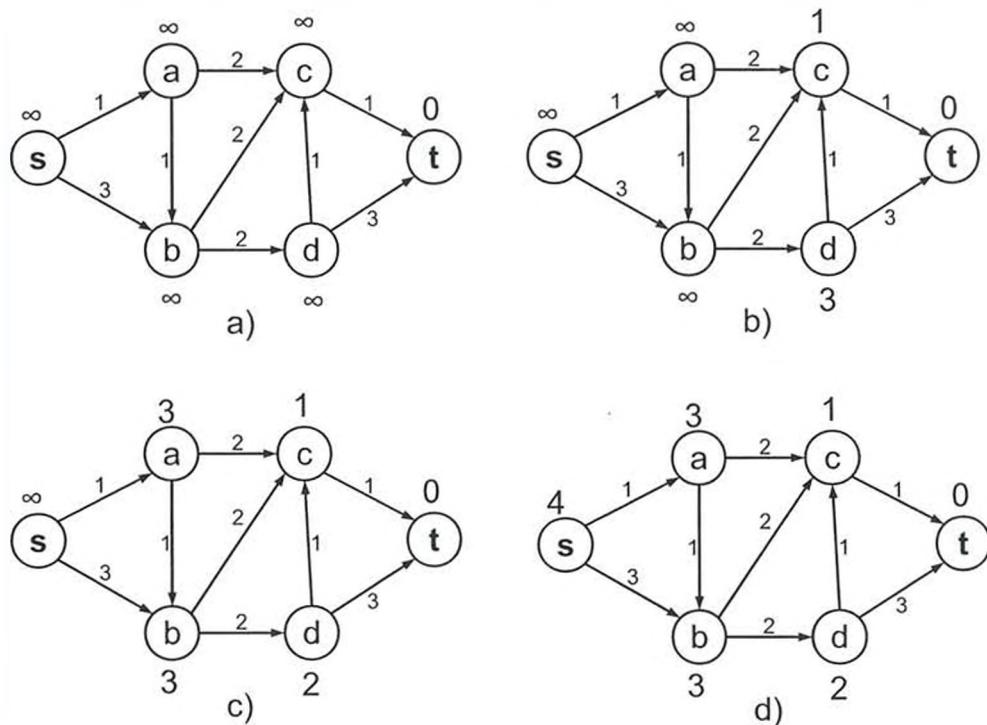
Slaba stran algoritma je, da potrebuje agenta za vsako vozlišče grafa, kar je v primeru realističnih ogromnih grafov (npr. šah) nesprejemljivo. Drugačen pristop s hevrističnem iskanjem, ki zahteva manj agentov, predstavlja algoritem *LRTA\**.

```

// algoritmom se izvaja na vsakem od vozlišč i danega grafa
void ADP(Vozlišče i) {
    if ( i je ciljno vozlišče )
        h(i) = 0 ; // točna vrednost
    else
        h(i) = ∞ ; // inicializacija
    do {
        foreach ( vozlišče j sosedno i )
            f(j) = c(i,j) + h(j) ;
            h(i) = minj f(j) ;
    } while (true) ;
}

```

Slika 14.16: Asinhrono dinamično programiranje za iskanje najkrajših poti.



Slika 14.17: Zaporedje spremenjanja vrednosti za iskanje najkrajših poti z asinhronim dinamičnim programiranjem.

### Hevristični algoritem $LRTA^*$

Algoritem  $LRTA^*$  (*learning real-time A<sup>\*</sup>*) uporablja enega ali več agentov za iskanje najkrajših poti. Podan je s psevdokodo na sliki 14.18. Zaradi enostavnosti predstavimo najprej delovanje enega agenta.

```
// algoritem lahko izvaja eden ali več agentov
void LRTA*() {
    i = s; // začetno vozlišče
    while ( i ni ciljno vozlišče ) {
        foreach ( vozlišče j sosedno i )
            f(j) = c(i, j) + h(j);
        b = arg minj f(j); // med enakima vozliščema izberi naključno
        h(i) = max(h(i), f(b));
        i = b;
    }
}
```

Slika 14.18: Algoritem  $LRTA^*$  za iskanje najkrajših poti.

Agent v danem vozlišču poišče najboljše ocenjeno sosednje vozlišče, se premakne vanj in nadaljuje tako dolgo, da pride do cilja. Začetna vrednost  $h$  vseh vozlišč je nastavljena na 0. Med potjo agent osvežuje  $h$ -vrednosti vozlišč. Agent na grafu izvede več poskusov in če med različnimi poskusi hrani vrednosti  $h(i)$ , se te vrednosti postopno izboljšujejo. Za nenegativno ceno povezav in dopustno oceno  $h$  (torej  $h(i) \leq h^*(i)$  za vsa vozlišča  $i$ ), agent zanesljivo najde najboljšo rešitev. Najboljšo rešitev identificira tako, da se v dveh zaporednih izvajanjih pot agenta ne spremeni.

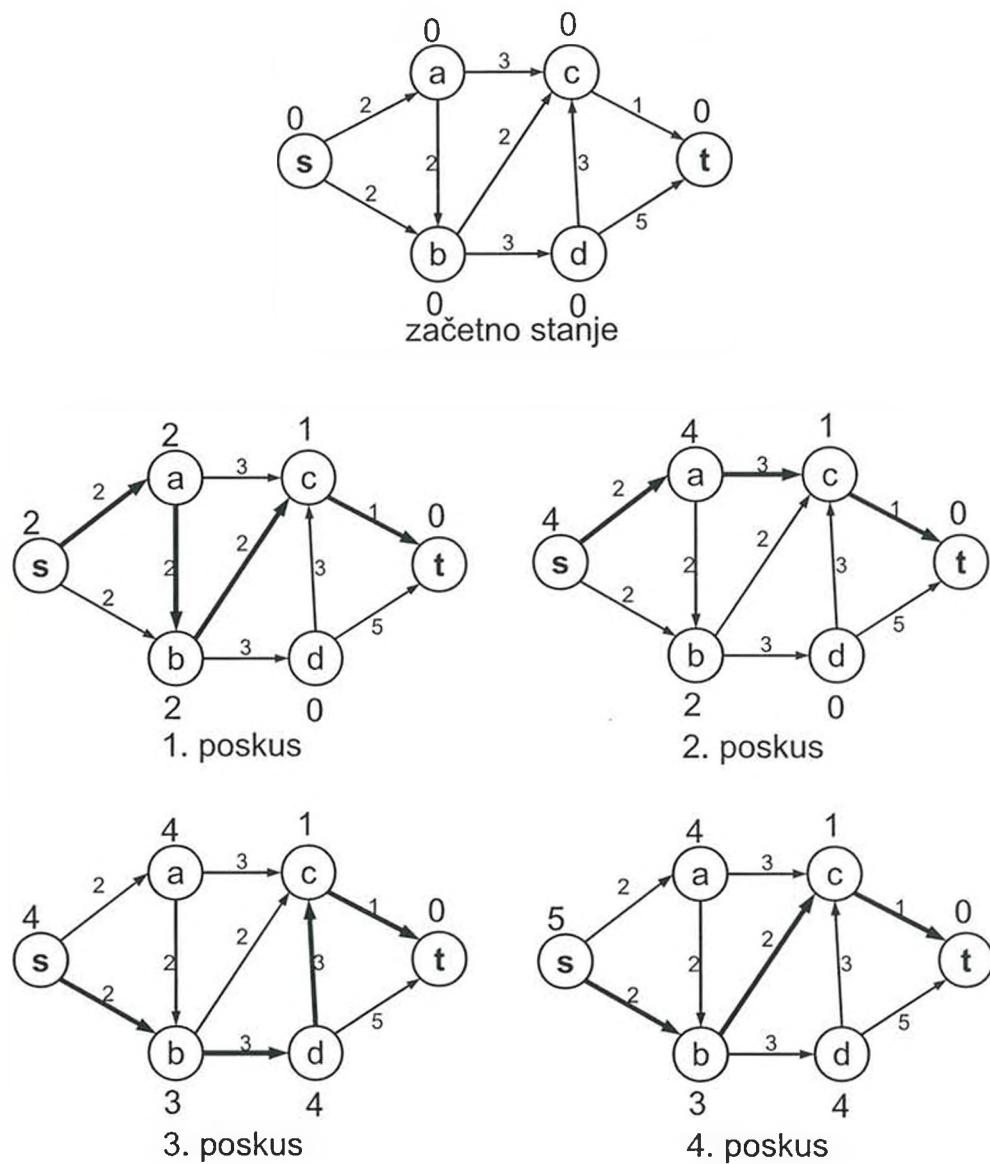
Slika 14.19 prikazuje štiri zaporedne poskuse, ki jih je izvajal agent. S krepkimi povezavami so označene povezave, ki jih je ubral.

Algoritem  $LRTA^*$  lahko izvaja tudi več agentov hkrati, v tem primeru ga označimo z  $LRTA^*(n)$ , kjer  $n$  pomeni število agentov, ki hkrati izvajajo algoritem. Izvajanje algoritma se lahko precej pospeši, saj agenti pri naključni izbiri enako ocenjenih poti uberejo različne poti in lahko nekateri cilj dosežejo prej. Če imajo do ocen  $h$  dostop vsi agenti, se učijo drug od drugega, npr. po vsakem poskusu se lahko vrednosti za vsa vozlišča  $i$  osvežijo takole:  $h(i) = \max_k h_k(i)$ , kjer je  $h_k(i)$  ocena vozlišča  $i$ , ki jo je našel agent  $k$ .

Slika 14.20 prikazuje tri poskuse izvajanja algoritma  $LRTA^*(2)$ , torej delovanje dveh agentov vzporedno. Z različnim črtkanjem so označene povezave, ki jih je izbral le eden od agentov, s polno krepko črto pa povezave, ki sta jih izbrala oba agenta.

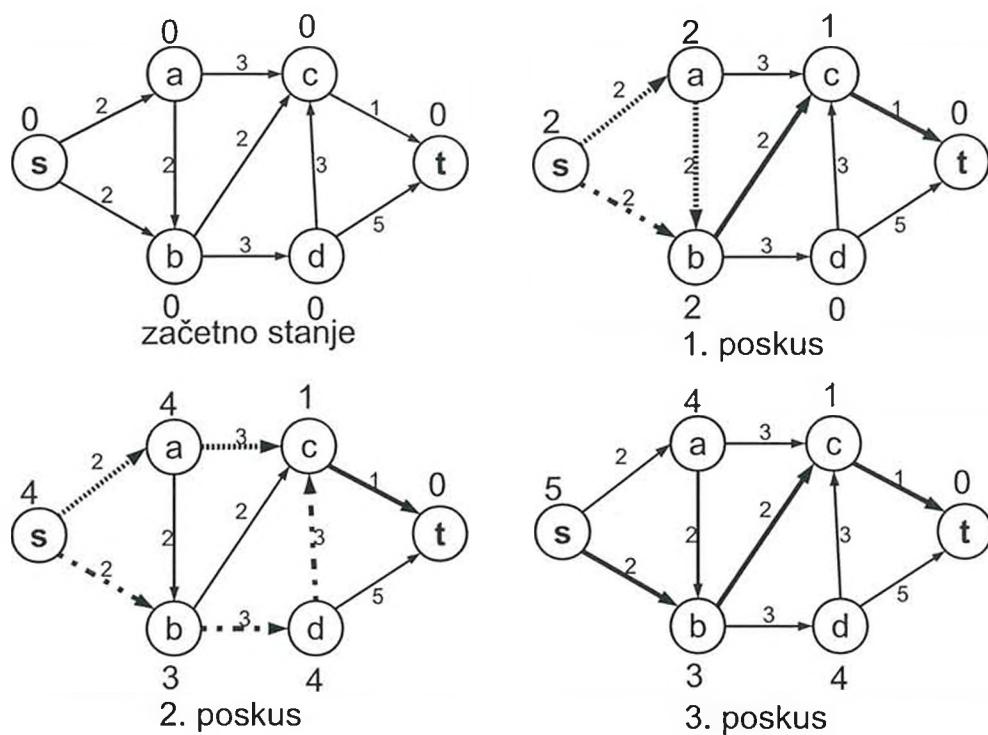
## Literatura

- [1] Daniel Kahneman, Amos Tversky. *Choices, values, and frames*. Cambridge University Press, 2000.



Slika 14.19: Širje poskusi agenta, ki izvaja LRTA\* za iskanje najkrajših poti.

- [2] Richard E. Korf. Real-time heuristic search. *Artificial intelligence*, 42(2-3):189–211, 1990.
- [3] David L. Poole, Alan K. Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.



Slika 14.20: Trije poskusi izvajana algoritma  $LRTA^*(2)$  za iskanje najkrajših poti.

- [4] Stuart J. Russell, Peter Norvig. *Artificial intelligence: a modern approach, 3rd edition.* Prentice Hall, 2009.
- [5] Yoav Shoham, Kevin Leyton-Brown. *Multiagent systems: algorithmic, game-theoretic, and logical foundations.* Cambridge University Press, 2009.
- [6] Gerhard Weiss, (ur.). *Multiagent systems: a modern approach to distributed artificial intelligence.* The MIT press, 2000.

## Poglavlje 15

# Spodbujevano učenje

*Ko brez miru okrog divjam,  
prijatlji prašajo me, kam?  
Prašájte raj' oblak nebá,  
prašájte raji val morjá,  
kadar mogočni gospodar  
drví jih semtertje vihar.*

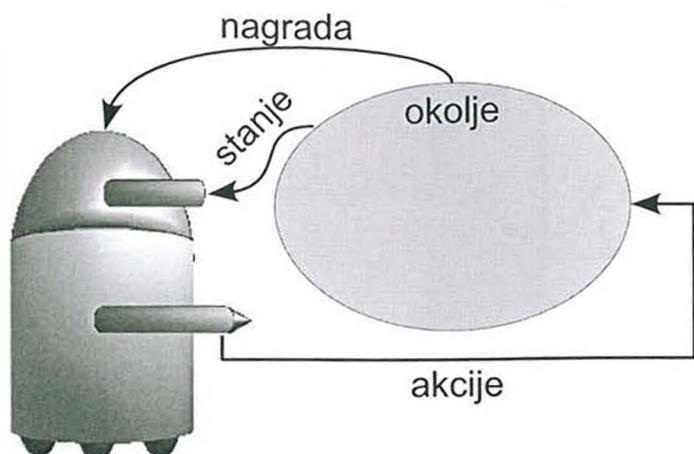
France Prešeren

Ime spodbujevano učenje (*reinforcement learning*) se nanaša na skupino računskih metod, kjer mora agent sam odkriti, katere akcije naj izvede v okolju, da bo dobil skupno kar največjo nagrado. Ne gre za nadzorovano učenje, pač pa za poizkušanje in raziskovanje na podlagi drugačne povratne informacije. Nagrada ali kazen iz okolja agenta spodbujata, da se uči in poskuša izdelati strategijo (*policy*), ki mu bo omogočila doseči cilje, oziroma kar največjo skupno nagrado. Pri tem lahko izbira med majhnimi kratkoročnimi in morebitnimi večjimi dolgoročnimi nagradami. Gre za dilemo med raziskovanjem in izkoriščanjem (*exploration or exploitation*): ali naj agent izkorišča to, kar že ve, da bi maksimiziral nagrado, ali naj razešče nova stanja, kjer je morebiti nagrada višja.

Odziv okolja ni nujno takojšen. Agent okolja ne pozna, temveč z njim eksperimentira. O njem si zgradi model, da bi se naučil, katere akcije mu prinesejo največjo nagrado. Okolje mu ne pove, zakaj je bila neka akcija uspešna. Kot primer si lahko predstavljamo, da agent igra igro, katere pravil ne pozna, čez 100 potez pa mu soigralec sporoči, da je izgubil. Če bo agent igrал mnogo iger, bo morda kdaj slučajno zmagal. Na podlagi teh končnih odzivov si bo izdelal model pravil igre in poskušal najti strategijo, ki ga bo večkrat vodila do zmage.

Shema delovanja agenta, ki predstavlja spodbujevano učenje, predstavlja slika 15.1. Njegovo delovanje lahko predstavimo takole:

- agent deluje v času,
- neprestano se uči in načrtuje,



Slika 15.1: Delovanje agenta z vidika spodbujevanega učenja.

- vpliva na okolje,
- okolje je negotovo in stohastično.

Navedimo nekaj področij, kjer je bilo spodbujevano učenje uspešno:

- uporaba v (industrijskih) nadzornih sistemih (denimo nadzor nad dvigali) in industrijskih robotih,
- upravljanje s premoženjem,
- dinamično prirejanje kanalov v mobilnih komunikacijah,
- v splošnih robotih se uporablja za različne naloge, denimo navigacijo, premikanje in prijemanje,
- v robotskem nogometu,
- v igrah; denimo, v backgammonu uporabljajo ta pristop najuspešnejši pristopi, kot sta TD-Gammon in Jellyfish.

Spodbujevano učenje v tem poglavju predstavljamo le z najpomembnejšimi koncepti in nekaj primeri. Najprej predstavimo komponente spodbujevanega učenja in jih ilustriramo na preprostem zgledu. Nato posamezne komponente formalno definiramo in si jih po vrsti ogledamo. Uvedemo markovsko lastnost in predstavimo nekaj načinov učenja vrednostne funkcije z eksplicitnim in implicitnim modelom okolja. Več o spodbujevanem učenju lahko bralec najde v [4, 2, 1].

## 15.1 Komponente spodbujevanega učenja

Obstajajo štiri bistvene sestavine spodbujevanega učenja.

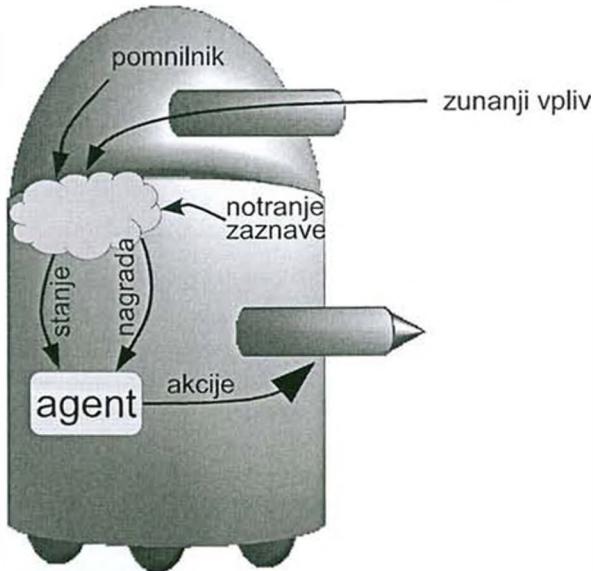
**Strategija** odgovarja na vprašanje, kaj storiti v danem trenutku, oziroma definira agentove izbire in akcije. Strategija je pogosto predstavljena z odločitvenimi pravili, v manjših problemskih prostorih pa s tabelo akcij. Lahko je rezultat iskanja ali načrtovanja, lahko je deterministična ali stohastična.

**Nagrada** je povratna informacija iz okolja v obliki realnega števila. Agent poskuša maximizirati skupno vsoto nagrad.

**Vrednost stanja** je agentovo interno vrednotenje stanja, v katerem se trenutno nahaja. Opisuje agentovo pričakovanje, kaj lahko dolgoročno dobi v danem stanju. Pri šahu bi to ustrezalo vrednotenju trenutne pozicije. Vrednost stanja zato implicitno vključuje tudi ocene naslednjih stanj, skozi katera se bo agent morda premikal. Vrednosti stanj se poskušamo naučiti, ponavadi jih agent poskuša z večkratnim ponavljanjem (vzorčenjem) zanesljivo oceniti.

**Model okolja** ni nujno prisoten v spodbujevanem učenju. Agentu, ki ima to komponento, omogoča, da na njegovi podlagi ocenjuje vrednosti stanj in vrednosti akcij, ne da bi jih dejansko izvedel.

Shemo notranje sestave agenta, ki ustreza našemu modelu, prikazuje slika 15.2.



Slika 15.2: Shema sestave agenta z vidika spodbujevanega učenja.

Poglejmo si, kako bi se s spodbujevanjem učenjem lotili igranja igre križci in krožci. Pravila igre so preprosta. Nasprotnika izmenoma na polje  $3 \times 3$  postavlja križce in krožce, zmaga pa tisti, ki ima tri svoje znake v ravni črti navpično, vodoravno ali diagonalno. Igre se lotimo pod predpostavko, da imamo pred sabo nepopolnega nasprotnika, ki dela napake. Sestavimo tabelo stanj, kot jo prikazuje tabela 15.1. Za vsako stanje hranimo njegovo vrednost  $V(s)$ , ki predstavlja verjetnost zmage. Z mnogokratnim igranjem igre lahko te verjetnosti zanesljivo ocenimo.

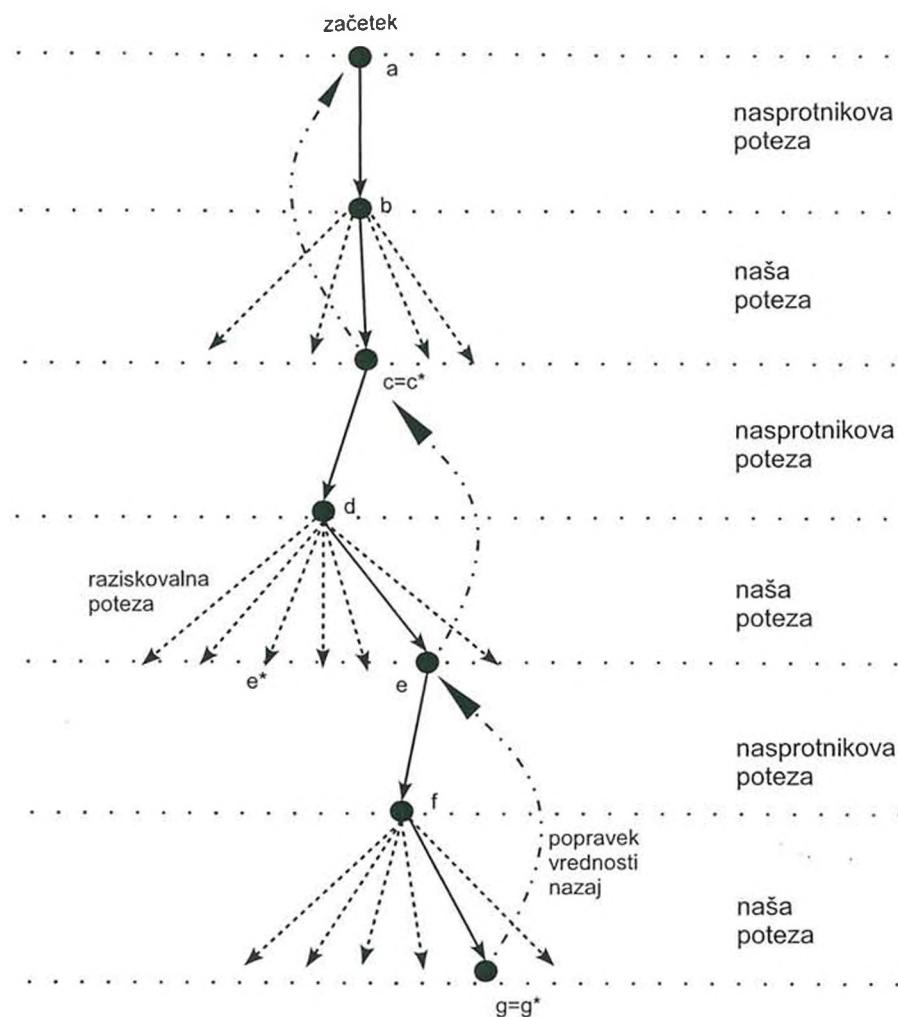
stanje	$V(s)$	opomba
	0.5	
	0.5	
	⋮	⋮
	1	zmaga
	⋮	⋮
	0	poraz
	⋮	⋮
	0	neodločeno
	⋮	⋮

Tabela 15.1: Tabela stanj in njihovih vrednosti za zmago pri igri križev in krožev.

Če poznamo vrednosti stanja  $V(s)$ , je zmagovalna strategija, ki se je moramo držati, preprosta in jo prikazuje slika 15.3. Na vsakem koraku maksimiziramo verjetnost za zmago, zato požrešno izberemo tistega od naslednikov, ki ima najvišjo vrednost  $V(s)$  (na sliki 15.3 je označen z \*).

Za zanesljive ocene kvalitete bomo igrali mnogo iger. Rezultate iger, zmago, poraz ali neodločen izid, vzvratno upoštevamo na stanjih, skozi katera smo prišli do izida. Seveda je vprašanje, kako določiti vrednosti stanj  $V(s)$ . Preprosta strategija uspeh vzvratno razširja nazaj. Če z  $s$  označimo stanje pred našo požrešno potezo, z  $s'$  pa po potezi, želimo, da se del vrednosti  $s'$  prenese tudi na  $s$ :

$$V(S) = V(s) + \alpha(V(s') - V(s)),$$



Slika 15.3: Potek preiskovanja in vzvratnega osveževanja vrednosti za križce in krožce.

kjer  $\alpha$  imenujemo velikost koraka in ga nastavimo na majhno pozitivno vrednost, recimo 0.1. Pri tem omenimo, da z majhno verjetnostjo občasno izberemo tudi neoptimalno potezo, kar imenujemo raziskovalni korak. Na sliki 15.3 smo namesto stanja z najvišjo vrednostjo  $e^*$  izbrali stanje  $e$ . S tem omogočimo tudi odkrivanje novih in morda bolj perspektivnih poti.

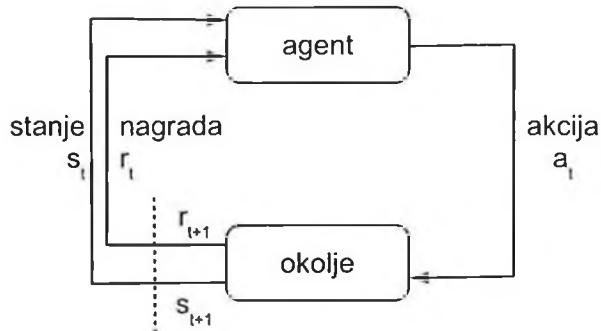
Primer s križci in krožci je poučen, vendar bi ga lahko še precej izboljšali. Pri hranjenju stanj bi lahko upoštevali simetrijo, učni sistem bi se igrala lahko učil tudi z igranjem proti samemu sebi, ali pa bi izdelali model obnašanja nasprotnika (denimo njegovih priljubljenih potez).

V nasprotju s primerom križev in krožev, v splošnem spodbujevano učenje ni omejeno

na probleme s končnim številom stanj. Pri neskončnem ali zelo velikem številu stanj generiramo le tista stanja, na katera naletimo med preiskovanjem in dejansko uporabimo pri rešitvi. Spodbujevano učenje tudi ni omejeno le na igre ali na probleme, kjer je potreben odgovor nasprotnika.

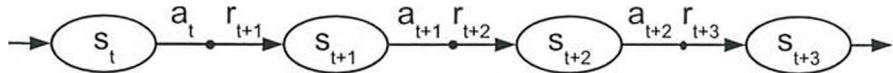
## 15.2 Formalizacija

Zapišimo zdaj bolj formalno, kar smo v grobem že predstavili. Formalizacijo predstavlja slika 15.4.



Slika 15.4: Formalna predstavitev spodbujevanega učenja.

Interakcija agenta z okoljem poteka v diskretnih časovnih korakih  $t = 0, 1, 2, \dots$ . V časovnem koraku  $t$  je agent v stanju  $s_t \in S$ . Izvede akcijo  $a_t \in A(s_t)$ , dobi nagrado  $r_{t+1} \in \mathcal{R}$  ter preide v naslednje stanje  $s_{t+1}$ . Zaporedje prehajanj med stanji prikazuje slika 15.5.



Slika 15.5: Zaporedje akcij, nagrad in stanj pri spodbujevanem učenju.

Agent izvaja akcije glede na svojo strategijo. S  $\pi_t$  označimo strategijo v koraku  $t$ . Strategija ni nujno deterministična, akcije lahko izberemo tudi z določeno verjetnostjo. V splošnem je strategija preslikava iz stanja v verjetnost akcije. Izraz  $\pi_t(s, a)$  tako predstavlja verjetnost, da agent v času  $t$  iz stanja  $s_t = s$  izvede akcijo  $a_t = a$ . S  $\pi^*$  označimo optimalno strategijo.

Vrednost stanja označimo z  $V(s)$ . Funkcija  $V^\pi(s)$  je vrednost stanja  $s$  pri dani strategiji  $\pi$ . Agent spreminja svojo strategijo glede na pridobljene izkušnje. Njegov cilj je dolgoročno pridobiti kar največjo nagrado. Različne metode spodbujevanega učenja se med seboj razlikujejo predvsem v načinu spreminjanja strategije. Podajmo nekaj splošnih napotkov, kako predstaviti problem, da bo primeren za reševanje s spodbujevanim učenjem.

Naj ima agent neko predstavo o okolju, ki ga ne more poljubno spremenijati. Predpostavimo, da je okolje agenta nedeterministično, vendar stacionarno, kar pomeni, da prehode med stanji in nagrade vedno generira enaka verjetnostna porazdelitev – "naravni zakoni" se ne spreminja. Nagrado obravnavamo, kot da prihaja iz okolja, ker je agent ne more poljubno spremenijati.

Agentove akcije so poljubne interakcije z okoljem. Lahko so nizkonivojske, denimo spremenjanje napetosti motorja, ali višjenivojske, kot so ukazi: kupi delnico, premakni figuro, postavi krožec na polje (1,3), ipd. Podobno široko lahko razumemo tudi stanja. Ta so lahko konkretne meritve na podlagi okolja (npr. pozicija v prostoru, stanje na igrальнem polju) ali pa abstraktne, simbolične in tudi agentove predstave (npr. izgubljen).

Nagrada je realno število, ki jo agent dobi iz okolja, in jo želi maksimizirati. Določa, kaj želimo doseči, ne pa tudi, kako bomo to dosegli. Način, kako doseči cilj, je prepuščen agentu, ki bo s svojim eksperimentiranjem rešil nalogu. Velja, da lahko agent eksplicitno in večkrat izmeri svoj uspeh. Predpostavka v spodbujevanem učenju je, da lahko cilje predstavimo tako, da maksimizirajo kumulativno vsoto delnih nagrad. Takšen način nagrajevanja je dovolj prožen, da ga lahko prilagodimo in uporabimo v številnih primerih.

### 15.3 Optimizacijski kriteriji

Označimo niz nagrad od časa  $t$  naprej z  $r_{t+1}, r_{t+2}, r_{t+3}, \dots$ . V splošnem želimo za vsako stanje optimizirati pričakovano nagrado  $E(R_t)$ , vendar imamo več možnosti, kako natančno bomo definirali ta kriterij. Tri najbolj razširjene definicije so:

**Končen horizont,** kjer agent upošteva fiksno število naslednjih stanj. To zapišemo

$$R_t = r_{t+1} + r_{t+2} + \dots + r_{t+h},$$

kjer je  $t+h$  zadnji časovni korak, denimo ko agent doseže cilj. Takšna definicija uspeha je smiselna pri epizodnih problemih, kjer je življenska doba agenta znana. Ena epizoda bi bila denimo eno iskanje prehoda skozi labirint. Agent maksimizira pričakovano nagrado v tem obdobju

$$E\left(\sum_{k=1}^h r_{t+k}\right).$$

Uporabljata se dve strategiji:

**h-koračna optimalna strategija:** agent na 1. koraku naredi akcijo, ki je najboljša ob predpostavki, da lahko naredi še  $h-1$  akcij, na 2. koraku akcijo, ki je optimalna za možnih še  $h-2$  akcij, ...

**h-premično koračna strategija:** agent na vsakem koraku naredi akcijo, ki je najboljša ob predpostavki, da lahko naredi še  $h$  akcij.

Pri izbiri končnega horizonta se je potrebno zavedati omejenosti pogleda vnaprej.

**Neskončen horizont:** delovanje agenta nima naravnega konca in se ne konča, a bližnja stanja so pomembnejša kot bolj oddaljena. Agent optimizira dolgoročno zaporedje nagrad, ki se v prihodnosti geometrijsko zmanjšujejo. Kumulativna nagrada v vsakem stanju je utežena vsota:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, 0 \leq \gamma < 1,$$

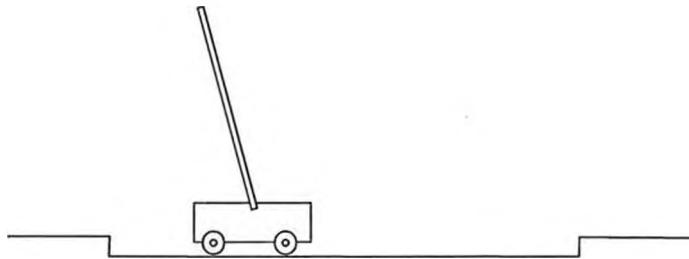
kjer je  $\gamma$  faktor zmanjševanja (*discount factor*). Faktor lahko interpretiramo kot neke vrste obresti, za katere se zmanjša nagrada, preden iz prihodnosti prispe v sedanost. Druga interpretacija je, da gre za zaporedje poskusov in je faktor zmanjševanja verjetnost preživetja še enega koraka. Nenazadnje lahko na ta faktor gledamo zgolj kot na način, kako omejimo vsoto neskončne vrste. Z  $\gamma$  kontroliramo vpliv bolj oddaljenih stanj napram bližnjim stanjem. Predstavlja kratkovidnost oz. daljnovidnost agenta (večji je  $\gamma$ , pomembnejša je prihodnost).

**Pričakovana povprečna nagrada**, kjer agent optimizira dolgoročno povprečno nagrado

$$\lim_{h \rightarrow \infty} E\left(\frac{1}{h} \sum_{k=1}^h r_{t+k}\right).$$

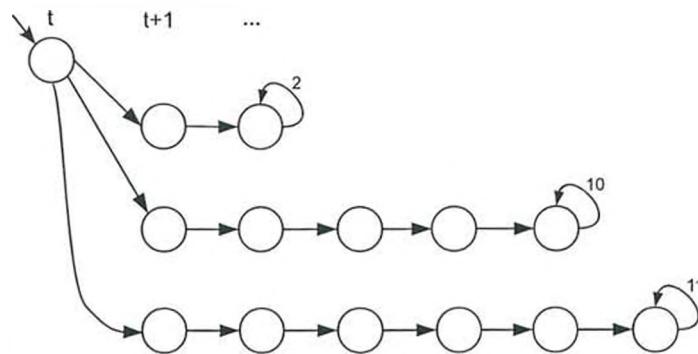
Slabost tega načina je, da ne loči med bližnjimi in oddaljenejšimi nagradami.

Za ilustracijo različnega obravnavanja nagrade si poglejmo problem balansiranja premičnega vozička s palico, kot ga prikazuje slika 15.6.



Slika 15.6: Problem vodenja vozička s palico. Palica se med premikanjem ne sme nagniti pod kritični kot, ker bi se prevrnila, in voziček ne sme udariti v rob.

S pomočjo spodbujevanega učenja se želimo naučiti voditi voziček tako, da palica med premikanjem čimdlje ne pada in da voziček ne zadane roba. Če problem obravnavamo kot epizodno nalogu, je nagrada enaka +1 za vsak korak, dokler palica ne pada. Uspeh je torej enak številu korakov pred padcem. V primeru, da gre za ponavljajočo se nalogo (neskončen horizont), za nagrado določimo -1 ob padcu in 0 sicer. Uspeh agenta je določen z  $-\gamma^k$  za  $k$  uspešnih korakov pred padcem. V obeh primerih je uspeh tem večji, čim dlje agent uspešno vodi voziček.



Slika 15.7: Različni kriteriji optimalnosti izberejo različne poti: končen horizont s  $h = 5$  izbere zgornjo pot, neskončen horizont z  $\gamma = 0.9$  izbere srednjo pot in povprečna nagrada izbere spodnjo pot.

Da ni vseeno, kakšen način obravnavanja nagrad izberemo, ilustrira slika 15.7.

Stanja, skozi katera se premika agent, so povezana s puščicami. Nagrade so 0 za vse prehode razen označenih. Trenutno stanje je v zgornjem levem kotu, označenem s puščico. Agent s končnim horizontom  $h = 5$  bi izbral zgornjo pot, saj je njegova nagrada v tem primeru  $2 + 2 + 2 = 6$ , po ostalih dveh poteh pa 0. Agent z neskončnim horizontom in  $\gamma = 0.9$  bi izbral srednjo pot, saj je njegova nagrada v tem primeru  $10 \cdot \gamma^5 + 10 \cdot 0.9^6 + \dots = 10\gamma^5 \frac{1}{1-\gamma} \approx 59.0$ , medtem ko znaša v zgornjem primeru 16.2 in v spodnjem primeru 58.5. Agent, ki maksimizira pričakovano povprečno nagrado, bi izbral spodnjo pot, saj mu v limiti prinese v povprečju 11 točk, kar je več kot 2, ki jih pridobi po zgornji poti, ter 10 po srednji poti.

Kako pomembna je izbira parametrov, si lahko ogledamo, ko spremenimo horizont na  $h = 10$  in postane za agenta s končnim horizontom najboljša srednja pot. S spremembou  $\gamma = 0.2$  bi agent z neskončnim horizontom izbral zgornjo pot. Za agenta s povprečno pričakovano nagrado pa vedno ostane najprivlačnejša spodnja pot.

## 15.4 Kriteriji za učenje

Poleg kriterijev za oceno strategije, ki smo jih obravnavali v prejšnjem razdelku, je potrebno omeniti še kriterije za oceno uspešnosti samega spodbujevanega učenja. Navedimo tri, medsebojno le deloma združljive kriterije.

- Konvergenca k optimalni rešitvi: nekateri algoritmi imajo dokazano asimptotično konvergenco, kar je s teoretičnega vidika sicer odlično, za praktično rabo pa manj pomembno kot to, kako hitro algoritem pride do zadovoljive rešitve.
- Hitrost konvergencije: ker je optimalnost večinoma dosegljiva šele v asimptoti, je bolje definirati hitrost konvergencije k skoraj optimalni rešitvi ali nivo optimalnosti po določenem času. V obeh primerih je potrebno določiti parameter - bodisi bližino optimalnosti bodisi čas. Slabost tega kriterija je tudi, da ima algoritem s hitro konvergenco

morda večjo skupno napako kot algoritem, ki konvergira počasi, a zbere večjo skupno nagrado.

- Razlika do optimalne strategije (*regret*) je vsota razlik med optimalno strategijo in strategijo, ki jo je našel učni algoritem. Ta kriterij upošteva razlike, kadarkoli se med izvajanjem pojavi, vendar je zanj težko dobiti rezultate, saj optimalna strategija pogosto ni znana.

## 15.5 Markovsko spodbujevano učenje

Pomemben podproblem spodbujevanega učenja upošteva med stanji markovsko lastnost. Stanje je markovsko, če je naslednje stanje odvisno le od trenutnega. Formalno zapišemo, da za vsa naslednja stanja  $s'$  in nagrade  $r$  ter za vsa pretekla stanja, akcije in nagrade  $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0$  velja

$$P(s_{t+1} = s' | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0) = P(s_{t+1} = s' | s_t, a_t).$$

Denimo za šah markovska lastnost velja, ker trenutna pozicija vsebuje vso potrebno informacijo za nadaljevanje in nam ni potrebno poznati zgodovine potez. Markovska lastnost je koristna aproksimacija, ki poenostavi učne algoritme, celo če okolje modeliramo.

Če v spodbujevanem učenju velja markovska lastnost, gre za markovski odločitveni problem (*MDP - Markov decision process*), ki je najpomembnejši primer spodbujevanega učenja. Kadar je množica stanj in akcij končna, govorimo o končnem markovskem odločitvenem problemu. Zanj definiramo množico stanj  $S$ , množico akcij  $A$ , verjetnosti prehodov za en korak

$$T(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a) \text{ za vse } s, s' \in S, a \in A(s)$$

ter vrednosti nagrad

$$R(s, a) = E(r_{t+1} | s_t = s, a_t = a) \text{ za vse } s \in S, a \in A(s).$$

Vrednost stanja  $s$  je pričakovana nagrada, ko začnemo iz danega stanja in sledimo agentovi strategiji  $\pi$ :

$$V^\pi(s) = E_\pi(R_t | s_t = s).$$

Za neskončen horizont, ki ga zaradi enostavnosti izračunov in izpeljav najpogosteje uporabljamo, zapišemo

$$V^\pi(s) = E_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right).$$

Marsikdaj je bolj prikladno namesto z vrednostjo stanj operirati z vrednostmi akcij. Definiramo funkcijo  $Q$ , ki pomeni vrednost akcije v danem stanju in pri dani strategiji. Tudi funkcija  $Q$  je definirana s pričakovano nagrado

$$Q^\pi(s, a) = E_\pi(R_t | s_t = s, a_t = a),$$

oziroma za neskončen horizont

$$Q^\pi(s, a) = E_\pi \left( \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right).$$

Vrednosti  $V^\pi(s)$  in  $Q^\pi(s, a)$  agent oceni na podlagi izkušenj. Če si agent za vsako stanje/akcijo zapomni povprečje nagrad, ki jih je dobil med svojim delovanjem, te vrednosti v limiti konvergirajo v dejansko vrednost. Pri velikih prostorih si vseh stanj ne moremo zapomniti, pač pa jih posplošimo, oziroma modeliramo. Največkrat jih ocenimo oziroma izračunamo z učnimi metodami Monte-Carlo, ki temeljijo na vzorčenju.

Dobra stran markovske lastnosti z neskončnim horizontom je, da lahko vrednosti stanj in akcij zapišemo z rekurzivnimi enačbami, ki jih imenujemo Bellmanove enačbe:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots + \gamma^3 r_{t+4} \dots \\ &= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \dots + \gamma^2 r_{t+4} \dots) \\ &= r_{t+1} + \gamma R_{t+1}. \end{aligned}$$

Za vrednost stanja tako dobimo rekurzivno enačbo

$$V^\pi(s) = E_\pi(R_t | s_t = s) = E_\pi(r_{t+1} + \gamma V(s_{t+1} | s_t = s)).$$

Če namesto operatorja za matematično upanje zapišemo vsoto, dobimo:

$$V^\pi(s) = \sum_a \pi(s, a) \left( \sum_{s'} T(s, a, s') (R(s, a) + \gamma V^\pi(s')) \right).$$

Zgornji zapis Bellmanove enačbe je v bistvu množica linearnih enačb in sicer po ena za vsako stanje. Edina rešitev enačb je funkcija vrednosti za  $\pi$ .

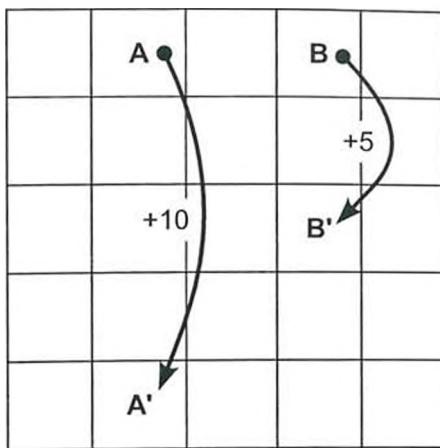
Kot primer si poglejmo kvadratno mrežo, kot jo prikazuje slika 15.8.

Agent se po njej lahko premika v štiri smeri: sever, jug, vzhod in zahod. Ti premiki tvorijo množico akcij, izmed katerih lahko izbiramo. Če bi agent zaradi akcije padel iz mreže, ostane na istem mestu z nagrado  $-1$ . Druge akcije imajo nagrado  $0$ , razen premikov v točko A ali v točko B. Iz A skočimo v A' z nagrado  $10$  in iz B skočimo v B' z nagrado  $5$ .

Predpostavimo, da uporabljamo naključno strategijo, kjer se v vse smeri premaknemo z enako verjetnostjo. Desna tabela prikazuje vrednosti stanj za to strategijo z neskončnim horizontom in faktorjem zmanjševanja  $\gamma = 0.9$ .

### Iskanje strategije pri danem modelu

Preden se lotimo reševanja splošnega markovskega odločitvenega problema, opišimo iskanje optimalne strategije pri že znanem modelu okolja. Spomnimo se, da model okolja predstavlja funkcija verjetnosti prehodov  $T(s, a, s')$  ter vrednosti nagrad  $R(s, a)$ . V nadaljevanju predpostavljamo neskončen horizont, čeprav podobni rezultati obstajajo tudi za končni horizont in za povprečno nagrado.



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Slika 15.8: Leva slika prikazuje kvadratno mrežo, po kateri se agent premika, desna tabela pa vrednosti stanj za strategijo, kjer se v vseh stanjih v vse smeri premikamo z enako verjetnostjo in za neskončen horizont z  $\gamma = 0.9$ .

Optimalna vrednost stanja  $V^*(s)$  je definirana kot pričakovana vsota nagrad, ki jih bo agent pridobil, če začne v stanju  $s$  in izvaja optimalno strategijo. Zapišemo

$$V^*(s) = \max_{\pi} V^{\pi}(s) = \max_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right).$$

Obstaja ena sama optimalna funkcija vrednosti stanj, in sicer kot rešitev enačb

$$V^*(s) = \max_a (R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s')) \text{ za } \forall s \in S.$$

Enačba pravi, da je vrednost stanja pričakovana takojšnja nagrada, ki ji prištejemo zmanjšano vrednost naslednjega stanja, v katerega preidemo z izbiro najboljše akcije. Na podlagi optimalne vrednosti stanja zapišemo tudi optimalno strategijo

$$\pi^*(s) = \arg \max_a (R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s')).$$

Iz enačbe sledi, da če poznamo optimalne vrednosti stanj  $V^*$ , je najboljša kar požrešna strategija, saj optimalne rezultate dobimo tako, da med dosegljivimi stanji izberemo najboljšega.

Način, kako do optimalne strategije, je torej izračun optimalnih vrednosti stanj. Bellman je pokazal, da algoritem na sliki 15.9 konvergira k optimalnim vrednostim  $V^*$ .

Algoritem za nazaj (denimo od končnega stanja) popravlja vrednosti stanj. Nagrade se tako od končnih stanj širijo proti začetku.

Postavi se vprašanje, kdaj smo lahko zadovoljni s konvergenco vrednosti stanj. Obstaja izrek, ki pravi, da če je razlika dveh zapored izračunanih vrednosti stanja manjša od  $\varepsilon$ , je

```

void valueIteration() {
    poljubno inicializiraj vrednosti  $V(s)$ 
    do {
        foreach ( $s \in S$ ) {
            foreach ( $a \in A$ ) {
                 $Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V(s')$  ;
                 $V(s) = \max_a Q(s, a)$  ;
            }
        }
    } while (! zadovoljen) ;
}

```

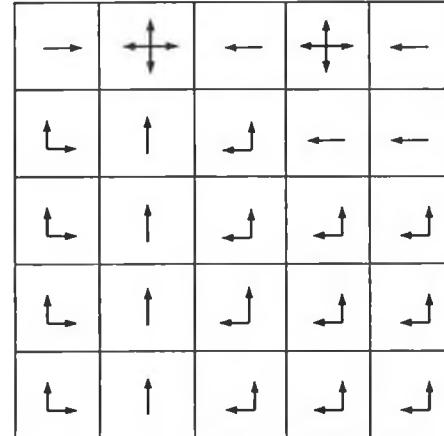
Slika 15.9: Iskanje optimalnih vrednosti stanj.

vrednost požrešne strategije, ki uporablja te ocene, v vsakem stanju od optimalne manjša za največ  $2\epsilon \frac{\gamma}{1-\gamma}$ . To omejitev lahko s pridom uporabimo za ustavitevni kriterij. V praksi je požrešna strategija optimalna večinoma že dolgo preden vrednosti stanj konvergirajo k optimalnim.

Algoritem z iteracijo vrednosti je precej robusten, saj lahko vrednosti stanjem pritejamamo tudi asinhrono in vzporедno, pa konvergenca k optimalnosti rešitev vseeno ni ogrožena.

Optimalne vrednosti stanj  $V^*(s)$  za kvadratno mrežo s slike 15.8 prikazuje tabela na levi strani slike 15.10. Optimalna strategija, ki izhaja iz nje, pa je za vsa stanja prikazana v tabeli na desni.

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

Slika 15.10: Optimalne vrednosti  $V^*(s)$  (levo) in optimalna strategija  $\pi^*(s)$  za problem s slike 15.8.

Namesto, da računamo optimalne vrednosti stanj in si iz njih s požrešno metodo določimo optimalno strategijo, lahko optimalno strategijo iščemo tudi direktno, kot to prikazuje algoritmom na sliki 15.11.

```
void policyIteration() {
    izberi poljubno strategijo  $\pi'$ 
    do {
         $\pi = \pi'$ 
        izračunaj vrednost strategije  $\pi$  tako, da rešiš linearne enačbe
         $V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$  ;
        izboljšaj strategijo v vsakem stanju
         $\pi'(s) = \arg \max_a R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s')$  ;
    } while ( $\pi \neq \pi'$ ) ;
}
```

Slika 15.11: Iskanje optimalnih strategij.

Vrednost strategije izračunamo kot pričakovano nagrado z neskončnim horizontom, če v vsakem stanju izvajamo to strategijo. Ko vrednost dobimo z rešitvijo linearnih enačb, preverimo ali lahko strategijo izboljšamo s spremembou prve akcije. Če ne moremo spremeniti strategije v nobenem stanju, je dobljena strategija zagotovljeno optimalna.

## Iskanje optimalne strategije

Kot rečeno, model sveta za markovske odločitvene probleme predstavlja funkciji prehajan med stanji  $T(s, a, s')$  in vrednosti nagrad  $R(s, a)$ . V splošnem ne moremo predpostavljati, da model sveta že imamo, ampak se v interakciji z okoljem učimo optimalne strategije. Ločimo dva načina:

- strategije se učimo direktno, ne da bi se naučili tudi modela okolja,
- naučimo se model okolja, na tej podlagi pa se kasneje ali istočasno učimo še strategije.

Za oba pristopa obstaja vrsta algoritmov. Poglejmo si najprej dva predstavnika prve vrste, nato pa še primer algoritma, ki se model okolja uči tudi neposredno.

## Učenje z razlikami v času

Ideja učenja z razlikami v času (*temporal difference (TD) learning*) je, da nazaj na prejšnja stanja prenesemo nekaj razlike z naslednjimi stanji. Vrednosti stanj popravljamo

$$V(s) = V(s) + \alpha(V(s') - V(s))$$

ozziroma za neskončni horizont

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))$$

kjer je  $\alpha$  parameter, ki se med učenjem postopno zmanjšuje in omogoča konvergenco. Vrednost stanja  $s$  popravimo med vsakim obiskom tako, da je bližje svojemu nasledniku, oziroma  $r + \gamma V(s')$ , kar je podobno algoritmu z iteracijo vrednosti na sliki 15.9, le da vzorčimo iz dejanskih interakcij z okoljem in ne iz znanega modela. Tudi za učenje z razlikami v času obstaja dokaz za konvergenco k optimalnim vrednostim stanj.

### Q-učenje

Namesto iz stanj lahko izhajamo tudi iz akcij. Definiramo  $Q^*(s, a)$  kot nagrado, ki jo dosegemo, če iz stanja  $s$  naredimo akcijo  $a$ , nato pa sledimo optimalni strategiji. Namesto optimalne akcije, ki smo jo predpostavili pri  $V^*(s)$ , pri  $Q^*(s, a)$  izberemo poljubno akcijo  $a$ . Rekurzivno definiramo

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a'} Q^*(s', a').$$

Ker velja  $V^*(s) = \max_a Q^*(s, a)$ , lahko optimalno strategijo dobimo kot

$$\pi^*(s) = \arg \max_a Q^*(s, a).$$

Q-učenje je zelo podobno učenju z razlikami v času, le da namesto vrednosti stanj  $V$  ocenjujejo vrednosti prehodov. Pravilo za njihovo iterativno osveževanje lahko zapišemo

$$Q^*(s, a) = Q(s, a) + \gamma \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a)).$$

Tudi za Q-učenje je pokazana konvergencia k optimalnim vrednostim  $Q^*(s, a)$  z ustreznim postopnim zmanjševanjem parametra  $\alpha$ . V praksi je ta metoda najbolj uporabljan učni algoritem spodbujevanega učenja.

### Učenje s prioritetnim osveževanjem

Predstavimo tudi enega od algoritmov, ki se modela okolja, predstavljenega s funkcijo prehajanj med stanji  $T(s, a, s')$  in funkcijo vrednosti nagrad  $R(s, a)$ , neposredno uči med postopkom učenja optimalnih vrednosti stanj. Vsako izkušnjo, ki jo algoritem pridobi iz okolja, lahko predstavimo s četverko  $(s, a, s', r)$ , ki pove, da smo v stanju  $s$  izvedli akcijo  $a$  ter zato prešli v stanje  $s'$  in dobili nagrado  $r$ . Vsaka nova izkušnja ustrezeno osveži verjetnosti, predstavljene v  $T(s, a, s')$  in  $R(s, a)$ . Algoritem s prioritetnim osveževanjem (*prioritized sweeping*) je prikazan na sliki 15.12.

Algoritem uporablja izkušnje zato, da gradi model okolja (funkciji  $T$  in  $R$ ) ter osveži svojo strategijo, istočasno pa za osveževanje strategije uporablja tudi model. Najprej za izbrano stanje izračuna spremembo vrednosti funkcije  $Q(s, a)$ . Če je ta sprememba pozitivna, želi prenesti del spremembe tudi na zaslужne predhodnike. Sprememba pri predhodnikih pa spet povzroči spremembo vrednosti pri predhodnikih predhodnikov itn. Dodatno si moramo zato v vsakem stanju zapomniti njegove predhodnike, ali pa jih pridobiti kako drugače. Ker bi osveževanje nazaj lahko kmalu zajelo ogromno (eksponentno mnogo) stanj, kandidate za

```

void prioritizedSweeping() {
    foreach ( $s \in S, a \in A$ ) {
        poljubno inicializiraj vrednosti stanj  $V(s)$  ;
        poljubno inicializiraj model:  $T(s, a, s')$  in  $R(s, a)$  ;
    }
    PQ = {} ; // prioritetna vrsta je prazna
    do {
         $s$  = trenutno stanje ;
        // izberemo najboljšo akcijo glede na trenutne ocene ali izvedemo raziskovalno potezo
         $a$  = izberi glede na  $\arg \max_{a'} Q(s, a')$  ali naključno ;
        izvedi akcijo  $a$  ; // preidemo v stanje  $s'$  in dobimo nagrado  $r$ 
        osveži model za  $T(s, a, s')$  in  $R(s, a)$  na podlagi izkušnje  $(s, a, s', r)$ 
         $p = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$  ;
        if (  $p > 0$  )
            PQ.add( $s, a, p$ ) ; // v prioritetno vrsto dodamo  $(s, a)$  s prioriteto  $p$  ;
        for ( $k$  elementov PQ) {
             $(s, a)$  = PQ.deleteMax() ; // vzemi element z najvišjo prioriteto
            iz  $T$  in  $R$  določi  $s'$  in  $r$  ;
             $Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$ ; // osvežimo
            foreach (  $(\bar{s}, \bar{a})$ , ki vodijo v  $s$  ) {
                 $\bar{r}$  = iz modela ocenjena nagrada ;
                 $\bar{p} = \bar{r} + \gamma \max_a Q(s, a) - Q(\bar{s}, \bar{a})$  ;
                if (  $p > 0$  )
                    PQ.add( $\bar{s}, \bar{a}, \bar{p}$ ) ; // v prioritetno vrsto dodamo  $(\bar{s}, \bar{a})$  s prioriteto  $\bar{p}$ 
            }
        }
    }
} while ( ! zadovoljen ) ;
}

```

Slika 15.12: Algoritem prioritetnega osveževanja.

osveževanje uvrstimo v prioritetno vrsto in jih osvežimo le omejeno število. Vlogo prioritete igra ponavadi velikost spremembe funkcije  $Q$ . Na ta način se spremembe propagirajo učinkovito in v najkoristnejših smereh, kajti velika sprememba  $Q$  ponavadi pomeni, da gre za obetavno smer veriženja nazaj.

Če med izvajanjem algoritma nalstimo na obetavno stanje (npr. ciljno), se sproži obsežno osveževanje po predhodnikih. Če pa naletimo na stanje, ki se le malo razlikuje od predhodnikov, bo osveževanja malo. S prioritetno vrsto se izognemo osveževanju v manj koristnih smereh in usmerimo pozornost k obetavnejšim stanjem.

V primerjavi s Q-učenjem je prioritetno osveževanje lahko računsko mnogo bolj učinkovito. Podrobnosti najde bralec v [3].

## Literatura

- [1] Lucian Busoniu, Robert Babuska, Bart De Schutter, Damien Ernst. *Reinforcement Learning and Dynamic Programming using Function Approximators*. Taylor & Francis CRC Press, 2010.
- [2] Leslie Pack Kaelbling, Michael L. Littman, Andrew W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
- [3] Lihong Li. *A unifying framework for computational reinforcement learning theory*. Doktorska disertacija, Rutgers University, 2009.
- [4] Richard S. Sutton, Andrew G. Barto. *Reinforcement learning: An introduction*. The MIT press, 1998.

# Poglavlje 16

## Načrtovanje

*Če načrtuješ za leto vnaprej, vzgajaj riž. Če načrtuješ za 20 let, vzgajaj drevesa. Če načrtuješ za stoletja, vzgajaj ljudi.*

kitajski pregovor

Naloga načrtovanja je, da najde zaporedje akcij, ki bodo omogočile izvedbeni enoti, demno agentu ali kontrolni enoti robota, da doseže določen cilj. Načrtovanje zahteva, da agent sklepa o posledicah svojih dejanj, zato je načrtovanje marsikdaj povezano z logičnim sklepanjem.

V tem poglavju predstavimo nekaj načinov za predstavitev akcij in njihovih učinkov na okolje. Uvodoma predstavimo problem načrtovanja. Začnemo z eksplicitno predstavitvijo stanja, nadaljujemo pa s predstavitvijo z značilnostmi ter s sistemom STRIPS. Omenimo preiskovanje naprej, ki v dani predstavitvi poišče dejanski načrt. Poglavlje končamo z načrtovanjem z delno urejenostjo, ki dopušča tudi vzporedno izvedbo nekaterih akcij.

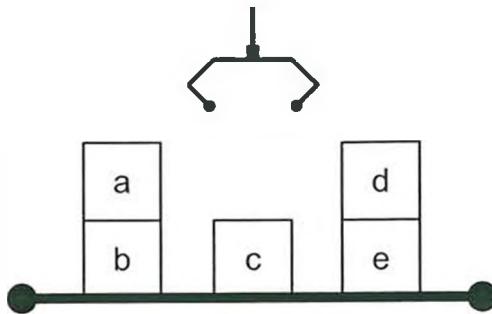
Naša predstavitev pokrije le delček obstoječih sistemov za načrtovanje. Več o tej temi lahko bralec najde v učbenikih umetne inteligence [4, 2, 5] ter v specializiranih publikacijah o načrtovanju, npr. [6, 3].

### 16.1 Klasično načrtovanje

Klasičen pristop k načrtovanju zahteva, da so akcije deterministične, torej da lahko predvidimo vse njihove učinke ter da ne obstajajo zunanji dejavniki, ki bi lahko spremenili stanje sveta. Prav tako predvideva, da lahko stanje sveta v celoti zaznamo in da je čas diskreten.

Da lahko sklepamo o potrebnih akcijah, moramo imeti cilj, model sveta in model učinkov akcij. Vsake akcije ne moremo izvesti v poljubnem stanju, ampak morajo biti izpolnjeni določeni predpogoji (npr., robot ne more zgrabiti objekta, če ni v njegovi bližini). Primer preprostega sveta kock in robotske roke, ki lahko kocke premika, predstavlja slika 16.1. To

okolje uporabimo za nekaj zgledov v nadaljevanju.



Slika 16.1: Svet kock in robotska roka.

Akcije povzročijo spremembe v okolici. Te spremembe so učinek akcije. Denimo, ko robot spusti kocko, povzroči spremembo v okolju: spremeni se lokacija kocke, robotska roka je spet prosta, kocka pokrije tam že stoječo drugo kocko, itd.

### Eksplisitna predstavitev

V najpreprostjem primeru lahko predpogoje za izvedbo akcij in njihove učinke predstavimo eksplisitno s tabelo, ki vsebuje za vsako možno stanje vse dovoljene akcije ter stanja, kamor nas te akcije privedejo. Shemo tovrstne predstavitev prikazuje tabela 16.1.

stanje	akcija	novo stanje
$s_3$	$a_{21}$	$s_{27}$
$s_9$	$a_7$	$s_{39}$
$s_9$	$a_4$	$s_{97}$
$s_{27}$	$a_{12}$	$s_{49}$
***	***	***

Tabela 16.1: Eksplisitna predstavitev stanj in akcij.

Eksplisitna predstavitev predstavlja usmerjeni graf, kjer so stanja vozlišča, akcije pa predstavljajo povezave med stanji. Iskanje načrta je v takšnem primeru poenostavljeno na iskanje poti v usmerjenem grafu, za kar lahko uporabimo algoritme iz 13. poglavja. Eksplisitna predstavitev stanj z grafom, razen v posebnih primerih, ni mogoča oziroma zaželena. Za to obstajajo trije razlogi:

1. število vseh stanj je preveliko, da bi jih lahko predstavili, zajeli in o njih sklepali,
2. že majhne spremembe v okolju povzročijo velike spremembe v predstavitvi. Če bi robotu dodali npr. senzor za napoljenost baterije, da bi se lahko usmeril k viru napajanja, bi bilo potrebno spremeniti vsa stanja,

3. predstavitev ne upošteva strukture in urejenosti, ki ponavadi veljajo za učinke akcij.

Z upoštevanjem strukture sta lahko specifikacija predpogojev, učinkov in sklepanje mnogo bolj učinkovita in kompaktna.

Alternativa eksplizitni predstaviti je, da učinke akcije predstavimo s spremembami, ki jih povzročijo na značilnostih okolja.

### Predstavitev akcij z značilnostmi

Namesto, da bi eksplizitno našeli akcije, ki jih je mogoče izvesti v določenem stanju, jih raje določimo z značilnostmi sveta, ki so potrebne za izvedbo akcije in ki predstavljajo predpogoj, da se lahko akcija izvede. Predpogoj sestavlja izjave, za katere zahtevamo, da so resnične. Te izjave omejujejo agenta, da lahko izvede le nekatere akcije. Denimo, v svetu kock je predpogoj, da lahko agent prime kocko, da je robotska roka prazna, pozicionirana nad kocko ter da kocke ne prekriva druga kocka.

Za specifikacijo učinka akcije uporabljamo pravila, ki določijo stanje okolja po izvedeni akciji. Telo pravil lahko vsebuje izvedene akcije in značilnosti predhodnega stanja. Pravila so dveh oblik:

- vzročno pravilo določa spremembo značilnosti,
- ohranitveno pravilo določa ohranitev značilnosti.

Pravila za lokacijo robota v svetu kock bi bila denimo:

$\text{lokacija} = \text{sredina} \Leftarrow (\text{lokacija} = \text{levo}) \wedge (\text{akcija} = \text{premik desno})$

$\text{lokacija} = \text{sredina} \Leftarrow (\text{lokacija} = \text{desno}) \wedge (\text{akcija} = \text{premik levo})$

$\text{lokacija} = \text{sredina} \Leftarrow (\text{lokacija} = \text{sredina}) \wedge \neg(\text{akcija} = \text{premik levo}) \wedge \neg(\text{akcija} = \text{premik desno}).$

Prvi dve pravili sta vzročni, zadnje pa je ohranitveno pravilo.

### Predstavitev STRIPS

Namesto da predstavljamo stanje okolja z njegovimi lastnostmi, lahko opišemo akcije in njihove učinke. Takšna predstavitev se po instituciji, kjer so jo razvili, imenuje STRIPS (STanford Research Institute Problem Solver). Vsaka akcija je opisana s

- predpogoji, ki jih sestavlja množica pogojev, ki morajo biti izpolnjeni, da se akcija izvrši,
- učinki, ki opisujejo spremembe lastnosti okolja, ki jih akcija povzroči. Učinke delimo na dodane lastnosti, ki predstavljajo zdaj resnične lastnosti, ter izbrisane lastnosti, ki predstavljajo lastnosti, ki po izvedbi akcije ne veljajo več.

Za akcijo "zgrabi kocko X" bi tako veljalo:

$\text{predpogoj} = (\text{roka} = \text{prosta}) \wedge (\text{lokacija} = \text{pozicija}(X)) \wedge (\text{naVrhu}(X))$

$\text{dodane lastnosti} = (\text{roka} = \text{drzi}(X)) \wedge (\text{naVrhu}(\text{pod}(X)))$

$\text{izbrisane lastnosti} = (\text{roka} = \text{prosta}) \wedge (\text{naVrhu}(X)).$

Predvidevali smo obstoj funkcij oz. predikatov pozicija( $X$ ), ki vrne lokacijo objekta  $X$ , ter na vrhu( $X$ ), ki vrne true, če je nad objektom  $X$  ni nobenega drugega objekta.

Manipulacija s tovrstno predstavljivo pravil je še posebej elegantna v programskejem jeziku prolog, ki omogoča enostavno manipulacijo tako pravil kot predikatov. Več najdemo v [1].

## Začetna stanja in cilji

Pod predpostavko, da je okolje deterministično ter da lahko opazujemo vse njegove lastnosti, lahko začetno stanje načrtovalcu podamo tako, da zapišemo vse na začetku veljavne lastnosti.

Pri ciljih, ki jih želimo doseči, ločimo končne lastnosti, ki morajo veljati, da bo načrt izpolnjen, in varnostne lastnosti, ki morajo veljati ves čas izvajanja načrta. S slednjimi se zavarujemo, da ne zaidemo v nevarna in neveljavna stanja, ko bi se robotska roka morda poškodovala ali zašla izven predvidenega območja. Včasih je smiseln uvesti vmesne cilje, ki jih dosežemo kot vmesne fazce pri izvajanjju načrta. Koristni so lahko tudi cilji, ki omejujejo vire, na primer porabljen čas ali energijo.

## Iskanje načrta

Najpreprostejši planer uporablja predstavitev s prostorom stanj ter algoritme za njihovo preiskovanje. Vhod v načrtovanje predstavlja ustrezna predstavitev podanega začetnega ter končnega stanja ter specifikacija akcij, ki so na razpolago agentu. Opis akcij zajema njihove predpogoje ter učinke. Preiskovanje poteka od podanega začetnega stanja do stanja, ki zadosti ciljnim pogojem. Povezave med stanji so akcije, ki so na voljo glede na trenutno stanje ter na izpolnjene predpogoje. Po izvedbi akcije morajo biti izpolnjeni tudi varnostni pogoji. Opozorimo, da tovrstno preiskovanje ni enako preiskovanju v grafu z eksplicitno predstavljenimi vsemi stanji, saj stanja generiramo dinamično med preiskovanjem.

Preiskovalni algoritmi, kot je  $A^*$ , zagotovo najdejo optimalno rešitev, vendar je za realne probleme težava velikanski prostor. Za nekatere probleme lahko najdemo dobre hevristike in tako rešitev vseeno najdemo.

Preiskovanje je mogoče začeti tudi v končnem stanju in stanja razvijati v smeri začetnega stanja. Težava takšnega preiskovanja je, da je cilj ponavadi izpolnjen v mnogih različnih stanjih sveta in že takoj na začetku dobimo ogromen preiskovalen prostor, zato iskanje nazaj večinoma ni praktično.

Mnogokrat je bolj učinkovito iskanje v drugačnem prostoru prostoru stanj, kjer vozlišča niso stanja, temveč cilji, ki jih želimo doseči. Takšno načrtovanje imenujemo regresijsko in postopno iz končnega stanja gradi predhodna stanja, ki so pripeljala do njega, pri tem pa lahko vsako od predhodnih stanj k ciljem prispeva eno ali več lastnosti, ki morajo veljati v naslednjem stanju. Postopek nadaljujemo, dokler ne pridemo do stanja, ki je začetno stanje originalnega problema. V dobljenem prostoru stanj tudi pri tem načinu načrt poiščemo z enim od preiskovalnih algoritmov, opisanih v 13. poglavju.

## 16.2 Načrtovanje z delnimi urejenostmi

Iskanje naprej in regresijsko načrtovanje definirata totalno urejenost akcij v vseh fazah načrtovanja, kar pomeni, da četudi bi lahko nekatere akcije izvajali sočasno, tega načrt ne upošteva. Ideja načrtovanja z delnimi urejenostmi je, da imamo med akcijami le delno urejenost, totalno urejenost pa le, kadar je to nujno. Pri predstavitevi tega pristopa se zgledujemo po [4].

Delno urejenost akcij definiramo tako, da velja  $a_1 \prec a_2$ , če akcija  $a_1$  nastopa v načrtu pred akcijo  $a_2$ . Uvedemo tudi dve posebni akciji, *start*, ki je pred vsemi ostalimi akcijami, ter *cilj*, ki je za vsemi akcijami. Prehod iz stanja *start* tako povzroči značilnosti, ki veljajo v začetnem stanju, stanje *cilj* pa kot predpogoj vsebuje značilnosti, ki veljajo v končnem stanju.

Zagotoviti moramo, da imajo akcije učinke, ki smo jih predvideli. Za vsak predpogoj  $P$  akcije  $a_2$  v načrtu moramo najti akcijo  $a_1$ , ki povzroči ta predpogoj. Trojko  $(a_1, P, a_2)$  imenujemo vzročna povezava. Delna urejenost določa, da je  $a_1 \prec a_2$ . Druge akcije, ki bi preprečile veljavnost  $P$ , morajo zato nastopati pred  $a_1$  ali po  $a_2$ . Algoritem je prikazan na sliki 16.2.

Algoritem začne z akcijama *start* in *cilj* ter delno urejenostjo  $start \prec cilj$ . Vzdržuje seznam odprtih nalog Agenda, ki vsebuje pare  $(P, A)$ , kjer je  $P$  ena od trditev o okolju, ki mora veljati za izvedbo akcije  $A$ . Na začetku Agenda vsebuje vse pare  $(G, cilj)$ , kjer so  $G$  vse trditve, ki morajo veljati v končnem stanju.

Na vsakem koraku izberemo eno izmed odprtih nalog  $(G, a_2)$ , kjer je  $G$  predpogoj za  $a_2$ , nato pa izberemo še akcijo  $a_1$ , ki doseže  $G$ . Izbrana akcija  $a_1$  je bodisi že v načrtu (npr. akcija *start*) bodisi gre za novo akcijo, ki jo dodamo v načrt. V delni urejenosti velja  $a_1 \prec a_2$ . Doda se tudi vzročna povezava  $(a_1, G, a_2)$ , ki določa, da  $a_1$  doseže  $G$  kot predpogoj za  $a_2$ . Vsaka druga akcija, ki razveljavlji  $G$ , mora torej nastopati pred  $a_1$  ali za  $a_2$ . Če je akcija  $a_1$  nova akcija, so njeni predpogoji dodani med odprte naloge v množici Agenda. Ves postopek se nadaljuje, dokler ni Agenda prazna.

Celotna procedura je nedeterministična zaradi ukazov izberi ter stavka **Either–Or**. Implementiramo jih kot iskanje preko vseh možnosti. Izbiramo med različnimi akcijami, ki dosežejo  $G$ , ter med tem, ali akcija, ki razveljavlji  $G$ , nastopa pred  $a_1$  ali po  $a_2$ .

Funkcija Omejitve.dodaj( $a_1 \prec a_2$ ) k trenutnim omejitvam doda  $a_1 \prec a_2$ . Če nova omejitev ni konsistentna z obstoječimi, postopek ne uspe in sproži se vračanje k drugim možnostim (nedeterminizem).

Funkcija Omejitve.ščiti( $(a_1, G, a_2), a$ ) med trenutnimi omejitvami preveri, če velja  $a \neq a_1$  in  $a \neq a_2$  ter če  $a$  razveljavlji trditev  $G$ . Če se razveljavitev zgodi, je med omejitve potrebno dodati  $a \prec a_1$  ali  $a_2 \prec a$ . Spet gre za nedeterministično izbiro, zato se sproži iskanje preko vseh možnosti.

Na koncu iz akcij, ki so vsebovane v seznamu Akcije, sestavimo načrt, ki se sklada s časovnimi omejitvami v množici Omejitve. Ker so Omejitve le delno urejene, za izvedljiv načrt, ki bi ga izvajal en sam agent, zahtevamo totalno urejenost in je mogočih ustvariti več različnih načrtov. Izberemo lahko najprimernejšega glede na dodatne kriterije, denimo čas ali porabo virov, možna pa je tudi sočasnna izvedba teh akcij v primeru, da imamo na voljo več agentov.

V večini načrtov nekatere akcije nastopajo večkrat. Da bo predstavljeni algoritem deloval tudi v takšnem primeru, je potrebno ločiti med posameznimi pojavitvami iste akcije, sicer

```

List partialOrderPlanner(Set Gs) {
    // Vhod: množica trditev o okolju Gs, ki jih želimo doseči
    // Vrača: načrt kako doseči Gs

    // lokalne spremenljivke:
    // Agenda: množica parov (P, A), kjer je P predpogoj za A
    // Akcije: seznam akcij v trenutnem načrtu
    // Omejitve: množica časovnih omejitev za akcije
    // Vzroki: množica trojk (a1, P, a2), kjer akcija a1 zagotavlja predpogoj P za a2

    // inicializacija
    Agenda = {(G, cilj) za vse G ∈ Gs} ;
    Akcije = {start, cilj} ;
    Omejitve = {start < cilj} ;
    Vzroki = {} ;
    do {
        izberi (G, a2) ∈ Agenda ; // še nedosežen par (predpogoj, akcija)
        Agenda.izbriši((G, a2)) ;
        Either {
            izberi a1 ∈ Akcije, kjer a1 doseže G ; // akcija je že del načrta
        }
        Or { // dodaj novo akcijo
            izberi a1 ∉ Akcije, kjer a1 povzroči G ;
            Akcije = Akcije ∪ {a1} ;
            Omejitve.dodaj(start < a1) ;
            foreach (v ∈ Vzroki )
                Omejitve.ščiti(v, a1) ;
            Agenda = Agenda ∪ {(P, a1) za vse P, kjer je P predpogoj za a1} ;
        }
        Omejitve.dodaj(a1 < a2) ;
        Vzroki = Vzroki ∪ {(a1, G, a2)} ;
        foreach (a ∈ Akcije ) // ščiti omejitve že izbranih akcij
            Omejitve.ščiti( (a1, G, a2), a ) ;
    } while ( Agenda != {} ) ;
    return totalno urejen seznam Akcije konsistenten z množico Omejitve
}

```

Slika 16.2: Postopek načrtovanja z delnimi urejenostmi.

algoritem poskuša najti delno urejenost, kjer so vse pojavitve iste akcije zgodijo ob istem času. To najlažje dosežemo z oštevilčenjem pojavitve posameznih akcij. Delna urejenost nato velja med oštevilčenimi pojavitvami, ne pa med akcijami samimi. Zaradi enostavnosti smo to podrobnost iz psevdokode izpustili.

## Literatura

- [1] Ivan Bratko. *Prolog programming for artificial intelligence*. Addison-Wesley, 2001.
- [2] George F Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6th edition*. Addison-Wesley Pearson Education, Boston, 2009.
- [3] Dana S. Nau. Current trends in automated planning. *AI magazine*, 28(4):43, 2007.
- [4] David L. Poole, Alan K. Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.
- [5] Stuart J. Russell, Peter Norvig. *Artificial intelligence: a modern approach, 3rd edition*. Prentice Hall, 2009.
- [6] Quiang Yang. *Intelligent Planning: A Decomposition and Abstraction Based Approach (Artificial Intelligence)*. Springer-Verlag, 1997.

## Dodatek A

# Ponovitev verjetnosti

*Zakoni verjetnosti: tako resnični v splošnem, tako varljivi v konkretnem.*

Edward Gibbon

V tem dodatku na kratko povzamemo poglavitev pojme glede verjetnosti, ki jih potrebujemo v ostalem besedilu, predvsem v poglavju 10. Govorimo o dogodkih, definiramo verjetnost in slučajne spremenljivke, razložimo pojem pogojne verjetnosti, Bayesovo pravilo in porazdelitve slučajnih spremenljivk.

### A.1 Računanje z dogodki

V okviru verjetnostnega računa in statistike je poskus opazovanje realizacije nekega pojava oziroma množice skupaj nastopajočih dejstev. Rezultat opazovanja imenujemo izid poskusa. Če rezultata ne moremo napovedati, je poskus naključen. Primeri naključnih poskusov so met igralne kocke, zaužitje novega zdravila skupine pacientov, ogled specifičnega spletnega mesta posameznega obiskovalca ali pa anketa na izbrani množici 1000 državljanov, če verjamejo v neznane leteče predmete.

Vzorčni prostor je množica vseh mogočih izidov poskusa. Za poskus z metom kocke je vzorčni prostor izid meta, ko pade 1, 2, 3, 4, 5 ali 6 pik. Za anketo vzorčni prostor predstavlja vse možne izbire tisočih prebivalcev.

Dogodek je katerikoli izid ali množica izidov slučajnega pojava. Lahko se v posameznem poskusu zgodi ali pa ne in je podmnožica vzorčnega prostora. Na primer dogodek je, da na igralni kocki vržemo 6 pik, da se pacientu po zaužitju zdravila stanje izboljša, obiskovalec pogleda določeno zaporedje spletnih strani ali da za anketo izberemo točno določene posameznike.

Nekatere vrste dogodkov poimenujemo. Gotov dogodek (označimo ga z  $G$ ) se zgodi ob vsaki ponovitvi poskusa, nemogoč dogodek (oznaka  $N$ ) se nikoli ne zgodi, slučajen dogodek se zgodi včasih. Primer slučajnega dogodka je, da na kocki pade 6 pik.

Dogodek  $A$  je poddogodek (ali način) dogodka  $B$ , če se vsakič, ko se zgodi  $A$ , zagotovo zgodi tudi  $B$ , kar zapišemo kot  $A \subset B$ . Primer tovrstnih dogodkov sta  $A$  (na kocki pade pade 6 pik) in  $B$  (na kocki pade več kot 4 pike).

Dogodka sta enaka, če sočasno velja  $A \subset B \wedge B \subset A \Leftrightarrow A = B$ . Takšna sta na primer dogodek  $A$  (na kocki pade 6 pik) in  $B$  (na kocki pade več kot 5 pik).

Z dogodki lahko tudi računamo. Ker gre dejansko za množice, računamo na enak način kot z množicami, čeprav nekateri avtorji uporabljajo tudi drugačno poimenovanje in zapis. Unija (vsota) dogodkov  $A$  in  $B$  se zgodi, če se zgodi vsaj eden od dogodkov  $A$  ali  $B$ , kar zapišemo  $A \cup B$ . Za dogodek  $A$  (na kocki pade 6 pik) in  $B$  (pade 5 pik) je unija dogodek  $C$  (pade več kot 4 pike)  $C = A \cup B$ .

Presek (produkt) dogodkov  $A$  in  $B$  se zgodi, če se zgodiha  $A$  in  $B$  hkrati:  $A \cap B$ . Na primer, za dogodek  $A$  (na kocki padejo več kot 4 pike) in  $B$  (pade manj kot 6 pik) je presek dogodek  $C$  (pade 5 pik),  $C = A \cap B$ .

Negacija dogodka  $A$  je njemu nasproten dogodek  $\bar{A}$ . K dogodku  $A$  (na kocki pade več kot 4 pike) je nasproten dogodek  $\bar{A}$  (pade manj ali enako 4 pike). Dogodke lahko predstavimo tudi z Vennovimi diagrami. Slika A.1 prikazuje po vrsti od leve proti desni in od zgoraj navzdol vzorčni prostor  $S$ , dogodek  $A$ , dogodek  $B$ ,  $A \cup B$ ,  $A \cap B$  in  $\bar{A}$ .

Popoln sistem dogodkov je množica dogodkov  $S = \{A_1, A_2, \dots, A_n\}$ , kjer se v vsaki ponovitvi poskusa zgodi natanko eden od dogodkov iz  $S$ . Velja, da so vsi dogodki mogoči:  $A_i \neq N$ , da so paroma nezdržljivi:  $A_i \cap A_j = N$ , za  $i \neq j$ , in da je njihova unija gotov dogodek:  $A_1 \cup A_2 \cup \dots \cup A_n = G$ . Za igralno kocko imamo več popolnih sistemov. Lahko ga sestavlja meti posameznih pik. Drug popoln sistem sta dva dogodka: vržem sodo število pik in vržem liho število pik.

## A.2 Verjetnost

Denimo, da  $n$ -krat ponovimo poskus in se  $k$ -krat zgodi dogodek  $A$ . Delež poskusov, v katerih se  $A$  zgodi, imenujemo relativna frekvanca (pogostost) dogodka  $A$

$$f(A) = \frac{k}{n}$$

Če poskus dolgo ponavljamo, se relativna frekvanca slučajnega dogodka ustali in sicer skoraj zmeraj toliko bolj, kolikor več ponovitev poskusa napravimo. Verjetnost (statistična definicija) vsakega od dogodkov je delež števila ponovitev tega dogodka v dolgem zaporedju poskusov. Verjetnost dogodka  $A$  v danem poskusu je število  $P(A)$ , pri katerem se navadno ustali relativna frekvanca dogodka  $A$  v velikem številu ponovitev tega poskusa.

Verjetnost ima naslednje lastnosti:

- $0 \leq P(A) \leq 1$ , kar izhaja iz definicije z relativno frekvenco, ki je vedno med 0 in 1.
- Za gotov dogodek velja  $P(G) = 1$  in za nemogoč  $P(N) = 0$ .
- Za poddogodke velja  $A \subset B \Rightarrow P(A) \leq P(B)$ .

### A.3 Pogojna verjetnost

Naredimo poskus in v njem opazujemo dogodka  $A$  in  $B$ . Verjetnost dogodka  $A$  je  $P(A)$ ,  $B$  pa je mogoč dogodek,  $P(B) > 0$ . Naj se dogodek  $B$  zgodi. Vprašamo se, kako se pri tem spremeni verjetnost dogodka  $A$ ? Če hočemo odgovoriti na to vprašanje, najprej opazimo, da se je spremenil (skrčil) vzorčni prostor, ki je zdaj omejen z dejstvom, da se je  $B$  zgodil.

Pogojna verjetnost  $P(A|B)$  je verjetnost dogodka  $A$  pri pogoju, da se zgodi dogodek  $B$ . Izračunamo jo kot

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

Izraz lahko napišemo tudi drugače in dobimo  $P(A \cap B) = P(B)P(A|B)$ . Če pogojno verjetnost zapišemo za  $B$ , dobimo  $P(A \cap B) = P(A)P(B|A)$ . Oba izraza združimo in dobimo  $P(B)P(A|B) = P(A)P(B|A)$ , iz česar sledi Bayesovo pravilo za pogojno verjetnost:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}.$$

V podkrepitev pravila izračunajmo naslednjo nalogu. Med 100 pacienti v bolnišnici so jih 10 izbrali za terapijo, ki poveča možnost ozdravitve iz 50% na 75%. Kolikšna je verjetnost, da je pacient prejel terapijo, če vemo, da je ozdravljen.

Z  $A$  označimo dogodek, da je pacient prejel terapijo, z  $B$  pa dogodek, da je ozdravljen. Iz podatkov preberemo  $P(B|A) = 0.75$ ,  $P(A) = 0.1$ . Izračunamo verjetnost za  $B$  kot  $P(B) = \frac{0.5 \cdot 0.90 + 0.75 \cdot 10}{100} = 0.525$ . Po Bayesu dobimo:  $P(A|B) = \frac{0.1 \cdot 0.75}{0.525} \approx 0.143$

Dogodka  $A$  in  $B$  sta neodvisna, če na verjetnost dogodka  $A$  ne vpliva izid dogodka  $B$  (npr. met dveh kovancev zapored).

Ker neodvisni dogodki ne vplivajo drug na drugega, velja  $P(A|B) = P(A)$  in  $P(B|A) = P(B)$ . Tedaj velja tudi  $P(A \cap B) = P(A)P(B)$ . Za nezdružljive dogodke pa velja  $P(A|B) = 0$  (kakšen je tedaj verjetnostni prostor?).

### A.4 Slučajne spremenljivke in porazdelitve

V slučajnem poskuusu se vrednost določenih opazovanih vrednosti lahko spreminja med različnimi ponovitvami poskusa. Takšne opazovane vrednosti imenujemo slučajne spremenljivke in jih označimo z velikimi črkami s konca abecede, npr.  $X, Y, Z$ .

V zvezi s spremenljivkami nas zanima njihova verjetnostna porazdelitev: katere vrednosti lahko zavzame spremenljivka (zaloga vrednosti) in kakšne so verjetnosti posameznih vrednosti ali intervalov vrednosti.

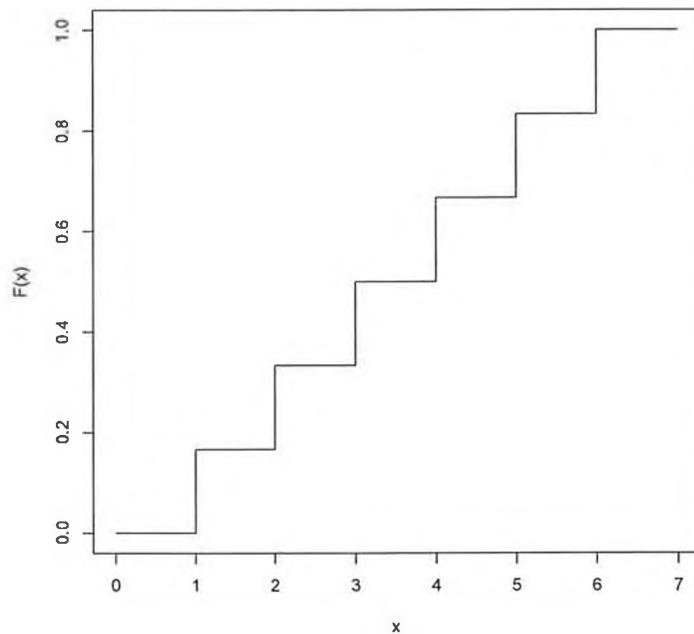
Slučajne spremenljivke so npr. število pik pri metu igralne kocke, število okvarjenih izdelkov pri testiranju ene šarže v proizvodnji, število klikov na spletni oglas v enem dnevu in izmerjeni električni tok na merilnem mestu.

Porazdelitveni zakon imenujemo predpis, ki določa, kolikšna je verjetnost vsake izmed možnih vrednosti ali intervala vrednosti. Vrednosti slučajne spremenljivke označujemo z malimi črkami, tako je ( $X = x_i$ ) dogodek, da slučajna spremenljivka  $X$  zavzame vrednost  $x_i$ . Vrednosti in njihove verjetnosti sestavljajo verjetnostno porazdelitev (distribucijo) in jih lahko zapišemo v tabelo. Za igralno kocko ima slučajna spremenljivka zalogo vrednosti  $\{1, 2, 3, 4, 5, 6\}$  in vsaka od vrednosti ima verjetnost  $\frac{1}{6}$ , kar prikazuje tabela A.1.

vrednost $x_i$	1	2	3	4	5	6
$p_i$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

Tabela A.1: Verjetnostna porazdelitev poštene igralne kocke.

Če je mogoče za vsako realno število  $x$  določiti verjetnost  $F(x) = P(X < x)$ , pravimo, da je porazdelitveni zakon slučajne spremenljivke  $X$  poznan.  $F(x)$  imenujemo (kumulativna) porazdelitvena funkcija. Primer kumulativne porazdelitvene funkcije za igralno kocko prikazuje slika A.2.



Slika A.2: Kumulativna porazdelitvena funkcija za met igralne kocke.

Diskretna slučajna spremenljivka  $X$  ima števno zalogo vrednosti  $(x_1, x_2, \dots, x_m, \dots)$ . Dogodki  $X = x_k$ ,  $k = 1, 2, \dots$  sestavljajo popoln sistem dogodkov. Verjetnost posameznega dogodka označimo s  $P(X = x_i) = p_i$ . Vsota verjetnosti vseh dogodkov je enaka 1:  $p_1 + p_2 + \dots + p_m + \dots = 1$ .

Zvezne slučajne spremenljivke lahko zavzamejo vsako realno število znotraj določenega

intervala. Slučajna spremenljivka  $X$  je zvezno porazdeljena, če obstaja taka integrabilna funkcija  $p(x)$ , imenovana gostota verjetnosti, da za vsak  $x \in \mathfrak{R}$  velja

$$F(X) = P(X < x) = \int_{-\infty}^{\infty} p(t)dt, \text{ kjer } p(x) = 0.$$

Velja tudi

$$P(x_1 \leq X < x_2) = \int_{x_1}^{x_2} p(t)dt.$$

Včasih nas zanima odnos med več naključnimi spremenljivkami, npr.  $X$  predstavlja starost pacienta,  $Y$  pa krvni tlak. Definiramo skupno porazdelitveno funkcijo  $F(X, Y) = P(X < x, Y < y)$ . Iz nje lahko (načeloma) dobimo porazdelitev posamezne spremenljivke  $F(x) = F(x, \infty) = P(X < x, Y < \infty)$ . Takšno porazdelitev imenujemo robna porazdelitev.

Slučajne spremenljivke in njihove porazdelitve lahko posplošimo na več dimenzij. Slučajni vektor je  $n$ -terica slučajnih spremenljivk  $X = (X_1, X_2, X_3, \dots, X_n)$ , njegova porazdelitvena funkcija za  $x_i \in \mathfrak{R}$  pa

$$F(x_1, x_2, \dots, x_n) = P(X_1 < x_1, X_2 < x_2, \dots, X_n < x_n).$$

Robno porazdelitev spremenljivke  $X_i$  iz vektorja dobimo kot

$$F(X_i) = F(\infty, \infty, \dots, x_i, \dots, \infty) = P(X_1 < \infty, X_2 < \infty, \dots, X_i < x_i, \dots, X_n < \infty).$$

Diskretne večrazsežne porazdelitve opisuje verjetnostna funkcija  $p_{k_1, k_2, \dots, k_n} = P(X_1 = x_{k_1}, X_2 = x_{k_2}, \dots, X_n = x_{k_n})$ , kjer je vsaka  $X_i$  diskretna slučajna spremenljivka.

V dvorazsežnem primeru s spremenljivkama  $X$  in  $Y$ , ki imata zalogu vrednosti  $(x_1, x_2, \dots, x_k)$  in  $(y_1, y_2, \dots, y_m)$ , lahko skupno porazdelitev  $P(X = x_i, Y = y_j)$  predstavimo z verjetnostno tabelo, kot to ilustrira tabela A.2.

	$y_1$	$y_2$	$\dots$	$y_m$	$X$
$x_1$	$p_{1,1}$	$p_{1,2}$	$\dots$	$p_{1,m}$	$p_1$
$x_2$	$p_{2,1}$	$p_{2,2}$	$\dots$	$p_{2,m}$	$p_2$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$x_k$	$p_{k,1}$	$p_{k,2}$	$\dots$	$p_{k,m}$	$p_k$
$Y$	$q_1$	$q_2$	$\dots$	$q_m$	1

Tabela A.2: Dvorazsežna diskretna porazdelitev spremenljivk  $X$  in  $Y$ .

Robno porazdelitev za spremenljivko  $X$  v tem primeru predstavlja vsote verjetnosti po vrsticah  $p_i$ , robno porazdelitev za  $Y$  pa vsote verjetnosti po stolpcih  $q_j$ .

## Dodatek B

# Slovar nekaterih angleških strokovnih izrazov

**accuracy** – točnost; glej *classification accuracy*

**admissability** – dopustnost hevristike; dopustna hevristika zagotavlja preiskovalnemu algoritmu  $A^*$ , da najde optimalno rešitev

**alpha-beta pruning** rezanje alfa-beta; omejevanje preiskovanja pri minimaxu

**ant colony optimization** – optimizacija s kolonijo mravov

**anytime planning** – neprestano načrtovanje; vrsta načrtovanja, kjer je potrebno neprestano prilagajanje načrtov spremenjenim okoliščinam; načrt se postopno izboljšuje s časom, a je vedno pripravljen

**artificial neural networks** – umetne nevronske mreže; glej *neural networks*

**association rules** – povezovalna pravila

**asynchronous backtracking** – asinhrono vračanje; algoritem za asinhrono distribuirano upoštevanje omejitev

**AUC** – Area Under the ROC Curve - ploščina pod krivuljo ROC; glej *ROC curve*

**background knowledge** – predznanje

**backpropagation (of error)** – vzvratno razširjanje (napake); algoritem za učenje večnivojskih usmerjenih nevronskeh mrež

**bagging** – kratica za "bootstrap aggregating"; princip generiranja zaporedja različnih učnih množic po metodi bootstrap, generiranje zaporedja hipotez iz tako dobljenih učnih množic in kombiniranje napovedi hipotez z glasovanjem; glej tudi *boosting*

**bag of words** – vreča besed; atributna predstavitev besedil

**Bayesian classifier** – Bayesov klasifikator; teoretični klasifikator z maksimalno klasifikacijsko točnostjo

**Bayesian belief network** – Bayesova (verjetnostna mreža); pristop k modeliranju podatkov (verjetnostne distribucije), kjer vozlišča predstavljajo dogodke, povezave pa odvisnosti med dogodki in kjer je vsakemu vozlišču prirejena tabela pogojnih verjetnosti

**beam search** – iskanje v snopu

**belief, desire, intention (BDI) architecture** – arhitektura BDI (verjetje, želja, namen) je sestavljena iz treh komponent; verjetje je izjava o okolju, za katero agent verjame, da je resnična; množica takih izjav opisuje agentov pogled na okolje; želja je ciljno stanje, ki ga želi agent doseči; namen je načrt, ki ga ima agent za dosego cilja

**best-first search** – metoda preiskovanja “najprej najboljši”; po tej metodi vedno nadaljujemo z najbolj perspektivno hipotezo

**best-only search** – metoda “samo najboljši”; glej *greedy search*

**bias** – pristranskost; napaka hipoteze je sestavljena iz pristranskosti in variance (glej *variance*; pristranskost je sistematična napaka učnega algoritma; pri učenju: pristranskost algoritma pri izbiri različnih alternativ

**biased** – pristranski; a priori daje prednost določenim alternativam

**blackboard architecture** – arhitektura skupne delovne površine omogoča sodelovanje agentov preko vsem dostopnih struktur, ki vsebujejo informacije o okolju in vmesne rezultate procesiranja

**boosting** – princip generiranja zaporedja različnih hipotez s pomočjo uteževanja učnih primerov, tako da so težji primeri bolj uteženi, in kombiniranje napovedi hipotez z uteženim glasovanjem; glej tudi *bagging*

**bootstrapping** – metoda razmnoževanja učnih primerov

**branch and bound** – metoda “razveji in omeji” ali omejeno izčrpno preiskovanje; metoda preiskovanja, kjer generiramo naslednike trenutne hipoteze (razvezimo), nato pa množico naslednikov omejimo samo na perspektivne hipoteze

**breadth-first search** – iskanje v širino

**building blocks hypothesis** – hipoteza zidakov; poskus razlage delovanja genetskih algoritmov, ki shemo oziroma zidak definira kot nizkonivojsko predstavitev, ki prispeva k nadpovprečni kvaliteti osebka; hipoteza trdi, da genetski pristopi implicitno in učinkovito identificirajo in sestavljajo zidake ter na ta način sestavljajo vse boljše rešitve iz delnih nizkonivojskih podrešitev

**classification** – klasifikacija, uvrščanje

**clustering** – razvrščanje (tudi: rojenje, grupiranje, gručenje, grozdenje)

**collaborative agents** – sodelujoči agenti; delujejo v velikih skupinah in sodelujejo pri doseganju cilja

**collaborative filtering** – skupinsko filtriranje; gre za izločanje koristnih vzorcev s sodelovanjem večih virov informacij (npr. ljudi, agentov, podatkovnih virov)

**confidence** – zaupanje; pri povezovalnih pravilih mera, ki oceni točnost pravila; glej tudi *support*

**confidence interval** – interval zaupanja

**confusion matrix** – matrika zmot

**constraint satisfaction problem** – problem upoštevanja omejitev; reševanje problema, kjer je potrebno upoštevati vrsto omejitev

**counterpropagation** – protipretočni algoritem za učenje večnivojskih nevronskih mrež

**covering algorithm** – prekrivni algoritem; algoritem, ki med učenjem iz učne množice odstrani učne primere, ki jih trenutna hipoteza pravilno razlaga (klasificira)

**cross-validation** – prečno preverjanje; glej *k-fold cross-validation* in *stratified cross-validation*

**crossover** – križanje; izraz se uporablja v genetskih algoritmih in označuje izmenjavo genetskega materiala med osebki

**curse of dimensionality** – prekletstvo dimenzionalnosti; problem prevelikega števila atributov pri danem učnem problemu

**data mining (DM)** – podatkovno rudarjenje, tudi odkrivanje zakonitosti (znanja) iz podatkov

**decision tree** – odločitveno drevo

**deduction** – dedukcija; sklepanje od splošnega h konkretnemu; glej tudi *induction*

**delta learning rule** – učno pravilo delta, pri katerem se uteži na sinapsah spreminjajo proporcionalno velikosti razlike med želenim in dejanskim izhodom nevrona

**density estimation** – ocenjevanje gostote verjetnosti

**depth-first search** – iskanje v globino

**document classification** – klasifikacija dokumentov v vnaprej določene kategorije

**document retrieval** – pridobivanje dokumentov; glede na določene kriterije želimo pridobiti relevantne dokumente

**document summarization** – povzemanje dokumentov iz enega ali več virov

**“don’t care” value** – poljubna vrednost; vrednost atributa lahko zavzame katerokoli vrednost iz zaloge vrednosti

**“don’t know” value** – neznana, manjkajoča vrednost (atributa)

**dynamic Bayesian network (DBN)** – dinamična Bayesova mreža; verjetnostni model, ki vsebuje časovne odvisnosti

**ensemble methods** – skupinske metode; metode strojnega učnja, ki namesto enega modela zgradijo zaporedje modelov; primeri skupinskih metod so *bagging*, *boosting* in naključni gozdovi

**error correcting output codes** – kode za popravljanje izhodnih napak razbijajo klasifikacijski problem na več podproblemov po principu maksimizacije razdalje med kodami razredov in se lahko uporablja za kodiranje razredov pri klasifikacijskih problemih z več kot tremi razredi

**exhaustive search** – izčrpano preiskovanje

**exploration or exploitation dilemma** – dilema med raziskovanjem in izkoriščanjem; ali naj agent izkoristi to kar že ve, da bi maksimiziral nagrado, ali naj raje razširi nova stanja, kjer je morebiti nagrada višja

**feature** – značilka; atribut

**feedforward neural networks** – usmerjene nevronske mreže

**game-tree complexity** – kompleksnost igralnega drevesa pri preiskovanju v igrah

**generalized delta learning rule** – pospoljeno učno pravilo delta; glej *backpropagation*

**genetic algorithms** – genetski algoritmi

**goal oriented agents** – ciljno usmerjeni agenti se na okolje ne le odzivajo, pač pa zasledujejo določen cilj

**greedy search** – požrešno iskanje; metoda preiskovanja, kjer izberemo vedno samo najbolj perspektivnega naslednika trenutne hipoteze in vse ostale zavrhemo

**heuristic search** – hevristično preiskovanje; glej *best-first search*

**hidden Markov model (HMM)** – skriti markovski model; statistični model z markovsko predpostavko, kjer je edino stanje modela skrito, viden pa je izhod, ki je odvisen od stanja

**impurity measure** – mera nečistoče

**incomplete data** – nepopolni podatki; podatki z manjkajočimi vrednostmi

- incremental learning** – inkrementalno učenje, kjer učenec sproti spreminja teorijo ob sprejemanju novih podatkov
- induction** – indukcija; sklepanje od konkretnega k splošnemu, kar je osnovni princip učenja; glej tudi *deduction*
- inductive logic programming (ILP)** – induktivno logično programiranje, tudi (več)relacijsko učenje
- information extraction** – pridobivanje informacij; pridobivanje relevantnih informacij iz dokumentov
- information gain** – informacijski prispevek; razlika med apriorno in pričakovano aposteriorno entropijo
- information-gathering agents** – informacijski agenti za uporabnika zbirajo, filtrirajo in klasificirajo različne informacije, največkrat na internetu
- information score** – informacijska vsebina odgovora; mera uspešnosti klasifikatorja, ki upošteva apriorne verjetnosti razredov
- interface agents** – vmesniški agenti so neke vrste osebni pomočniki, ki za uporabnika opravljajo različne naloge
- inverted file** – poseben indeks (slovar), ki vsebuje vse besede iz vseh dokumentov
- iterative deepening** – iterativno poglavljanje; pri iskanju postopno poglabljamamo globino preiskovanja
- iterative deepening  $A^*$**  – algoritem  $A^*$  z iterativnim poglavljanjem; podobno iterativnemu poglavljanju, le da poglabljamamo zgornjo mejo ocene kvalitete naslednikov in ne globine preiskovanja
- hill climbing** – glej greedy search
- kernel function** – jedrna funkcija; funkcija, ki definira vpliv enega primera na gostoto verjetnosti v prostoru
- $k$ -fold cross-validation** –  $k$ -kratno prečno preverjanje; učno množico razdelimo na  $k$  delov in za vsak del zgradimo hipotezo iz preostalih primerov in jo testiramo na izloženem delu primerov
- knowledge discovery in databases (KDD)** – odkrivanje znanja iz podatkov, glej tudi data mining
- $k$ -nearest neighbors** –  $k$ -najbližjih sosedov; algoritem, ki novemu primeru priredi razred, ki je najbolj pogost med  $k$  najbližjimi sosedmi v učni množici primerov
- lazy learning** – leno učenje; učenje, kjer učne primere preprosto shranimo in je glavnina procesiranja potrebna šele pri napovedovanju razreda novemu primeru; primer tega učenja je metoda  $k$ -najbližjih sosedov

**learning agents** – učeči se agenti se z učenjem prilagajajo okolju

**learning rate** – stopnja učenja; parameter, ki definira hitrost učenja

**leave-one-out** – metoda “izloči enega”; za vsak učni primer zgradimo hipotezo iz preostalih primerov in jo testiramo na izločenem primeru

**linearly separable** – linearno ločljivi; dva podprostora sta linearno ločljiva, če ju lahko ločimo s hiperravnino

**link farms** – farme povezav; velika množica spletnih mest, ki vsebujejo mnogo povezav in na ta način poskušajo umetno dvigniti pomembnost nekaterih spletnih mest, ter za to dobijo plačilo

**lookahead** – pogled naprej (pri preiskovanju)

**machine learning** – strojno učenje

**margin** – rob - ocena zanesljivosti klasifikacije; razlika med verjetnostjo najbolj verjetnega razreda in verjetnostjo drugega najbolj verjetnega razreda; pri metodi podpornih vektorjev je rob razdalja podpornih vektorjev od hiperravnine

**Markov decision process** – markovski odločitveni proces; vrsta spodbujevanega učenja, kjer velja markovska lastnost

**MDL** – glej *minimal description length principle*

**m-estimate** – m-ocena (verjetnosti)

**mean absolute error** – srednja absolutna napaka; mera za ocenjevanje uspešnosti napovedovanja vrednosti zvezne spremenljivke v regresijskih problemih

**mean squared error** – srednja kvadratna napaka; mera za ocenjevanje uspešnosti napovedovanja vrednosti zvezne spremenljivke v regresijskih problemih

**minimal description length principle** – princip najkrajšega opisa; teorija, ki minimizira dolžino opisa teorije in podatkov pri dani teoriji, je najboljša razlaga teh podatkov

**misclassification cost** – cena napačne klasifikacije

**missing value** – manjkajoča vrednost; glej *don't know value*

**mobile agents** – mobilni agenti se lahko premikajo iz kraja v kraj fizično ali virtualno

**MSE** – glej *mean squared error*

**multiagent system** – večagentni sistem je način, kako združimo moč več posameznih avtonomnih agentov, in omogoča porazdeljeno reševanje problemov

**multilayered neural networks** – večnivojske nevronske mreže

- mutual information** – medsebojna informacija; drugo ime za informacijski prispevek, ki je mera kvalitete atributa, hkrati pa mera skupne informacije dveh atributov
- naive Bayesian classifier** – naivni Bayesov klasifikator; Bayesov klasifikator, ki predpostavlja pogojno neodvisnost atributov pri danem razredu
- neural net(work)s** – nevronske mreže; simulacija ali implementacija algoritmov za učenje, ki oponašajo učenje bioloških nevronskeih mrež
- no free lunch theorem** – teorem zastonjskega kosila, ki trdi, da sta poljubna dva učna algoritma ekvivalentna, če njuno uspešnost povprečimo čez vse možne probleme
- noise** – šum; napake v podatkih
- noisy data** – šumni podatki; podatki z napakami
- Occam razor principle** – princip Ockhamove britve, ki pravi, da je najpreprostejša razlaga najbolj zanesljiva
- odds** – razmerje verjetnosti (tudi: upanje); verjetnost dogodka deljena z ena minus verjetnost dogodka
- outlier** – osamelec; podatek, ki odstopa od ostalih
- overfitting** – preveliko prileganje; mnogi učni algoritmi vsebujejo mehanizme za preprečevanje prevelikega prileganja podatkom
- parse tree** – drevo izpeljav; večinoma rezultat sintaktične analize besedila predstavimo kot drevo izpeljav
- parsing** – sintaktična analiza teksta; prva faza pri razumevanju naravnega jezika
- particle swarm optimization** – optimizacija z rojem delcev
- plausibility** – možnost; dvojiški logaritem razmerja verjetnosti
- precision** – preciznost; mera za ocenjevanje klasifikatorjev (na področju iskanja pomembnih dokumentov ocenjuje odstotek pomembnih dokumentov glede na vse dokumente, ki so bili označeni kot pomembni) (glej tudi *recall*)
- principle of multiple explanations** – princip večkratne razlage pravi, da je treba obdržati vse hipoteze, ki so konsistentne z vhodnimi podatki
- prioritized sweeping** – algoritem za spodbujevano učenje s prioritetnim osveževanjem
- POS tagging** (part-of-speech tagging) – označevanje besed v besedilu z besednimi vrstami, npr. samostalnik, glagol, pridevnik, veznik, ...
- postpruning** – naknadno rezanje odločitvenih in regresijskih dreves
- pruning** – rezanje odločitvenih in regresijskih dreves

**recursive best first search (RBFS)** – rekurzivno iskanje najprej najboljši

**reactive agents** – odzivni agenti se na okolje odzivajo glede na vnaprej definirana pravila

**recall** – priklic; mera za ocenjevanje klasifikatorjev, definirana enako kot senzitivnost (na področju iskanja pomembnih dokumentov ocenjuje odstotek pomembnih odkritih dokumentov glede na vse pomembne dokumente)(glej tudi *precision*)

**refinement graph** – ostritveni graf; graf hipotez, kjer naslednike trenutno obravnavane hipoteze (vozlišča v grafu) generira ostrilni operator

**refinement operator** – ostrilni operator; operator ostritve

**regression tree** – regresijsko drevo

**regret** – razlika do optimalne strategije pri spodbujevanem učenju

**reinforcement learning** – spodbujevano učenje; skupina računskih metod, kjer mora agent sam odkriti, katere akcije naj izvede v okolju, da bo dobil skupno kar največjo nagrado

**ROC curve** – krivulja ROC podaja razmerje med senzitivnostjo in specifičnostjo; glej tudi *sensitivity in specificity*

**scatterplot** – razsevni diagram

**search tree** – iskalno drevo

**semantic web** – semantični splet, ki bo za najrazličnejše vire prinesel uporabo metapodatkov ter jih tako naredil bolj dosegljive

**sensitivity** – senzitivnost; mera za ocenjevanje klasifikatorjev, ki ocenjuje odstotek pravilno klasificiranih pozitivnih primerov (glej tudi *specificity*)

**simple Bayesian classifier** – glej *naive Bayesian classifier*

**simulated annealing** – simulirano ohlajanje; metoda preiskovanja, kjer naključno generira nega naslednika sprejmemo z verjetnostjo, ki je sorazmerna njegovi ocjenjeni kvaliteti

**spam** – slama; nekoristna elektronska sporočila

**specificity** – specifičnost; mera za ocenjevanje klasifikatorjev, ki ocenjuje odstotek pravilno klasificiranih negativnih primerov (glej tudi *sensitivity*)

**standard deviation** – standardno odstopanje

**standard error** – standardna napaka

**stop words** – besede, ki jih ignoriramo pri analizi besedila

**stratified cross-validation** – sorazmerno prečno preverjanje; prečno preverjanje, kjer je ohranjena enaka ali vsaj podobna distribucija razredov v vsaki podmnožici primerov

**subsumption architecture** – arhitektura vsebovanosti je večnivojska robotska arhitektura, namenjena primitivnim robotom, ki ne predvideva centralne inteligence, pač pa poleg vhoda in izhoda vsebuje več plasti, kjer vsaka zasleduje svoje cilje oz. nadzira določeno agentovo obnašanje

**support** – podpora; pri povezovalnih pravilih mera, ki oceni število primerov, ki izpolnjujejo pogojni del pravila; glej tudi *confidence*

**template** – shema

**text mining** – tekstovno rudarjenje; pridobivanje novega znanja iz teksta

**tokenization** – leksikalna analiza besedila oziroma določitev mej med besedami

**traveling salesman problem (TSP)** – problem trgovskega potnika

**transduction** – transdukcija; sklepanje od konkretnega h konkretnemu; glej tudi *deduction* in *induction*

**tree augmented naive Bayes (TAN)** – drevesno razširjen naivni Bayes; Bayesova verjetnostna mreža, kjer je atributni podgraf drevo

**unbiased** – nepristranski

**utility based agents** uporabnostni agenti poleg doseganja ciljev maksimirajo še druge kriterije, vsebovane v funkciji uporabnosti

**validation set** – nastavitevna množica; množica primerov, ki se ne uporabi za generiranje teorije, ampak za čim optimalnejšo nastavitev parametrov algoritma

**variance** – varianca; napaka hipoteze je sestavljena iz pristrankosti in variance (glej *bias*); varianca je napaka zaradi vhodnih podatkov

**weight of evidence** – teža evidence

**wrapper approach** – metoda notranje optimizacije; algoritem strojnega učenja uporablja za optimizacijo notranje prečno preverjanje, t.j. prečno preverjanje brez uporabe testnih primerov; učne primere razdelimo na začasne učne in začasne testne primere, zato da ocenimo optimizacijski kriterij; postopek ponavljamo z različnimi nastavtvami parametrov učenja, dokler ne najdemo (lokalnega) optimuma