Strojno učenje

Zapiski pri predmetu

Kazalo

Predgovor

Pred vami so zapiski pri predmetu Strojno učenje na magistrskem študiju univerzitetnega študijskega programa Računalništvo in informatika na Fakulteti za računalništvo in informatiko Univerze v Ljubljani.

V trenutni različici so zapiski povzetek profesorjevih prosojnic in predavanj v skoraj izključno besedilni obliki. Kot vizualno podporo branju lahko zaenkrat uporabite profesorjeve prosojnice. Če želite zapiske dopolniti s slikami in primeri ali popraviti kakšno morebitno napako, lahko to brez zadržkov storite.

Na koncu se nahaja tudi zbirka (delno) rešenih starih izpitov, ki vam je lahko pomoč pri pripravah na izpit.

Opomniti velja, da je uporaba zapiskov na lastno odgovornost, saj lahko vsebujejo napake in ni nujno, da pokrivajo vso vsebino predmeta.

Na koncu hvala vsem, ki ste s popravki in dopolnitvami prispevali h kakovostnejšim zapiskom.

Poglavje 1

Uvod

Herbert A. Simon je dejal: "Učenje pomeni adaptivne spremembe v sistemu, ki mu omogočajo bolj učinkovito opravljanje iste naloge."

Inteligenca je sposobnost prilagajanja okolju in reševanja problemov. Znanje je lahko rezultat učenja ali pa je dano vnaprej.

Umetna inteligenca je veja računalništva za razvoj sistemov, ki se obnašajo inteligentno in ki so sposobni reševati relativno težke probleme. Sem med drugim uvrščamo strojno učenje, podatkovno rudarjenje, predstavitev znanja, hevristično preiskovanje in reševanje problemov, genetske algoritme, robote, računalniško zaznavanje, obdelava jezika, igranje iger, razvoj priporočilnih sistemov, planiranje ...

Med umetno inteligenco uvrščamo sisteme, ki se vedejo ali razmišljajo kot človek. Sam cilj umetne inteligence je razumeti in zgraditi inteligentne sisteme na osnovi razumevanja človeškega razmišljanja, sklepanja, učenja in komuniciranja.

Ali je sistem inteligenten, lahko preverimo s Turingovim testom. V tem testu izpraševalec postavlja vprašanja, vendar ne ve, ali je na drugi strani človek ali računalnik. Računalnik opravi test, če izpraševalec po odgovorih ne more ugotoviti, ali je na drugi strani človek ali računalnik. Ena od največjih slabosti testa je, da je subjektiven in ga ni mogoče ponoviti ali matematično analizirati.

Spomin je ustvarjanje novih povezav z drugimi nevronim spreminjanje moči povezav na sinapsah, proizvodnja proteinov ... Človek si zapomni čisto vse, kar se mu pripeti v življenju, problematično je le naslavljanje spomina.

Nekateri ljudje imajo izjemen spomin. Solomon Šereševski, znan tudi kot S., je bil novinar in mnemonik, ki je bil znan po svojem neverjetnem spominu. Neka anekdota pravi, da je sredi 20-ih let 20. stoletja sodeloval

na sestanku, kjer si ni smel zapisati ničesar, po koncu sestanka pa je do besede natančno ponovil celoten govor.

V prihodnjih letih je sodeloval v več raziskavah, v katerih si je uspešno zapomnil matematične enačbe, velike matrike in pesmi v tujih jezikih, ki jih spoh ni govoril, v nekaj minutah.

1.1 Uvod v strojno učenje

Najprej je potrebno razumeti naš problem, kakšne so metodologije, cilji in uspeh, ki se glede na zahtevnost problema razlikuje. Podatke moramo razumeti, ugotoviti, kdo jih je zbiral, kako so sestavljeni, identificirati morebitne izjeme, napake, šume, pomanjkljivosti (npr. manjkajo kje kakšni atributi) ... Zatem moramo podatke ovrednotiti, poenotiti, očistiti in filtrirati (izločiti kakšne očitne napake) in transformirati, torej jih pretvoriti v obliko za uspešno strojno učenje. Šele zatem pridemo do strojnega učenja, kjer moramo izbrati ustrezne metode glede na naše podatke, recimo primerne za obdelavo velike količine podatkov, če je naših podatkov veliko, in zgraditi ter ovrednotiti modele in po potrebi postopek ponoviti, na primer ponovno zbrati podatke, če so se uporabljeni izkazali za slabe. Ko smo z modelom zadovoljni, moramo rezultate ovrednotiti, torej glede na različne kriterije določiti, kateri modeli so učinkoviti in če so praktično uporabni. Na koncu določimo, če smo z doseženim zadovoljni, sicer se lahko vrnemo na katerega od prejšnjih korakov, kdo bo uporabljal rezultate, ali je model možno prenesti na nove podatke ...

1.2 Napovedovanje

Napovedovanje je proces, kjer napovedujemo bodisi razred bodisi neko število. Če napovedujemo enega od možnih razredov, gre za klasifikacijo, pri kateri uporabljamo diskretno funkcijo. Če napovedujemo število, pa gre za regresijo.

Cilj nadzorovanega učenja je glede na podatke (opise) rešenih problemov zgraditi model, ki poskuša razložiti te podatke. Če modelu posredujemo nove probleme, ta glede na podatke, na katerih je bil zgrajen, vrne rešitev.

Pri nadzorovanem učenju najpogosteje učne primere predstavimo z atributi. Imamo torej množico učnih primerov $(x_1, y_1), ..., (x_n, y_n)$, kjer so x_i atributi, y_i pa vrednost neznane funkcije $y = f(x)$. Naloga je najti funkcijo

h, ki je čim boljši približek funkciji f. Atributi ali značilke x_i so neodvisne, y_i je ciljna ali odvisna spremenljivka, funkcijo h pa imenujemo model.

Pri gradnji modela se praviloma držimo principa preprostosti: problem poskušamo modelirati na čim preprostejši način.

Preprosta razlaga lahko sicer pogosto tudi zgreši precej primerov, zato se je pred izbiro optimalnega modela potrebno posvetovati z avtorjem podatkov, ali gre v teh primerih morda za napako.

Prostor hipotez se zelo hitro širi. Recimo, da imamo n binarnih atributov. Iz tega sledi, da imamo lahko vsega skupaj 2^n različnih učnih primerov in 2^{2^n} hipotez.

1.3 Metode strojnega učenja

1.3.1 Odločitvena in regresijska drevesa

Odločitvena in regresijska drevesa so drevesa, v notranjih vozliščih katerih se nahajajo atributi, veje iz vozlišč ustrezajo vrednostim atributov, listi pa odločitvam. Pri odločitvenih drevesih listi torej predstavljajo razrede, pri regresijskih pa točkovne napovedi.

1.3.2 k-najbližjih sosedov in lokalno utežena regresija

Ideja k-najbližjih sosedov je, da učne primere predstavi v prostoru glede na atribute, za nov primer pa izračuna razdalje do k najbližjih točk in na podlagi tega določi, kateremu razredu pripada.

Če je k nizek, je metoda precej občutljiva na šum v podatkih. Z večanjem števila k se lahko temu izognemo, vendar pa pri preveliki vrednosti metoda postane nenatančna.

Lokalno utežena regresija deluje na podoben način kot k-najbližjih sosedov, le da za nov primer poiščemo k najbližjih točk in kot napoved upoštevamo povprečje njihovih ciljnih vrednosti.

1.3.3 Naivni Bayes

Naivni Bayes predpostavi, da so atributi pri danem razredu med seboj neodvisni. Metoda deluje tako, da izračunamo verjetnost, da nek primer pripada določenemu razredu pri danih vrednostih atributov, in ga klasificiramo v

tistega z najvišjo verjetnostjo. Ker predpostavljamo, da so atributi neodvisni, verjetnost preprosto izračunamo kot produkt neodvisnih prispevkov posameznih atributov. Problem metode je v tem, da je verjetnosti potrebno zanesljivo oceniti in nimamo dovolj učnih primerov, da bi pokrili celoten prostor.

1.3.4 Linearna regresija

Pri linearni regresiji so učni primeri predstavljeni kot točke v prostoru. Učne primere nato poskušamo modelirati z linarno funkcijo, ki se čim bolje prilega točkam.

Uporabimo lahko seveda tudi bolj zapleteno funkcijo (npr. kvadratno, logaritemsko, ...), ki se lahko bolje prilagodijo točkam v prostoru, vendar moramo pri tem paziti, da se učnim podatkom ne prilagodimo preveč.

1.3.5 Metoda podpornih vektorjev

Ideja je, da originalni prostor z učnimi primeri transformiramo v večdimenzionalni prostor, za katerega predpostavimo, da je linearno rešljiv. V večdimenzionalnem prostoru nato učne primere ločimo s hiperravnino, ki maksimizira razdaljo med primeri enega in drugega razreda in hkrati minimizirati napako. Učne primere, ki so najbližje tistim iz drugaga razreda, imenujemo podporni vektorji.

Pri regresiji se funkcija, ki opisuje podatke, linearizira, metoda pa nato deluje na enak način kot pri klasifikaciji.

1.3.6 Naključni gozdovi

Naključni gozd je množica odločitvenih ali regresijskih dreves, v katerem vsako drevo glasuje za neko rešitev. Vsako drevo je zgrajeno na nekoliko drugačni učni množici, s čimer dobimo veliko varianco zgrajenih dreves. Končna rešitev je tista, za katero glasuje največ dreves.

1.3.7 Umetne nevronske mreže

Ideja nevronskih mrež je modelirati poenostavljen proces v možganih. Model, torej nevron, zna zgolj sešteti vhode oziroma signale od sosednjih nevronov, poslati skozi pragovno funkcijo, normalizirati in poslati naprej. Bistvo modela so uteži na povezavah med nevroni, ki določajo, kakšno funkcijo računa posamezen nevron oziroma kakšno funkcijo računa celotna mreža, ki

na vhodu dobi podatek in na koncu vrne rezultat. Naloga metode je torej določiti uteži na povezavah med nevroni.

Poglavje 2

Osnovni principi strojnega učenja

Učenje je opisovanje oziroma modeliranje podatkov. Uporabljamo dve vrsti algoritmov: učni (proces učenja modela) in izvajalni (izvajanje modela na novih podatkih). Učni podatki so opisi problemov in njihovih rešitev, novi podatki pa opisi novih, še nerešenih problemov. Vsako učenje temelji na nekem predznanju, ki je lahko prostor možnih modelov, kriterij optimalnosti, začetna hipoteza, množica hevristik ...

Hipoteze vrednotimo na podlagi več kriterijev:

- maksimizirati napovedno točnost hipoteze,
- minimizirati velikost hipoteze,
- maksimizirati prileganje hipoteze vhodnim podatkom,
- maksimizirati razumljivost hipoteze,
- minimizirati časovno zahtevnost napovedovanja,
- minimizirati število parametrov, potrebnih za napovedovanje (pridobitev nekaterih atributov je, recimo, lahko drago, npr. določene znanstvene meritve),
- **maksimizirati verjetnost hipoteze**.

Strojno učenje lahko definiramo kot optimizacijo, pri kateri imamo podan prostor možnih rešitev in kriterijsko funkcijo. Naloga strojnega učenja je poiskati rešitev, ki optimizira to kriterijsko funkcijo. V praksi se pogosto zadovoljimo z nekimi suboptimalnimi rešitvami, saj so prostori možnih rešitev preveliki.

2.1 Princip najkrajšega opisa

Princip najkrajšega opisa (angl. Minimum Description Length (MDL)) izvira iz principa Ochamove britve. Pravi, da je najpreprostejša razlaga, ki čim bolj ustreza vhodnim podatkom in predznanju, najbolj zanesljiva oz. verjetna.

Velikost in razumljivost hipoteze, časovna zahtevnost napovedovanja in podobno so le nekateri kriteriji kakovosti hipoteze, ki so pogojeni s principom najkrajšega opisa.

MDL je definiran s tem, da želimo maksimizirati verjetnost hipoteze pri danih podatkih in predznanju. Naj bo H ena od hipotez v prostoru vseh možnih hipotez $\mathcal{H}$, B predznanje, E vhodni podatki in $P(H|B)$ apriorna verjetnost informacije.

Apriorna količina informacije hipoteze H predstavlja število bitov za predstavitev hipoteze z znanim predznanjem in je definirana kot:

$$I(H|B) = -\log_2 P(H|B)$$

Aposteriorna količina informacije hipoteze H predstavlja število bitov za predstavitev hipoteze z znanim predznanjem in podatki in je definirana kot:

$$I(H|E,B) = -\log_2 P(H|E,B)$$

Optimalna hipoteza je definirana kot:

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

Po Bayesovem teoremu velja:

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

Ker je $I(E|B)$ konstanta, neodvisna od hipoteze, dobimo:

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

Iz tega izvemo, da je optimalna hipoteza tista, ki je točna (ima majhno napako $I(E|H,B)$) in preprosta (majhen $I(H|B)$).

2.2 Pristranskost in varianca

Napako, ki jo naredi hipoteza, lahko razdelimo na pristranskost in varianco.

Pristranskost izhaja iz samega učnega algoritma oziroma predstavitve modelov in se ji ne moremo izogniti (gre za sistematično napako, ki jo povzroča

algoritem). Če je t neznana vrednost, ki jo želimo napovedati, t' vrednost, ki jo dejansko napovemo, in $E[t']$ povprečje napovedanih vrednosti preko vseh učnih množic, je pristranskost definirana kot

$$Bias(t) = E[t'] - t$$

Visoka pristranskost pomeni veliko napako na učnih in testnih podatkih. Želimo imeti model, ki ima nizko pristranskost, saj se s tem izognemo premajhnemu prileganju (angl. underfitting).

Varianca je v nasprotju od pristranskosti odvisna od učne množice in nam torej pove, koliko je algoritem občutljiv na učno množico. Če je t' vrednost, ki jo dejansko napovemo, in $E[t']$ povprečje napovedanih vrednosti preko vseh učnih množic, je varianca definirana kot

$$Variance(t') = E[(E[t'] - t')^2]$$

. Varianca nam torej pove spremenljivost predikcij modela pri različnih učnih množicah - odvisnost od učne množice.

V praksi je potrebno najti optimalno razmerje med pristranskostjo in varianco. Če želimo minimizirati varianco, moramo poenostaviti hipotezo, in sicer tako, da zmanjšamo število parametrov, vendar s tem hkrati povečujemo pristranskost. Če želimo minimizirati pristranskost, pa moramo povečati kompleksnost hipoteze, kar naredimo tako, da povečamo število parametrov, vendar lahko pri tem pride do prevelikega prileganja učni množici, zato bo model zelo odvisen od nje. Iz tega sklepamo, da je torej potrebno najti optimalno število parametrov.

2.3 Inkrementalno učenje

Inkrementalno učenje govori o tem, da se teorija po vsakem novem učnem primeru spreminja. To lahko naredimo tako, da zgradimo nov model, ali pa tako, da posodobimo obstoječega.

2.4 Princip večkratne razlage

Če je z opažanji konsistentna več kot ena teorija, obdrži vse teorije.

- Epikur

Princip večkratne razlage (ansambelski pristop) govori o tem, da naj bi

obdržali vse konsistentne hipoteze z vhodnimi podatki. Na tak način lahko zmanjšamo varianco, saj je lahko vsak model zgrajen na drugačnih atributih. Pri klasifikaciji tako izberemo razred, ki ga je napovedalo največ modelov, pri regresiji pa upoštevamo povprečje napovedi. Slabost tega principa je, da ne razumemo več, kako zadeva deluje.

Principa večkratne razlage in MDL sta si komplementarna. S principom MDL iščemo (v povprečju) najboljše hipoteze (najbolj preproste in verjetne), kar pomeni, da lahko s tem algoritmom usmerjamo učni algoritem. Princip večkratne razlage pa kombinira več verjetnih hipotez in pri katerem gre za usmerjanje izvajalnega algoritma (uporabimo vse modele za napovedovanje novega primera).

Pomembna razlika v omenjenih principih je tudi v smislu razumevanja odločitev. Odločitve modelov zgrajenih po principu MDL lažje razumemo, kot odločitve modelov zgrajenih po principu večkratne razlage.

2.5 Ocenjevanje verjetnosti

Učenje se dogaja iz podatkov, iz katerih ocenjujemo verjetnosti določenega zaključka, kar pomeni, da moramo iz podatkov dobiti neke aproksimacije verjetnosti. Več kot imamo podatkov na voljo, boljšo aproksimacijo verjetnosti lahko dosežemo. Seveda pa morajo biti podatki dovolj zanesljivi.

Na začetku moramo imeti neko apriorno verjetnost, ki jo bo vsak naslednji vhodni podatek spremenil. Apriorna verjetnost je definirana po beta porazdelitvi, ki ima dva parametra a in b, ki jih lahko interpretiramo (še predno vidimo kakršenkoli podatek) kot a uspešnih in b neuspešnih poiskusov (met kovanca - ne da izvedemo met kovanca lahko za 100 metov predvidimo, da bosta a in b dokaj podobna, okrog 50).

Beta porazdelitev nam poda naslednjo gostoto verjetnosti:

$$Beta(\alpha, \beta) : \ prob(x|\alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)} \tag{2.1}$$

kjer je B beta funkcija

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}dt$$

Matematično upanje in varianca beta porazdelitve:

$$E(x) = \mu = \frac{\alpha}{\alpha + \beta} \tag{2.2}$$

$$Variance(x) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \tag{2.3}$$

Za r uspešnih in n vseh primerov ocenimo verjetnost uspeha po naslednji enačbi ($\beta(a+r, b+n-r)$):

$$p = \frac{r+a}{n+a+b} \tag{2.4}$$

Z začetno porazdelitvijo $\beta(0,0)$ dobimo relativno frekvenco: $p = \frac{r}{n}$, z $\beta(1,1)$ pa dobimo Laplaceov zakon zaporednosti. Pri tem velja za k možnih izidov: $0 < p = \frac{r+1}{n+k} < 1$.

Po tem principu dobimo m-oceno verjetnosti. Pri tem velja, da je $m = a + b$ (teža apriornega znanja) in $p_0 = \frac{a}{a+b}$. Tako iz prejšnje enačbe dobimo:

$$p = \frac{r+mp_0}{n+m} = \frac{n}{n+m} \cdot \frac{r}{n} + \frac{m}{n+m} \cdot p_0 \tag{2.5}$$

Za $m = k$ in $p_0 = 1/k$ dobimo Laplacov zakon zaporednosti. M-oceno uporabljamo, kadar imamo majhno število primerov (n).

2.6 Bootstrapping

Stremljenje (angl. bootstrapping) je postopek, ki nam omogoča ocenjevanje zanesljivosti ocen, ki jih dobimo iz učne množice.

Denimo, da imamo veliko vrečo modrih in rumenih kroglic in da iz nje naključno izvlečemo 100 kroglic, od tega 70 modrih in 30 rumenih. Iz tega sklepamo, da je delež modrih kroglic v vreči 0.7. Zanima nas, kako dober je ta približek?

Postopek ponovimo, le da tokrat kroglice vračamo, s tem pa dobimo nove učne primere. Ko torej izvlečemo kroglico, pogledamo, kakšne barve je, in jo vrnemo nazaj v vrečo. Poskus lahko večkrat ponovimo in na podlagi več izračunanih deležev modrih kroglic sestavimo histogram ter na tak način z nekim intervalom zaupanja ocenimo zanesljivost naše prvotne ocene.

Poglavje 3

Preiskovalni algoritmi

V strojnem učenju je potrebno rešiti mnoge probleme. Že samo učenje je iskanje hipoteze, soočamo pa se tudi z izbiro najboljše podmnožice atributov, diskretizacijo atributov (določanje števila intervalov in meje med njimi), binarizacijo atributov, konstruktivno indukcijo, nastavljanjem vrednosti parametrov (kako nastaviti parametre, da bodo rezultati čim boljši) ...

3.1 Izčrpno in delno izčrpno preiskovanje

Izčrpni preiskovalni algoritmi preiščejo ves prostor in zagotavljajo, da bodo v preiskanem prostoru našli najboljšo možno rešitev. Pri delnem izčrpnem preiskovanju prostor preiskovanja z določenimi pogoji lahko deloma omejimo.

3.1.1 Iskanje v širino

Iskanje v širino deluje tako, da drevo stanj razvijamo po nivojih. Njegova prednost je, da zagotovo najde najkrajšo pot do rešitve, slabost pa, da prostor in čas z globino eksponentno rasteta.

3.1.2 Iskanje v globino

Iskanje v globino deluje tako, da drevo stanj rekurzivno razvijamo po vozliščih. V nasprotju z iskanjem v širino prostor raste linearno. Slaba stran algoritma je, da lahko zgreši rešitev, ki je zelo blizu začetku, ali pa se zacikla v neskončni veji.

3.1.3 Iterativno poglabljanje

Iterativno poglabljanje je kombinacija iskanja v širino in globino, kjer maksimalno globino sproti povečujemo. Njegova prednost je, da zagotovo najde najkrajšo pot do rešitve in z globino prostorska zahtevnost raste linearno, slabost pa, da v vsaki iteraciji ponovno preiščemo ves že preiskan prostor, zaradi česar čas raste eksponentno.

3.1.4 Omejeno izčrpno iskanje

Pri omejenem izčrpnem iskanju preiščemo samo stanja, za katera ocenimo, da so perspektivna. S tem lahko sicer zavržemo prostor, v katerem se nahaja optimalno stanje, bomo pa do rešitve, čeprav morda ne optimalne, prišli hitreje kot sicer.

3.1.5 Najprej najboljši

Ideja algoritma najprej najboljši (angl. best-first search) je, da razvijemo naslednike nekega vozlišča, jih ocenimo in iskanje nadaljujemo v najbolje ocenjenem izmed vseh do zdaj ocenjenih. Če algoritem kombiniramo z omejenim izčrpnim iskanjem, s čimer določen prostor stanj zavržemo, je eden izmed najbolj učinkovitih. Slaba stran algoritma je, da prostorska zahtevnost eksponentno raste, saj moramo imeti shranjena vsa še nerazvita potencialna vozlišča.

3.2 Lokalno preiskovanje

Lokalni preiskovalni algoritmi preiščejo le lokalni del prostora, pri čemer se zanašajo na lokalno informacijo. Poznamo tri lokalne ekstreme: lokalni in globalni maksimumi, plato in greben. Pri teh algoritmih je koristen pogled naprej (angl. lookahead), kar pomeni, da ne upoštevamo zgolj trenutnega stanja, temveč tudi naslednike stanj, s čimer lahko zmanjšamo kratkovidnost in tako lažje rešimo dan problem, vendar je potrebno upoštevati, da je precej časovno potraten.

3.2.1 Požrešno iskanje

Ideja požrešnega iskanja je, da v nekem stanju razvijemo naslednike, jih ocenimo in zavržemo vse, razen najbolje ocenjenega. Algoritem je zato hiter, saj se s tem, da gremo zgolj po eni poti, naš eksponentni prostor spremeni v

linearnega. Vendar pa rešitev zaradi kratkovidnosti algoritma ni optimalna, temveč je le približna, je pa v številnih praktičnih primerih zadovoljiva.

3.2.2 Gradientno iskanje

Pri gradientnem iskanju v nekem stanju izračunamo gradient funkcije, ki nam pove, kam se moramo premakniti, da najdemo boljšo rešitev. Težava algoritma je, da ne vemo, kako velik korak moramo narediti, da pridemo do boljše rešitve. Ker gre v praksi za posplošitev požrešnega iskanja v zveznem prostoru, je tudi ta algoritem kratkoviden.

3.2.3 Iskanje v snopu

Iskanje v snopu je podobno požrešnemu iskanju, le da tukaj obdržimo več naslednikov na določenem nivoju in ne zgolj enega. Koliko naslednikov obdržimo, je odvisno od velikosti snopa.

3.2.4 Lokalna optimizacija

Lokalna optimizacija je posplošitev požrešnega algoritma. Njena ideja je v tem, da večkrat ponovimo požrešno iskanje iz naključnega začetnega stanja. Zadevo ponavljamo, dokler ne poteče dodeljen čas za izvajanje algoritma.

3.3 Stohastično preiskovanje

Stohastični preiskovalni algoritmi se premikajo po prostoru glede na verjetnosti.

3.3.1 Simulirano ohlajanje

Ideja algoritma simulirano ohlajanje (angl. simulated annealing) je, da v iskanje vključimo naključje. Naključje bi lahko simulirali z žogico, ki se kotali po hribu navzdol in pristane v enem izmed lokalnih minimumov, zadevo pa nato pretresemo in če imamo srečo, bo žogica padla v še nižji lokalni minimum.

Simulirano ohlajanje izhaja iz fizikalnih lastnosti v metalurgiji. Višja je temperatura jekla, bolj gibljive so molekule v njem, med ohlajanjem pa se jeklo strjuje in gibanje molekul v njem se umirja.

3.3.2 Genetski algoritmi

Genetski algoritmi se zgledujejo po evolucijski teoriji: parjenju in boju za preživetje. Dobri organizmi skozi čas torej preživijo, slabi pa odmrejo. Pri tem tako velja, da imajo najboljši organizmi največ naslednikov.

Genetski algoritmi so sposobni preiskati zelo velik prostor, njihovo bistvo je izbira pravega kodiranja.

Problem	**Strategija**
Gradnja odločitvenih dreves	Požrešno/pogled naprej/genetski alg.
Binarizacija diskretnih atributov	Požrešno/izčrpno
Delno naivni Bayes	Omejeno izčrpno
Učenje odločitvenih pravil	Požrešno/lokalna optimizacija/genetski alg.
Nevronske mreže	Gradientno iskanje/simulirano ohlajanje
Povezovalna pravila	Omejeno izčrpno
Iskanje podmnožice atributov	Požrešno/lokalna optimizacija/genetski alg.

Tabela 3.1: Primeri uporabe preiskovalnih algoritmov v strojnem učenju

Poglavje 4

Ocenjevanje atributov

4.1 Mera nečistoče

Pri nekem klasifikacijskem problemu imamo podano neko množico razredov. Mera nečistoče je potem definirana kot verjetnostna porazdelitev po razredih.

Maksimum mere nečistoče dosežemo, ko so vsi razredi enako verjetni, minimum pa, ko je verjetnost enega izmed razredov 1, ostalih pa 0. Funkcija je simetrična, konkavna in zvezna ter zvezno odvedljiva.

Kakovost atributa potem definiramo tako, da od apriorne nečistoče odštejemo povprečje nečistoč vseh možnih vrednosti tega atributa. Kakovost atributa, definirana na tak način, je vedno nenegativna. Njen maksimum dosežemo, če imamo za vsako možno vrednost atributa čisto množico, saj bodo tedaj te nečistoče enake 0, kakovost atributa pa bo enaka kar apriorni nečistoči. Slabost mere je, da precenjuje večvrednostne atribute, čemur se lahko izognemo z normalizacijo ali pa jo uporabljamo na atributih, ki imajo enako število vrednosti, kar lahko dosežemo z binarizacijo.

4.1.1 Entropija

Količina prejete informacije je definirana kot:

$$I(X_i) = -\log_2 P(X_i)$$

Entropija je mera nečistoče, ki je definirana kot povprečna pričakovana količina informacije:

$$H(X) = -\sum_i P(X_i) \log_2 P(X_i)$$

Pogojno entropijo definiramo kot razliko med skupno entropijo in entropijo atributa.

4.1.2 Informacijski prispevek

Iz entropije lahko izpeljemo informacijski prispevek, ki je definiran kot razlika med apriorno in pogojno entropijo. Informacijski prispevek je nenegativen in ima maksimum pri apriorni entropiji, kar se zgodi, če atribut povsem ločuje razrede med seboj. Njegova slabost je, da precenjuje večvrednostne atribute.

4.1.3 Razmerje informacijskega prispevka

Z namenom eliminacije te slabosti je nastalo razmerje informacijskega prispevka, ki ga dobimo tako, da informacijski prispevek preprosto delimo z entropijo atributa. Sedaj algoritem sicer ne precenjuje več večvrednostnih atributov, precenjuje pa atribute z majhno entropijo. Temu so se v praksi izognili tako, da so razmerje informacijskega prispevka zato izračunali zgolj za atribute, ki so bili nadpovprečni.

4.1.4 Mera razdalje

Druga rešitev slabosti informacijskega prispevka je normalizacija informacijskega prispevka s skupno entropijo razreda in atributov. Če to normalizacijo odštejemo od 1, dobimo mero razdalje. Mera razdalje je vselej nenegativna, enaka 0, ko sta atribut in razred identična, zanjo pa veljata tudi simetričnost razreda in atributa ter trikotniška neenakost med razredom in dvema različnima atributoma.

4.1.5 Teža evidence

Teža evidence temelji na alternativni definiciji količini informacije, imenovani tudi *odds*, ki je definirana kot razmerje verjetnosti dogodka in verjetnosti negacije istega dogodka.

Teža evidence je definirana kot logaritem razmerja med aposteriornim oziroma pogojnim *odds*, ko poznamo vrednost nekega pogoja, in apriornim *odds*. Zanjo velja, da je simetrična in definirana na celotnem intervalu $[-\infty, \infty]$.

Pri ocenjevanju kakovosti atributa uporabljamo povprečno absolutno težo evidence, ki jo definiramo tako, da vzamemo absolutno vrednost povprečja tež evidence pri vseh možnih vrednostih atributa. Absolutno vrednost potrebujemo zato, ker je teža evidence lahko manjša od 0. Uporablja se tudi pri naivnem Bayesu.

4.1.6 MDL

MDL sledi principu najkrajšega opisa, ki pravi, da mora biti hipoteza čim boljša in čim preprostejša. Sledimo temu, da so zakodirana hipoteza in zakodirani podatki pri dani hipotezi skupaj manjši od izvornih podatkov. Velja, da je hipoteza boljša, če je čim bolj kompresivna.

Enak princip lahko uporabimo tudi pri ocenjevanju atributov. Apriorna entropija nam pove povprečno količino informacije, potrebne za kodiranje razredov. Pogojna entropija nam pove, koliko informacije v povprečju potrebujemo za kodiranje razreda pri določeni vrednosti atributa.

Mera MDL z optimalnim kodiranjem je negativna takrat, ko atribut ni kompresiven.

4.1.7 J mera

J mera v nasprotju z prej definiranimi ocenjuje kakovost zgolj ene vrednosti atributa. Definirana je kot povprečje količine informacije pri dani vrednosti atributa glede na posamezne razrede, uteženo z verjetnostjo, da nek atribut dejansko zavzame to vrednost.

J mera je nenegativna in zanjo velja, da je vsota J mer po vseh vrednostih atributov kar informacijski prispevek atributa. Ker J mera ocenjuje vrednost le enega atributa, zanjo ne velja, da bi precenjevala večvrednostne atribute. J mera je tudi statistično zanimiva, saj sledi χ^2 porazdelitvi. Zaradi vseh teh lastnosti je danes ena od najpogosteje uporabljenih mer pri ocenjevanju if-then pravil.

4.1.8 Mera ortogonalnosti

Mera ortogonalnosti je definirana kot razlika med 1 in kosinusom kota med dvema vektorjema verjetnostnih distribucij za vsako vrednost atributa posebej. Uporabna je pri binarnih atributih.

Mera ortogonalnosti je simetrična glede na razrede, njena vrednost je vselej med 0 in 1 in velja, da je enaka 0, ko je informacijski prispevek atributa enak 0, in 1, ko sta pogojni entropiji za obe vrednosti atributa enaki 0, saj tedaj atribut idealno ločuje razreda med seboj. Obratno ne velja; če je mera ortogonalnosti enaka 1, to še ne pomeni, da je informacijski prispevek maksimalen, saj pri binarnih atributih nimamo garancije, da bomo vse razrede ločili med seboj.

4.1.9 Gini indeks

Gini indeks (prior / mera nečistoče) je definiran kot razlika med 1 in vsoto kvadratov apriornih verjetnosti po razredih.

Kakovost atributa ocenimo z razliko med priornim in pričakovanim Gini indeksom. To nam pove, kakšna je povprečna verjetnost, da dva naključna primera pripadata istemu razredu pri določeni vrednosti atributa. Ima enake lastnosti kot informacijski prispevek in prav tako precenjuje vrednosti večvrednostnih atributov. V praksi se uporablja za ocenjevanje linearnih atributov.

4.1.10 RELIEF

Vse do sedaj predstavljene mere so kratkovidne, saj se osredotočijo zgolj na ocenjevanje enega atributa, medtem ko ostale ignorirajo.

RELIEF temelji na ideji, da iz množice vzamemo naključen primer in opazujemo njegov najbližji primer iz nasprotnega in najbližji primer iz istega razreda. Glede na to, kakšne so razlike v vrednostih, atributom določimo različne uteži. Postopek ponovimo za več, lahko pa tudi za vse učne primere. Algoritem ima kvadratno časovno zahtevnost glede na število primerov in je zato precej časovno potraten, a še vedno v krajšem času reši problem, ki je bil sprva videti eksponenten. RELIEF je sposoben ocenjevati tako diskretne kot zvezne atribute in njihov kontekst in ne precenjuje večvrednostnih atributov, vendar pa je občutljiv na šum v podatkih in omejen na dvorazredne probleme.

4.1.11 ReliefF

Problem neznanih vrednosti atributov, šumnosti podatkov in večrazrednih problemov rešuje ReliefF. Če atributu manjka ena vrednost, se upošteva verjetnost, da imata primera, ki pripadata nekemu razredu, različno vrednost atributa, če manjkata obe vrednosti, pa se izračunajo pogojne verjetnosti po vseh možnih vrednostih atributa glede na razred posameznega primera. Šumnosti podatkov se izogne z iskanjem ne le najbližjega soseda, temveč več najbližjih sosedov. Posplošitev na več razredov pa naredimo tako, da poiščemo najbližje sosede za vse razrede, prispevke k utežem atributov pa utežimo z apriornimi verjetnostmi posameznih razredov.

4.1.12 Sprememba variance

Varianca je mera nečistoče v zveznem prostoru. Definirana je kot kvadrat odstopanja primera od povprečja. Bolj se učni primeri v prostoru nahajajo skupaj, manjši sta varianca in nečistoča (in obratno).

Ocena atributa je razlika variance, torej razlika med apriorno varianco in posteriorno varianco, torej ko že poznamo vrednost atributa. Razlika variance je nenegativna, ima pa enako slabost kot večina že predstavljenih metod, in sicer precenjuje večvrednostne atribute. Metoda se zato v praksi uporablja za binarne atribute.

4.1.13 RReliefF

V regresiji ne moremo uporabiti najbližjih sosedov, zato namesto tega skušamo oceniti "verjetnost", da dva primera pripadata različnim razredom, ki jo modeliramo z razdaljo. Z njim lahko ocenjujemo tako diskretne kot zvezne atribute, pri čemer ne precenjujemo večvrednostnih atributov. Časovna zahtevnost je enaka kot pri običajnem RELIEFu.

4.1.14 MDL v regresiji

Pri MDL moramo kodirati realna števila, ciljno spremenljivko, ki je prav tako realno število, in napako. Za kodiranje števil lahko uporabimo fiksno natančnost.

Pri kodiranju ciljnih spremenljivk ne smemo delati razlik, saj so si vse vrednosti med seboj enakovredne. Pri kodiranju napake to ne velja, saj morajo imeti manjše napake krajšo kodo, za to lahko uporabimo recimo Rissanenovo kodo.

4.2 Izbira podmnožice atributov

4.2.1 Filter

Atribute lahko filtriramo in izberemo k najbolje ocenjenih. Pristop je hiter, vendar kratkoviden, saj ne upošteva interakcij med atributi. Tako se lahko zgodi, da sta dva atributa, ki sta si zelo podobna, oba ocenjena kot zelo dobra, vendar s tem pridobimo kopije atributov.

4.2.2 Ovojnica

Druga možnost je ovojnica, ki deluje tako, da izberemo podmnožico, zgradimo model in ga testiramo, ves postopek pa ponavljamo tako dolgo, dokler nismo zadovoljni. Odvisno od modela ta možnost implicitno upošteva interakcije med atributi, je pa počasna. Že testiranje ene podmnožice je časovno zahtevno, ob tem pa moramo upoštevati še, da je prostor vseh podmnožic

eksponenten, zato lahko pregledamo le njen majhen del. Pri izbiri podmnožic si zato pomagamo z različnimi strategijami:

- **izbira naprej**: začnemo s prazno podmnožico in dodajamo po en atribut, dokler se rezultati izboljšujejo,
- **odstranjevanje nazaj**: začnemo z vsemi atributi in odstranjujemo po en atribut, dokler se rezultati izboljšujejo,
- **kombinacija**: začnemo z naključnim začetnim stanjem ali začetnim stanjem, ki ga tvori filter.

4.2.3 Vgrajen princip

Ena od možnosti je tudi, da algoritem sam izbira atribute med gradnjo modela, kar imenujemo vgrajen princip.

Poglavje 5

Umetne nevronske mreže

Umetne nevronske mreže so poenostavljen model bioloških nevronskih mrež. Uporabljajo preproste procesne enote, imenovane nevroni (ali perceptroni), ki z vhodnih povezav z določenimi utežmi dobivajo signale od sosednjih nevronov, jih seštevajo in pošiljajo skozi pragovno funkcijo. Izhod nevrona je tako določen z vhodom in vrednostmi uteži in prav uteži na povezavah med nevroni so tiste, ki predstavljajo spomin nevronske mreže.

Učna naloga nevronske mreže je torej izbrati topologijo mreže in nastaviti vrednosti uteži.

Možgani so sestavljeni iz veliko med seboj povezanih nevronov, ki si med seboj pošiljajo signale. Vsak nevron je tako povezan s številnimi drugimi. Pošiljanje signala med nevroni je v obliki vse ali nič, saj mora električni potencial v telesu celice nevrona preseči nek določen prag. Kako močan signal bo poslan, je odvisno od moči povezave oziroma sinapse med dvema nevronoma.

Nevronske mreže se zmorejo prilagajati spreminjajočemu se okolju tako, da sproti posodabljajo uteži na povezavah. So konstruktivnega značaja, kar pomeni, da posamezen nevron kombinira več osnovnih atributov. Nevronske mreže so tudi robustne, saj so sposobne sprejeti pomanjkljive vhodne podatke, poleg tega pa odstranitev enega nevrona povzroči le delni padec v natančnosti za več podatkov in ne popolne izgube enega. Mreža bo v tem primeru tako še vedno delovala, le natančnost bo nekoliko slabša. Za nevronske mreže velja tudi, da čim večja je porazdeljenost, tem natančneje lahko shranimo iste podatke z enakim številom nevronov. Slaba stran nevronskih mrež je, da so kompleksne, zaradi česar pogosto nimamo prave razlage, zakaj na izhodu dobimo določen rezultat.

Nevronske mreže delimo po različnih kriterijih:

- **topologija**: brez nivojev, dvonivojske, večnivojske (globoke),
- **namen**: klasifikacija, regresija, razvrščanje, razpoznavanje signalov, slik, videoposnetkov, besedil ...,
- **pravilo učenja**: Hebbovo pravilo, delta pravilo - gradientno, tekmovalno, pozabljanje,
- **funkcija kombiniranja vhodov nevrona v izhod**: aktivacijska funkcija (utežena vsota, sigma-pi, naivni Bayes), izhodna funkcija/normalizacija (pragovna, (ne)deterministična, sigmoidna (odvedljiva), ReLu).

5.1 Dvonivojske usmerjene nevronske mreže

Pri dvonivojskih nevronskih mrežah gre pravzaprav samo za uteženo vsoto, zato lahko tovrstne nevronske mreže rešijo le linearne probleme. Število vhodov v nevronsko mrežo ustreza številu atributov, število izhodov pa pri klasifikaciji ustreza številu razredov, pri regresiji pa imamo le en sam izhodni nevron. Učna naloga je nastaviti uteži na povezavah tako, da bo mreža uspešno rešila čim več učnih primerov. Vse učne primere bo taka mreža rešila le, če je primer linearen.

5.1.1 Učenje enega nevrona

Najprej naključno inicializiramo uteži na povezavah. Nato pokažemo vzorec in izhod, izračunamo dejanski izhod, posodobimo uteži na povezavah in ta postopek ponavljamo tako dolgo, dokler ne dosežemo sprejemljive točnosti.

5.2 Hopfieldove nevronske mreže

Hopfieldova nevronska mreža je poln graf, kar pomeni, da je vsak nevron povezan z vsakim, vključno s samim seboj. Uteži se nastavljajo po posplošenem Hebbovem pravilu: če sta vrednosti nevronov enaki, se povezava ojača, če različni, pa oslabi. Utež je torej korelacija aktivnosti med dvema nevronoma. Učenje je linearno, saj vsak primer pogledamo natanko enkrat in glede na to popravimo uteži, izvajanje pa je asinhrono. Hopfield je dokazal, da se izvajanje garantirano konča v končnem številu korakov, zato je stabilno.

Taka nevronska mreža ima tudi sposobnost avto-asociacije, kar pomeni, da podatek asociira isti podatek oziroma pomanjkljiv ali pokvarjen podatek lahko asociira pravilen podatek (delno sliko ključa lahko, na primer, povežemo s celotno sliko ključa).

Pri hetero-asociaciji gre za priklic sorodnega podatka, recimo delno sliko ključa lahko povežemo s ključavnico.

5.3 Večnivojske usmerjene nevronske mreže

Večnivojske nevronske mreže imajo enega ali več skritih nivojev, zato lahko rešijo poljuben nelinearni problem. Učna naloga je izbrati ustrezno število skritih nivojev in število skritih nevronov na vsakem nivoju, torej sestaviti topologijo mreže, ter nastaviti uteži na povezavah tako, da bo mreža uspešno rešila čim več učnih primerov.

Pri učenju tovrstnih nevronskim mrež začnemo z naključno nastavitvijo uteži na povezavah med nevroni. Mreži na vhodu podamo vhodni vzorec in ta izračuna izhod. Najprej se izračuna razlika med dejanskim in želenim izhodom, nato pa se spremenijo uteži med zadnjim in predzadnjim nivojem. Zatem se izračunajo želene vrednosti nevronov na predzadnjem nivoju, izračuna razlika med želenimi in dejanskimi vrednostmi na predzadnjem nivoju in spremenijo uteži med prepredzadnjim in predzadnjim nivojem. Postopek se ponavlja, dokler ne pridemo do vhodnega nivoja nevronov.

Kot izhodna funkcija se najpogosteje uporablja sigmoidna funkcija $f(X) = \frac{1}{1+e^{-X}}$, ker je izračun njenega odvoda preprost.

Pri tem uporabimo vzvratno propagiranje napake, kar pomeni, da se napaka v nekem nevronu propagira nazaj kot vsota preko vseh povezav, pri čemer se pri vsaki povezavi pomnoži z utežjo na tej povezavi.

5.3.1 Pravilo delta

Pravilo delta se uporablja za izračun uteži v nevronski mreži. Uporablja se gradientno pravilo, kar pomeni, da se izračuna odvod napake glede na uteži. Napako predstavimo kot razliko med želenim in dejanskih izhodom nevrona, jo kvadriramo in odvajamo po utežeh. Izračunan gradient nam pove, kako moramo nastaviti uteži, da minimiziramo napako. Pri tem uporabljamo stopnjo učenja, ki nam pove, kako velik korak moramo narediti pri spreminjanju uteži. Če je ta parameter prevelik, bomo uteži nastavljali preveč naključno, če pa je premajhen, pa bo učenje prepočasno. V praksi moramo torej najti optimalen parameter, da se izognemo obema skrajnostma.

Obstaja tudi paketna varianta pravila delta, kjer namesto enega primera upoštevamo vse učne primere. Izračunamo torej napako za vse učne primere in postopek nadaljujemo na enak način kot prej. Tako učenje je hitrejše, vendar manj natančno, ker uteži nastavljamo preveč splošno. V praksi se zato ne vzame vseh učnih primerov, ampak nek manjši del, recimo 10.

Pravilo delta ima določene slabosti:

- ker gre za gradientno pravilo, ne konvergira vedno (lahko obtiči v lokalnem minimumu),
- izbira topologija mreže (koliko skritih nivojev, koliko nevronov na skritih nivojih ...),
- preveliko prileganje učni množici (več nivojev pomeni več možnih parametrov),
- stopnja učenja vpliva na stabilnost (včasih jo je lahko težko določiti),
- propagiranje nazaj je zelo počasno zaradi veliko prehodov preko učnih primerov,
- pravilo nima biološke analogije z možgani.

Prvi problem lahko omilimo tako, da vpeljemo dodatni člen, ki utež usmeri v smeri, v kateri smo se gibali prej. S tem se lahko izognemo temu, da obtičimo v kakšnem vmesnem lokalnem minimumu. Drugi problem je precej težji. Ponavadi ga rešujemo tako, da kaznujemo velike uteži na povezavah. Tako lahko učenje začnemo z veliko mrežo in odvečne nevrone sproti odstranjujemo, obenem pa se s tem izognemo tudi prevelikemu prileganju učni množici, ker se mreža ustrezno zmanjša. Slaba stran rešitve je, da metoda vpelje dodatne parametre, katerih nastavitev ni trivialna.

5.3.2 Načrtovanje mreže

Vhode lahko kombiniramo z vsoto, vsoto kvadratov, produktom ...

Za aktivacijsko funkcijo po navadi izbiramo med sigmoidno funkcijo, hiperboličnim tangensom, Gaussovo funkcijo, linearno funkcijo, funkcijo softmax, ReLU ...

Za mrežo izbiramo med sprotnim, paketnim in mini-paketnim učenjem, opcijsko lahko dodamo tudi momentni člen, s čimer se lahko izognemo morebitnemu obstanku v lokalnem minimumum, določiti moramo regularizacijsko metodo oziroma metodo za eliminacijo uteži, za kar danes v praksi uporabljamo L1 in L2, izbrati pa moramo tudi funkcijo napake, kjer izbiramo med absolutno napako, kvadratno napako, napako 0-1, logaritemsko napako ...

5.3.3 Preprečevanje prevelikega prileganja

Najpreprosteje in najučinkoviteje se prevelikemu prileganju izognemo tako, da ustavimo učenje, ko napaka na validacijski množici začne naraščati. Druga možnost je, da uporabimo dodatno, večjo učno množico. Zgradimo lahko tudi več nevronskih mrež in jih povežemo v ansambel, kar sledi principu večkratne razlage, vendar je to pri nevronskih mrežah precej časovno potratno. Po principu najkrajšega opisa pa lahko kaznujemo velike uteži, eliminiramo premajhne in na tak način zmanjšamo kompleksnost modela.

5.3.4 Regularizacija

L1 regularizacija k napaki prišteje člen, ki sešteje absolutne vrednosti uteži. S tem prisili nekatere uteži, da postanejo 0, na ta način pa eliminiramo nekatere nevrone, s čimer zmanjšamo kompleksnost modela.

L2 regularizacija k napaki prišteje člen, ki sešteje kvadrate vrednosti vseh uteži v mreži. S tem zmanjšamo vrednosti vseh uteži, a ne spremenimo topologije mreže. Metoda po navadi privede do dobrih rezultatov oziroma majhne napake.

Max-norm navzgor omeji vrednost vsake uteži oziroma dolžine vektorjev uteži za vsak nevron.

Princip preskakovanja (angl. dropout) temelji na tem, da pri učenju za vsak učni primer upoštevamo vsak nevron le z določeno verjetnostjo, nekatere tako preprosto izpustimo. Na tak način prisilimo mrežo, da se nauči reševati problem tudi pri pomanjkljivi informaciji. Pri tem vsak učni primer uči drugačno mrežo, zato je rezultat unija več manjših pomanjkljivih mrež. Postopek sledi principu ansambla in v praksi dobro deluje.

	Tipično	Možne vrednosti
μ stopnja učenja	0.1	$[0.01, 0.99]$
α momentum	0.8	$[0.1, 0.9]$
λ cena uteži	0.1	$[0.001, 0.5]$

Tabela 5.1: Nastavitve parametrov v nevronskih mrežah

Eno od vprašanj je tudi, kako nastavimo uteži. Najlažje je uteži nastaviti naključno. Lahko se odločimo za princip, ki pravi, da morajo imeti vozlišča z velikim številom vhodnih povezav manjše uteži, kar lahko pomaga preprečevati preveliko prileganje že takoj na začetku. Obstaja tudi pravilo ad-hoc, ki pravi, da začetne vrednosti navzgor omejimo s $\pm\frac{1}{k}$, kjer je k število uteži na vhodnih povezavah nekega vozlišča. Učinkovito nastavljanje uteži

je še posebej pomembno pri globokih nevronskih mrežah, saj lahko s tem bistveno pohitrimo učenje.

5.3.5 Osnovni napotki

- poskusimo z najboljšo znano metodo,
- pridobimo čim večjo učno množico,
- poskusimo z mrežo brez skritih nivojev,
- smiselno kodirajmo vhodne spremenljivke,
- uporabimo regularizacijo,
- uporabljajmo validacijsko množico za izogib prevelikemu prileganju,
- s prečnim preverjanjem se prepričajmo, da je rešitev dobra.

Nevronske mreže se dobro obnesejo v praksi, saj se lahko naučijo poljubne, tudi nelinearne funkcije, dobro delujejo na šumnih podatkih, ob zadostnem številu učnih primerov jih je možno dobro posplošiti in čeprav je učenje počasno, je predikcija hitra.

5.4 Nevronske mreže z radialnimi baznimi funkcijami

Ideja je, da imamo vhodne in izhodne nevrone ter en skriti nivo, pri čemer je učenje skritega nivoja nenadzorovano, izhodnih nevronov pa nadzorovano. Pri tem skriti nevroni skušajo vhodne vzorce povezati v skupine tako, da se vsak vhodni vektor skuša približati enemu od nevronov, ti pa izračunavajo razdaljo med vhodnim vektorjem in središči skritih nevronov ter na tak način učne podatke združiti v skupine. Izhodna funkcija v skritih nevronih je po navadi Gaussova, medtem ko izhodni nevroni izračunavajo linearno funkcijo. Vhodne podatke tako najprej pretvorimo v večdimenzionalni prostor, ki jih nato klasificiramo z linearnimi funkcijami.

Središča skritih nevronov so najprej določena naključno. Nato se vsakemu dodeli najbližji učni primer in središče se posodobi v skladu s hitrostjo učenja μ. Zatem za vsako središče poiščemo najbolj oddaljen učni primer in izračunamo skalar, ki se uporablja za določanje izhoda bazne funkcije. Učenje izhodnega nivoja medtem poteka enako kot pri običajnih nevronskih mrežah.

Tovrstne nevronske mreže nimajo lokalnih minimumov, ker imamo na izhodu linearno funkcijo. Njihova težava je, da ne vemo, koliko je središč. Če jih je preveč, lahko pride do prevelikega prileganja. Metoda je sicer zelo podobna metodi podpornih vektorjev.

5.5 Globoko učenje

Pri plitvem učenju gre za neposredno preslikavo vhoda v izhod. Pri globokem učenju pa gre za preslikavo vhoda v vmesne koncepte in šele iz njih se tvori izhod.

Temelj globokega učenja je učenje vsakega nivoja nevronske mreže posebej, ki jim nato dodamo končni nivo in jih izboljšamo z vzvratnim propagiranjem napake.

Težava večnivojskih nevronskih mrež, ki jih učimo z vzvratnim propagiranjem napake je, da učenje traja zelo dolgo, obstaja velika verjetnost za preveliko prileganje učni množici, gradientno iskanje se lahko ustvai v lokalnem minimumu, poleg tega pa potrebujemo veliko učno množico, da dobimo dobre rezultate.

Globoke nevronske mreže delujejo po principu večkratne razlage, saj gre v bistvu za ansambel podmrež, ki vse rešujejo isti problem.

Globoke nevronske mreže lahko rešujejo nelinearne probleme in so kompaktna predstavitev vhodnih podatkov, kar hkrati omogoča tudi, da iz nje lahko do določene mere pridobimo izvorno predstavitev. Težava je, da med učenjem lahko naletimo na množico lokalnih minimumov, da še vedno ne vemo, kakšna je optimalna struktura, torej koliko nivojev in koliko nevronov potrebujemo na vsakem nivoju, in dolgotrajno učenje, njihova slabost pa je tudi nerazumevanje njihovega delovanja.

Na začetku imamo vhod, ki ga preslikamo v atributni prostor. Skonstruirati ga želimo tako, da bo sposoben rekonstruirati vhod. Nato definiramo naslednji nivo z več abstraktnimi atributi, ki mora znati rekonstruirati prejšnji nivo in tako naprej, dokler z globino mreže nismo zadovoljni. Šele na zadnjem nivoju uporabimo ciljno spremenljivko, nakar uporabimo vzvratno propagiranje napake za nastavljanje uteži na povezavah.

Globoko učenje se zelo dobro obnese pri videih, avdiu in obdelavi naravnega jezika.

Za razumevanje konteksta v slikah se zelo dobro obnesejo tudi rekurentne nevronske mreže, ki imajo vzvratne povezave.

Poglavje 6

Nenadzorovano učenje

Pri nenadzorovanem učenju je vsak podatek opisan z vektorjem atributov. Tipični problemi so avto-asociacija, kjer se učimo korelacij med deli vzorca, ki omogočajo dopolnjevanje manjkajočega ali pokvarjenega dela vzorca, razvrščanje ali gručenje, kjer podobne primere združujemo v skupine, faktorizacija, kjer iščemo skrčeno predstavitev, ki omogoča približno rekonstrukcijo vhodnih podatkov, in povezovalna pravila, ki iščejo zakonitosti v podatkih, kjer nimamo ciljne spremenljivke.

6.1 Razvrščanje

Pri razvrščanju se soočamo z naslednjimi izzivi:

- različnih gručenj je eksponentno mnogo,
- kako določiti podobnost med primeri (kako izmeriti razdalje med primeri),
- kakšen je kriterij združevanja primerov,
- kakšna je pravilna razvrstitev (nimamo ciljne spremenljivke) ...

6.1.1 k-means

Pri algoritmu k-means so skupine predstavljene s centroidi. Na začetku so ti določeni naključno. Nato vsak primer priredimo najbližjemu centroidu, ponovno izračunamo centroide in postopek ponavljamo do konvergence.

6.2 Faktorizacija

Pri matrični faktorizaciji skušamo matriko učnih podatkov predstaviti z dvema veliko manjšima matrikama. Če ti dve matriki zmnožimo, dobimo nazaj matriko, ki je približno enaka izvorni. Ker lahko postopek odkrije manjkajoče vrednosti v matriki, se pogosto uporablja v priporočilnih sistemih.

Zadeva deluje tako, da matriki inicializiramo z naključnimi vrednostmi, izračunamo, za koliko se njun produkt razlikuje od izvorne matrike in nato v več korakih poskušamo minimizirati to razliko (npr. z gradientnim iskanjem lokalnega minimuma razlike z odvajanjem napake).

6.3 Povezovalna pravila

Povezovalna pravila so se razvila v želji ugotoviti navade kupcev. Problem je eksponenten, zato si izčrpnega iskanja ne moremo privoščiti, obstaja pa algoritem apriori, ki z omejenim izčrpnim iskanjem najde optimalno rešitev.

Problem je definiran tako, da imamo podatkovno bazo s transakcijami. V vsaki transakciji so bili kupljeni določeni izdelki. Cilj je poiskati vsa povezovalna pravila med izdelki, torej če kupimo določene izdelke, bomo kupili tudi nekatere druge. Ker moramo generirati vse možne kombinacije izdelkov, vpeljemo dva parametra, in sicer podporo s in zaupanje α. Podpora nam pove, kolikokrat se neka kombinacija izdelkov pojavi v podatkovni množici, zaupanje pa nam pove, v koliko primerih se zgodi, da če kupimo neko množico izdelkov, kupimo tudi ostale.

Recimo, da imamo povezovalno pravilo **kruh** $\rightarrow$ **marmelada**. V tem primeru je podpora definirana kot delež transakcij, kjer nastopata kruh in marmelada, zaupanje pa, v koliko transakcijah, kjer nastopa kruh, nastopa tudi marmelada.

Algoritem tako usmerjata parametra zaupanje in podpora, ki sta smiselno definirana, saj nas navadno zanimajo najpogostejša in najnatančnejša pravila. Najprej zato poiščemo množice izdelkov, ki imajo zadostno podporo, nato pa iz njih generiramo vsa povezovalna pravila, ki zadoščajo minimalnemu zaupanju.

Algoritem temelji na dejstvih, da je vsaka podmnožica pogoste množice pogosta in da nobena nadmnožica nepogoste množice ni pogosta. Pri generiranju vseh možnih podmnožic pa velja pravilo, da v pravilu, ki ima zadostno podporo in zaupanje in ima daljši sklepni del, lahko del sklepnega dela premaknemo v pogoj. To lahko uporabimo tako, da najprej generiramo pravila s krajšim sklepnim delo, in če katero izmed teh pravil nima zadostnega zaupanja, nam pravila z daljšim sklepnim delom sploh ni treba preverjati.

Prednosti algoritma so, da je preprost in ga lahko paraleliziramo, največja slabost pa je, da zahteva veliko prehodov čez podatkovno bazo. Problem lahko omilimo s pozabljanjem transakcij, ki ne vsebujejo nobenih pogostih množic. Druga možnost je, da podatkovno bazo razbijemo na particije in pogoste množice iščemo le v posameznih particijah in šele na koncu med kandidati izberemo pogoste množice v celotni bazi. Obstajata še vzorčenje in zelo konservativni vrednosti za podporo in zaupanje, pri čemer ti vrednosti postopoma spreminjamo, dokler ne dosežemo želenega števila pravil.

Poglavje 7

Predstavitev znanja

Simbolična predstavitev znanja je eksplicitna in logična. Sem uvrščamo pravila if-then, odločitvena in regresijska drevesa, povezovalna pravila, logiko prvega reda (logično programiranje) in verjetnostne porazdelitve (naivni Bayes in k najbližjih sosedov).

Numerična predstavitev znanja je implicitna in podzavestna. Sem uvrščamo linearne funkcije, polinomske funkcije, ϕ-funkcije in nevronske mreže kot nelinearne funkcije.

Obstaja še porazdeljena predstavitev znanja, ki jo srečamo pri matrični faktorizaciji in nevronskih mrežah, saj je preslikava naučena iz vseh učnih primerov, hkrati pa se vse uteži uporabijo za preslikavo enega primera.

7.1 Predstavitev besedil

Na področju besedil je veliko izzivov: povzemanje, kategorizacija, razvrščanje besedil, vizualizacija informacije (npr. ključne besede), določanje avtorstva, zaznavanje plagiatorstva ...

Obdelava besedila je še danes težko, saj algoritmi ne zmorejo razumeti pomena besedil. To je še posebej očitno pri več besedah z istim pomenom (npr. muca in mačka) in pri enakih besedah z več pomeni (npr. zmaj, ki lahko pomeni mitološko bitje ali igračo).

7.1.1 Vreča besed

Vreča besed (angl. bag-of-words) je predstavitev podatkov s številom pojavitev posamezne besede iz nekega slovarja v besedilu. Težava predstavitve je, da izgubimo vrstni red in s tem tudi kontekst besed, poleg tega je število atributov lahko zelo veliko. Algoritmi na tem področju so naivni Bayes, metoda

podpornih vektorjev in globoke nevronske mreže.

Namesto posameznih besed lahko vzamemo n zaporednih besed. S tem ohranimo kontekst besed v besedilu, a še povečamo atributni prostor. V praksi se najpogosteje uporabljajo 2, 3 in 4-terke. Zadevo lahko posplošimo tako, da posamezne besede vmes izpuščamo (angl. skip-grams).

7.1.2 TF-IDF

Ideja TF-IDF je v tem, da skušamo najti besede, ki so specifične za posamezen dokument. Če se torej beseda pojavlja v vseh dokumentih, je nepomembna. Slabost je, da nam tudi ta predstavitev ne pove nič o kontekstu besed.

7.1.3 Word2Vec

Word2Vec preslika besede v večdimenzionalne vektorje. Začnemo z naključno predstavitvijo matrike, ki vsebuje vektorje besed. Nato iz matrike izluščimo za nas zanimive vektorje, jih združimo v en sam vektor in uporabimo gradientno metodo, da jih popravimo. Metodo je potrebno učiti na velikih množicah podatkov z od 100 milijonov do milijarde besedami.

Arhitektura CBOW poskuša napovedati trenutno besedo glede na kontekst (npr. na podlagi 2 besed pred in 2 besed za trenutno besedo). Arhitektura skip-gram pa skuša doseči ravno nasprotno; iz trenutne besede skuša napovedati okoliške. Medtem ko je pri prvem učenje hitrejše, ker ne upošteva zaporedja besed, a manj natančno, je učenje pri skip-gram počasnejše, a natančnejše, ker upošteva vrstni red besed.

Pri Word2Vec gre za porazdeljeno predstavitev podatkov, ki upošteva tudi kontekst besed v besedilu. Besede, ki nastopajo v podobnem kontekstu, bodo tako predstavljene s podobnimi vektorji, kar lahko izračunamo s kosinusno podobnostjo.

Postopek je možno posplošiti tako, da ne uporabljamo besed, ampak povedi, odstavke ali celo besedila. To lahko na preprost način naredimo tako, da povprečimo vse besede v povedi, odstavku ali dokumentu, vendar s tem izgubimo vrstni red besed. Celotno poved, odstavek ali dokument nato lahko predstavimo z enim samim vektorjem.

Poglavje 8

Bayesovsko učenje

Bayesov klasifikator je teoretično najboljši možen klasifikator. Če pri določenih pogojih vemo pravilno izračunati verjetnosti razredov, bo rezultat optimalen. Ker pa v praksi teh verjetnosti ne poznamo, jih moramo aproksimirati. Izračunamo ga kot verjetnost hipoteze pri danem opažanju:

$$P(hipoteza|opažanje) = \frac{P(opažanje|hipoteza) \cdot P(hipoteza)}{P(opažanje)}$$

8.1 Naivni Bayes

Pri strojnem učenju nas zanima verjetnost razreda pri danih vrednostih atributov. Ker število zahtevanih učnih primerov s številom atributov za ocenjevanje verjetnosti eksponentno raste, uporabimo predpostavko, da so atributi pri danem razredu medsebojno neodvisni. Rezultati so zato dobri le, če so atributi $(X_1, X_2, ..., X_n)$med seboj dovolj neodvisni.

$$P(C|X_1, X_2, ..., X_n) = \frac{P(C)\prod_i P(X_i|C)}{\prod_i P(X_i)} \tag{8.1}$$

Učenje poteka tako, da ocenimo apriorne in pogojne verjetnosti razreda pri danem atributu, napovedovanje pa poteka tako, da izračunamo verjetnosti za vse razrede pri določenih vrednostih atributov in primer klasificiramo v najbolj verjeten razred.

Naivni Bayes vedno uporabi vse znane atribute. Za neznane vrednosti primere preprosto izpustimo. Ker gre pri njem za izračunavanje verjetnosti, se učni množici ne more preveč prilagoditi, preprosto pa lahko pridobimo razlago tako, da logaritmiramo Bayesovo formulo.

$$m - ocena : P(r_k|v_i) = \frac{N_{k,i} + mP(r_k)}{N_i + m} \tag{8.2}$$

8.2 Delno naivni Bayes

Ideja delno naivnega Bayesa je, da če se za dva atributa izkaže, da sta odvisna, ju združimo. Postopek ponavljamo tako dolgo, dokler je to smiselno.

Dva atributa sta odvisna, ko velja:

$$P(AB|C) = P(A|C) \cdot P(B|C)$$

Komentar: zgornja trditev ne velja (tisti je pogoj za pogojno neodvisnost glede na C). Odvisna sta če P(AB) != P(A) P(B).

Dva atributa združimo, če sta si produkt in faktor dovolj različna in če je ocena združene verjetnosti dovolj zanesljiva. Velja, da več pogojev združimo, slabša je aproksimacija. Zanesljivost aproksimacije navadno izračunavamo s teoremom Čebiševa.

Učenje začnemo z učenjem naivnega Bayesa in nato združujemo atribute po zgoraj opisani metodi. Gre za požrešno omejeno izčrpno iskanje, saj v postopek vključimo pogoj, kdaj prenehamo z združevanjem atributov.

8.3 Bayesovske mreže

Bayesovska mreža opredeljuje odvisnosti spremenljivk med seboj. Mreža je vselej brez ciklov. Neko vozlišče v mreži je neodvisno, če poznamo vse vrednosti staršev, staršev otrok in otrok.

Najpreprostejša Bayesovska mreža ima v korenu razred, iz katerega izhajajo vsi možni atributi. Odvisnosti ne veljajo vedno, zato je mrežo oziroma povezave v njej potrebno smiselno preurediti, kar ponavadi naredimo v sodelovanju s strokovnjakom, ki nam lahko razloži odvisnosti atributov.

Za Bayesovsko mrežo za klasifikacijo velja, da je ciljna spremenljivka vselej brez staršev, za atribute pa velja, da so potomci razreda. Povezave so možne tudi med atributi, ki definirajo odvisnost potomca od starša.

8.4 Drevesno razširjen naivni Bayes

Ideja je, da naivnemu Bayesu dodamo podgraf, ki v obliki drevesa povezuje vse atribute med seboj. S tem algoritmu omogočimo, da sam poišče odvisnosti med atributi, za katere naivni Bayes predpostavlja, da jih ni, drevesna struktura pa obenem bistveno zmanjša možnost pojava prevelikega prilegaanja.

Odvisnost atributov se določa z medsebojno pogojno informacijo. Za vsak par atributov se tako izračuna vsota pogojnih entropij za vsak atribut

posebej, od tega pa se odšteje pogojna entropija za oba atributa skupaj. Tako poiščemo maksimalno vpeto drevo in za koren izberemo katerikoli atribut ali pa nam ga pomaga izbrati strokovnjak.

Za iskanje minimalnega vpetega drevesa v grafu z neusmerjenimi povezavami se uporablja Primov algoritem. Gre za požrešen algoritem, katerega časovna zahtevnost je $O(m \log n)$, kjer je m število povezav in n število vozlišč v kopici.

Poglavje 9

Kalibracija verjetnosti

Klasifikatorji tipično vrnejo verjetnostno distribucijo po razredih, ki mora biti čim bolj zanesljiva, da jo uporabnik lahko ustrezno interpretira. Različni klasifikatorji imajo različne pristranskosti:

- naivni Bayes precenjuje ali podcenuje verjetnosti, zato tipično dobimo verjetnosti, ki so bližje 0 in 1,
- k najbližjih sosedov, SVM in boosting naredijo obratno, in sicer verjetnosti ocenjujejo preveč konservativno, zaradi česar so verjetnosti bolj oddaljene od 0 in 1, torej bližje 0.5,
- odločitvena drevesa imajo visoko varianco in pristranskost, ki sta precej odvisni od vhodnih podatkov,
- naključni gozdovi, nevronske mreže, logistična regresija in včasih k najbližjih sosedov so navadno že privzeto dobro kalibrirani.

Za kalibracijo potrebujemo kalibracijsko množico, ki ne sme biti del učne ali testne množice. Kalibracijska množica se uporablja izključno za nastavljanje parametrov pri kalibraciji. V splošnem velja, da večja je velikost kalibracijske množice, boljša je kalibracija.

9.1 Plattova metoda

Kalibrirane verjetnosti dobimo tako, da jo pošljemo skozi sigmoidno funkcijo, zato je primerna zgolj pri modelih s sigmoidno pristranskostjo, kot so SVM, boosting in včasih k najbližjih sosedov. Cilj metode je zmanjšati razliko med napovedano verjetnostjo razreda in dejanskim razredom, za kar uporabimo gradientno metodo. Da se izognemo prevelikemu prileganju, uporabimo Laplaceovo metodo.

9.2 Izotonična regresija

Izotonična regresija je splošnejša od Plattove metode, vendar zanjo potrebujemo večjo kalibracijsko množico, česar si vselej ne moremo privoščiti. Ideja izotonične regresije je, da se izračuna funkcija, ki monotono narašča. Če funkcija ne bi naraščala monotono, bi namreč lahko spremenili vrstni red in posledično tudi klasifikacijo primerov, česar pa ne smemo storiti. Cilj metode je torej najti funkcijo, ki minimizira kvadratno napako.

Poglavje 10

Ocenjevanje zanesljivosti

Zanesljivost napovedi nam pove, kako dobro metoda opravlja svoje delo. V strojnem učenju nas navadno zanima kompleksnost modela, razlaga modela, predvsem pa točnost napovedi.

Naš cilj je napovedati, kolikšna je verjetnost, da je posamezna napoved zanesljiva.

10.1 Pristopi, vezani na model

Pri odločitvenih drevesih nek učni primer pride do lista (ali več listov, če imamo manjkajoče vrednosti). Zanesljivost napovedi potem lahko ocenimo glede na porazdelitev razredov v listu.

Pri linearni regresiji lahko izračunamo interval zaupanja, kjer lahko z določeno gotovostjo trdimo, da je naša napoved točna.

Prednost pristopov, vezanih na model, je, da je možna verjetnostna razlaga, slabost pa, da so vezani na določen model in zato manj splošni.

10.2 Pristopi, neodvisni od modela

Imamo model, ki deluje kot črna škatla. Z nastavljanjem različnih parametrov spreminjamo vhode in opazujemo, kako spremembe vplivajo na izhode. Spreminjamo lahko, recimo, učno množico.

Pri analizi senzitivnosti uporabimo nov primer, ki mu priredimo bodisi napovedano, bodisi neko naključno ciljno vrednost, in ga dodamo v učno množico. Na tak način smo spremenili učno množico. Sedaj uporabimo spremenjeno učno množico, ji podamo ta primer in opazujemo, kako se spreminja verjetnostna distribucija v primerjavi s prvotnim modelom. Na tak

način ugotovimo, koliko je model občutljiv na ta učni primer in bolj se verjetnostna porazdelitev spremeni, bolj je model občutljiv nanj. Enak pristop lahko uporabimo pri regresiji, le da novemu primeru namesto razreda priredimo realno število. Metoda včasih deluje dobro, po navadi takrat, ko ima model visoko varianco, kar lahko analiza senzitivnosti celo popravi.

Pri "bagging variance"uporabimo množico različnih modelov in vse uporabimo za napovedovanje. Iz tega nato izračunamo varianco napovedi, in večja kot je varianca, manj zanesljiva je napoved. Je kompleksnejša, a v praksi navadno deluje dobro.

Lokalno prečno preverjanje deluje tako, da za nek nov učni primer poiščemo najbližje sosede, njihove dejanske ciljne spremenljivke in njihove napovedi. Na ta način izvemo, kako dobro model napoveduje sosede in velja, da če to drži, je tudi napoved novega primera zanesljivejša. Zahteva veliko dela, vendar pogosto deluje dobro.

Ocena gostote se v praksi redko uporablja. Deluje tako, da skušamo oceniti gostoto učne množice na območju novega primera. Gostejša je učna množica na tem območju, zanesljivejša je napoved. Ker metoda ne upošteva atributov, se navadno ne obnese najbolje.

Pri lokalnem modeliranju napake za nov učni primer preprosto izračunamo variance oznak najbližjih sosedov. Višja je varianca, manj zanesljiva je napoved. Metoda deluje hitro in včasih deluje dobro.

V klasifikaciji poznamo še verjetnost najbolj verjetnega razreda. Če vzamemo verjetnost najbolj verjetnega razreda za oceno zanesljivosti, se izkaže za najboljšo možno metodo.

Poglavje 11

Kombiniranje in prilagajanje algoritmov strojnega učenja

11.1 Ansambli

Ansambli so sestavljeni iz več različnih modelov. Sledijo principu večkratne razlage, le da napovedi določamo s kombiniranjem napovedi posameznih modelov glede na verjetnosti modelov. Uporablja se lahko za zmanjševanje pristranskosti in variance. Če imamo veliko učno množico, lahko posamezne modele učimo z različnimi deli učnih množic, če imamo majhno podatkovno množico, pa lahko uporabimo "bootstrapping". Prednost ansamblov je tudi v tem, da zmore rešiti kompleksnejše probleme, saj jih lahko razbije na več manjših; modeli tako rešujejo le del kompleksnega problema.

11.1.1 Stacking

Z bootstrappingom zgradimo različne modele in se nato naučimo združevanja. Cilj metode se je tako dodatno naučiti, kako učinkovito združiti zgrajene modele. Za ta ti. meta klasifikator si lahko privoščimo le uporabo preprostejših modelov, kot sta navni Bayes za klasifikacijo in linearna regresija za regresijo, saj sicer lahko pride do prevelikega prileganja. Za kombiniranje odgovorov modelov navadno uporabimo metodo glasovanja oziroma povprečenja, če gre za regresijski problem. Za nove primere lahko uporabimo lokalno uteženo glasovanje, ki upošteva natančnosti napovedi najbližjih sosedov.

11.1.2 Bagging

Generiramo veliko število podmnožic učne množice z bootstrappingom (vzorčenjem z vračanjem). Vsaka učna množica ima tako enako število učnih primerov,

vendar pri tem pride do tega, da se nekateri primeri pojavijo v več množicah, nekateri pa v nobeni (v povprečju izpustimo približno tretjino primerov). Na podlagi teh množic zgradimo modele, ki vsi uporabljajo enak algoritem, vendar različne učne množice, in na koncu vsi glasujejo za končen rezultat. Izkaže se, da metoda izboljša učinkovitost nestabilnih algoritmov, ki so odvisni od variance v učni množici, hkrati pa ne pride do prevelikega prileganja učni množici.

Naključni gozdovi

Specialna verzija bagginga, uporablja bootstraped podatkovne množice. Nadgradnja bagginga z odločitvenimi oziroma regresijskimi drevesi je naključni gozd. Učne množice definiramo enako kot pri baggingu, bagging pa vpeljemo tudi pri sami gradnji dreves: izmed množice atributov jih izberemo določeno število, izmed katerih izberemo najboljši atribut. Na tak način zgradimo 100 ali več dreves, pri čemer je vsako zgrajeno na drugačni učni množici in naključno izbranih atributih. Rezultat določimo na podlagi glasovanja pri klasifikaciji oziroma povprečenja pri regresiji. Njihova prednost je, da zmanjšujejo varianco v podatkih in zelo dobro delujejo pri problemih, kjer je atributni prostor dobro definiran, medtem ko pri problemih, kjer je potrebno atributni prostor dopolniti s kakšno konstruktivno indukcijo, kakor to počnejo nevronske mreže, niso tako učinkoviti, hkrati pa tudi nimamo razlage, kako metoda deluje.

11.1.3 Boosting

Pri boostingu predpostavljamo, da zna naš algoritem delati z uteženimi podatki, sicer to simuliramo z razmnoževanjem učnih primerov. V prvem koraku zgradimo model, pri čemer imajo vsi učni primeri utež 1.0, in ga testiramo na učni množici, nakar pravilno napovedanim primerom zmanjšamo uteži, napačno napovedanim pa jih povečamo. Pri tem uteži pomnožimo z $\frac{e}{1-e}$, kjer je e napaka napovedi in na koncu uteži normaliziramo tako, da je njihova vsota enaka velikosti učne množice n. Postopek ponavljamo toliko časa, dokler ne rešimo vseh primerov ali dokler izboljšave niso več možne, kar pomeni, da je problem pretežek. Zadnji model zavržemo, da se v prvem primeru izognemo prevelikemu prileganju, v drugem pa, ker je preprosto preslab in ne pomaga pri rešitvi problema, medtem ko ostale obdržimo. Končno napoved modelov izračunamo z uteženim glasovanjem, kjer utež predstavlja točnost modela na svoji učni množici. Ta se izračuna s formulo $-\log \frac{f}{1-f}$, kjer f predstavlja napako. Boosting se pogosto izkaže za natančnejšega od bagginga in ga lahko uporabljamo tudi s stabilnimi algoritmi z nizko vari-

anco, ki jim na ta način zmanjšamo pristranskost. Slabost boostinga je, da včasih pride do prevelikega prileganja.

11.2 Transdukcija

Navadno od nekega primera do njegove rešitve v strojnem učenju pridemo s pomočjo hipoteze. Pri transdukciji pa gre za sklepanje rešitve neposredno iz učnega primera.

Če za nov primer ne poznamo vrednosti odvisne spremenljivke, zanjo določimo več vrednosti in vsak tako spremenjen primer ločeno dodamo v učno množico, na kateri zgradimo model. Izberemo model, ki je najboljši in na tak način izberemo tudi oznako novega primera. To lahko ponavljamo tako dolgo, dokler se oznaka še spreminja.

11.3 Cenovno občutljivo učenje

Za posamezne predikcije imamo lahko različne cene za različne napake, na primer, en razred je lahko pomembnejši kot drugi. Ideja je utežiti primere. Če algoritem ne zmore upoštevati uteži, lahko to simuliramo z razmnoževanjem učnih primerov. Metoda deluje le v dvorazrednih problemih, pri večrazrednih je ena od možnih rešitev razbitje učne množice na več dvorazrednih.

11.4 Učenje iz neuravnovešenih učnih množic

Pri neuravnovešenih učnih množicah nam manjkajo primeri iz enega razreda, zato je primerom iz manjšinskega razreda potrebno dati višjo težo.

To najpreprosteje dosežemo z naključnim nadvzorčenjem, s čimer razmnožimo primere iz manjšinskega razreda. Druga možnost je podvzorčenje, s čimer se znebimo dela primerov iz večinskega razreda.

Ena od možnosti je tudi generiranje umetnih primerov, ki so generirani iz učne množice. Ti primeri so umetno sestavljeni, zato navadno poteka v sodelovanju s strokovnjakom, ki presodi, ali so smiselno sestavljeni. To naredimo tako, da vzamemo en primer iz manjšinskega razreda, poiščemo najbližjega soseda, generiramo naključno utež med 0 in 1 in nato za vsak atribut novo generiranega primera izračunamo nov atribut, tako da vrednosti atributa primera iz manjšinskega razreda prištejemo razliko med vrednostma atributa najbližjega soseda in atributa iz manjšinskega razreda, pomnoženo z utežjo.

11.5 Imitacija funkcije

Recimo, da imamo dobro metodo za napovedovanje, vendar želimo uporabljati metodo, ki jo dejansko razumemo.

Ideja je, da s pomočjo dobre napovedovalne metode generiramo dodatne učne primere: recimo da imamo nevronsko mrežo, z njo generiramo nove dodatne primere in nato s poljubno drugačno metodo zgradimo model (recimo klasifikacijsko drevo, ki se ga da enostavno razložiti) s pomočjo teh novih dodatno generiranih primerov.

11.6 Uporaba klasifikatorja v regresiji

Regresorsko diskretno spremenljivko diskretiziramo in uporabimo klasifikator. Seveda je potrebno potem diskretne napovedi spremeniti nazaj v zvezne. S tem, ko ciljno spremenljivko diskretiziramo, zanemarimo majhne razlike v vrednostih regresijske spremenljivke (kar je dobro, če je to, recimo, šum). Končno napoved nato določimo s pomočjo verjetnostne porazdelitve.

Zadevo je možno tudi obrniti tako, da z regresijo rešujemo klasifikacijski problem. Prvemu razredu določimo vrednost 0, drugemu pa 1 in tako klasifikacijski problem rešujemo kot regresijski. Končno napoved podobno kot prej določimo s pomočjo verjetnostne porazdelitve.

11.7 Tabele za popravljanje napak

Razred	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	0	1	1	1	1	1	1	1
R3	0	0	0	0	1	1	1	1	0	0	0	0	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 11.1: Primer tabele za popravljanje napak

Recimo, da imamo petrazredni klasifikacijski primer. Namesto grajenja klasifikatorja za pet razredov razstavimo primer na 15 manjših binarnih problemov, kjer v vsakem skušamo razločiti enega ali več razredov od ostalih. S tem dobimo zaporedje bitov glede na klasificiran razred. Končen rezultat je razred, katerega vrstica se najbolje ujema z napovedmi.

Poglavje 12

Razumevanje v strojnem učenju

12.1 Razumevanje podatkov in problema

Za razumevanje podatkov, ki so povezani z našim problemom, je zelo pomembna vizualizacija podatkov. Same številke namreč ljudje težko dojamemo, še posebej, če jih je veliko, medtem ko smo ljudje dobri pri zaznavalju vizualnih vzorcev. Vizualizacija nam lahko namreč pokaže tako interakcije med atributi kot tudi med atributi in ciljno spremenljivko. Na tak način lahko bolje razumemo zakonitosti v podatkih in problem, ki ga rešujemo.

Z vizualizacija lahko odkrijemo sama pravila za klasifikacijo, skupine podobnih primerov in osamelce, interakcije med atributi, strukture, trende in druge zakonitosti v podatkih.

Pri tem se soočamo z velikim številom možnih vizualizacij, saj je že pri majhnem številu atributov možnih veliko različnih diagramov. V bioinformatiki, recimo, kjer imajo primeri lahko tudi po več tisoč atributov, je možno izrisati nekaj milijonov različnih diagramov. Diagramov tako ročno ne moremo pregledovati, zato nalogo iskanja zanimivih vizualizacij prepustimo računalniku.

Osnovna metoda je histogram, ki nam prikaže frekvence primerov po posameznih vrednosti. S tem lahko odkrijemo zakonitosti v podatkih, včasih pa tudi odstopanja ali celo napake, kot, na primer, da so v podatkih neznane vrednosti kodirane z 0. Druga možnost je vizualizacija posameznega atributa skozi čas.

Dva atributa naenkrat lahko vizualiziramo z diagramom raztrosa. Če želimo na tak način vizualizirati več atributov, generiramo več diagramov in na vsakem prikažemo le izbran par atributov.

Več atributov naenkrat lahko vizualiziramo tudi ločeno po primerih s ti. zvezdastimi grafi, kjer vsak krak zvezde predstavlja vrednost določenega atributa.

Vizualizacija se lahko prilagodi tudi človeku s ti. vizualizacijo obraza, kjer posamezne atribute predstavimo z lastnostmi obraza (npr. oči, nos, usta ...), pri čemer vsak obraz predstavlja en primer.

Na enem grafu je možna tudi vizualizacija več kot dveh dimenzij. Primer takega grafa je recimo diagram raztrosa, kjer za prikaz razreda uporabimo barve. Radviz uporablja krog, ki ga razbije na toliko delov, kolikšno je število atributov. Primer je prikazan tam, kjer je vsota vrednosti njegovih atributov najbolj utežena. Vizualizaciji lahko dodamo tudi barve, ki predstavljajo razrede. Podobno lahko atribute predstavimo tudi z različnimi dolžinami vektorjev in koti med njimi, kar imenujemo linearna vizualizacija. Pri vzporednem grafu posamezna os predstavlja en atribut, primeri pa so prikazani s črtami. Tudi tu lahko uporabimo barvo, da ločimo razrede primerov med seboj. Nazadnje imamo še mozaični diagram, ki je namenjen prikazu največ štirih diskretnih atributov, saj ima pravokotnik štiri stranice. Za izbiro optimalnih atributov pri tej vizualizaciji je najlažje izračunati kartezični produkt atributov, da dobimo nov atribut, ki ga ocenimo z različnimi merami nečistoče (MDL, informacijski prispevek, razdalja, χ^2 ...), lahko pa uporabimo tudi njegovo točnost.

Vizualizacija nam lahko pomaga tudi pri iskanju optimalnega modela. Na enem grafu lahko izrišemo lego primerov in točnosti posameznih modelov ter na podlagi tega sklepamo, kateri model je najustreznejši. Navadno izberemo takega, ki se razmeroma dobro prilega množici podatkov in je čim preprostejši.

Večina vizualizacij je nezanimivih, saj v njih ne dobimo nobenih dodatnih informacij. Za nas zanimive vizualizacije so tiste, ki razločujejo primere iz različnih razredov.

12.1.1 Vizrank

Najprej iz množice atributov izberemo atribute za vizualizacijo in za dano metodo vizualizacije izračunamo koordinate. Ker je teh vizualizacij lahko zelo veliko, skušamo poiskati tiste, ki so zanimive. To naredimo tako, da generiramo učno množico, sestavljeno iz koordinat in ciljnih vrednosti, za katero napovemo točnost. Dobljena točnost predstavlja zanimivost vizualizacije.

Za ocenjevanje kakovosti vizualizacij uporabimo k najbližjih sosedov, točnost pa ocenjujemo s prečnim preverjanjem (običajno 10-kratnim) in povprečno verjetnostjo pravilnega razreda.

V primerih, ko imamo veliko atributov, je vizualizacij toliko, da jih tudi računalnik ne zmore pregledati. Vizrank to rešuje s hevristiko, kjer da bolje ocenjenim atributom večje možnosti za pojav v vizualizaciji. Na ta način ne pregledamo vseh vizualizacij, temveč le omejeno število.

Vizualizacije se lahko uporabljajo tudi za oceno kakovosti atributov. To preprosto naredimo tako, da štejemo, kolikokrat se določen atribut pojavi v najboljših vizualizacijah.

Vizualizacije se lahko uporabijo tudi za iskanje osamelcev. Hkrati jim z različnimi vizualizacijami lahko sledimo in ugotavljamo, kaj se dogaja z njim.

Poglavje 13

Razlaga modelov v strojnem učenju

V strojnem učenju imamo vedno podatke, iz katerih zgradimo model, ki te podatke razlaga in ki ga lahko uporabimo za napovedovanje novih primerov. Tipično gre za črno škatlo, za katero ne vemo, kako natanko deluje. Želeli bi si, da ko modelu podamo nek nov primer, dobimo napoved in razlago, zakaj se je model odločil zanjo.

Pri odločitvenih drevesih je razlaga že samo drevo oziroma v primeru napovedi pot, ki nas je pripeljala do ciljnega vozlišča. Pri odločitvenih pravilih, ki sicer niso drevesno urejena, lahko razlago za napoved modela izluščimo na podoben način. Pri k najbližjih sosedov imamo shranjene le učne primere. Razlaga napovedi so kar vrednosti atributov k najbližjih sosedov novemu primeru. Pri naivnem Bayesu, ki potrebuje apriorno in pogojne verjetnosti razreda za dane vrednosti atributov, je razlaga logaritem Bayesove formule, od koder dobimo, koliko posamezni atributi govorijo za ali proti določenemu razredu.

Nevronska mreža po drugi strani ne nudi jasne razlage. Tudi če za nek nov primer ročno izračunamo vrednosti v vseh nevronih, nam ne bo jasno, kako zadeva deluje.

13.1 Splošna razlaga napovedi modela

Ljudem najbližja razlaga je tista, ki jo ponuja naivni Bayes: koliko posamezna vrednost atributa pripomore k določenemu razredu.

Model obravnavamo kot črno škatlo. Cilj je ugotoviti funkcijo, ki se je je model naučil glede na učne podatke tako, da opazujemo spremembo napovedi glede na podane vrednosti atributov.

Prvotni sistemi so se reševanja problematike lotili z računanjem razlike napovedi pri vseh znanih atributih in napovedi, ko izbran atribut skrijemo, ter na tak način določili pomembnost izbranega atributa. Težava metode je, da je kratkovidna, zato ne upošteva interkacije med atributi, kar lahko vodi do napak. Primer, kjer ta metoda odpove, je recimo disjunkcija treh vrednosti.

Boljša rešitev je preverjanje vpliva vseh podmnožic atributov. Če imamo n atributov, je torej potrebno pregledati 2^n podmnožic.

Naj bo Δ_Q razlika med neko podmnožico atributov Q in apriorno napovedjo in I_W interakcijski prispevki (prispevki, ki jih prispevajo interakcije), pri čemer je W podmnožica atributov. Potem je

$$\Delta_Q = \sum_{W \subseteq Q} I_W$$

To pomeni, da razlika v predikciji pride iz interakcij vseh podmnožic Q. Interakcijski pripevek množice Q pa je razlika med Δ_Q in zgoraj zapisano vsoto.

Prispevek posameznega atributa izračunamo tako, da pogledamo interakcijske prispevke za vse podmnožice, v katerih se ta atribut nahaja, in ga normaliziramo tako, da vsak atribut v vsaki podmnožici prispeva enakovredno.

Da izračunamo vse interakcijske prispevke napovedi modela, moramo omogočiti skrivanje določenega atributa. To lahko dosežemo na več načinov:

- za izbran atribut nastavimo neznano vrednost, pri čemer se lahko pojavi težava, če naš model ne dopušča neznanih vrednosti atributov ali če ne vemo, kako model obravnava neznane vrednosti,
- za dano podmnožico atributov vsakič zgradimo nov model, kar je zaradi eksponentne zahtevnosti (imamo namreč 2^n podmnožic atributov) nesprejemljivo, hkrati pa tudi predpostavimo, da imamo na voljo izvirno učno množico modela, kar ni nujno,
- utežena vsota, kjer pridobimo napovedi za vse vrednosti izbranega atributa in napovedi povprečimo, s čimer se izognemo zgoraj omenjenim težavam.

Pomembno je poudariti, da nas zanima razlaga modela in ne razlaga domene, saj domene ne moremo razložiti, ker je ne poznamo. Velja, da imajo boljši modeli boljše razlage.

Ta razlaga je torej preprosta, človeku razumljiva, neodvisna od modela, nekratkovidna in lahko jo uporabljamo za razlago posameznega primera ali

celotnega modela. Slaba stran metode je časovna zahtevnost, ker imamo eksponentno število podmnožic atributov.

Časovno zahtevnost lahko zmanjšamo z nekaj spremembami. Najprej se znebimo potrebe po skrivanju atributov tako, da gremo z vzorčenjem preko celotne množice primerov, ki imajo določene vrednosti vseh atributov, in opazujemo razliko v napovedi, ki jo prispeva neka podmnožica atributov.

Za aproksimacijo prispevka vrednosti atributa uporabimo Shapleyjevo vrednost. Shapleyjeva vrednost temelji na naslednjih Shapleyjevih aksiomih:

- vsota vseh kakovosti atributov je enaka kakovosti celotne množice,
- če dva atributa v vseh množicah prispevata enako, potem je tudi njuna kakovost enaka,
- če za nek atribut vselej velja, da je prispevek množice z atributom in množice brez atributa enak, je kakovost atributa 0, saj ne prispeva ničesar,
- če ocenjujemo atribute v dveh različnih ocenjevanjih prispevkov, je vseeno, če to počnemo ločeno.

Algoritem deluje tako, da najprej izberemo število vzorcev m, kakovost izbranega atributa nastavimo na 0 in začnemo z vzorčenjem. Vzamemo naključno permutacijo atributov, naključno generiramo primer v prostoru atributov in uporabimo model za napoved, katerega model želimo razložiti, pri čemer enkrat izbran atribut uporabimo, enkrat pa ne. Na ta način dobimo razliko v predikciji, ki jo prispeva dana vrednost našega primera pri tem atributu, s katero posodobimo prispevek izbranega atributa. Postopek ponovimo m-krat in prispevek atributa na koncu normaliziramo z m. Višji kot je m, boljša je ocena (v praksi navadno zadošča 10000 vzorčenj), je pa odvisen od števila atributov: več kot je atributov, več vzorčenj je potrebnih.

13.2 Razlaga napovedi v regresiji

V regresiji namesto posameznih razredov ciljno spremenljivko obravnavamo kot en sam razred. Naš cilj je ugotoviti, koliko posamezni atributi prispevajo za oziroma proti napovedani točkovni napovedi. Poleg prispevka atributa tukaj dodamo tudi pomembnost atributa oziroma varianco, saj lahko neka vrednost atributa v različnih kontekstih govori za ali proti neki vrednosti.

Razlago modela dobimo tako, da vzorčimo preko celotne populacije, pri čemer zopet naključno izbiramo primere, za dani primer izberemo atribute v dani podmnožici, ki smo jo izbrali za ta atribut, opazujemo razliko in

posodabljamo kakovost. To naredimo za vse atribute in vse vrednosti tega atributa ter zadevo na koncu normaliziramo.

Če želimo imeti razlage v kontekstu, algoritem popravimo le toliko, da namesto naključno generiranega primera izberemo enega od primerov iz učne množice.

Poglavje 14

Aktivno učenje

Pri aktivnem učenju učenec dejansko vpliva na to, katere podatke bo dobival iz realnega sveta. Da izboljša svoj model, od sveta zahteva konkretnejše podatke in sam izbira, iz katerih učnih podatkov se bo učil.

S tem ko učenec izbira, katere točke v atributnem prostoru so zanj zanimive za to, da izboljša svoj model, je lahko število učnih primerov, ki jih potrebujemo, manjše, posledično pa lahko tudi zmanjšamo čas in strošek označevanja novih primerov ter izboljšamo sam model. Majhna učna množica pa je hkrati lahko tudi slabost, saj naš model potem ni občutljiv na majhne spremembe ciljne hipoteze in majhne spremembe v učni množici - pride torej do prevelikega prileganja. Hkrati pa je dejstvo tudi, da če aktivno izbiramo učne primere, tako dobljena verjetnostna distribucija ni reprezentativna. Vpeljali smo namreč pristranskost vzorčenja, kar pomeni, da na situacijo nimamo pravega pogleda, to pa lahko hkrati privede tudi do slabšh napovedi.

Pri aktivnem učenju predpostavljamo, da imamo majhno množico označenih modelov in da je označevanje novih primerov drago, medtem ko je neoznačenih primerov običajno veliko (ali pa jih skonstruiramo sami). Te bodisi konstantno pridobivamo v obliki podatkovnega tokova bodisi jih imamo podane vnaprej.

Na eni strani imamo prostor primerov, iz katerega vzorčimo:

- če na voljo ni učnih primerov, jih bo generiral učenec in uporabnika (strokovnjaka, drug model ...) prosil, da jih označi,

- če so primeri na voljo in prihajajo v obliki podatkovnega toka, jih je preveč, da bi jih uporabnik lahko označil, zato se aktivni učenec za vsakega odloči, ali je zanimiv za učenje ali ne, nakar prosi uporabnika, da zanimive primere označi,

- če so primeri na voljo vnaprej, recimo v podatkovni bazi, aktivni učenec izbere primere, ki bi jih bilo smiselno označiti.

14.1 Izbira primerov za označevanje

Model lahko primere, za katere meni, da bi jih bilo smiselno označiti, izbira na več načinov.

14.1.1 Najmanj zanesljiva napoved

Imamo nov neoznačen primer, ki ga model označi. Za oceno nezanesljivosti lahko upoštevamo nizko verjetnost za najbolj verjeten razred, majhno razliko med verjetnostma dveh najbolj verjetnih razredov ali največjo izmed entropij distribucije razredov.

14.1.2 Največja redukcija prostora hipotez

Ideja pristopa je, da minimiziramo prostor hipotez. To lahko naredimo tako, da zmanjšamo število odločitvenih mej med dvema razredoma. Učne primere vzorčimo po postopku bisekcije, dokler ne najdemo ustrezne meje.

14.1.3 Ansambli

Najprej zgradimo več različnih modelov (bagging, boosting ...) in ta ansambel uporabimo za napovedovanje novih neoznačenih primerov. Nov primer je nezanimiv, če se modeli z napovedjo strinjajo in zanimiv, če se ne strinjajo. Nestrinjanje lahko merimo z razliko med števili glasov za posamezen razred, razliko med največjima verjetnostma napovedi dveh različnih razredov, z najmanjšo verjetnostjo napovedan najbolj verjeten razred, iz porazdelitve frekvenc glasov za posamezen razred izračunamo entropijo ali pa uporabimo Kullback-Leiblerjevo divergenco.

14.1.4 Maksimalna redukcija napake

Če želimo dodati nov učni primer za označevanje, si želimo, da je napaka napovedi po tem čim manjša, zato poskušamo predvideti napako na novih primerih. Učni primer posodobimo z vsemi možnimi napovedmi, vsakič posodobimo model in napovemo vse ostale primere. Princip lahko zato uporabimo samo, če so podatki dani vnaprej, ena od slabosti postopka pa je tudi velika časovna zahtevnost.

14.1.5 Kombinirani pristopi

Postopke lahko tudi kombiniramo, na primer, da iščemo največjo razliko od označenih primerov in hkrati najmanjšo oddaljenost od neoznačenih primerov, da se izognemo šumnim primerom in osamelcem.

14.1.6 Struktura podatkov

Sestavimo graf sosedov. Začnemo z naključno izbranimi primeri, zatem pa nas zanimajo samo oznake tistih, ki niso podobni že označenim primerom. Zanimajo nas torej novi učni primeri, ki se razlikujejo od že označenih.

Poglavje 15

Strojno učenje iz podatkovnih tokov

Dandanes se srečujemo s presežkom podatkov. V vsakem trenutku se namreč s pomočjo senzorjev in drugih naprav meri velika količina podatkov, ki jih ne moremo ne shraniti ne prenašati po komunikacijskih kanalih.

Podatkovni tokovi so časovno urejeno zaporedje podatkov, ki je potencialno neskončno, saj podatki prihajajo neprekinjeno v realnem času, navadno tudi z veliko hitrostjo. Pri tem velja tudi, da na njihovo zaporedje ni mogoče vplivati, temveč je, kakršno pač je. Zaradi njihovega obsega je podatke nemogoče shraniti v celoti in težko obdelovati v realnem času, zato se moramo odločati, katere bomo sploh obdelali, in nato še, katere bomo shranili. Izziv, s katerim se soočamo, je tudi morebitna sprememba porazdelitev podatkov oziroma koncepta, zaradi česar se mora model prilagajati novejšim podatkom, ki so pomembnejši od starejših. Uporabiti moramo torej tako vzorčenje, da bodo novejši podatki imeli višjo težo kot starejši.

Časovne vrste so v nasprotju s podatkovnimi tokovi zaporedje meritev v tipično enakih časovnih intervalih, saj ima v njih čas bistveno vlogo. Pri njih nas zanima trend in kratkoročne in dolgoročne periode. Pri podatkovnih tokovih pa je poudarek na zaporedju, ne glede na časovne točke.

Klasične metode so bile razvite za statične podatke, torej podatke zberemo, jih prečistimo, zgradimo model in ga uporabimo, ko model ni več dovolj dober, pa cikel ponovimo. Če imamo podatkov zelo veliko, lahko klasičen pristop nadgradimo z ovojnico, kjer podatke vzorčimo in uporabljamo algoritme, ki se hitro učijo. Vendar za oba pristopa velja, da naučenega modela ni možno posodobiti, temveč moramo celoten učni postopek ponoviti z dodanimi novimi podatki.

Ena od možnih rešitev bi lahko bila uporaba ansambelskega pristopa, kjer za vsak vzorec podatkov zgradimo nov model. Modele lahko tudi uredimo

po starosti in novejšim damo višjo težo. Obenem pa s tem pristopom rešimo tudi morebitno spremembo koncepta v podatkih.

15.1 Sprotno učenje

Učne primere obdelujemo zaporedoma. Vsak primer je obdelan največ enkrat in model sproti posodabljamo. Algoritem uporablja navzgor omejena količino pomnilnika in čas za učenje. Ko je model naučen, mora biti takoj pripravljen za napovedovanje, da ga lahko takoj uporabimo ali preizkusimo.

Tipično se zadovoljimo z aproksimacijskimi algoritmi. Definiramo parametra majhno napako ϵ in visoko točnost $1 - \delta$., ki definirata razliko med pravo hipotezo in naučeno aproksimacijo. Želimo, da je verjetnost, da je razlika med pravo in aproksimirano hipotezo manjša od ϵ, večja od $1 - \delta$.

Naravni inkrementalni in učinkovit algoritem je naivni Bayes, kjer primere preprosto štejemo in izpustimo neznane vrednosti. Pri tem moramo upoštevati, da je klasifikator linearen in zato namenjen le reševanju linearnih problemov, čemur se lahko izognemo z množico linearnih klasifikatorjev, ki pa lahko rešijo tudi nelinearne probleme. V regresijskih problemih lahko uporabimo linearno regresijo, ki jo lahko rešujemo z enonivojsko nevronsko mrežo, ki računa uteženo vsoto. Z gradientnim pravilom lahko popravljamo uteži na povezavah in model prilagajamo novim podatkom.

Tudi umetne nevronske mreže so po naravi inkrementalne. Posodabljamo jih tako, da za vsak učni primer popravljamo uteži na povezavah po pravilu vzvratnega razširjanja napake. To lahko storimo tudi v paketu nekaj desetih učnih primerov, kar je časovno manj zahtevno, hkrati pa zmanjšamo verjetnost prevelikega prileganja.

Pri k najbližjih sosedov bi morali shraniti vse primere, kar je nepraktično, ker je podatkov preprosto preveč. Namesto primerov zato shranimo le prototipe, tipične predstavnike podatkov, ki jih lahko identificiramo z metodo k voditeljev (k-means). Ker je število prototipov omejeno, je prototipe potrebno sproti popravljati. Podobno kot pri nevronskih mrežah lahko to storimo za posamezen učni primer ali v paketu nekaj deset učnih primerov.

Sproti lahko posodabljamo tudi drevesa. Za ta namen uporabimo zelo hitra Hoeffdingova drevesa. Pri tem upoštevamo, da je majhen vzorec podatkov pogosto dovolj za skoraj optimalno odločitev. Odločiti se je potrebno le, kako velik mora biti vzorec, ki bo zadoščal za zanesljivo razlikovanje med alternativnimi odločitvami. Ideja je iz majhnega vzorca izračunati zadostne statistike in izračunati kakovost vsakega atributa. Začnemo z enim listom in se sproti odločamo, ali naj list razširimo v poddrevo. To storimo v primeru, če je razlika med oceno kakovosti najbolje ocenjenega atributa

in oceno kakovosti drugega najbolje ocenjenega atributa večja od napake $\epsilon = \sqrt{\frac{(\max G - \min G)^2 \ln \frac{1}{\delta})}{2n}}$, kjer je n število učnih primerov. Velja, da se ϵ z večanjem števila učnih primerov zmanjšuje, zaradi česar se bo model posodobil pri manjši razliki. Hoeffdingova drevesa vračajo stabilne odločitve (model ima nizko varianco), do prevelikega prileganja navadno ne more priti, saj vsak primer obdelamo samo enkrat, drevesa ni potrebno rezati in v teoriji imamo nizko napako in visoko verjetnost. Slabost modela je neupoštevanje sprememb koncepta.

Ko pride do spremembe koncepta, atribut, ki je bil izbran, morda ni več najboljši za sam model. Čeprav je poddrevo zastarelo, je nekaj časa gotovo boljše kot najboljši list, še posebej, če je ta izbrani atribut bližje korenu drevesa. Ideja je zgraditi alternativno poddrevo, kjer izberemo novi najboljši atribut, pri čemer staro poddrevo še vedno uporabimo. Potem vsake toliko časa testiramo kakovost obeh poddreves in ko je staro drevo slabše od novega, ga zamenjamo z novim.

Sprotno učenje je torej robustno, saj se morebitne napake in manjkajoči podatki med učenjem sproti popravljajo, če so na voljo novejši podatki, lahko učenje na starih podatkih prekinemo in začnemo uporabljati novejše, in model se sproti prilagaja novim trendom in porazdelitvam v podatkih. Je pa standardiazcija in normalizacija podatkov lahko problematična, saj za dan atribut ne poznamo maksimuma in minimuma, povprečja in standardnega odklona, vendar v praksi aproksimacija navadno zadošča.

15.1.1 Merjenje uspešnosti

Kakovost sprotnega učenja merimo z uspešnostjo skozi čas glede na število primerov. Zanima nas:

- uspešnost v določenem časovnem obdobju,
- hitrost naraščanja uspešnosti v začetku učenja, saj tedaj še nimamo veliko učnih primerov,
- vztrajnost uspešnosti skozi čas.

Pri sprotnem učenju k-kratnega prečnega preverjanja ne moremo uporabiti, ker zanj nimamo časa, poleg tega pa sproti pozabljamo učne primere. Namesto tega se uporablja enkratno razbitje na učne in testne primere in na ta način merimo uspešnost skozi čas.

Uspešnost sprotnega učenja lahko testiramo periodično, ko občasno uporabimo primere za testiranje in izmerimo kakovost modela, lahko pa se testiranje in učenje prepletata in kakovost modela izmerimo za vsak učni primer,

kar poteka tako, da uporabimo nov primer za posodobitev modela, potem dobimo nov primer, ki ga uporabimo za testiranje in nato za posodobitev modela ...

Napako merimo tako, da seštejemo napake od začetka do trenutnega primera in normalizirano vsoto sproti ažuriramo. Tako so izkoriščeni vsi podatki, graf uspešnosti skozi čas je neprekinjen, napako računamo v realnem času in primere z manjkajočimi vrednostmi lahko izpustimo. Slabost metode je, da imamo na začetku nezanesljivo oceno točnosti, zaradi česar je natančno merjenje časa, potrebno za uspešno učenje, težavno.

15.1.2 Sprememba koncepta

Sprememba koncepta pomeni, da se vir podatka zamenja z nekim drugim virom, ki ima drugačno porazdelitev. To predstavlja težavo, ker gre za nepredvidljivo spremembo. Pri tem velja omeniti, da ponavljanje dogodkov z neko periodo ni sprememba koncepta, saj mora model take vzorce upoštevati.

Poznamo tri vrste spremembe koncepta:

- sprememba apriorne porazdelitve ciljne spremenljivke (pogosteje se pojavljajo primeri iz različnih razredov; relacija med razredi in atributi ostane enaka),
- sprememba pogojne vrednosti atributov pri danih vrednostih ciljne spremenljivke (če pride samo do spremembe vrednosti neodvisnih spremenljivk, se spremenijo samo apriorne verjetnosti atributov),
- sprememba pogojne porazdelitve ciljne spremenljivke pri danih vrednostih atributov (spremeni se relacija med atributi in ciljno spremenljivko).

Osnovni tipi spremembe koncepta so naslednji:

- nenadna: preklop z enega vira na drugega,
- postopna: spremembe se zvezno in počasi dogajajo skozi čas, zato jih je težje zaznavati,
- mešanje: koncepta se nekaj časa preklapljata in sčasoma prevlada nov,
- ponavljajoča: koncepti se ponavljajo, a ne periodično,
- redek dogodek: začasna kratkotrajajoča sprememba,
- šum.

Tipi se med seboj lahko tudi mešajo, zato je vseh možnih načinov spremembe koncepta pravzaprav neskončno mnogo.

Sprememba koncepta je lahko globalna ali lokalna. Pri lokalni spremembi se spremembe dogajajo samo v delu atributnega prostora, pri globalni pa se spremembe dogajajo v celotnem atributnem prostoru. Ideja detekcije spremembe koncepta je, da porežemo poddrevo, kjer je do spremembe koncepta prišlo, ali pa ponovno naučimo model v listu.

Ključna ideja pri detekciji spremembe koncepta je pozabiti staro informacijo. Za to potrebujemo inkrementalne algoritme, ki znajo vključiti novo informacijo, in dekrementalne algoritme, ki znajo pozabiti starejšo informacijo. Za samo detekcijo spremembe koncepta potrebujemo delo s podatki, v okviru česar se je potrebno odločiti, katere podatke shraniti v pomnilnik, da bo prisotna prava informacija za posodabljanje modela, metode za samo detekcijo in adaptacijo odločitvenega modela na novo porazdelitev.

V okviru spremembe koncepta ločimo polni in delni pomnilnik. Pogosteje se uporablja delni pomnilnik, kjer shranjujemo vzorec vseh podatkov, za kar uporabljamo drseža okna (FIFO), da so v vrsti novejši podatki, in vzorčenje, da v pomnilnik shranjujemo le reprezentativen vzorec. Pri polnem pomnilniku shranjujemo statistike preko vseh primerov. Te se lahko razlikujejo od statistik v novejših učnih podatkih, kar nam lahko pove, ali je prišlo do spremembe koncepta ali ne.

Pri delnem pomnilniku imamo okno, kamor shranjujemo podatke, pri čemer novejši podatki ostajajo, starejše pa odstranjujemo. Če uporabljamo fiksno velikost okna, je majhno okno lahko dobro za adaptacijo na spremembo koncepta, vendar je naše učenje manj natančno, ker uporabljamo manjšo množico podatkov, če uporabljamo veliko okno, pa je naše učenje stabilno, a na spremembe koncepta reagira počasi. Namesto teh se uporabljajo okna s spremenljivo velikostjo, ki jih uporabljamo skupaj z modeli za detekcijo sprememb koncepta. Ko ugotovimo spremembo koncepta, bomo torej zmanjšali velikost okna, zavrgli starejše podatke in upoštevali le novejše. Dokler pa se koncept ne spreminja, pa lahko okno povečujemo do maksimalne velikosti, da ohranjamo dobro učinkovitost modela.

Pri vzorčenju je pomembno, da je vzorec reprezentativen (ustrezno velik in zajet z ustrezno tehniko vzorčenja). Navadno se uporablja rezervoarsko vzorčenje, ki ohranja naključni vzorec, kar je pomembno, da ne vpeljemo prevelike pristranskosti. Pri tem uporabimo parameter k, ki predstavlja velikost rezervoarja, in ko pride nov primer, ga ohranimo z verjetnostjo $\frac{k}{n}$, kar pomeni, da kasneje se podatek pojavi v podatkovnem toku, manjša je verjetnost, da ga bomo obdržali. Preden v rezervoar shranimo nov primer, moramo enega od obstoječih iz njega izbrisati, kar navadno naredimo naključno. Metoda je učinkovita, saj je časovna zahtevnost ne glede na velikost rezervoarja $O(1)$.

Metoda nam torej zagotavlja, da imamo naključen vzorec iz celotne do zdaj videne učne množice.

Pogosto pa želimo novejšim primerom dati večjo težo, za kar moramo vsem primerom na novo izračunati uteži. Časovna zahtevnost je zato proporcionalna z velikostjo rezervoarja, torej $O(k)$, kar je sicer še vedno konstanta, a vendarle traja nekoliko dlje.

Spremembo koncepta lahko zaznamo tako, da skozi čas merimo točnost napovedi skozi čas in v skladu s tem spreminjamo velikost okna. Hkrati pa moramo spremljati tudi lastnosti podatkov skozi čas tako, da opazujemo spremembo različnih porazdelitev skozi čas. Pri drevesih lahko, na primer, primerjamo porazdelitve v različnih vozliščih in se glede na to odločamo, ali je prišlo do spremembe koncepta ali ne. Za samo identifikacijo, kdaj je pride do spremembe koncepta, se lahko uporablja Chernoffova meja, da ocenimo statistično verjetnost, da sta dve porazdelitvi enaki oziroma različni. Če nista, to nakazuje na spremembo koncepta.

Algoritem ADWIN (Adaptive Sliding Window) ocenjuje povprečje števil oziroma bitov v trenutnem oknu podatkovnega toka, s čimer ocenjujemo porazdelitev podatkov. V oknu ohranjamo samo zadnjih n elementov, pri čemer se n dinamično spreminja. Dokler je model skladen s hipotezo in dokler ne pride do sprememb, se n povečuje do neke sprejemljive meje, ko pa zaznamo spremembo koncepta, pa starejši del okna odstranimo, če se preveč razlikuje od novejših podatkov. Dopustno razliko imenujemo stopnja zaupanja $\delta \in [0, 1]$.

Pri odločitvenih drevesih lahko definiramo statično in vzvratno napako, s katerima ocenjujemo porazdelitev, ki prihaja z novimi podatki. Velja, da pride do spremembe koncepta, ko vzvratna napaka preseže vrednost statične napake. To pomeni, da je koncept, ki smo ga zgradili na podatkih v preteklosti, slabši, kot če bi ga odrezali in obdržali le list. Ko do tega pride, je model neuporaben, zato ga je potrebno zavreči in zgraditi novega.

Adaptacija je lahko slepa, kjer preprosto ne zaupamo temu, da do sprememb koncepta ne prihaja, ampak vsake toliko časa zavržemo stare podatke, obdržimo le nove in na novo zgradimo model. Na ta način zagotavljamo, da je model vedno ažuriran glede na nove podatke. Za adaptacijo lahko uporabimo tudi informirane metode, ki spremenijo model le takrat, ko pride do detekcije spremembe koncepta.

Za rešitev vseh teh problematik lahko uporabimo ansambelski pristop. Imamo torej množico modelov določene velikosti, ki jo bomo vzdrževali. Predpostavljamo, da podatki prihajajo iz več porazdelitev, pri čemer za napovedovanje uporabljamo ansambelski pristop. Glede na uspešnost posameznih modelov, ki jo sproti merimo, lahko določene modele zavržemo in generiramo nove. Na tak način sistem ves čas deluje nemoteno, hkarti pa se

tudi prilagajamo različnim spremembam koncepta.

Še ena metoda, ki deluje na drevesih, je Ultra Fast Forest of Trees (UFFT), kjer inkrementalno učimo množico dreves na podatkovnih tokovih. Ta drevesa so tako učinkovita, da v konstantnem času in prostoru procesirajo vsak nov primer, saj gremo skozi podatke le enkrat in jih lahko tudi pozabimo. Drevesa za ocenjevanje atributov uporabljajo hitre odločitve in naivni Bayesov klasifikator v vsakem vozlišču za detekcijo spremembe koncepta. Drevesa lahko medtem tudi uporabljamo za klasifikacijo. Imamo torej nek nov primer, ki pade v nek list, in v vsakem vozlišču ocenjujemo klasifikacijsko točnost naivnega Bayesa oziroma njegovo napako. Ko se ta napaka spremeni za določeno vrednost, ocenimo, da gre za spremembo koncepta, ustrezno poddrevo porežemo in ga zgradimo na novo.

Poglavje 16

Osnove teorije naučljivosti

Zanima nas, ali obstaja program, ki se po končanem branju iz (potencialno neskončnega) vhoda v nekem jeziku nauči pravilo, ki bo znalo razlikovati besede iz tega jezika od besed, ki niso v tem jeziku.

Iz tega izhaja, da se program, ki se zna naučiti dovolj preprosto pravilo za razlikovanje besed, ki so v danem jeziku, od besed, ki niso v tem jeziku, lahko nauči le končno mnogo jezikov. Hkrati velja tudi, da omejitve vplivajo na fleksibilnost učenja, čeprav je omejitev smiselna. Če se bo, recimo, program učil le določene množice jezikov, se morda ne bo mogel naučiti vseh jezikov v tej množici. Za avtomatsko učenje nekaterih neskončnih jezikov mora program včasih nujno predpostaviti, da je na vhodu končen jezik, ki ne presega do sedaj videnih vhodnih podatkov.

16.1 Turingovi stroji

Teorija izračunljivosti pravi, da se vse, kar se da izračunati, da izračunati s Turingovim strojem.

V Turingovem stroju imamo nadzorno enoto, ki se lahko nahaja v enem izmed množice stanj. Stroj začne v definiranem začetnem stanju in konča v enem izmed končnih. Vsak stroj ima tudi bralno-pisalno glavo, od koder bere in zapisuje simbole iz vhodne abecede, ki je podmnožica tračne abecede, ki ima poleg vhodne abecede lahko še prazen simbol (angl. blank). Delovanje Turingovega stroja definira prehodna funkcija, ki pove, v katero stanje se bo stroj prestavil iz določenega stanja pri določenem znaku, kateri znak bo zapisal na trak in ali se bo trak premaknil v levo ali desno.

Turingov stroj je torej sprejemnik jezika, torej vseh besed iz vhodne abecede. Jezik so tiste besede iz vhodne abecede, pri katerih se Turingov stroj iz začetnega stanja v končnem številu korakov ustavi v končnem stanju. Če Turingov stroj besede ne sprejme, se ali ne ustavi v končnem stanju ali pa se nikoli ne ustavi. Turingov stroj lahko intepretiramo tudi kot funkcijo ali kot generator jezika.

16.2 Rekurzivne funkcije

Funkcije preslikajo naravna števila v naravna števila $f : \mathbb{N} \to \mathbb{N}$. Z $\mathcal{F}$ označujemo vse možne funkcije, z $\mathcal{F}^{total}$ opisujemo funkcije, ki so totalne, torej funkcije, ki so povsod definirane, in z $\mathcal{F}^{rec}$ opisujemo rekurzivne funkcije.

Poznamo naslednje osnovne funkcije:

- ničelna funkcija: vsak element preslika v 0,
- funkcija naslednika: vrne naslednika podanega elementa,
- funkcija projekcije: izmed vseh argumentov funkcije vrne enega,
- pravilo substitucije: če imamo k rekurzivnih funkcij, lahko definiramo novo rekurzivno funkcijo f, ki kot argumente sprejme teh k rekurzivnih funkcij,
- pravilo primitivne rekurzije:

$$f(0, x_2, ..., x_k) = g(x_2, ..., x_k)$$

$$f(y + 1, x_2, ..., x_k) = h(y, f(y, x_2, ..., x_k), x_2, ..., x_k)$$

- operator minimizacije: če je f rekurzivna funkcija, ki ima $k + 1$ argumentov, potem je z operatorjem minimizacije μx definirana nova rekurzivna funkcija tako, da je operator minimizacije prva vrednost, pri kateri velja $f = 0$; iščemo torej tak x, za katerega je $f = 0$.

S temi funkcijami lahko definiramo hierarhijo jezikov. Algoritem je podan s prehodno funkcijo, kar lahko zakodiramo z naravnimi števili. To pomeni, da je vsak Turingov stroj naravno število, množica jezikov pa podmnožica potenčne množice vseh naravnih števil. To nakazuje na to, da je vseh možnih Turingovih strojev številno neskončno in vseh možnih jezikov (problemov) neštevno neskončno.

Turingovi jeziki ustrezajo množici rekurzivnih funkcij. Odločljivi jeziki, torej tisti, za katere obstaja Turnigov stroj, ki jih sprejme in se vedno ustavi,

ustrezajo totalnim rekurzivnim funkcija, medtem ko so neturingovi jeziki vsi ostali. Ker lahko vsak Turingov stroj predstavimo z naravnim številom, lahko vsakemu Turingovemu stroju dodelimo tudi indeks. Iz tega sklepamo, da obstaja neskončno Turingovih strojev, ki sprejmejo enak jezik.

Problem izračunljivosti izhaja iz problemov Turingovega stroja. Turingov stroj se bo namreč ustavil, če beseda na vhodu pripada jeziku, sicer pa morda nikoli. Rezultat je, da algoritem, ki bi se vedno ustavil in za vsak Turingov stroj ter vsako vhodno besedo znal odgovoriti na vprašanje, ali beseda pripada Turingovemu stroju, ne obstaja. Podobno velja za rekurzivne funkcije. Funkcija je za dane vrednosti parametrov izračunljiva, če je definirana, sicer pa morda ni izračunljiva. Enako tudi ne obstaja algoritem, ki bi za vsako funkcijo in število znal odgovoriti na vprašanje, ali je funkcija za to število definirana. Ta dva izsledka lahko posplošimo tudi na gramatike in predikatni račun prvega reda.

16.3 Rekurzivno naštevne množice

Naučiti se je možno le Turingovih jezikov, zato je smiselno učence omejiti na tiste jezike, ki so dejansko naučljivi. Razred rekurzivno naštevnih jezikov vsebuje torej tiste jezike, ki so Turingovi.

Definiramo lahko tudi razred rekurzivnih ali odločljivih jezikov, to so jeziki, ki jih nek Turingov stroj sprejme v končnem številu korakov. Zanje velja, da sta rekurzivni jezik in njegov komplement rekurzivno naštevna. Posebej označujemo tudi končne jezike, ki so podmnožica naravnih števil in ki so končni. Pri tem velja: $RE_{fin} \subset RE_{rec} \subset RE$.

Z enovrednostni totalnimi jeziki definiramo vhode, ki predstavljajo funkcije. Če se učenec uči teh jezikov, se v bistvu uči funkcije. Jezik definiramo kot par števil, pri čemer je prva vrednost poljubno naravno število, druga pa njegova preslikava z neko totalno rekurzivno funkcijo. Na tak način se učenec torej uči funkcije.

16.4 Formalna teorija učenja

Na vhodu imamo besedilo, torej neskončno število opazovanj oziroma pozitivnih učnih primerov ali besed iz jezika. Naloga učenca je identificirati jezik. To pomeni, da morajo vsa števila v besedilu in nobeno drugo biti v našem jeziku. Na vhodu imamo vselej končno mnogo opazovanj, saj bi verifikacija hipoteze, ki jo postavi učenec zahtevala neskončno mnogo opazovanj. Učenec se mora torej naučiti indeksa Turingovega stroja, ki preslika neko naravno

število v neko drugo naravno število, a nikoli ne bo vedel, ali se je naučil pravilne hipoteze.

Pravzaprav pa za dokaz, da beseda ni v jeziku, zadošča tudi končno mnogo opazovanj, saj je funkcija enolična in z enim pozitivnim učnim primerom dobimo neskončno negativnih učnih primerov.

Funkcija φ identificira nek jezik, če identificira vsako besedilo iz tega jezika.

16.4.1 Izreki teorije učenja

Množica končnih jezikov je naučljiva. Velja tudi, da se končne jezike da naučiti z rekurzivnimi učenci.

Naučljiva je tudi množica vseh enovrednostnih totalnih jezikov, vendar to velja le za nerekurzivne učence.

Če imamo nek končen jezik, ki mu dodamo nek poljuben neskončen rekurzivno naštevni jezik, dobljena množica ni naučljiva, ne z rekurzivnimi ne z nerekurzivnimi učenci.

Če neka množica jezikov ni naučljiva, potem ni naučljiva tudi nobena njena nadmnožica. Iz tega sledi, da množica vseh rekurzivno naštevnih jezikov in množica vseh rekurzivnih jezikov nista naučljivi. Velika večina problemov torej s Turingovimi stroji ni rešljiva.

16.4.2 Naučljivost glede na lastnosti učnih funkcij

Strategija učenja je neka lastnost, ki jo damo učencem. Podana je z množicami jezikov, ki jih identificirajo funkcije z dano lastnostjo.

Pravimo, da strategija učenja omejuje drugo strategijo, če velja, da je njun presek podmnožica druge strategije. Če torej učencu dodamo lastnost izgubimo na naučljivosti.

Če se omejimo na rekurzivne učence, je naučljivost slabša. Naučljiva množica jezikov ni naučljiva z rekurzivnimi učenci, medtem ko totalnost ne omejuje naučljivosti.

Preprosta hipoteza je taka, pri kateri porabimo najmanj simbolov za zakodiranje ustreznega Turingovega stroja. Pri tem jezik Turingovega stroja mora ustrezati vhodu. Zanima nas torej, koliko se hipoteza razlikuje od najpreprostejše hipoteze, za kar uvedemo izračunljivo totalno funkcijo. Ta je preprosta, če je število porabljenih simbolov manjše od minimalnega števila simbolov za zakodiranje Turingovega stroja, ki sprejme ta jezik. Vendar se s tem ko zahtevamo preprostost, določenih jezikov ne moremo naučiti. Če je učenec Turingov in preprost, se lahko torej nauči le končno mnogo jezikov.

Netrivialna funkcija je tista, ki vedno vrne neskončno hipotezo. Če učence omejimo na to, da vedno postavljajo neskončne hipoteze, se bo naučljivost omejila tudi v prostoru neskončnih jezikov.

Odgovornost učenca je ta, da vedno postavi hipotezo, ki je nadmnožica tega, kar je do zdaj prebral na vhodu. Če zahtevamo, da je odgovoren, se naučljivost zmanjša. Velja tudi, da je vsak netrivialen učenec odgovoren.

Pri Popperovskih funkcijah učenec vedno predpostavi, da je na vhodu funkcija, ki se je mora naučiti, torej enovrednostne totalne jezike. Ti jeziki so vsi neskončni, saj je funkcija definirana na vseh naravnih številih. Vsak jezik, ki je enovrednostno totalen je neskončen. Tudi s Popperovskimi funkcijami omejujemo naučljivost rekurzivnih funkcij, ne omejujemo pa naučljivosti nerekurzivnih funkcij.

Te tri vrste funkcij so v relaciji $\mathcal{F}^{Popperian} \subset \mathcal{F}^{nontrivial} \subset \mathcal{F}^{accountable}$.

Preudarna je tista funkcija, hipoteza katere je sposobna identificirati jezik. Preudarni učenci se lahko naučijo razlikovati vse končne jezike med seboj. Pri tem ne omejuje ne rekurzivnih ne nerekurzivnih učencev.

Konsistentnost pomeni, da hipoteza vsebuje vse do zdaj videne besede, torej da ne postavlja nesmiselnih hipotez. S konsistentnostjo ne omejujemo naučljivosti zbirke končnih jezikov, omejujemo pa naučljivost učenca za določene zbirke neskončnih jezikov.

Konservativnost pomeni, da učenec vztraja pri hipotezi, dokler na vhod ne dobi protiprimera. Naučljivost končnih jezikov se s tem ne spremeni, za ostale jezike pa izgubimo naučljivost.

Previden učenec je tak, ki samo generalizira svoje hipoteze, torej mora biti toliko previden, da ne generalizira preveč, ker nikoli ne specializira svoje hipoteze. Previdnost že sama po sebi omejuje naučljivost, če jo dodamo h konservativnosti, pa ne omejimo naučljivosti konservativnih učencev. Prav tako previdnost ne omejuje naučljivosti končnih jezikov.

Odločen učenec nikoli ne ponovi hipoteze, ki jo je že zavrgel. Odločnost ne spremeni naučljivosti končnih jezikov, konsistentni in konservativni učenci pa se lahko naučijo manj kot odločni. Odločnost tudi ne omejuje naučljivosti neskončnih jezikov.

Zanesljivost govori o tem, da če funkcija na vhodu konvergira, potem je funkcija tudi sposobna identificirati besedilo na vhodu. Zanesljivost je omejena samo na končne jezike, ne glede na to, ali je učenec rekurziven ali ne.

Samozavesten je tisti učenec, ki vedno konvergira za poljuben vhod. Zanesljivost in samozavestnost ne gresta skupaj, saj bi to pomenilo, da bi se učenec lahko naučil poljubnega jezika, kar pa vemo, da ni možno. Samozavesten učenec omejuje naučljivost neskončnih jezikov.

16.4.3 Naučljivost glede na lastnosti vhodnih podatkov

Evidenčna relacija govori o tem, kakšne lastnosti ima tekst.

Šumno besedilo je tako, da jeziku dodamo neko množico besed, ki niso v tem jeziku. Šumno besedilo omejuje naučljivost tako za rekurzivne kot za nerekurzivne učence. Dva jezika lahko pri šumnem tekstu ločimo med sabo le, če se ta dva jezika razlikujeta v neskončnem številu besed. Če sta dva jezika med seboj različna, ne samozavestna ne odločna funkcija ne moreta ločiti jezikov med seboj.

Besedilo je pomanjkljivo, če mu manjka končno mnogo besed iz tega jezika. Tudi pri pomanjkljivih besedilih se naučljivost zmanjša tako za rekurzivne kot za nerekurzivne učence. Enako kot pri šumnih besedilih velja, da če želimo ločiti dva jezika med seboj s pomanjkljivim tekstom, mora biti razlika med njima neskončna. Je pa naučljivost pomanjkljivega besedila večja od šumnega. Bolje je torej imeti manjkajoč podatek kot napačnega.

Imamo tudi kombinacijo šuma in pomanjkljivosti, čemur pravimo nepopolnost. Dodane so torej nekatere besede, ki niso v jeziku, in manjkajo nekatere besede, ki so v jeziku. Tudi nepopolni tekst omejuje naučljivost, v splošnem pa velja, da nepopoln in šumni tekst enako omejujeta naučljivost.

Če v besedilo dodamo pozitivne in negativne učne primere, to imenujemo obveščevalec. K samemu učnemu primeru torej dodamo tudi oznako, ali primer pripada jeziku ali ne. Pri tem se naučljivost poveča tako za nerekurzivne kot za rekurzivne učence. Pomemben rezultat je, da pozitivni in negativni učni primeri omogočajo nerekurzivnim učencem, da se naučijo razlikovati vse rekurzivno naštevne jezike, ne pa tudi rekurzivnim.

Tekst s prerokom pomeni, da na vhodu postavljamo vprašanja. Učenec je torej aktivni učenec, ki generira besedilo tako, da vsakič postavi vprašanje preroku za neko besedo. Izkaže se, da sta naučljivosti obveščevalca in preroka enaki. Poleg običajnega imamo še končnega preroka, kjer imamo tekst, vendar lahko preroku postavimo končno mnogo vprašanj za neke zanimive primere. Končni prerok in navadno besedilo sta enako naučljiva za rekurzivne in nerekurzivne učence.

16.5 Implikacije za strojno učenje

Sama teorija predpostavlja, da imamo neskončna besedila in da pri učenju nismo ne časovno in ne prostorsko omejeni, kar v praksi skoraj nikoli ne drži. Kljub temu lahko iz teorije povzamemo nekatere izsledke:

- velika večina stvari ni naučljivih,

- totalne rekurzivne funkcije niso naučljive z rekurzivnimi učenci, torej ne obstaja univerzalni algoritmični učenec, ki bi se naučil vseh rekurzivnih jezikov ali funkcij,
- časovna omejitev učencev je bolj omejujoča za inkrementalne kot za neinkrementalne učence,
- če želimo, da učenec vrne relativno preproste hipoteze, je omejen na končno število jezikov,
- če algoritmu omejimo prostor hipotez na jezike, ki se jih dejansko uči, se lahko zgodi, da zaradi te omejitve naučljivost pade, zato je pomembno algoritmu dovoliti, da dela očitne napake,
- za učenje nekaterih neskončnih jezikov je pomembno, da algoritem naredi napako in postavi končno hipotezo,
- algoritmu moramo dovoliti, da je nekonsistenten,
- če imamo šumno besedilo, algoritem ne sme vedno konvergirati in biti preveč odločen,
- šum v podatkih je večja slabost kot pomanjkljivi podatki,
- negativni primeri močno izboljšajo naučljivost.

Rešitve starih izpitov

Legenda:

- ✗: naloga ni rešena ali je delno rešena,
- ?: rešitev naloge ni nujno pravilna,
- ✓: rešitev naloge je pravilna.

1. izpit, 24. 1. 2020 ?

Naloga 1 ✓

MDL (minimum description length) je temeljni princip strojnega učenja.

a) Definiraj ga!

- Princip najkrajšega opisa pravi, da je najpreprostejša razlaga, ki čim bolj ustreza vhodnim podatkom in predznanju, najbolj zanesljiva ali verjetno. Naj bo $\mathcal{H}$ prostor vseh možnih hipotez in $H \in \mathcal{H}$, ena od možnih hipotez. B naj bo predznanje, E vhodni podatki in $P(H|B)$ apriorna verjetnost informacije. Apriorna količina informacije hipoteze H je definirana kot:

$$I(H|B) = -log_2 P(H|B) \tag{16.1}$$

Aposteriorna količina informacije hipoteze H je definirana kot:

$$I(H|E,B) = -log_2 P(H|E,B) \tag{16.2}$$

Optimalna hipoteza je definirana kot:

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

Po Bayesovem teoremu velja:

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

Ker je $I(E|B)$ konstanta, neodvisna od hipoteze, dobimo:

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

Iz tega izvemo, da je optimalna hipoteza tista, ki je točna (ima majhno napako $I(E|H,B)$) in preprosta (majhen $I(H|B)$).

b) Kako ga uporabljajo algoritmi:

- gradnja odločitvenega drevesa: *uporabljamo ga za rezanje odločitvenih dreves. Ocenjuje se dolžina kodiranja drevesa in porazdelitve razredov učnih primerov v listih.*
- delno naivni bayesov klasifikator: *pri združeavnju atributov - iščemo kompromis med nenaivnostjo in zanesljivostjo aproksimacije verjetnosti.*
- usmerjene večnivojske umetne nevronske mreže: *uporabljajo ga v stilu, da se mreža ne nauči preveč natančno na podatkih in preprečujejo prenaučenost (overfitting/overlearning).*

c) Kako ga lahko uporabimo za ocenjevanje atributov?

- *V klasifikacijiskih problemih, je atribut tem bolj pomemben, čim bolj je kompresiven. Ne precenjuje večvrednostih atributov. Odkrije neuporabne (nekompresivne) atribute. MDL je najprimernejša metoda glede pristranskosti pri ocenjevanju večvrednostnih atributov. Če je $MDL < 0$, potem je atribut nekompresiven in zato neuporaben.*
- *V regresiji moramo izbrati ustrezno kodiranje realnih števil, ki ga potem uporabimo za kodiranje razredov in napak. Ko izberemo kodiranje, lahko ocenjujemo kvaliteto atributa tako kot pri klasifikaciji. Namesto porazdelitve po razredih kodiramo povprečno vrednost razreda in odstopanje od povprečne vrednosti razreda za vask učni primer.*

d) Kaj pomeni, da je hipoteza kompresivna?

- *Če je hipoteza kompresivna, pravimo da je hipoteza dobra. Problem si želimo opisati s čim manj biti. Če je MDL za nek atribut < 0, potem pravimo da atribut ni kompresiven.*

- *Zakodirana hipoteza in zakodirani podatki so pri dani hipotezi skupaj manjši od izvornih podatkov*

Naloga 2 ✓

Opiši ideje metod za ocenjevanje atributov. Za katere probleme se lahko uporabljajo, kakšne so njihove prednosti in slabosti?

a) Gini index

- *Gini indeks je definiran kot razlika med 1 in vsoto kvadratov apriornih verjetnosti po razredih.*

$$1 - \sum_{i=1}^{n} (P_i)^2$$

Kakovost atributa ocenimo z razliko med priornim in pričakovanim Gini indeksom. To nam pove, kakšna je povprečna verjetnost, da dva naključna primera pripadata istemu razredu pri določeni vrednosti atributa.

- *Ima enake lastnosti kot informacijski prispevek in prav tako precenjuje vrednosti večvrednostnih atributov.*
- *V praksi se uporablja za ocenjevanje linearnih atributov.*

b) razlika (sprememba) variance

- *Varianca je mera nečistoče v zveznem prostoru. Definirana je kot kvadrat odstopanja primera od povprečja.*
- *Ocena atributa je razlika med apriorno in posteriorno varianco. Razlika variance je nenegativna in njena slabost je, da precenjuje večvrednostne atribute.*
- *Metoda se zato v praksi uporablja za binarne atribute.*

c) ReliefF

- *Relief temelji na ideji, da iz množice vzamemo naključen primer in opazujemo njegov najbližji primer iz nasprotnega in najbližji primer iz istega razreda.*
- *Algoritem ima časovno zahtevnost $O(n*m*a)$, kjer je n število primerov, m število iteracij in a število atributov. Glede na časovno zahtevnost je precej potraten, a še vedno v krajšem času reši problem, ki je bil sprva videti eksponenten.*

- *Je sposoben ocenjevati tako diskretne kot zvezne atribute in njihov kontekst. Obenem ne precenjuje večvrednostnih atributov, vendar pa je občutljiv na šum v podatkih in omejen na dvorazredne probleme.*

d) RReliefF

- *V regresiji ne moremo uporabiti najbližjih sosedov, zato namesto tega skušamo oceniti "verjetnost", da dva primera pripadata različnim razredom, ki jo modeliramo z razdaljo.*
- *Časovna zahtevnost je enaka kot pri navadnem Relief-u, torej* $O(n * m * a)$.
- *Z njim lahko ocenjujemo tako diskretne kot zvezne atribute, pri čemer ne precenjujemo večvrednostnih atributov.*

Naloga 3 ?

Kakšno preiskovalno strategijo uporabljajo naslednji algoritmi (odgovore obrazloži)?

a) Algoritem apriori za gradnjo povezovalnih pravil

- *Uporablja omejeno izčrpno iskanje.*
- *Algoritem usmerjata parametra zaupanje in podpora, ki sta smiselno definirana, saj nas navadno zanimajo najpogostejša in najnatančnejša pravila. Najprej poiščemo množice izdelkov, ki imajo zadostno podporo, nato pa iz njih generiramo vsa povezovalna pravila, ki zadoščajo minimalnemu zaupanju.*

b) Algoritem za gradnjo odločitvenih dreves

- *Uporablja požrešno iskanje, iskanje s pogledom naprej ali pa genetski algoritem. (TBA)*
- *Požrešno: Na vsakem nivoju izberemo najboljši atribut, pri čemer smo kratkovidni.*

c) Usmerjene večnivojske nevronske mreže

- *Gradientno iskanje/simulirano ohlajanje (TBA)*
- *Uporabljajo gradientno iskanje pri backpropagation-u. Odvod narekuje, kako spremeniti uteži na povezavah, da bo napaka čim manjša.*

d) Naivni bayesov klasifikator

- *Ne opravlja raziskovanja, samo poračuna potrebne verjetnosti!*
- *Naivni Bayes predpostavi, da so atributi pri danem razredu med seboj neodvisni. Metoda deluje tako, da izračunamo verjetnost, da nek primer pripada določenemu razredu pri danih vrednostih atributov, in ga klasificiramo v tistega z najvišjo vrednostjo.*

Naloga 4 ✓

a) Kaj pomeni preveliko prilagajanje učni množici (overfitting)? Zakaj do njega pride?

- *Preveliko prilagajanje učni množici pomeni, da se je model naučil podrobnosti in šuma v učni množici do te mere, da negativno vpliva na rezultate.*
- *Do prevelikega prilagajanja učni množici pride, če imamo previsoko varianco. To pa se zgodi takrat, ko so parametri modela slabo nastavljeni.*

b) Kako se proti overfittingu borijo naslednji algoritmi?

- Gradnja odločitvenega drevesa
  - *Ne zgradimo celotnega drevesa, določimo največjo globino ter največjo velikost lista.*
  - *Zgradimo celotno drevo ter ga nato obrežemo (ang. prune)*
- Naivni Bayesov klasifikator
  - *Je precej odporen na overfitting.*
- K-najbližjih sosedov
  - *Povečamo vrednost parametra k.*
- linearna regresija
  - *Če imamo dovolj podatkov je model, zaradi svoje preprostosti, precej odporen na overfitting.*
  - *(Lahko uporabimo lokalno uteženo regresijo.*
  - *Lahko ocenimo optimalno λ ki mimizira napako testne množice pri prečnem preverjanju. Napaka testne množice bi se morala zmanjšati, ko povečamo lambdo.*
  - *L1 regularizacija ali Lasso*

  - *L2 regularizacija ali Ridge*
- Hopfieldova nevronska mreža
  - *Je odporna na overfitting.*
- Usmerjena večnivojska nevronska mreža
  - *Overfittingu se lahko izognemo z avtomatskim iskanjem primerne topologije mreže.*

2. izpit, 4. 2. 2020 ✓

Naloga 1 ✓

a) Opiši metodo radviz za vizualizacijo podatkov pri klasifikaciji. Koliko atributov naenkrat lahko vizualiziraš s to metodo? Zakaj?

- *Pri metodi Radviz so atributi enakomerno razporejeni na stranici kroga, primeri pa so izrisani v notranjosti kroga, in sicer tam, kjer je vsota vrednosti njihovih atributov najbolj utežena. Razredi primerov so označeni z barvami, da lahko opazimo, koliko kombinacija izrisanih atributov pripomore k razločevanju razredov. Z metodo lahko teoretično vizualiziramo poljubno število atributov, vendar jih v praksi zaradi lažje vidljivosti vizualiziramo do največ 10.*

b) Koliko je vseh različnih vizualizacij podatkov z metodo radviz, ki so opisani z a zveznimi atributi, pri čemer vsaka vizualizacija uporabi k atributov $k < a$?

- $\frac{(k-1)!}{2}$, pri čemer k predstavlja število atributov

c) Kako algoritem vizrank razvrsti vizualizacije pri metodi radviz po zanimivosti za uporabnika?

- *Vizrank vzame koordinate točk in njihove razrede kot učno množico, za katero napove točnost. Dobljena točnost predstavlja zanimivost vizualizacije.*

Naloga 2 ✓

a) Zakaj je potrebno napovedne verjetnosti pri nekaterih klasifikatorjih kalibrirati?

- *Ker so nekateri klasifikatorji pristranski, zaradi česar je porazdelitev verjetnosti ciljnih razredov pri njih nepravilna.*

b) Katere metode za kalibracijo poznaš (definiraj jih) in kakšne so njihove lastnosti?

- *Plattova metoda uporablja sigmoidno funkcijo in je zato primerna samo za klasifikatorje, ki imajo zgoščeno porazdelitev verjetnosti okrog 0.5.*
- *Izotonična regresija je splošnejša, saj skuša poiskati monotono naraščajočo funkcijo, ki minimizira kvadratno napako. Rabimo večjo kalibracijsko množico. Vzame sosedne primere, če nista monotona, združi in izračuna povprečje - to ponavlja.*

c) Ali lahko s kalibracijo izboljšamo klasifikacijsko točnost? Zakaj?

- *Ne, saj s kalibracijo samo spremenimo verjetnostno porazdelitev razredov, ne pa tudi dejanske klasifikacije posameznih primerov. Izboljšuje pravilnost interpretacije.*

Naloga 3 ✓

a) Čemu je namenjena regularizacija pri umetnih nevronskih mrežah?

- *Regularizacija je namenjena zmanjševanju prileganja učni množici. To lahko, recimo, dosežemo z odstranjevanjem nekaterih nevronov, s čimer poenostavimo topologijo mreže ali pa z zmanjševanjem ali omejevanjem vrednosti uteži.*

b) Katere metode regularizacije poznaš (definiraj jih) in kakšne so njihove lastnosti?

- *L1 regularizacija k napaki prišteje vsoto absolutnih vrednosti uteži, s čimer lahko izloči nekatere nevrone.*
- *L2 regularizacija k napaki prišteje vsoto kvadratov vrednosti uteži, s čimer se vrednost uteži v mreži zmanjša.*
- *Max-norm omeji vrednost posamezne uteži.*
- *Pri principu preskakovanja (droupout) za vsak učni primer posamezen nevron upoštevamo le z določeno verjetnostjo, zato bomo nekatere preprosto izpustili.*

c) Če bi imel na voljo neomejeno količino časa, kako bi lahko poiskal čim bolj optimalno strukturo usmerjene večnivojske nevronske mreže?

- *Uporabili bi zelo veliko mrežo z veliko nevroni in skritimi nivoji in jo z vzvratnim razširjanjem napake in regularizacijo postopoma krčili oziroma prilagajali učenju našega problema.*

d) Zakaj Hopfieldova nevronska mreža ne potrebuje regularizacije?

- *Ker se Hopfieldova nevronska mreža navadno ne prilagodi preveč učni množici, saj je njeno izvajanje stabilno.*

Naloga 4 ✓

Pri rudarjenju besedil je zelo pomembna predstavitev podatkov tj. besedil.

a) Katere predstavitve besedil poznaš (definiraj jih) in kakšne so njihove lastnosti?

- *Vreča besed (Bag of words): štejemo, kolikokrat se vsaka beseda iz neke množice besed pojavi v našem besedilu. Metoda je preprosta, a s tem izgubimo vrstni red besed, hkrati pa je atributni prostor lahko zelo velik.*
- *N-grams: upošteva tudi N besed okoli same besede. S tem besedi doda kontekst, saj upošteva tudi besede s katerimi se ta beseda pojavi.*
- *Pri TF-IDF skušamo identificirati besede, ki so specifične za vsako besedilo, s čimer prav tako ne upoštevamo konteksta besed.*
- *Word2Vec vsako besedo predstavi z vektorjem v večdimenzionalnem prostoru, pri čemer velja, da so besede, ki nastopajo v podobnem kontekstu, predstavljene s podobnimi vektorji.*

b) Ali lahko katero od njih uporabiš za predstavitev proteinov? Zakaj?

- *Da, za predstavitev proteinov lahko uporabimo Word2Vec. Z metodo zaradi njenih zgoraj opisanih lastnosti lahko potem primerjamo med seboj podobne proteine.*

3. izpit 24.8.2020 ?

Naloga 1 ✓

a) V kakšni relaciji sta J-mera in informacijski prispevek? Ali je katera od teh dveh mer lahko negativna? Kaj pomeni, če je J-mera enaka 0?

- *Informacijski prispevek nam pove kako pomemben je določen atribut. J-mera ocenjuje le eno vrednost atributa.*
- *Relacija med informacijskim prispevkom in J-mero: vsota vseh J-mer preko vseh vrednosti atributa je enaka ravno informacijskemu prispevku.*
- *Ne, nobena od mer ne more biti negativna.*
- *Če je J-mera = 0, to pomeni, da je vrednost atributa slaba.*

b) Katere mere se lahko neposredno uporabljajo za ocenjevanje zveznih atributov v klasifikaciji in regresiji? Kako lahko prilagodimo ostale mere za ocenjevanje zveznih atributov?

- *Regresija:*
  - *sprememba variance (change of variance)*
  - *RReliefF*
  - *MDL*
- *Klasifikacija:*
  - *gain ratio*
  - *info gain*
  - *ReliefF*
  - *MDL*
  - *merjenje razdalj*
  - *J-mera*
- *Zvezne atribute lahko diskretiziramo in jih uporabimo pri klasifikacijskih metodah (npr. z izdelavo intervala).*

c) V čem je ideja RReliefF-a glede na ReliefF? Kakšna je njuna časovna zahtevnost?

- *ReliefF je algoritem, ki se ga uporablja za izbiranje značilk. RReliefF je nadgradnja algoritma Relief, ki se uporablja za regresijske probleme. RReliefF namesto najbližjih sosedov, kakor razdaljo med atributi ocenjuje ReliefF, oceni verjetnost, da imata dva primera različno ciljno vrednost, kar modeliramo z razdaljo.*
- *RELIEF:* $O(n^2)$
- *ReliefF:* $O(n * m * a)$
- *RReliefF:* $O(n * m * a)$

Naloga 2 ✓

a) V kakšni relaciji sta princip MDL in kompromis med pristanskostjo in varianco? V kakšnih primerih obstaja nevarnost prevelikega prileganja podatkom?

- *Če želimo minimizirati varianco, moramo poenostaviti hipotezo (najkrajši opis), tako, da zmanjšamo število parametrov, vendar s tem hkrati povečujemo pristranskost. V nasprotnem primeru, če želimo minimizirati pristranskost, moramo povečati kompleksnost hipoteze (kar naredimo s povečevanjem števila parametrov), vendar pri tem pride od prevelikega prileganja učni množici (Do overfittinga pride, če imamo previsoko varianco).*

b) Kako se lahko uporabi princip MDL pri razvrščanju primerov v gruče (clustering)?

- *Ideja MDL pri razvrščanju v gruče je, da želimo minimizirati število gruč, hkrati pa minimizirati premer vsake gruče.*
- *Ce imamo premalo gruc, bomo porabili prevec podatkov za kodiranje elementov v teh grucah, ce pa imamo prevec gruc (ekstrem je to, da je vsak element v svoji gruci), bomo pa porabili prevec podatkov za kodiranje gruč.*

c) Opiši stremljenje (bootstrapping). Ali ga lahko uporabimo za oceno variance točnosti/napake?

- *Stremljenje je postopek, ki nam omogoča ocenjevanje zanesljivosti ocen, ki jih dobimo iz učne množice.*
- *Pri stremljenju (bootatrappingu) z vračanjem naključno izberemo določeno število primerov izmed vseh. Z njim je možno oceniti točnost, in sicer tako, da postopek večkrat ponovimo, zberemo točnosti v obliki histograma in tako vidimo, kakšna je porazdelitev napovednih točnosti.*
- *Da, lahko ga uporabimo za oceno variance točnosti/napake.*
- *Denimo, da imamo veliko vrečo modrih in rumenih kroglic in da iz nje naključno izvlečemo 100 kroglic, od tega 70 modrih in 30 rumenih. Iz tega sklepamo, da je delež modrih kroglic v vreči 0.7. Zanima nas, kako dober je ta približek? Postopek ponovimo, le da tokrat kroglice vračamo. Ko torej izvlečemo kroglico, pogledamo, kakšne barve je, in jo vrnemo nazaj v vrečo. Poskus lahko večkrat ponovimo in na podlagi več izračunanih deležev modrih*

kroglic sestavimo histogram ter na tak način ocenimo zanesljivost naše prvotne ocene.

Naloga 3 ✓

Ali lahko naslednje pristope uporabimo za izboljšanje klasifikacijske točnosti (zakaj in kako):

a) Razlaga posameznih napovedi?

- *Neposredno ne, saj s tem dobimo le razlago odločitve modela. Lahko pa na ta način identificiramo primere, ki ne prispevajo k izboljšanju, in jih po potrebi odstranimo.*

b) Ocenjevanje zanesljivosti posameznih napovedi?

- *Neposredno ne, saj s tem samo ocenimo, s kakšno verjetnostjo je dobljen rezultat pravilen, s čimer ne spreminjamo klasifikacijske točnosti. Lahko pa tako identificiramo napačno klasificirane primere in skušamo ugotoviti, zakaj je do tega prišlo.*

c) Kode za popravljanje napak (error correcting output codes)?

- *Da, saj večrazredni problem razstavi na kombinacije le-teh, kjer v vsakem skušamo razločiti enega ali več razredov od vseh ostalih. Na ta način lahko popravimo določeno število napak pri klasifikaciji, s čimer lahko izboljšamo klasifikacijsko točnost.*

d) Prečno preverjanje?

- *Ne, saj se prečno preverjanje uporablja samo za ocenjevanje klasifikacijske točnosti, ne pa tudi za njeno izboljšanje.*

e) Kalibracija napovednih verjetnosti?

- *Ne, kalibracija verjetnosti samo izenači verjetnostno porazdelitev napovedi in neposredno ne izboljšuje klasifikacijske točnosti.*

Naloga 4 ✓

a) Definiraj TAN (tree augmented Naive Bayes). Kakšna je časovna zahtevnost gradnje TAN?

- *Ideja je, da naivnemu Bayesu dodamo podgraf, ki v obliki drevesa povezuje vse atribute med seboj. S tem algoritmu omogočimo, da sam poišče odvisnosti med atributi, za katere naivni Bayes predpostavlja, da jih ni. Drevesna struktura pa ob enem bistveno zmanjša možnost pojava prevelikega prileganja.*
- *Časovna zahtevnost iskanja minimalnega vpetega drevesa je* $O(m \log n)$ *ali* $O(a^2 \log a)$
- *Časovna zahtevnost gradnje TAN drevesa je:* $O(na^2c+a^2 \log a$*, pri čemer je n število primerov, a število binarnih atributov, c število razredov.*
- *Algoritem:*
  - *Vhod: set učnih primerov*
  - *Izhod: Tree augmented Naive Bayes klasifikator*
  - *Najprej se izračuna pogojena skupna informacija* $I(A_i; A_j|C)$*, kjer C predstavlja labelo razreda med vsemi pari atributov* A_i *in* A_j *po enačbi 16.3. Nato zgradimo neusmerjen atributni podgraf in označimo robove med vsemi pari atributov* A_i *in* A_j *s pripadajočo skupno pogojeno informacijo* $I(A_i; A_j|C)$*.*
  - *Sledi iskanje največjega vpetega drevesa v atributnem podgrafu. Nato spremenimo neusmerjeno maksimalno vpeto drevo v usmerjen graf tako, da je eno vozlišče koren, robovi pa so usmerjeni stran od korena. V atributnem podgrafu, ki je sedaj drevo vključimo razred in robove, ki povezujejo razred z vsakim atributom.*

$$I(A_i; A_j|C) = H_{A_1|C}+H_{A_2|C}-H_{A_1A_2|C} = H_{A_1C}+H_{A_2C}-H_C-H_{A_1A_2C} \tag{16.3}$$

b) Definiraj napovedni problem, kjer pričakujemo, da bo Naivni Bayesov klasifikator boljši od TAN.

- *Pričakujemo, da se bo naivni Bayes obnesel bolje, ko med atributi ni nobene korelacije ali pa ko imamo, na primer, dva atributa, od katerih eden vpliva na ciljno spremenljivko, eden pa ne.*

c) Opiši idejo gradnje napovednega problema, kjer pičakujemo, da bo TAN deloval bolje kot Naivni Bayesov klasifikator.

- *Pričakujemo, da se TAN obnese bolje v primerih, ko obstaja korelacija med atributi, na primer pri modeliranju funkcije XOR.*

Poleg tega se bo bolje odnesel, ko imamo opravka z zveznimi atributi.

1. izpit, 24. 1. 2018

Naloga 1 ✓

a) What is the idea of RReliefF in comparison to ReliefF? What is the time complexity of both algorithms?

- *RReliefF deluje na enak način kot ReliefF, le da je prilagojen za regresijo. Namesto najbližjih sosedov, kakor razdaljo med atributi ocenjuje ReliefF, oceni verjetnost, da imata dva primera različno ciljno vrednost, kar modeliramo z razdaljo.*
- *Časovna zahtevnost obeh algoritmov je kvadratna glede na število primerov.*

b) Define sensitivity and specificity. Which one is more important?

- *Senzitivnost nam pove, koliko primerov, ki pripadajo nekemu razredu, smo dejansko pravilno identificirali.*
- *Specifičnost pa nam pove, koliko primerov, ki ne pripadajo nekemu razredu, smo dejansko tako identificirali.*
- *Katera je pomembnejša, je odvisno od problema, ki ga rešujemo. V medicinski diagnostiki bi na primer, bila pomembnejša senzitivnost, saj ta predstavlja delež vseh obolelih, ki smo jih identificirali za bolne. Če bolnega ne zdravimo, lahko ima to namreč precej večje posledice kot če zdravimo zdravega človeka.*

c) What is the idea of the MDL evaluation of the attribute quality?

- *MDL informacijski prispevek nekega atributa enači s kompresivnostjo atributa in pravi, da je atribut tem boljši, čim bolj kompresiven je (se pravi je MDL tega večji od 0).*

Naloga 2 ✓

a) Describe bootstrapping. Can it be used to estimate the variance of the accuracy/error?

- *Stremljenje je postopek, ki nam omogoča ocenjevanje zanesljivosti ocen, ki jih dobimo iz učne množice.*

- *Pri stremljenju (bootatrappingu) z vračanjem naključno izberemo določeno število primerov izmed vseh. Z njim je možno oceniti točnost, in sicer tako, da postopek večkrat ponovimo, zberemo točnosti v obliki histograma in tako vidimo, kakšna je porazdelitev napovednih točnosti.*
- *Da, lahko ga uporabimo za oceno variance točnosti/napake.*
- *Denimo, da imamo veliko vrečo modrih in rumenih kroglic in da iz nje naključno izvlečemo 100 kroglic, od tega 70 modrih in 30 rumenih. Iz tega sklepamo, da je delež modrih kroglic v vreči 0.7. Zanima nas, kako dober je ta približek? Postopek ponovimo, le da tokrat kroglice vračamo. Ko torej izvlečemo kroglico, pogledamo, kakšne barve je, in jo vrnemo nazaj v vrečo. Poskus lahko večkrat ponovimo in na podlagi več izračunanih deležev modrih kroglic sestavimo histogram ter na tak način ocenimo zanesljivost naše prvotne ocene.*

b) Which methods for calibrating prediction probabilities do you know and what are their properties?

- *Za kalibracijo verjetnosti poznamo Plattovo metodo in izotonično regresijo. Pri Plattovi metodi porazdelitev verjetnosti pošljemo skozi sigmoidno funkcijo, zato je ta metoda uporabna samo pri modelih s sigmoidno pristranskostjo, na primer pri SVM, boostingu in včasih k najbližjih sosedov. Pri izotonični regresiji pa skušamo najti monotono naraščajočo funkcijo, ki minimizira kvadratno napako. Ker uporabljamo monotono naraščajočo funkcijo, je splošnejša, a zanjo potrebujemo večjo kalibracijsko množico.*

Naloga 3 ✓

a) Define TAN (Tree augmented Naïve Bayes). What is time complexity of building TAN?

- *Ideja TAN je, da naivnemu Bayesu dodamo podgraf, ki v obliki drevesa povezuje vse atribute med seboj. Tako omogočimo iskanje odvisnosti med atributi, za katere naivni Bayes predpostavlja, da jih ni.*
- *Časovna zahtevnost iskanja minimalnega vpetega drevesa je* $O(m \log n)$ *ali* $O(a^2 \log a)$

- *Časovna zahtevnost gradnje TAN drevesa je: $O(na^2c+a^2 \log a$, pri čemer je n število primerov, a število binarnih atributov, c število razredov.*

b) Define a prediction problem where Naïve Bayes is expected to be better than TAN.

- *Pričakujemo, da se bo naivni Bayes obnesel bolje, ko med atributi ni nobene korelacije ali pa ko imamo, na primer, dva atributa, od katerih eden vpliva na ciljno spremenljivko, eden pa ne.*

c) Give the idea of construction of a prediction problem where TAN is expected to be better than Naïve Bayes.

- *Pričakujemo, da se TAN obnese bolje v primerih, ko obstaja korelacija med atributi, na primer pri modeliranju funkcije XOR.*

Naloga 4 ✓

a) What kind of text representations are used for text mining and what are their properties?

- *Poznamo vrečo besed, TF-IDF in Word2Vec. Pri vreči besed preprosto preštejemo, kolikokrat se določena beseda iz neke množice besed pojavi v našem besedilu. Pri TF-IDF skušamo najti besede, ki so specifične za vsak dokument. Pri Word2Vec besede preslikamo v večdimenzionalne vektorje, za katere velja, da so si blizu, če besede nastopajo v podobnem kontekstu.*

b) Do deep neural networks use the principle of multiple explanations? What about the MDL principle?

- *Globoke nevronske mreže uporabljajo princip večkratne razlage, saj v osnovi gre za ansambel množice podmrež, ki rešujejo enak problem.*
- *Princip MDL uporabljajo z regularizacijo (npr. kaznovanje prevelikih uteži, odstranjevanje povezav s premajhnimi utežmi ...).*

c) What is the purpose of convolutional layers in CNN (convolutional neural networks)? What about the pooling layers?

- *Convolutional layer konvulira dobljen vhod in posreduje rezultat naslednjemu sloju. Vsak konvolucijski nevron sprocesira podatke za njegov dovzeten prostor.*

- *Pooling layer zmanjšuje dimenzijo podatkov z združevanjem izhodov gruč nevronov iz ene plasti v en nevron na naslednji plasti.*

Naloga 5 ✓

a) What is the difference between recursive and recursively enumerable sets?

- *Rekurzivno naštevna množica je tista, za katero velja, da se za določen vhod Turingov stroj ustavi le, če se vhod nahaja v tej množici. Rekurzivna ali odločljiva množica jezikov pa je tista, za katero velja, da za določen vhod Turingov stroj zna po končnem številu korakov povedati, ali se vhod nahaja v tej množici ali ne.*

b) Is every single valued total recursively enumerable set decidable? Explain.

- *Da, vendar le za nerekurzivne učence. Za rekurzivne učence to ne velja.*

c) Describe the learner which is reliable and confident.

- *Tak učenec ne obstaja. To bi namreč pomenilo, da bi se učenec lahko naučil poljubnega jezika, kar pa ni mogoče.*

d) Does text with negative examples help in learning single valued total recursively enumerable sets? Why?

- *Ne, saj že z enim pozitivnim primerom pri enovrednostnih totalnih rekurzivnih funkcijah dobimo neskončno mnogo negativnih primerov. Dodajanje negativnih primerov v tem primeru tako ne izboljšuje naučljivosti.*

2. izpit, 13. 2. 2018 je enak 3. izpit, 1. 9. 2017

Naloga 1 ✓

a) In what relation are J-measure and Information gain? Can any of them be negative? What does it mean that J-measure is equal to 0?

- *Vsota J mer po vseh vrednosti je informacijski prispevek atributa. Ne J mera in ne informacijski prispevek ne moreta biti negativna.*

Če je J mera enaka 0, to pomeni, da ta vrednost atributa ne prispeva k razločevanju ciljnega razreda.

b) Which evaluation measures can be used to evaluate continuous attributes in a) classification and b) in regression? How can you adapt other evaluation measures to evaluate the continuous attributes?

- *Pri klasifikaciji lahko uporabimo Relief in ReliefF, pri regresiji pa RReliefF. Zvezne atribute lahko diskretiziramo in na tak način omogočimo ocenjevanje zveznih atributov tudi metodam, ki sicer ne morejo ocenjevati zveznih atributov.*

c) Why we sometimes want to use recall and precision instead of sensitivity and specificity?

- *Uporabimo ju, ko nas zanima delež pravilno klasificiranih primerov izmed vseh, ki smo jih uvrstili v ta razred, in izmed vseh, ki so dejansko bili v tem razredu. Primerno ju je uporabiti, ko želimo izmeriti relevantnost klasificiranih primerov. Recimo, da imamo sliko, na kateri je nekaj psov in mačk in želimo ugotoviti, koliko mačk je na sliki. V tem primeru nam recall (prevod, če ve kdo) in preciznost povesta več kot senzitivnost in specifičnost.*

d) Which evaluation measures in classification can take into account the predicted (posterior) probabilities of classes? What about the prior probabilities of classes?

- *Posteriorne verjetnosti uporabljajo teža evidence, MDL in sprememba variance. Apriorne verjetnosti uporabljajo informacijski prispevek, razmerje informacijskega prispevka, mera razdalje, teža evidence, MDL in sprememba variance.*

Naloga 2 ✓

In what relation is the MDL principle with the bias-variance tradeoff? In what situations do we have a danger of overfitting?

- *Princip MDL upoštevamo, če poenostavljamo hipotezo. S tem minimiziramo varianco, a povečujemo pristranskost. Do prevelikega prileganja pride v nasprotnem primeru, če hipotezo naredimo kompleksnejšo. S tem minimiziramo pristranskost, a povečujemo varianco in tvegamo pojav prevelikega prileganja.*

Naloga 3 ✓

a) Describe the idea of Radviz visualization.

- *Pri vizualizaciji Radviz so atributi, ki jih želimo vizualizirati, enakomerno označeni na stranici kroga. Vsak primer je prikazan tam, kjer je vsota vrednosti njegovih atributov najbolj utežena. Vizualizaciji lahko dodamo tudi barve, ki označujejo, kateremu razredu pripada nek primer in na tak način ocenimo, kako dobro vizualizirani atributi razločujejo razrede med seboj.*

b) How many attributes can be visualized with this method?

- *Teoretično bi lahko vizualizirali poljubno število atributov, zaradi preglednosti pa navadno prikazujemo do največ 10 atributov.*

c) How does the Vizrank algorithm evaluate the quality of Radviz visualizations?

- *Generira učno množico, sestavljeno iz koordinat in ciljnih vrednosti, za katero napove točnost. Dobljena točnost predstavlja zanimivost vizualizacije.*

Naloga 4 ✓

a) Define the recursive enumerable sets. What is the difference between recursive and recursive enumerable sets?

- *Rekurzivno naštevna množica jezikov je tista, za katero velja, da se za določen vhod Turingov stroj ustavi le, če se vhod nahaja v njej. Pri rekurzivnih množicah se Turingov stroj ustavi v končnem številu korakov za vsak vhod in zna odgovoriti, ali je vhod v množici ali ne.*

b) Usually it is advantageous to have positive and negative training instances. Why this is not true for single valued total recursive enumerable sets?

- *Ker že z enim pozitivnim primerom dobimo neskončno veliko množico negativnih. Za te množice tako velja, da se z dodatkom pozitivnih in negativnih primerov naučljivost ne spremeni.*

c) Is the set of all finite languages learnable with recursive learners? Is it learnable with nontrivial learners?

- *Da, množica vseh končnih jezikov je naučljiva tako z rekurzivnimi kot z nerekurzivnimi učenci.*

Naloga 5 ✓

Can the following approaches be used to improve the prediction accuracy (why and how):

a) explaining the individual predictions?

- *Neposredno ne, saj s tem dobimo le razlago odločitve modela. Lahko pa na ta način identificiramo primere, ki ne prispevajo k izboljšanju, in jih po potrebi odstranimo.*

b) evaluating the reliability of individual predictions?

- *Neposredno ne, saj s tem samo ocenimo, s kakšno verjetnostjo je dobljen rezultat pravilen, s čimer ne spreminjamo klasifikacijske točnosti. Lahko pa tako identificiramo napačno klasificirane primere in skušamo ugotoviti, zakaj je do tega prišlo.*

c) error correcting output codes?

- *Da, saj večrazredni problem razstavi na kombinacije le-teh, kjer v vsakem skušamo razločiti enega ali več razredov od vseh ostalih. Na ta način lahko popravimo določeno število napak pri klasifikaciji, s čimer lahko izboljšamo klasifikacijsko točnost.*

d) cross validation?

- *Ne, saj se prečno preverjanje uporablja samo za ocenjevanje klasifikacijske točnosti, ne pa tudi za njeno izboljšanje.*

e) calibration of prediction probabilities?

- *Ne, kalibracija verjetnosti samo izenači verjetnostno porazdelitev napovedi in neposredno ne izboljšuje klasifikacijske točnosti.*

3. izpit, 28. 8. 2018 je enak 1. izpit, 2. 2. 2017

Naloga 1 ✗

What are the time complexities of the learning phase of the following algorithms with respect to the number of classes c and the number of binary

attributes a – give arguments for the formula!

a) Tree augmented Naive Bayes (TAN)

   - $O(na^2c + a^2 \log a)$, *kjer je* n *število učnih primerov. Potrebno je naučiti naivnega Bayesa in poiskati minimalno vpeto drevo.*

b) decision tree with ReliefF as a selection criterion

   - $O(a^2n^2c)$, *kjer je* n *število učnih primerov. Sama zahtevnost ReliefF je kvadratna glede na število primerov, zahtevnost gradnje drevesa pa je odvisna od števila razredov in kvadratno odvisna od števila atributov.* **Komentar:** Preveriti, ali je res (najverjetneje ni)

c) decision tree with Gini index as a selection criterion

   - $O(a^2nc)$, *kjer je* n *število učnih primerov.*

d) Bagged Naïve Bayes

   - $O(na)$, *kjer je* n *število učnih primerov. Časovna zahtevnost naivnega Bayesa je* $O(na)$, *število generiranj podmnožic učne množice pa zanemarimo.*

e) K-nearest neighbors

   - $O(1)$, *saj se učenje v bistvu ne izvaja.*

Naloga 2 ?

From the following training set (columns correspond to training instances):

A1	1	0	1	0	1	0	1	0
A2	1	1	0	0	1	1	0	0
A3	1	0	1	0	1	0	1	1
Class	1	0	0	1	1	0	0	1

one has generated 400 training instances by making 50 copies of each training instance and by adding 5 random attributes A4-A8.

a) Rank the attributes A1 to A8 with:

   (a) Gini index

      - *Gini indeks oceni vse atribute od A1 do A8 približno enako.*

   (b) ReliefF

- *ReliefF najbolje oceni atributa A1 in A2. Vsi ostali pa imajo slabšo oceno. Funkcija, ki jo modeliramo, je namreč ekvivalenca* $A1 \Leftrightarrow A2$.

b) Define one or more new attributes from existing ones, so that the ranks by two methods will be equal.

- *Nov atribut bi moral imeti vrednosti, ki ustrezajo funkciji* $A1 \Leftrightarrow A2$.

Naloga 3 ?

a) What is active learning?

- *Pri aktivnem učenju se model sam odloča, iz katerih primerov se bo učil in katere atribute bo pri tem uporabil.*

b) What is the difference between query synthesis, stream-based-selective sampling, and pool-based sampling for active learning?

- *Query synthesis: ni primerov, jih sintetično generiramo,*
- *Stream-based-selective sampling: je tok primerov in izbiramo, katere bomo označili*
- *Pool-based sampling: je statična baza primerov, ravno tako izbiramo katere bomo označili.*

c) Describe the query by committee approach to active learning.

- *Zgradimo več modelov in ta ansambel uporabimo za napovedovanje novih primerov. Nov primer je nezanimiv, če se modeli pri napovedi strinjajo, in zanimiv, če se ne strinjajo.*

Naloga 4

a) Describe the advantages of modeling text as a sequence of letters as opposed to bag of words.

- *Z uporabo zaporedja črk lahko zmanjšamo prostor atributov, če vzamemo majhno število zaporednih črk, ali ohranimo kontekst besedila, če vzamemo večje število zaporednih črk* **Komentar:** Smo to sploh delali?

b) Define representative sampling from large set of text documents. How can be compression algorithms used for an implementation of representative sampling?

- **Komentar:** This year left out?

c) How can representative sampling help the K-means clustering of documents?

- Mogoče si lahko pomagamo pri izbiri centroidov? **Komentar:** This year left out?

Naloga 5 ?

a) Define deep learning. Why backpropagation of error is not enough for training deep neural networks?

- *Pri globokem učenju gre za preslikavo vhoda v vmesne koncepte, na podlagi katerih se generira izhod. Vzvratno razširjanje napake ni dovolj, ker bi potrebovali ogromno učno množico, obstaja velika verjetnost za preveliko prileganje in ker se iskanje lahko ustavi v lokalnem minimumu, poleg tega bi bilo tudi preveč dolgotrajno. Namesto tega naučimo vsak nivo posebej in na koncu izvedemo vzvratno razširjanje napake.*
- *Če bi globoke nevronske mreže učili samo z vzvratnim razširjanjem napake, bi se obnesle slabše kot plitve.*

b) How can matrix factorization be used for deep learning?

- *Skriti nevroni predstavljajo skrčeno predstavitev vhodnih podatkov, iz katere lahko vhodne podatke reproduciramo nazaj. Več nivojev imamo, bolj skrčena je predstavitev in tem višjenivojske koncepte imamo.*

c) How can matrix factorization be used for clustering?

- *Na faktorizacijo lahko gledamo kot na gručenje. Ena od matrik namreč vsebuje moči asociacij med uporabnikom in značilkami, druga pa moči asociacij med izdelki in značilkami.*

d) Describe Convolutional neural networks. Explain the idea of the convolution step.

- *Ideja konvolucijskih nevronskih mrež je, da v nevronsko mrežo dodamo povratne povezave. Konvolucijski korak iz slike izlušči značilke, pri čemer se okoliške odvisnosti med piksli ohranjajo. Pooling korak pa zmanjša dimenzijo vsake preslikave značilk, pri čemer ohranja najpomembnejše informacije.*

2. izpit, 17. 2. 2017

Naloga 1 ?

a) What is incremental learning?

Učni algoritem dobiva vhodne podatke enega za drugim. Algoritem sproti spreminja (prilagaja) teorijo (model spremeni ali pa naredi novega). Algoritem poskuša poiskati najmanjšo potrebno spremembo trenutne hipoteze, tako da le-ta ustreza vsem do sedaj videnim podatkom. Nasprotje inkrementalnega je paketno učenje, kjer morajo biti podatki na voljo vsi takoj

b) Which of the following algorithms are able to learn incrementally without modifications (explain your claims):

(a) Naïve Bayes: *Da, posodobimo ocene verjetnosti razredov pri določenih vrednostih atributov.*

(b) Decision trees: *Ne, ko drevo zgradimo, ga brez modifikacije osnovnega algoritme ne moremo spremeniti. Če spremenimo osnovni algoritem, pa lahko na podlagi novih podatkov drevo popravljamo.*

(c) Random forest: *Brez modifikacije algoritma jih ne moremo inkrementalno učiti, razlog je enak kot pri drevesih. Če algoritem spremenimo, pa drevesa lahko učimo sproti in se postopno znebimo starih dreves.*

(d) Hopfields neural network: *Da, ko je za učenje uporabljen nek nov vzorec podatkov, so nove uteži odvisne od starih vrednosti in novega vzorca.*

(e) Multilayered perceptron with backpropagation *Da, mreži prikazujemo nove primere, uteži se osvežujejo.*

(f) SVM: *Ne, saj ne vemo, na kakšen način smo preslikali atribute v večdimenzionalni prostor. Trditev velja le, če je to inkrementalni SVM - ko dodamo nove primere, se preračuna nov vektor.*

(g) K-nearest neighbours: *Da, ker shranjuje nove primere.*

(h) Semi-naïve Bayes: *Ne, ker model gradimo tako, da združujemo atribute, če so odvisni, takega modela pa ni možno posodabljati.*

Naloga 2 ✗

We want to classify the researchers into good/bad. Out of 100 researchers, 25 have published less than 3 papers. Out of those 25, 15 are considered to be bad researchers. Out of those with 3 papers or more, 50 are considered to be good researchers. Good researchers all have more than 50 citations, and 30 of them have written a textbook. Bad researchers have all less than 50 citations and 15 of them have also written a textbook.

a) Calculate the following probabilities with Naïve Bayes using m = 2:

(a) The probability that the researcher is bad if we know that he has written 4 papers and hasn't writen any textbooks?

Iz danih podatkov sestavimo spodnjo tabelo in izračunamo:

Razred	< 3 cl.	≥ 3 cl.	tb	ntb	> 50 cit.	< 50 cit.	Skupaj
good	10	50	30	30	60	0	60
bad	15	25	15	25	0	40	40
Skupaj	25	75	45	55	60	40	100

$$P(bad) = 0.4, P(\geq 3cl.) = 0.75, P(ntb) = 0.55$$

$$P(\geq 3cl.|bad) = 0.625, P(ntb|bad) = 0.625$$

$$P(bad| \geq 3cl., ntb) = \frac{P(\geq 3cl.|bad) \cdot P(ntb|bad) \cdot P(bad)}{P(\geq 3cl.) \cdot P(ntb)}$$

Dobimo končni rezultat 0.3788.

Komentar: Verjetno se je pozabila upoštevati m-ocena. Lahko nekdo potrdi?

Komentar: Tole pomojem ni prav, ker se uporablja Laplaceva ocena verjetnosti. Najbrž bi bila prava rešitev, če bi šli po temu postopku: https://ucilnica.fri.uni-lj.si/mod/url/view.php?id=39927

(b) The probability that the researcher is good if we know that he has written 2 papers, he hasn't write any textbooks, and has 51 citations?

- *Zadevo izračunamo na popolnoma enak način kot zgoraj in dobimo* $P(good| < 3cl, ntb, > 50cit.) = 0.6073$

b) Rank the three binary attributes using the gini-index.
Komentar: TODO!

Naloga 3 ✗

a) Can semi-naïve Bayes be considered as an artificial neural network algorithm? Provide arguments for and against?

b) How can MDL be used to guide the search in clustering algorithms?

c) Can an arbitrary clustering algorithm be used to initialize a deep neural network?

Naloga 4 ✗

a) Define the principle of multiple explanation. Can it be used without MDL?

b) Define the approaches to prediction which use the principle of multiple explanation and describe their properties.

c) Do neural networks use the principle of multiple explanations?

Naloga 5 ✗

a) Describe the idea of the algorithm ReliefF for evaluating the attribute quality.

   (a) Princip večkratne razlage pravi, da je potrebno zadržati vse hipoteze, ki so konsistentne z vhodnimi podatki.

b) How is it related do gini-index?

c) What is the difference between gini-index and information gain? What are their properties and which one is better?

d) For each of the three mentioned evaluating measures describe how they can be used to evaluate the quality of continuous attributes.

1. izpit, 29. 1. 2016

Naloga 1 ?

a) What is the difference between weight of evidence and information gain?

- *Informacijski prispevek je razlika med apriorno in pogojno entropijo, teža evidence pa uporablja alternativno definicijo prejete informacije, imenovano odds, in je definirana kot logaritem razmerja med pogojnim odds in apriornim odds.*

b) Can J-measure be used to evaluate the quality of the whole attribute? Explain.

- *Da, lahko seštejemo J mere po vseh vrednostih atributa, pri čemer dobimo informacijski prispevek.*

c) Can gini-index be used to evaluate the quality of attributes in regression? Explain.

- *Ne, Gini indeksa ne moremo uporabiti za ocenjevanje kakovosti atributov v regresiji. Gini indeks uporablja verjetnosti po razredih, česar pa pri regresiji ne moremo izračunati. Uporabili bi ga lahko le, če bi diskretizirali ciljno spremenljivko, torej jo razbili na več intervalov in za vsak primer označili, v katerem intervalu se nahaja.*

Naloga 2 ?

a) Define stacking.

- Pri stackingu z metodo bootstrap zgradimo več različnih modelov in se nato učimo združevanja. Pri tem uporabimo preprost klasifikator, kot sta naivni Bayes ali linearna regresija, saj lahko sicer pride do prevelikega prileganja.

b) What are error correcting output codes used for and what are their properties?

- *Tabele za popravljanje napak so namenjene popravljanju napačnih klasifikacij. Večrazredni problem razdelimo na več binarnih, pri čemer v vsakem skušamo razločiti enega ali več razredov od ostalih. S tem dobimo zaporedje bitov glede na klasificiran razred in končen*

rezultat je tisti razred, katerega vrstica z 0 in 1 se najbolje ujema z napovedmi.

c) Can boosting improve SVM? What about bagging? Explain.

- *Preprost boosting ne more bistveno izboljšati SVM. Sam boosting sicer lahko zmanjša pristranskosti stabilnih algoritmov, vendar ker imata tako SVM kot boosting sigmoidno pristranskost, bistvene spremembe s tem verjetno ne bomo dosegli. Z baggingom imamo boljše možnosti za izboljšavo, saj lahko zmanjšamo preveliko prileganje učni množici.*

Naloga 3 ✗

How (if at all) can the following algorithms be used for clustering?

a) any data compression algorithm

- **Komentar:** ???

b) Archetypal analysis

- **Komentar:** ???

c) any ML algorithm for classification

- *Primere lahko klasificiramo v razrede in tako identificiramo skupine, v katerih so primeri s podobnimi lastnostmi.*

Naloga 4 ✗

a) Describe the idea of deep neural networks. What are their (dis)advantages in comparison to standard multilayered perceptron with backpropagation learning?

- *Pri večnivojskem perceptronu sestavimo celotno mrežo naenkrat, ki jo potem učimo z vzvratnim razširjanjem napake. Pri globokem učenju pa učimo vsak nivo mreže posebej in mrežo sestavljamo postopoma. Ideja globokih nevronskih mrež je preslikati vhod v vmesne koncepte in šele iz teh tvoriti izhod. V primerjavi z večnivojskimi perceptroni je njihovo učenje počasnejše in verjetnost za preveliko prileganje učni množici je večja, vendar pa lahko zato rešuje zelo kompleksne probleme, na primer v videih, avdiu in obdelavi naravnega jezika.*

b) Let us have a ML algorithm which uses MDL to guide the search. Can MDL be replaced using the cross validation procedure? Explain.

- **Komentar:** ???

Naloga 5 ✓

Describe the Apriori algorithm for generating association rule. What are its properties and how can you control the number of generated rules? What are time complexities?

- *Apriori algoritem poišče vse možne podmnožice izdelkov in izmed njih izbere take, ki se v podatkovni bazi transakcij pojavijo pogosto, nato pa izmed pogostih množic izdelkov generira še množico povezovalnih pravil. Število generiranih pravil lahko omejimo s dejstvoma, da je vsaka podmnožica pogoste množice pogosta in da nobena nadmnožica nepogoste množice ni pogosta, pri generiranju pravil pa lahko najprej generiramo pravila s krajšim sklepnim delom in če pravilo nima zadostne podpore in zaupanja, nam pravila z daljšim sklepnim delom sploh ni treba preverjati. Časovna zahtevnost je $O(2^d)$, kjer je d število izdelkov v podatkovni bazi.*