UNIVERZA V LJUBLJANI

Fakulteta za elektrotehniko in računalništvo

Marko Robnik

KONSTRUKTIVNA INDUKCIJA Z ODLOČITVENIMI DREVESI

diplomsko delo

Mentor: doc.dr. Igor Kononenko

Ljubljana, december 1993

Kazalo

Seznam uporabljenih simbolov:

oznaka	*pomen*
$\mathcal{L}$	učna množica primerov
L	moč učne množice primerov (število primerov)
$\mathcal{A}$	množica atributov
A	moč množice atributov (število atributov)
X_i	i-ti atribut
V_i	število vrednosti atributa X_i
$v_{i,l}$	l-ta vrednost atributa X_i
$\mathcal{C}$	množica razredov
C	moč množice razredov (število razredov)
c_j	j-ti razred
$\mathbf{x}$	vektor vrednosti atributov, ki opisuje posamezen primer
$\Phi(Y,F)$	funkcija, ki ocenjuje razdelitev
$H(Y)$	nedoločenost slučajne spremenljivke Y
$H(Y\|F)$	nedoločenost slučajne spremenljivke Y pri hipotezi realizacije slučajne spremenljivke F
$p(y)$	verjetnost dogodka y
$p(y\|t)$	verjetnost dogodka y pri pogoju t
$\Psi(F)$	zmanjšanje nedoločenosti o konceptu po realizaciji konstrukta F
$\Psi(p,n)$	zmanjšanje nedoločenosti o konceptu po realizaciji konstrukta, ki mu ustreza p pozitivnih in n negativnih primerov
m	parameter začetne β porazdelitve pri splošnem bayesovskem ocenjevanju verjetnosti (m-oceni verjetnosti)
$\mathrm{Var}(p)$	varianca verjetnosti p
Inf	informacijska vsebina klasifikacij testnih primerov
Inf_i	informacijska vsebina klasifikacije i-tega testnega primera
Cl_i	razred, ki mu pripada i-ti testni primer
$P(Cl_i)$	apriorna verjetnost razreda Cl_i
$p(Cl_i)$	verjetnost razreda Cl_i, kot jo vrne klasifikator
$\tau(A)_f$	natančnost klasifikacije algoritma A v f-tem poskusu
$N(\mu,\sigma)$	normalna porazdelitev verjetnosti s parametroma μ in σ
$H(\mu = a)$	hipoteza: vrednost parametra μ je a
$\overline{z}$	vzorčno povprečje vektorja poskusov
s^2	popravljeni vzorčni odklon vektorja poskusov
$\triangle$ in $\triangle'$	zamegljenost koncepta (meri težavnosti)
$\bigtriangledown$	povprečna medsebojna informacija
R	število pomembnih atributov

Povzetek

Standardni algoritmi za strojno učenje iz primerov imajo težave s problemi, kjer obstaja med atributi visoka stopnja odvisnosti. Problem odvisnosti med atributi rešujemo s konstruktivno indukcijo. Pred kratkim se je pojavil algoritem LFC, ki s pogledom naprej in gradnjo konstruktov uspešno zmanjšuje svojo kratkovidnost. Konstrukti so koristni zaradi boljše razumljivosti odločitvenega drevesa, preprečevanja podvajanja poddreves in zaradi tega tudi boljše statistične podpore listom in večje natančnosti klasifikacije.

Gradnja konstruktov je kombinatorično zelo zahteven problem, ki ga LFC deloma omili z več hevristikami, ki izhajajo iz geometrijske predstavitve kriterijske funkcije. LFC omeji prostor koristnih konstruktov s predpreiskovanjem, ki je računsko mnogo manj zahtevno kot sama gradnja konstruktov. Dodatno sta omejeni tako globina kot širina preiskovanja problemskega prostora.

Mnogi realni problemi so opisani s šumnimi in nepopolnimi podatki. Takšnim domenam sem prilagodil algoritem z verjetnostnim obravnavanjem manjkajočih podatkov, pri šumu v učnih primerih pa sem si pomagal z dvema vrstama rezanja odločitvenih dreves. Za ocenjevanje verjetnosti sem uporabil m-oceno verjetnosti.

LFC sem na testiral na nekaj umetnih in realnih problemih. Z algoritmom brez konstruktivne indukcije sem ga primerjal glede natančnosti klasifikacije, informacijske vsebine in razumljivosti dreves ter določil značilnosti odstopanj. Med seboj sem primerjal tudi različne oblike konstruktov ter analiziral pomen zgrajenih konstruktov za eno od domen. Da bi razložil razlike med LFC in algoritmom brez konstruktivno indukcije, sem obravnaval težavnost različnih problemov in kot merilo predstavil zamegljenost koncepta.

1.

Uvod

Naloga se ukvarja s konstruktivno indukcijo z odločitvenimi drevesi, ki je postavljena v širše okolje strojnega učenja.

Učenje razumemo kot adaptivne spremembe v sistemu, ki sistemu omogočajo, da naslednjič bolj učinkovito opravi isto nalogo ali nalogo iz iste populacije nalog *(Thornton, 1992)*. Na področju računalništva se s tem pomembnim pojmom ukvarja umetna inteligenca, oziroma njeno podpodročje strojno (avtomatsko) učenje. V grobem prevladujejo trije pristopi: simbolično učenje, konekcionistične metode in numerične metode (razpoznavanje vzorcev), razlike med njimi pa so vse bolj zabrisane. Učenje z odločitvenimi drevesi, ki je danes najbolj razširjena metoda na področju strojnega učenja *(Winston, 1992)*, je ena od simboličnih metod.

Če se omejimo na učenje iz primerov, ki temelji na podobnosti (similarity based learning), je eden od možnih pristopov tudi induktivno učenje. Njegov cilj je iz primerov inducirati pravila, ki bodo sposobna identificirati določen koncept. Odločitveno drevo določa obliko teh pravil. Obstaja množica algoritmov in učnih sistemov, ki generirajo pravila v obliki odločitvenih dreves npr. CART *(Breiman in sod., 1984)*, ID3 *(Quinlan, 1979)*, ASSISTANT *(Cestnik in sod., 1987)*, itd. Vsi našteti spadajo v družino TDIDT (Top Down Induction of Decision Trees) in delujejo po načelu, da v vsako vozlišče drevesa postavijo atribut, ki glede na nek kriterij najbolje razdeli primere

po razredih, ki jim pripadajo. Slabost pristopa je njegova kratkovidnost, saj je izbira atributa le lokalno najboljša, globalno pa je morda zgrešena, kar še posebej pride do izraza pri problemih z veliko stopnjo odvisnosti med atributi. Možno rešitev tega problema nam nudijo algoritmi s konstruktivno indukcijo, ki poskušajo namesto posameznega atributa v vozlišča drevesa postaviti konstrukte izpeljane iz atributov. Le-ti naj bi bolje opisovali odvisnosti med atributi in bili bolje statistično podprti. Tvorba takšnih konstruktov je kombinatorično zelo zahteven problem.

Algoritem LFC (Lookahead Feature Construction), ki sta ga razvila Ragavan in Rendell *(1993; Ragavan in sod., 1993; 1993a)*, ponuja nekaj novih pristopov, ki omejujejo problemski prostor.

Cilj diplomske naloge je implementirati algoritem LFC in primerjati njegove rezultate z algoritmom brez konstruktivne indukcije. Kot primer algoritma brez konstruktivne indukcije sem implementiral program, ki je podoben sistemu ASSISTENT, opisanem v *(Cestnik in sod., 1987)*. Rezultate obeh pristopov sem primerjal na nekaj različnih problemskih domenah.

Algoritem sem implementiral v programskem jeziku C++, na operacijskih sistemih MS-DOS in OS/2. Zaradi prenosljivosti sem se odločil za strogo podporo standardu tega jezika, kot je opisan v *(Stroustrup 1991)*. Strokovni termini v tem delu so vzeti iz Računalniškega slovarčka *(1993)*.

V drugem poglavju je opisan algoritem LFC in še posebej lastnosti, po katerih se loči od drugih algoritmov.

Tretje poglavje opisuje dodatke, ki so potrebni za obravnavanje nepopolnih in šumnih podatkov.

V četrtem poglavju so podani rezultati algoritma LFC in primerjava z algoritmom brez konstruktivne indukcije.

Zadnje poglavje podaja zaključke in ponudi ideje za nadaljnje izboljšave.

V dodatku najdemo podrobno analizo zgrajenih konstruktov za enega od testnih problemov, opis programa, ki sem ga izdelal ter kratka navodila za njegovo uporabo.

2.

Opis algoritma LFC

Algoritem LFC (Lookahead Feature Construction) sta Harish Ragavan in Larry Rendell objavila v različnih publikacijah *(Ragavan in Rendell, 1993; Ragavan in sod., 1993; 1993a)*.

To poglavje povzema delovanje algoritma in opisuje tudi mnoge probleme, ki sta jih avtorja algoritma v svojih člankih izpustila, na začetku pa zgoščeno definira področje, kamor uvrščamo algoritem.

2.1 Atributno učenje iz primerov

Definirajmo najprej oznake, ki jih bomo uporabljali v nadaljevanju; njihov pomen je označen tudi na sliki 2.1.

V učni množici primerov $\mathcal{L}$ velikosti L je vsak primer opisan z A atributi $X_1, \ldots, X_A$ iz množice atributov $\mathcal{A}$ in pripada natanko enemu od C razredov $c_1, \ldots, c_C$ iz množice razredov $\mathcal{C}$. Atribut X_i lahko zavzame V_i različnih vrednosti $v_{i,1}, \ldots, v_{i,V_i}$.

Primer lahko opišemo z urejenim parom $(\mathbf{x}, c)$, kjer $\mathbf{x}$ označuje vektor vrednosti atributov $(x_1, \ldots, x_A)$ dimenzije A, $c \in \mathcal{C}$ pa predstavlja razred, v katerega spada atribut.

	X_1	X_2	...	X_A	$\mathcal{C}$
$\mathbf{x_1}$	$x_{1,1}$	$x_{1,2}$	...	$x_{1,A}$	c_1
$\mathbf{x_2}$	$x_{2,1}$	$x_{2,2}$	...	$x_{2,A}$	c_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$\mathbf{x_L}$	$x_{L,1}$	$x_{L,2}$	...	$x_{L,A}$	c_C

Slika 2.1: Tabelarično zapisana množica primerov $\mathcal{L}$

2.2 Gradnja odločitvenega drevesa

Osnovni algoritem za gradnjo odločitvenega drevesa je podoben modelu, ki ga je s CLS postavil *Hunt s sod. (1966)*, citirano v *(Cestnik in sod., 1987)*. Prikazan je na sliki 2.2.

```
če je ustavitveni pogoj izpolnjen,
      napravi list in ga označi z najverjetnejšim razredom ;
sicer
{
      glede na učno množico in funkcijo ocene razdelitve določi
            najboljši atribut X_i ;
      postavi X_i v koren drevesa in razbij učne primere v
            V_i podmnožic glede na vrednosti atributa X_i ;
      za vse vrednosti atributa X_i ponovi:
            rekurzivno zgradi poddrevo s pripadajočo učno podmnožico ;
}

ko je drevo zgrajeno, ga obreži ;
```

Slika 2.2: Psevdo koda algoritma CLS za gradnjo odločitvenih dreves

LFC se pri gradnji drevesa od tega modela razlikuje v bistvu le v eni točki, ki je na sliki 2.3 označena s puščico.

Namesto, da bi v vozlišče izbral en sam najboljši atribut, zgradi iz atrib-

če je ustavitveni pogoj izpolnjen,
    napravi list in ga označi z najverjetnejšim razredom ;
sicer
{
    glede na učno množico in množico atributov $\mathcal{A}$
        sestavi najbolj informativen konstrukt f_{naj} ; ⟸
    postavi f_{naj} v koren drevesa in glede na njegovo vrednost
        razbij učne primere v dve podmnožici ;
    za obe vrednosti f_{naj} **ponovi:**
        rekurzivno zgradi poddrevo s pripadajočo učno podmnožico ;
}

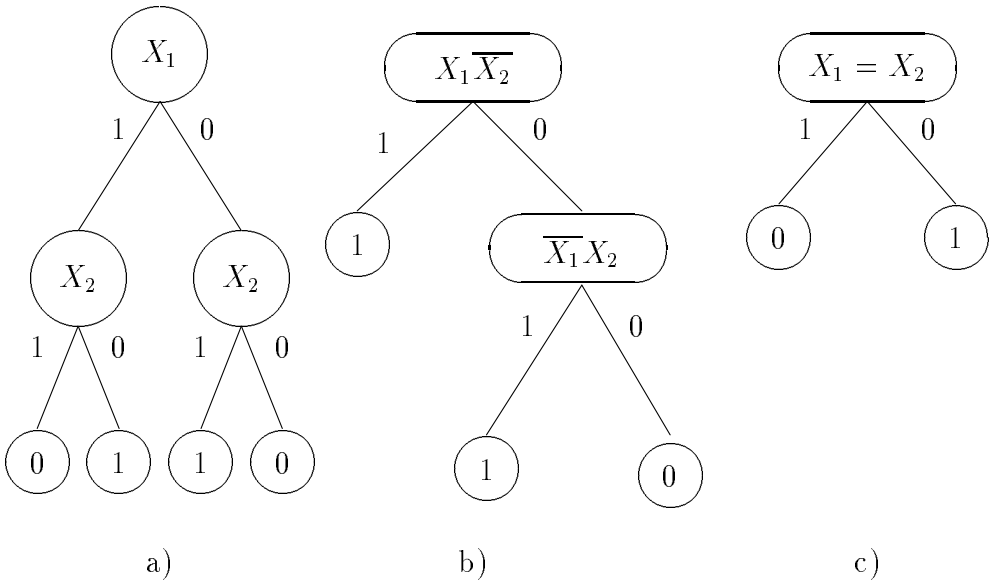
ko je drevo zgrajeno, ga obreži ;

Slika 2.3: Psevdo koda za izgradnjo odločitvenega drevesa pri LFC

utov konstrukt. Ta je v odvisnosti od parametrov izgradnje lahko poljuben logični izraz, v katerem na mestu spremenljivk nastopajo atributi. Ko delimo učne primere v podmnožici glede na zgrajeni konstrukt, postavimo vrednosti atributov posameznega primera v logični izraz, ki ga določa f_{naj}; če vrednosti izrazu ustezajo, oziroma dobimo logično 1, razvrstimo primer v prvo, sicer pa v drugo podmnožico. Konstrukt nam torej opravlja vlogo binarnega atributa.

Na sliki 2.4 si lahko kot primer ogledamo drevesa, ki ju zgradita algoritma brez in s konstruktivno indukcijo, za problem določitve vrednosti funkcije XOR, ki je podana z izrazom $XOR(X_1, X_2) = (X_1 = 1 \wedge X_2 = 0) \vee (X_1 = 0 \wedge X_2 = 1)$ ali krajše $XOR(X_1, X_2) = X_1\overline{X_2} \vee \overline{X_1}X_2$ ali drugače $XOR(X_1, X_2) = (X_1 \neq X_2)$.

LFC bi zaradi načina kako gradi konstrukte, zgradil drevo b). Če bi hoteli kot rezultat dobiti drevo c), bi morali poleg konjunkcije, disjunkcije in negacije vpeljati še druge operatorje.



Slika 2.4: Odločitvena drevesa za problem XOR; a) običajno drevo b) in c) drevesi s konstrukti

Najzanimivejši del algoritma LFC je zagotovo način, kako poskuša učinkovito zgraditi uspešne konstrukte. Poglejmo, zakaj je gradnja konstruktov sploh koristna.

2.3 Zakaj graditi konstrukte ?

Na sliki 2.4 vidimo razliko med drevesom brez konstruktivne indukcije in z njo. Prednosti konstruktivne indukcije so očitne. Opišimo najpomembnejše:

- **projekcija:** Z enim samim atributom projeciramo prostor učnih primerov v eno dimenzijo, konstrukt pa projicira v večdimenzionalen prostor, kjer so zakonitosti, ki se v enodimenzionalnem prostoru izgubijo, morda bolj očitne.

- **razumljivost:** S stališča znanja pomenijo konstrukti nekakšno vmesno stopnjo med konceptom, ki se ga učimo, in atributi. Opisujejo določene zakonitosti v konceptu in so tako koristni za človekovo razumevanje odločitvenega drevesa. O konceptu parnosti tako konstrukti $X_1\overline{X_2}$, $\overline{X_1}X_2$ in $X_1 = X_2$ povedo mnogo več kot atributa $X1$ in X_2. Konstrukti nam torej omogočajo bolje razumeti koncept.
- **pogled naprej:** Z gradnjo konstruktov dolžine d gledamo pri gradnji d korakov naprej. Trenutno dobri atributi se lahko izkažejo za širše gledano slabe in obratno: trenutno slabi atributi lahko prispevajo k opisu kakšne pomembne zakonitosti.
- **konstrukti ponazarjajo odvisnosti med atributi:** Če so atributi med seboj odvisni, potem posamezen atribut bolj malo prispeva k razločevanju med razredi, konstrukt pa ima možnost, da to odvisnost zajame na mah. Primer za to vidimo na sliki 2.4.
- **preprečujejo podvajanje v drevesu:** Pri odvisnih atributih potrebujemo večje poddrevo za opis neke zakonitosti koncepta. Mnogo podobnih vej v drevesu (replication problem), povzroči nerazumljivost in nenatančnost opisa koncepta. Posedica je dostikrat tudi nenatančnost klasifikacije s takšnim odločitvenim drevesom *(Yang in sod., 1991)*.
- **boljša statistična podpora:** Razdrobljenost primerov v mnogo listov, kar je posledica večjih poddreves za opis neke zakonitosti, nosi s seboj posledico nezadostne statistične podpore listom. Težava se še zaostri pri problemih s šumnimi in nepopolnimi podatki ter pri problemih, kjer je podatkov malo.
- **manjša drevesa:** Zaradi združitve pomembnih atributov v konstrukt odpade potreba po velikem drevesu.
- **večja natančnost:** Je posledica boljše statistične podpore, manjše kratkovidnosti, projekcije v večdimenzionalne podprostore in zajetja pomembnih odvisnosti med atributi.

Konstruktivna indukcija nosi s seboj tudi težave, ki jih pri navadnih odločitvenih drevesih ni bilo, ali pa niso bile tako izrazite:

- **prevelika prilagoditev podatkom:** Mogoče je, da so konstrukti preveč prilagojeni učnim primerom. Zaradi tega se zmanjša natančnost klasifikacije na testnih podatkih. Problem prevelike prilagoditve podatkov

ni neznan niti v gradnji navadnih odločitvenih dreves, pri konstruktivni indukciji pa pridobi še dodatno dimenzijo. Problem škodljive prilagoditve podatkom rešujemo z rezanjem, pri konstruktivni indukciji pa je potrebno obrezovati tudi konstrukte, za kar potrebujemo dodatne učne primere.

- **težja razumljivost:** Kompleksni konstrukti so za človeka težko razumljivi. Ragavan in sod. *(1993)* trdijo, da so imeli strokovnjaki, ki so poskušali razumeti odločitvena drevesa, največje težave z disjunktno povezanimi izrazi. Težavo lahko rešimo tako, da omejimo nabor operatorjev: namesto nabora $\{\neg, \wedge, \vee\}$ vzamemo le $\{\neg, \wedge\}$, ki pa je še vedno funkcijsko poln. Velja namreč: $A \vee B = \neg(\neg A \wedge \neg B)$. Lahko se omejimo tudi na disjunktivno normalno obliko (DNO).
- **večja kompleksnost izračunov:** Gradnja konstruktov je računsko zahteven proces. Številne omejitve problemskega prostora lahko zadržijo čas procesiranja v razumnih mejah.

Poglejmo si, kako konstruktivno indukcijo realizira LFC.

2.4 Gradnja konstruktov

Psevdo kodo procedure za gradnjo konstruktov najdemo na sliki 2.5.

Konstrukte gradimo postopno. Da bi se izognili padcu v lokalni ekstrem, realiziramo omejeno iskanje v širino. V enem koraku poskušamo razširiti *maxŠirina* konstruktov. Na začetku zato v množico *Konstrukti* izberemo *maxŠirina* najboljših vrednosti atributov, v naslednjih korakih pa širimo toliko najboljših konstruktov iz prejšnjega koraka.

Vsak konstrukt poskušamo razširiti z vsemi vrednostmi atributov, ki ostanejo po omejitvi z oknom. Če bi hoteli preskusiti vse možnosti, bi morali konstrukte razširjati tudi z zanikanimi vrednostmi atributov. Da se temu izognemo, uporabimo v nadaljevanju opisano hevristično merilo, ki pove, kdaj je negacija smiselna.

Preden konstrukt dejansko konjunktivno ali disjunktivno povežemo z neko vrednostjo atributa, opravimo manj zahteven predizračun, ki pokaže ali bo novi konstrukt sploh koristen. Če je odgovor pozitiven, zgradimo s

Vhodni podatki: množica originalnih atributov $\mathcal{A}$, največja širina iskanja $maxŠirina$, največja globina iskanja $maxGlobina$, širina okna W, množica učnih primerov D

Izhod: najboljši konstrukt f_{naj}

f_{naj} = najboljša vrednost atributov ;
$globina = 0$;
$Konstrukti = maxŠirina$ najboljših vrednosti atributov iz $\mathcal{A}$;
dokler $globina < maxGlobina$
{
    $NoviKonstrukti = \{\}$;
    za vse $f \in Konstrukti$ **ponovi**
    {
        v $\mathcal{O}$ izberi atribute iz $\mathcal{A}$, ki so v oknu širine W okoli f
        za vse $x \in \mathcal{O}$ **ponovi**
        {
            po potrebi negiraj x ;
            če je možen koristen konstrukt
            {
                f_{novi} = boljši $(f \wedge x, f \vee x)$;
                dodaj f_{novi} v množico $NoviKonstrukti$;
            }
        }
    }
    če obtaja $a \in NoviKonstrukti$, ki je boljši kot f_{naj}
        $f_{naj} = a$;
    če je vsak $a \in NoviKonstrukti$ slabši kot vsak $b \in Konstrukti$
        $globina = maxGlobina$;
    $Konstrukti = maxŠirina$ najboljših iz $NoviKonstrukti$;
    $globina = globina + 1$;
}
obreži f_{naj} ;
vrni f_{naj} ;

Slika 2.5: Psevdo koda procedure za izgradnjo konstruktov pri LFC

konjunkcijo in disjunkcijo dva nova konstrukta. Postopek ponovimo z vsemi kandidati.

Ko smo razširili vse konstrukte iz množice *Konstrukti*, povečamo globino iskanja za 1 in ponovimo širitev konstruktov. Poglabljanje ustavimo, ko doseže največjo dovoljeno globino *maxGlobina*, ali ko nobeden od novo zgrajenih konstruktov ni boljši od kateregakoli starega konstrukta.

Konstrukt z najboljšo vrednostjo kriterijske funkcije, kar smo jih našli v postopku poglabljanja, obrežemo in vrnemo kot rezultat.

Kot sem že omenil, je gradnja konstruktov kombinatorično zelo zahteven problem, zato mora imeti uspešen postopek vgrajene učinkovite omejitve. LFC ima več vgrajenih omejitev:

Omejitev iskanja v globino: Največja globina iskanja *maxGlobina* določa tudi dolžino logičnega izraza, s katerim je podan konstrukt. Pri problemu določanja paritete bi tako npr. potrebovali iskanje do globine, ki je enaka stopnji paritete.

Omejeno iskanje v širino (beam search): Da bi se izognil padcu v lokalni ekstrem, uporabi LFC omejeno iskanje v širino, ki daje možnost uspeha *maxŠirina* konstruktom hkrati. V programu je omejitev iskanja v širino realizirana z omejitvijo velikosti množice *Konstrukti* na največ *maxŠirina* elementov.

Omejitev možnih koristnih konstruktov: Ta omejitev izhaja iz geometrijske predstavitve kriterijske funkcije za izbiro konstruktov in je najbolj originalen prispevek algoritma LFC. Opisana je v nadaljevanju.

Omejitev z oknom: Pri izbiri kandidatov za gradnjo, uporablja LFC okno širine W; tudi ta omejitev izhaja iz geometrijske predstavitve.

Če hočemo razumeti, kako LFC omejuje gradnjo konstruktov, si moramo najprej ogledati kriterijsko funkcijo, s katero ocenjujemo kvaliteto konstruktov.

2.4.1 Ocena razdelitve

Obstaja več različnih funkcij oziroma algoritmov, s katerimi ocenjujemo pomembnost atributov, oziroma ocenjujemo razdelitev, npr.:

- Gini indeks *(Breiman in sod., 1984)*
- informativnost *(Quinlan, 1986)* citirano v *(Bratko, 1990)*
- informativnost s popravki *(Quinlan, 1986)* citirano v *(Cestnik, 1991)*
- j-ocena *(Smyth in Goodman, 1990)*
- RELIEF *(Kira in Rendell, 1992; Kononenko, 1994)*
- itd.

Breiman in sod. *(1984)* v svoji knjigi trdijo, da je kvaliteta odločitvenega drevesa skoraj neodvisna od funkcije, s katero ocenjujemo razdelitev (seveda, če le-ta izpolnjuje določene pogoje). Na žalost so vse zgoraj omenjene kriterijske funkcije (z izjemo RELIEF-a) kratkovidne, kar se pokaže pri problemih z odvisnimi atributi.

V LFC je uporabljena informativnost (information gain), vendar pri predpostavki binarnega koncepta Y (dva razreda: y in $\overline{y}$ oz. pozitivni in negativni razred) in binarnih atributov oz. konstruktov F (vrednosti f in $\overline{f}$). Definirana je na naslednji način:

$$\Phi(Y,F) = H(Y) - H(Y|F) \tag{2.1}$$

pri čemer je $H(Y)$ apriorna nedoločenost (entropija) koncepta, ki je definirana kot:

$$H(Y) = -\left[p(y)\log_2 p(y) + p(\overline{y})\log_2 p(\overline{y})\right] \tag{2.2}$$

$H(Y|F)$ pa je nedoločenost koncepta, pri pogoju, da poznamo razdelitev, ki jo opravi konstrukt:

$$H(Y|F) = \Psi(F) = \sum_{t=\{f,\overline{f}\}} p(t) \sum_{u=\{y,\overline{y}\}} -p(u|t)\log_2 p(u|t) = \tag{2.3}$$

$$= -\sum_{t=\{f,\overline{f}\}} p(t)[p(y|t)\log_2 p(y|t) + p(\overline{y}|t)\log_2 p(\overline{y}|t)]$$

$H(Y|F)$ imenujeta Ragavan in Rendell $\Psi(F)$ in s tem poudarjata, da imamo v nekem vozlišču možnost izbirati le konstrukt F, množica primerov pa je fiksna.

Osnovna ideja kriterija informativnosti izhaja iz Shannonove teorije informacij. Razdelitev primerov, ki jo opravi konstrukt (atribut), in pripadnost razredu si lahko predstavljamo kot dve slučajni spremenljivki. Funkcija $\Phi(Y,F)$ predstavlja razliko med a priori nedoločenostjo o slučajni spremenljivki Y (pripadnosti razredu) in nedoločenostjo slučajne spremenljivke Y ob hipotezi, da je znana realizacija slučajne spremenljivke F (pripadnost razredu po opravljeni razdelitvi). $\Phi(Y,F)$ izraža množino medsebojne informacije med obema naključnima spremenljivkama *(Gyergyék, 1988)*. Želimo si čim večjo medsebojno informacijo, zato gradimo konstrukte, ki bodo maksimizirali Φ. Da bi to dosegli, moramo minimizirati $\Psi(F)$, saj je apriorna nedoločenost $H(Y)$ konstantna za fiksno množico primerov. Minimizirati $\Psi(F)$ pomeni zgraditi tak konstrukt F, da bo nedoločenost koncepta Y, kar najmanjša. Ekstremna primera sta:

- $\Psi(F) = 0$, ni več nobene nedoločenosti, slučajni spremenljivki sta v funkcijski povezavi $Y = g(F)$
- $H(Y|F) = \Psi(F) = H(Y)$, medsebojna informacija slučajnih spremenljivk je 0, spremenljivki sta neodvisni

V nadaljevanju si bomo ogledali, kako LFC omeji prostor potencialno koristnih konstruktov.

2.4.2 Omejitev gradnje konstruktov

Za lažjo predstavo si oglejmo geometrijsko predstavitev funkcije $\Psi(F)$. Konstrukt F si zopet predstavljamo kot logični izraz. Ker razločujemo zgolj med dvema razredoma (y in $\overline{y}$ oz. pozitivnim in negativnim), imenujmo primere, ki spadajo v razred y pozitivne, tiste, ki spadajo v razred $\overline{y}$, pa negativne.

Da bi izračunali vrednost funkcije $\Psi(F)$, potrebujemo zgolj število pozitivnih in negativnih primerov v vozlišču, katerih vrednosti atributov ustrezajo izrazu F. Razložimo to trditev.

Skupno število primerov pozitivnega razreda v vozlišču označimo s P_{max}, skupno število primerov negativnega razreda pa z N_{max}. Vrednosti P_{max} in N_{max} v vsakem vozlišču so konstantne. Če poznamo število pozitivnih primerov p in negativnih primerov n, ki ustrezajo logičnemu izrazu F, lahko izračunamo tudi število pozitivnih $\overline{p}$ in negativnih primerov $\overline{n}$ v vozlišču, ki izrazu ne ustrezajo. Velja namreč:

$$p + \overline{p} = P_{max} \qquad\qquad n + \overline{n} = N_{max}$$

Skratka, izračunamo lahko funkcijo $\Psi(F)$, ki jo zaradi tega označimo kar s $\Psi(p, n)$.

Da bi izračunali funkcijo $\Psi(F)$, množice učnih primerov torej ni potrebno v resnici razdeliti, dovolj je, da ugotovimo vrednosti p in n. Če verjetnosti v enačbi 2.3 ocenimo z relativno frekvenco, dobimo naslednji izraz:

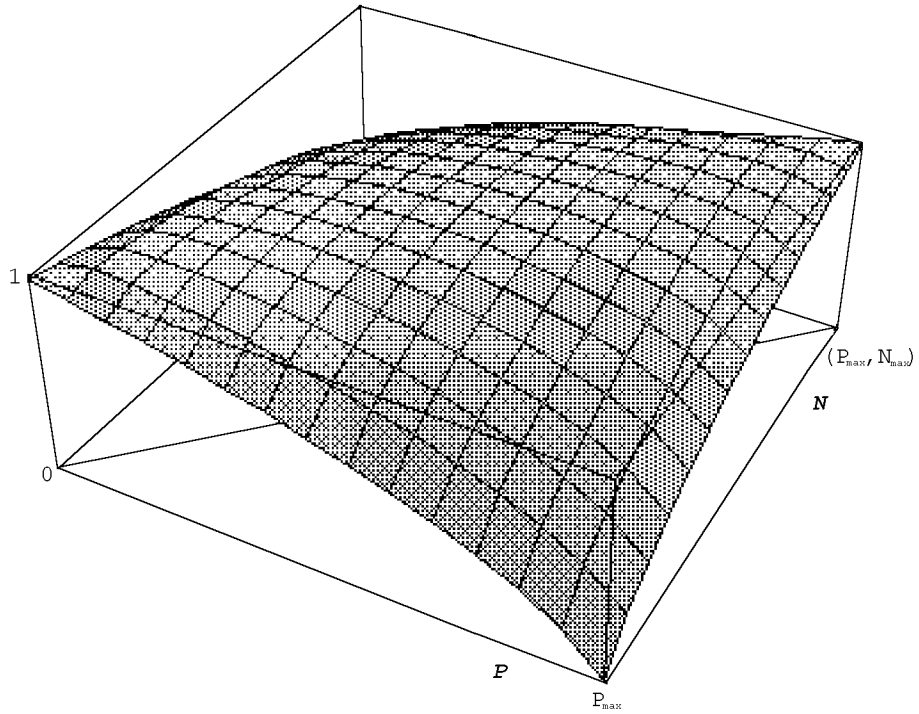
$$\begin{aligned}\Psi(p,n) &= -\left[\frac{p+n}{P_{max}+N_{max}}\left(\frac{p}{p+n}\log_2\frac{p}{p+n}+\frac{n}{p+n}\log_2\frac{n}{p+n}\right)\right. \\ &+ \left.\frac{\overline{p}+\overline{n}}{P_{max}+N_{max}}\left(\frac{\overline{p}}{\overline{p}+\overline{n}}\log_2\frac{\overline{p}}{\overline{p}+\overline{n}}+\frac{\overline{n}}{\overline{p}+\overline{n}}\log_2\frac{\overline{n}}{\overline{p}+\overline{n}}\right)\right] \quad (2.4)\end{aligned}$$

Izraz zaradi lažje predstave tudi narišemo; slika 2.6 prikazuje trodimenzionalno podobo funkcije $\Psi(F) = \Psi(p, n)$.

Vsak konstrukt enolično določa urejeni par (p, n), vrednost funkcije $\Psi(p, n)$ pa lahko poiščemo na sliki 2.6. Na diagonali med točkama $(0, 0)$ in (P_{max}, N_{max}) ležijo konstrukti, ki identificirajo enako število pozitivnih in negativnih primerov, njihova razločevalna vrednost je ničelna. Bolj ko se konstrukt približuje osi P ali N, večjo razločevalno vrednost ima in manjša je vrednost funkcije Ψ. Minimum funkcije Ψ je v točki $(P_{max}, 0)$, njena vrednost je tu 0. Če konstrukt identitificira samo pozitivne primere, mu pripada točka na osi P. Posvetimo pozornost spodnjemu trikotniku $[(0, 0), (P_{max}, 0), (P_{max}, N_{max})]$. Vidimo, da funkcija monotono pada od diagonale proti točki $(P_{max}, 0)$. V celoti je funkcija poševno simetrična glede na diagonalo. Površina funkcije Ψ je konveksna, brez lokalnih minimumov, ne glede na vrednosti P_{max} in N_{max}.

Algoritem vedno poskuša najti izraz, ki bi opisoval nekaj pozitivnih primerov in nobenega negativnega. Konstrukt se temu cilju tem bolj približuje, čim bližje je osi P. Tudi negacija je podrejena temu cilju. Kandidat za širitev je namreč bodisi konstrukt bodisi njegova negacija, glede na to kaj je bližje točki $(P_{max}, 0)$, ki predstavlja idealno pokritje vseh pozitivnih primerov.

Za omejitev gradnje konstruktov je potrebno na nek način povezati vrednost funkcije Ψ pri konstruktu z vrednostjo Ψ sestavnih delov konstrukta.



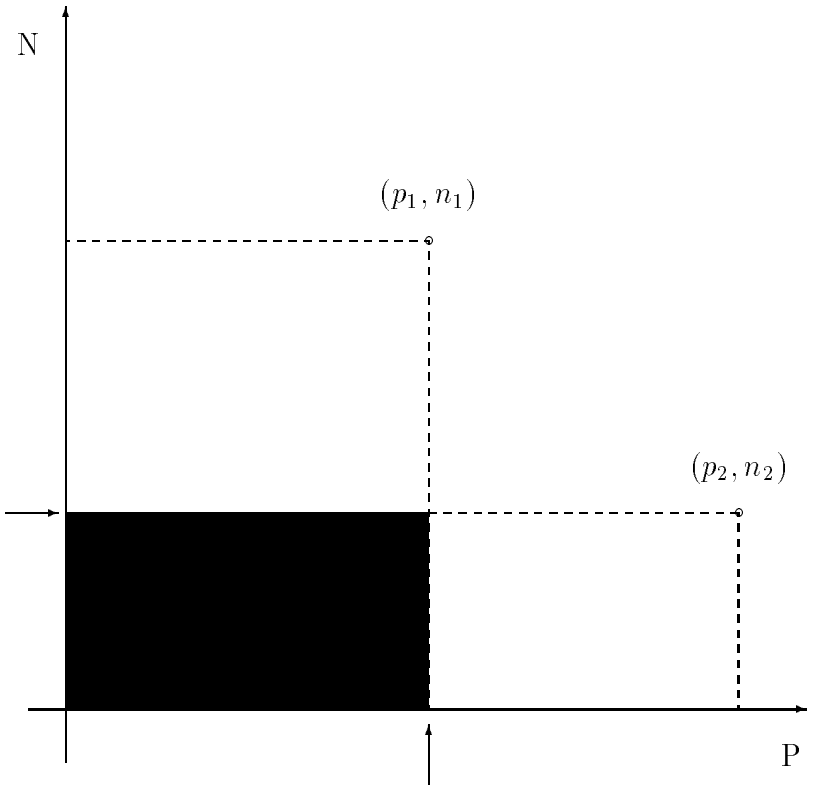
Slika 2.6: Funkcija $\Psi(F) = \Psi(p, n)$

Ko hočemo sestavne dele sestaviti konjunktivno, se ta povezava razlikuje od tiste pri disjunktivni sestavi. Tudi omejitev z oknom okoli konstrukta zahteva svoj razmislek. Poglejmo si jih po vrsti.

Omejitev konjunkcije

Slika 2.7 prikazuje položaj dveh konstruktov k_1 in k_2 s pripadajočima urejenima paroma (p_1, n_1) in (p_2, n_2). Naredimo njuno konjunkcijo $k = k_1 \wedge k_2$. Področje, kjer se lahko na sliki 2.7 nahaja k, je potemnjeno. Omejeno je s pravokotnikom $[(0, 0), (\min(p_1, p_2), 0), (\min(p_1, p_2), \min(n_1, n_2)), (0, \min(n_1, n_2))]$. Konjunkcija namreč povzroči, da sta lahko vrednosti p in n v k samo manjši od ustreznih vrednosti za k_1 in k_2.

Na počrnjenem področju doseže funkcija $\Psi(p, n)$ minimum v eni od točk



Slika 2.7: Omejevanje gradnje konstruktov pri konjunkciji

$(\min(p_1, p_2),\ 0)$ ali $(0,\ \min(n_1, n_2))$. Na sliki 2.7 sta označeni s puščicama. Če je v obeh točkah vrednost Ψ večja od vrednosti Ψ za k_1 in k_2, to pomeni, da novo zgrajeni konstrukt ne more imeti manjše vrednosti Ψ kot njegovi sestavni deli. Če se omejimo le na konjunktivne širitve, vemo, da nobena nadaljnja razširitev ne more postati boljša. Konstruktov, ki so slabši od svojih sestavnih delov, ne potrebujemo in jih ni potrebno zgraditi.

Preden zgradimo konjunktiven konstrukt, vedno preverimo najboljši možni točki. Če sta vrednosti Ψ tu slabši (večji) od vrednosti Ψ sestavnih delov, nam gradnje sploh ni potrebno opraviti.

Omejitev disjunkcije

S konjunktu dualno tehniko lahko omejimo tudi disjunkt.

Zgradimo disjunktni konstrukt: $k = k_1 \vee k_2$. Namesto primerov, ki ustrezajo logičnima izrazoma k_1 in k_2, moramo za omejitev upoštevati primere, ki izrazoma ne ustrezajo (ti določajo para $(\overline{p_1}, \overline{n_1})$ in $(\overline{p_2}, \overline{n_2})$). Disjunkcija namreč kvečjemu poveča število primerov, ki izrazu ustrezajo in zmanjša število primerov, ki mu ne ustrezajo. Slednji so omejeni na enak način kot je opisano v prejšnjem razdelku za konjunkcijo. Na omejitev za disjunkcijo lahko sklepamo tudi iz omejitve za konjunkcijo in De Morganovega zakona: $k_1 \vee k_2 = \overline{\overline{k_1} \wedge \overline{k_2}}$. Povejmo, da tu omejitev velja le, če bodo nadaljne širitve disjunktne.

Preden zgradimo disjunkt, zato preverimo vrednosti funkcije Ψ v točkah: $(\min(\overline{p_1}, \overline{p_2}), 0)$ in $(0, \min(\overline{n_1}, \overline{n_2}))$. Če sta obe vrednosti večji od $\Psi(p_1, n_1)$ in $\Psi(p_2, n_2)$, disjunkcije ne zgradimo.

Okno okoli konstrukta

Preden razmislimo o še eni omejitvi kandidatov za razširitev konstrukta, si ponovno oglejmo sliko 2.6. Slabši konstrukti (z veliko vrednostjo funkcije Ψ) ležijo blizu diagonale $[(0, 0), (P_{max}, N_{max})]$. Dobri konstrukti (z manjšimi vrednostmi Ψ) ležijo bližje osi P oziroma N.

Postavimo konstrukt k s točko (p, n) v sliko. S pridruževanjem novih členov želimo ustvariti konstrukte, ki bodo bližje osi P oz. N. Z oknom zato odstranimo tiste kandidate za pridružitev, ki so predaleč stran od konstrukta k ali od osi P oz. N.

Širina okna W, ki je vrednost med 0 in 1, nam pove, kolikšen delež kandidatov bomo obdržali v postopku. Tako vrednost 1 pomeni, da okno ne izloči nobenega kandidata, 0.2 pa, da bomo poskušali razširitev le s petino vrednosti atributov.

2.5 Problemi z več razredi

Ker omejitve problemskega prostora pri gradnji konstruktov delujejo le za probleme z dvema razredoma, moramo probleme z več razredi prevesti na probleme z dvema razredoma. To lahko storimo tako, da uvedemo dva super razreda *(Breiman in sod., 1984)*, od katerih vsak združuje več prvotnih razredov. Da bi v nekem vozlišču ugotovili, katera je najboljša razporeditev razredov v dve skupini, bi morali preskusiti več možnosti in za vsako ponoviti postopek gradnje konstruktov. Čas izračunov bi tako preveč narasel. Iz zagate se rešimo tako, da izberemo v prvo skupino le razred, ki mu v danem vozlišču pripada največ primerov, v drugo skupino pa uvrstimo vse ostale razrede. Poskusi so pokazali, da je takšna rešitev sprejemljiva.

2.6 Rezanje

Da bi se izognil preveliki prilagoditvi podatkom in prevelikim drevesom, uporablja LFC rezanje konstruktov in rezanje drevesa.

Oba postopka ocenjujeta napako pred in po rezanju. LFC napako ocenjuje s klasifikacijo na neodvisnem testnem vzorcu (test sample error estimation), ki je opisana v *(Breiman in sod., 1984)*. Za testni vzorec si moramo zagotoviti dodatne primere, ki v gradnji drevesa sicer ne smejo sodelovati.

2.6.1 Rezanje konstruktov

Z izrazom *rezanje konstruktov* označujemo v LFC naslednji postopek:

Število napačnih razvrstitev, ki jih zagreši konstrukt na testnem vzorcu, primerjamo s številom napačnih razvrstitev za en člen skrajšanega konstrukta. Če je število slednjih manjše, zadnji člen konstrukta odvržemo in nadaljujemo postopek. Z rezanjem končamo, ko se napaka ne manjša več ali ko ostane samo še en člen.

Načeloma bi lahko odstranjevali katerikoli člen v konstruktu in ne le zadnjega, vendar izkušnje kažejo, da so zadnji členi tisti, ki so najmanj pomembni in ki povzročijo preveliko prilagoditev podatkom.

2.6.2 Rezanje drevesa

Pod pojmom *rezanje odločitvenih dreves* razumemo odstranitev tistih poddreves, za katere nam nek kriterij pove, da ne bodo izboljšala natančnosti klasifikacije. Obstajata dve glavni skupini rezanja:

- predhodo rezanje (preprunning)
- kasnejše rezanje (postprunning)

Predhodno rezanje opravimo kar med samim postopkom izgradnje odločitvenega drevesa. Gradnja drevesa se ustavi, ko je izpolnjen eden od naslednjih ustavitvenih pogojev oziroma pogojev predhodnega rezanja *(Gams in Lavrač, 1987; Cestnik in sod., 1987; Breiman in sod., 1984)*:

1. Vsi primeri so iz istega razreda.
2. Delež primerov v vozlišču v primerjavi z vsemi primeri je manjši od nekega vnaprej določenega odstotka.
3. Število oziroma teža primerov v vozlišču, je manjše od vnaprej določene vrednosti.
4. Delež večinskega razreda v vozlišču preseže vnaprej določen odstotek.
5. Prikladnost atributa (attribute suitability) *(Cestnik in sod., 1987)* pade pod vnaprej določeno vrednost.

V literaturi sem našel dve uspešni metodi *kasnejšega rezanja*:

Niblett-Bratkova metoda z m-oceno verjetnosti *(Cestnik, 1991)*. Metoda neko vozlišče obreže, če ocenjena napaka neobrezanega vozlišča preseže ocenjeno napako obrezanega vozlišča. Napaki ocenjujemo z m-oceno verjetnosti.

Breimanova metoda *(Breiman in sod., 1984)*; Pri tej metodi zgradimo kar največje možno drevo, pri rezanju pa tvorimo zaporedje različno velikih dreves. Kot rezultat izberemo drevo, katerega ocenjena napaka je najmanjša. Napako ocenjujemo bodisi s klasifikacijo na neodvisni testni množici primerov bodisi s prečnim preverjanjem (cross-validation).

Originalno sta Ragavan in Rendell v LFC ustavila gradnjo, ko je večinski razred presegel določen odstotek. Za rezanje sta priredila Breimanovo metodo, tako da v vsakem vozlišču ocenjujeta napako z neodvisnim testnim vzorcem.

Sam sem eksperimentiral z vsemi zgoraj navedenimi ustavitvenimi pogoji in rezanji. Rezanje je nekoliko podrobneje razloženo v naslednjem poglavju.

Do sedaj opisani postopki so primerni za problemske domene, kjer ni manjkajočih, nepopolnih ali napačnih podatkov. Ocenjevanje verjetnosti je povsod izvedeno z relativno frekvenco.

Pristop do šumnih podatkov, boljši način ocenjevanja verjetnosti ter nekateri drugi koristni pripomočki graditeljev odločitvenih dreves so opisani v naslednjem poglavju.

3.

Orodjarna

V tem poglavju opisujem postopke in pristope, ki jih uporabljamo pri strojnem učenju iz šumnih ali nepopolnih podatkov. Deloma so opisani v *(Cestnik in sod., 1987; Gams in Lavrač, 1987; Cestnik, 1991; Breiman in sod., 1984)*.

Skupno tvorijo ti postopki dobro založeno orodjarno, ki jo lahko uporabimo ob raznih priložnostih, ko je v strojnem učenju potrebno delati s šumnimi in nezadostnimi podatki.

3.1 Bayesovsko ocenjevanje verjetnosti

V strojnem učenju najpogosteje najdemo ocenjevanje verjetnosti z *relativno frekvenco*. Verjetnost ocenimo kot količnik med številom uspešnih poskusov (n) in skupnim številom poskusov, ki smo jih opravili (N):

$$p = \frac{n}{N} \tag{3.1}$$

Ocenjevanje verjetnosti z relativno frekvenco je zlasti neustrezno v primerih, ko imamo na voljo malo podatkov. Če na primer ocenjujemo verjetnost cifre pri metu kovanca in dvakrat zapored pade grb, bomo z relativno frekvenco ocenili, da je verjetnost pojavitve cifre 0 ($p = \frac{0}{2}$), grba pa 1. Intuitivno nam je jasno, da je takšna ocena prenagljena in zavajajoča.

V takšnih primerih je boljša *Laplaceova ocena*:

$$p = \frac{n+1}{N+T} \tag{3.2}$$

kjer T predstavlja število možnih izidov poskusa. V gornjem primeru bi verjetnost cifre tako ocenili kot $p = \frac{0+1}{2+2} = \frac{1}{4}$.

Bojan Cestnik je razvil metodo, ki dovoljuje vključitev predznanja o problemski domeni v oceno verjetnosti *(Cestnik, 1991)*. Ocena izhaja iz splošnega bayesovskega ocenjevanja verjetnosti in predpostavlja β porazdelitev. Verjetnost dogodka x ocenjujemo z izrazom:

$$p = \frac{n + m \cdot p_a}{N + m} \tag{3.3}$$

kjer p_a predstavlja a priori verjetnost za dogodek x, ne negativen m pa je parameter β porazdelitve. Po parametru m je ta ocena dobila tudi svoje ime: *m-ocena verjetnosti*.

Varianca začetne porazdelitve je:

$$\mathrm{Var}(p) = \frac{p_a(1-p_a)}{m+1} \tag{3.4}$$

Izraz 3.3 si lahko predstavljamo, kot da smo enkrat v preteklosti že opravili m poskusov in pri njih ugotovili verjetnost p_a; zdaj opravimo še dodatnih N poskusov, od katerih jih je n v naš prid. Rezultate združimo: skupno število poskusov je $N+m$, od tega jih hipotezo potrjuje $n+p_a m$. Izračunamo verjetnost z relativno frekvenco $p = \frac{n+p_a m}{N+m}$ in dobili smo točno izraz 3.3.

Zanimivo (in skladno z intuitivno predstavo) je, da v primeru $m = 0$, preidemo na ocenjevanje z relativno frekvenco, ko pa gre $m \to \infty$ dobimo kot rezultat apriorno verjetnost.

Oglejmo si uporabnost m-ocene verjetnosti.

3.1.1 Uporaba m-ocene verjetnosti

Bayesovsko ocenjevanje verjetnosti sem uporabljal tako pri gradnji kot pri rezanju odločitvenih dreves.

Problem, ki ga moramo rešiti, preden lahko zamenjamo relativno frekvenco z m-oceno verjetnosti, je določitev vnaprejšnje verjetnosti p_a in parametra m.

Pri gradnji drevesa moramo v nekem vozlišču izračunati verjetnosti v izrazu za $\Psi(F)$ (2.3). Poglejmo si obe tipični predstavnici verjetnosti, ki jih moramo oceniti:

- $p(f)$ verjetnost, da razvrstimo primer v vejo, ki ustreza konstruktu,
- $p(y|f)$ verjetnost, da je primer iz razreda y, če pade v ustrezajočo vejo.

Apriorni verjetnosti za pogojno in brezpogojno verjetnost ocenimo iz cele učne množice primerov, parameter m [1] pa zahteva malo več razmisleka. Rešitve so lahko različne: vrednost oceni strokovnjak, izračunamo jo iz variance, nanjo gledamo kvalitativno. V mojem programu je m parameter, ki ga določa uporabnik.

Pri rezanju nam ocenjevanje verjetnosti z m-oceno koristi v Niblett-Bratkovi metodi rezanja. Ta v vseh nekončnih vozliščih primerja dve napaki:

statično E_s**:** verjetnost napake, če bi veje porezali in

dinamično E_d**:** verjetnost napake, če vej ne bi oklestili.

Vozlišče obrežemo, če je dinamična napaka večja od statične. Izraza za oceno obeh napak z relativno frekvenco sta:

$$E_s = 1 - \frac{n(c|t)}{n(t)} \qquad\qquad E_d = \sum_{v=1}^{st.vej} p_v E_v \tag{3.5}$$

kjer $n(c|t)$ pomeni število primerov iz večinskega razreda v vozlišču t, $n(t)$ število vseh primerov v tem vozlišču, p_v verjetnost v-te veje in E_v napako v-te veje.

Z m-oceno postane izraz za E_s naslednji:

$$E_s = 1 - \frac{n(c|t) + p(c)m}{n(t) + m} \tag{3.6}$$

Vidimo, da smo apriorno verjetnost večinskega razreda zopet ocenili iz cele učne množice (kot brezpogojno verjetnost). Prikladno je, da apriorno verjetnost ocenjujemo z Laplaceovim obrazcem 3.2. Kako izračunati vnaprejšnjo

[1] V principu bi morali za ocenjevanje vsake od obeh verjetnosti imeti poseben parameter m, vendar je Cestnik v svojih poskusih ugotovil, da to ne prinese izboljšanja v klasifikacijski natančnosti.

verjetnost, da pade primer v v-to vejo, ki jo potrebujemo za izračun dinamične napake z m-oceno, smo že opisali.

Tudi pri rezanju gledamo na parameter m kvalitativno in preskušamo natančnost za več velikostnih razredov npr. $m = 0, 0.01, 2, 5, 10, 20$. S spreminjanjem parametra m dobimo različno velika drevesa, kar je zaželena lastnost.

3.2 Manjkajoče vrednosti atributov

V mnogih realnih problemskih domenah se dogaja, da pri nekaterih primerih manjkajo vrednosti določenih atributov. V medicinskih domenah se na primer često zgodi, da na pacientu niso opravili vseh možnih preiskav in tako določenih vrednosti nimamo na razpolago.

Manjkajoče vrednosti lahko obravnavamo na več različnih načinov. Lahko dodamo posebno vrednost, ki označuje neznano vrednost, in razširimo možne vrednosti atributa še s to vrednostjo, vendar se zaradi tega odločitveno drevo poveča in potrebno je priskrbeti dodatne primere, da pokrijemo vse liste. Boljša se zdi rešitev, ki je predlagana v *(Cestnik in sod., 1987)*, vendar razširjena z m-oceno verjetnosti.

Atributu z neznano vrednostjo pripišemo vse možne vrednosti, vsako s pripadajočo pogojno verjetnostjo. Če primer pripada razredu y, potem ima v vozlišču t atribut X vrednost v z verjetnostjo:

$$p(v|y,t) = \frac{n(v\&y|t) + p_a(v|y)m}{n(y|t) + m} \tag{3.7}$$

kjer $n(v\&y|t)$ označuje število primerov v vozlišču t, ki imajo vrednost atributa v in pripadajo razredu y, $n(y|t)$ pa skupno število primerov iz razreda y v vozlišču t. Apriorno verjetnost $p_a(v|y)$ izračunamo iz cele učne množice z Laplaceovim obrazcem 3.2.

Primer z manjkajočo vrednostjo atributa se razdeli v obe veji s pripadajočima verjetnostma. Pri gradnji drevesa zato ne delamo s številom primerov, ampak z utežjo primerov. Utež primerov v vozlišču nam pomeni vsoto verjetnosti vseh primerov v vozlišču. Če v problemski domeni manjkajočih vrednosti ni, potem so verjetnosti vseh primerov 1 in delamo v bistvu s številom primerov.

Atributi z manjkajočimi vrednostmi nastopajo seveda tudi v konstruktih. Če je mogoče, logičen izraz, ki določa konstrukt, izračunamo tako, da atribut z manjkajočo vrednostjo ne vpliva na njegovo vrednost. V primeru, da to ni mogoče, izračunamo pogojne verjetnosti po obrazcu 3.7

3.2.1 Klasifikacija primera z manjkajočo vrednostjo

Primer klasificiramo tako, da sledimo vejam glede na njegove vrednosti atributov. Če v vozlišču ne moremo določiti vrednosti konstrukta zaradi manjkajoče vrednosti atributa, sledimo obema vejama, vsaki z verjetnostjo, ki smo jo že predhodno (pri gradnji) s pomočjo m-ocene verjetnosti izračunali iz učne množice primerov. Ko se klasifikacija v vseh vejah ustavi, pristanejo deli primera v listih z različno utežno porazdelitvijo razredov. Zdaj se proces obrne in porazdelitve uteži potujejo nazaj proti korenu drevesa. V vsakem vozlišču se glede na predhodne verjetnosti veje sestavijo v ustrezno utež. Porazdelitev uteži, ki jo dobimo v korenu drevesa, predstavlja klasifikacijo primera; izberemo razred z največjo utežjo.

4.

Rezultati

V tem poglavju bom predstavil rezultate algoritma LFC in jih poskušal razložiti.

Algoritem sem testiral na nekaj realnih problemih, za katere obstajajo tudi rezultati drugih algoritmov, ter na nekaj umetnih problemih.

Za začetek naj malo podrobneje predstavim testne domene.

4.1 Testni problemi

Slika 4.1 prikazuje nekaj značilnih podatkov o testnih domenah.

Kratek opis domen:

- ANKE (anketa) predstavlja problem s področja revmatologije
- ASIS je umetna domena, generirana za sistem ASSISTENT
- BAYS je umetna domena, prilagojena klasifikaciji z naivnim bayesovim klasifikatorjem
- BO1L je sestavljena iz logičnega izraza v Boolovi algebri
- BREA je problem diagnoze raka na dojki
- DIAB opisuje diabetes pri Indijankah iz plemena Pima
- E1PI opisuje problem iz fizike (razločevanje elektronov in pionov)

označba domene	število atributov	#vrednosti atributov	število razredov	število primerov	večinski razred (%)
ANKE	32	292	6	355	66
ASIS	11	22	2	200	57
BAYS	10	20	2	200	55
BO1L	6	12	2	640	67
BREA	10	27	2	288	80
DIAB	8	70	2	768	65
E1PI	18	299	2	500	50
HEPA	19	73	2	155	79
IRIS	4	24	3	150	33
K1RK	18	36	2	1000	67
K2RK	6	48	2	1000	67
LE1D	7	14	10	1000	11
LYMP	18	60	4	148	55
P27Y	7	14	2	128	50
P37Y	7	14	2	128	50
P2AR	12	24	2	200	54
P3AR	13	26	2	200	54
P400	14	18	2	400	50
PRIM	17	37	22	339	25
SCIT	15	137	4	884	56
SOYB	35	100	15	630	15
VO1E	16	32	2	435	61
Z2RM	14	66	4	254	42

Slika 4.1: Nekaj značilnih podatkov o testnih domenah

- HEPA opisuje diagnozo hepatitisa
- IRIS sestavljajo podatki za določitev vrste irisa
- K1RK in K2RK sta šahovski domeni (ilegalnost končnic trdnjave in kralja proti kralju)
- LE1D vsebuje primere napovedi prikaza na sedem segmentnem LED prikazovalniku
- LYMP pomeni limfografijo
- P27Y in P37Y sta umetni domeni, ki predstavljata problem parnosti, kjer za izračun parnosti upoštevamo dva oz. tri izmed sedmih atributov (2/7, 3/7)
- P2AR, P3AR in P400 so problemi parnosti 2/12, 3/13, 4/14
- PRIM označuje primarni tu mor
- v SCIT so opisani primeri obolenj ščitnice
- SOYB predstavlja diagnozo bolezni soje
- VO1E pomeni predvidevanja izidov volitev v ZDA
- Z2RM označujeje realen problem iz industrije (napovedovanje lastnosti betona)

Podatki o številu atributov, številu vrednosti atributov, številu razredov in številu primerov lahko služijo kot (sicer nezanesljiv) pokazatelj računske zahtevnosti, odstotek primerov iz večinskega razreda pa nam služi za grobo primerjavo uspešnosti učenja.

Vsi testi so sestavljeni iz gradnje dreves na desetih razbitjih vsake od testnih domen. V vseh poskusih sem uporabljal enaka razbitja. 70% primerov je uporabljenih za učenje, ostalih 30% sestavlja testno množico. Rezultati v tabelah so izračunani kot povprečje desetih primerov, dodan je tudi standardni odklon.

4.2 S konstruktivno indukcijo in brez nje

Algoritem LFC lahko gradi konstrukte različnih oblik, od enega samega atributa do zapletenih logičnih pravil. Ta njegova dinamičnost mu zago-

tavlja, da bo natančnost klasifikacije na testnih primerih gotovo vsaj tako dobra, kot pri algoritmu brez konstruktivne indukcije.

Ker želimo ugotoviti predvsem prednosti, ki nam jih daje konstruktivna indukcija, sem rezultate za LFC meril z vključeno konstrukcijo, čeprav je bil rezultat včasih slabši.

Primerjava natančnosti klasifikacije algoritma s konstruktivno indukcijo in brez nje, je prikazana na sliki 4.2. Navedeni so najboljši rezultati, kar sem jih dobil v več poskusih na vsaki od domen. Parametri, s katerimi sem dobil te rezultate, se od domene do domene razlikujejo. Vrednosti v oklepajih pomenijo standardni odklon. Stolpca z informacijsko vsebino povesta, kolikšna je vrednost posameznega rezultata v primerjavi z apriorno verjetnostjo. Večja je vrednost informacijske vsebine, boljši je klasifikator.

Informacijsko vsebino neke klasifikacije (information score) sta definirala Kononenko in Bratko *(1991)*, izraz zanjo povzemam po *(Kononenko, 1992)*:

$$Inf = \frac{\sum_{i=1}^{T} Inf_i}{T} \tag{4.1}$$

$$Inf_i = \left\{ \begin{array}{lll} -\log_2 P(Cl_i) + \log_2 p(Cl_i) & ; & p(Cl_i) \geq P(Cl_i) \\ -[-\log_2(1 - P(Cl_i)) + \log_2(1 - p(Cl_i))] & ; & p(Cl_i) < P(Cl_i) \end{array} \right\}$$

kjer je T število testnih primerov, Cl_i razred, ki mu pripada i-ti testni primer, $P(Cl_i)$ apriorna verjetnost razreda Cl_i, $p(Cl_i)$ pa verjetnost razreda Cl_i, kot jo vrne klasifikator.

Zanimiva je tudi primerjava kompleksnosti obeh dreves izražena s številom listov. Najdemo jo na sliki 4.3. Več o kompleksnosti najdemo v razdelku o razumljivosti konstruktov na strani 40.

Ocenimo, ali je napredek algoritma LFC značilen v primerjavi z algoritmom brez konstruktivne indukcije.

4.2.1 Statistična pomembnost razlik

Da bi pokazal značilnost odstopanj rezultatov algoritma LFC, sem uporabil dvostranski parametrični test zaupanja *(Jamnik, 1980; Cestnik, 1991)*.

V okviru ene domene bomo značilnosti odstopanj merili na $F = 10$ ponovitvah poskusa. Izračunamo razliko med rezultatom, ki ga da LFC, in tistim

		brez konstrukcije		LFC	
domena	večina	natan.(%)	inf. vsebina	natan.(%)	inf. vsebina
ANKE	66	64.1(4.5)	0.37(0.09)	67.5(4.1)	0.32(0.07)
ASIS	57	80.1(4.5)	0.53(0.07)	77.4(5.3)	0.50(0.09)
BAYS	55	72.2(3.6)	0.32(0.06)	74.8(3.5)	0.43(0.06)
BO1L	67	89.0(1.6)	0.55(0.03)	89.0(1.6)	0.56(0.03)
BREA	80	79.1(2.6)	-0.04(0.03)	78.5(5.2)	-0.05(0.03)
DIAB	65	74.6(2.4)	0.27(0.03)	72.6(3.3)	0.26(0.05)
E1PI	50	83.1(3.3)	0.64(0.04)	84.7(2.3)	0.58(0.03)
HEPA	79	80.4(4.5)	0.25(0.11)	81.8(2.8)	0.27(0.09)
IRIS	33	96.4(1.7)	1.47(0.04)	96.2(1.6)	1.49(0.02)
K1RK	67	98.9(0.6)	0.87(0.02)	99.1(0.9)	0.88(0.03)
K2RK	67	67.5(2.0)	0.02(0.02)	88.0(2.9)	0.56(0.05)
LE1D	11	72.3(3.5)	2.21(0.10)	72.2(2.8)	2.19(0.07)
LYMP	55	77.4(3.5)	0.61(0.09)	83.0(3.3)	0.78(0.09)
P27Y	50	67.9(23.1)	0.50(0.37)	100(0.0)	1.00(0.00)
P37Y	50	55.1(18.9)	0.25(0.32)	99.2(2.5)	0.98(0.05)
P2AR	54	65.7(11.5)	0.28(0.21)	95.6(3.6)	0.86(0.04)
P3AR	54	63.5(14.2)	0.25(0.27)	83.0(1.7)	0.63(0.20)
P400	50	55.7(6.4)	0.11(0.13)	61.3(13.9)	0.19(0.27)
PRIM	25	41.0(4.9)	1.21(0.11)	37.5(4.8)	1.06(0.14)
SCIT	56	67.4(1.9)	0.53(0.06)	67.9(2.3)	0.47(0.06)
SOYB	15	87.7(2.8)	3.07(0.12)	88.7(2.4)	3.07(0.09)
VO1E	61	96.1(1.2)	0.85(0.02)	95.5(0.9)	0.84(0.03)
Z2RM	42	58.7(4.6)	0.61(0.11)	59.5(2.9)	0.68(0.07)

Slika 4.2: Natančnost in informacijska vsebina klasifikacije brez konstruktivne indukcije in z njo

	brez konstrukcije		LFC	
domena	#pred	#po	#pred	#po
ANKE	34.2(3.5)	23.9(5.9)	31.6(3.5)	6.4(3.8)
ASIS	30.7(1.8)	17.5(1.8)	20.8(1.9)	17.3(2.9)
BAYS	12.1(2.1)	6.2(1.5)	32.5(2.8)	16.1(2.5)
BO1L	60.2(2.1)	25.8(5.9)	56.4(2.8)	5.6(1.0)
BREA	6.3(2.8)	4.3(2.0)	14.2(3.8)	2.7(1.9)
DIAB	19.3(3.4)	16.2(2.6)	37.1(2.1)	8.7(4.4)
E1PI	25.9(3.4)	24.7(2.9)	16.8(2.9)	3.8(2.1)
HEPA	11.6(2.2)	10.1(1.6)	3.9(0.8)	3.9(0.8)
IRIS	7.2(1.1)	5.2(0.4)	4.6(1.3)	3.8(0.4)
K1RK	10.6(0.8)	10.6(0.8)	5.2(1.0)	5.2(1.0)
K2RK	9.9(10.1)	3.7(4.2)	45.7(4.3)	22.9(7.3)
LE1D	74.8(3.5)	32.0(2.2)	72.1(4.0)	28.8(2.0)
LYMP	14.1(2.2)	8.4(2.1)	8.3(0.9)	7.7(1.2)
P27Y	17.8(8.5)	12.6(5.2)	3.3(0.5)	3.3(0.5)
P37Y	23.2(4.1)	18.2(3.7)	10.0(2.9)	9.9(2.9)
P2AR	35.4(2.4)	27.2(2.5)	5.6(1.7)	3.0(0.0)
P3AR	33.6(7.7)	25.5(7.8)	14.8(2.0)	11.8(3.1)
P400	79.9(4.0)	58.1(5.3)	32.4(7.1)	11.0(5.9)
PRIM	99.2(5.7)	58.7(4.4)	87.2(4.1)	53.2(3.7)
SCIT	43.9(5.4)	34.0(6.7)	37.3(8.1)	11.1(4.9)
SOYA	38.8(5.2)	33.6(3.3)	37.8(3.5)	34.1(4.8)
VO1E	8.8(1.2)	7.3(1.3)	6.7(1.4)	6.0(1.3)
Z2RM	54.8(4.3)	34.2(7.1)	27.4(5.6)	22.0(3.8)

Slika 4.3: Število listov v drevesih brez konstruktivne indukcije in z njo, pred in po rezanju

brez konstruktivne indukcije:

$$z_f = \tau(LFC)_f - \tau(BKI)_f$$

pri čemer je $\tau(LFC)_f$ rezultat algoritma LFC v f-ti ponovitvi poskusa, $\tau(BKI)_f$ pa rezultat algoritma **b**rez **k**onstruktivne **i**ndukcije v f-ti ponovitvi poskusa.

Predpostavimo, da je z_f porazdeljen normalno s parametri $N(z, \sigma)$ in preiskušamo hipotezo, da dosegata oba algoritma enake rezultate, oziroma $H(z = 0)$.

Izračunamo:

$$\overline{z} = \frac{1}{F}\sum_{f=1}^{F} z_f \qquad\qquad s^2 = \frac{1}{F-1}\sum_{f=1}^{F}(z_f - \overline{z})^2 \tag{4.2}$$

Iz česar dobimo parameter t za Studentovo porazdelitev:

$$t = \frac{\overline{z}}{s}\sqrt{F} \tag{4.3}$$

Če je $|t|$ večji od neke meje, lahko hipotezo $H(z = 0)$ (algoritma dajeta enake rezultate) zavrnemo z določenim zaupanjem. Stopnje zaupanja in meje za $|t|$ so podane na sliki 4.4.

Stopnja zaupanja	20%	40%	60%	80%
Meja za $\|t\|$:	0.26	0.542	0.879	1.372

Stopnja zaupanja:	90%	95%	98%	99%
Meja za $\|t\|$:	1.812	2.228	2.764	3.169

Slika 4.4: Tabela za parametrični test značilnosti odstopanja

Izračuni so pokazali, da razlike pri natančnosti klasifikacije primerov iz testne množice, niso slučajne z naslednjimi stopnjami zaupanja:

- 80% : E1PI, HEPA

- 90% : BAYS
- 95% : ANKE
- 99% : K2RK, LYMP, P27Y, P37Y, P2AR, P3AR, P400

Značilnosti odstopanj pri informacijski vsebini so naslednje:

- 80% : P400, Z2RM
- 99% : BAYS, K2RK, LYMP, P27Y, P37Y, P2AR, P3AR

Pri ostalih domenah je verjetnost, da ne gre za slučajno razliko, manjša od 80% in ne moremo govoriti o boljših rezultatih z algoritmom LFC.

4.3 Konjunktivni konstrukti

Ragavan in Rendell trdita, da strokovnjaki lažje razumejo rezultate, če so konstrukti zloženi le s konjunkcijo in negacijo. Ker sta konjunkcija in negacija funkcijsko poln nabor, z omejitvijo ne izgubimo na splošnosti.

Rezultati, ki jih dobimo, če se omejimo le na $\{\wedge, \neg\}$, so zbrani v tabeli na sliki 4.5.

Rezultati natančnosti klasifikacije, razen pri domenah parnosti, ne odstopajo dosti od tistih brez konjunktivne indukcije, marsikje pa so celo slabši. Očitno so odvisnosti, ki jih zajamemo s konjunkcijo, dosti dobro izražene že v drevesih brez konstruktrov, kjer vsaka pot od korena do lista predstavlja konjunktiven izraz.

Glede razumljivosti se omejitev na konjunkcijo izkaže kot koristna, parnosti smo na primer vajeni le v konjunktivni obliki.

4.4 Disjunktivni konstrukti

Ker je znana dualnost konjunkcije in disjunkcije v Boolovi algebri, sem poskušal tudi z omejitvijo le na disjunkcijo in negacijo (prav tako funkcijsko poln nabor operatorjev).

označba domene	natančnost (%)	število listov	inf. vsebina
ANKE	60.7(3.9)	46.1(4.4)	0.21(0.09)
ASIS	77.4(5.3)	17.3(2.9)	0.50(0.09)
BAYS	69.6(4.0)	16.8(2.2)	0.35(0.07)
BO1L	89.0(1.6)	5.6(1.0)	0.56(0.03)
BREA	76.4(3.9)	31.6(8.2)	0.03(0.06)
DIAB	69.5(1.6)	52.4(6.1)	0.27(0.04)
E1PI	82.1(1.8)	17.1(1.8)	0.63(0.04)
HEPA	78.4(5.2)	6.1(1.4)	0.19(0.13)
IRIS	93.4(3.8)	4.9(1.1)	1.43(0.07)
K1RK	99.0(0.8)	8.6(1.3)	0.88(0.04)
K2RK	63.7(3.7)	92.0(4.4)	0.11(0.07)
LE1D	71.2(2.6)	21.3(2.5)	2.15(0.05)
LYMP	76.8(6.8)	8.7(1.0)	0.64(0.12)
P27Y	100(0.0)	3.3(0.5)	1.00(0.00)
P37Y	99.2(2.5)	9.9(2.9)	0.98(0.05)
P2AR	93.6(3.0)	4.7(2.1)	0.84(0.04)
P3AR	83.0(10.7)	11.8(3.1)	0.63(0.20)
P400	60.0(8.2)	45.5(6.2)	0.19(0.17)
PRIM	33.1(6.1)	67.8(5.1)	0.95(0.08)
SCIT	56.9(3.0)	5.6(1.8)	0.25(0.09)
SOYB	88.7(2.4)	34.1(4.8)	3.07(0.09)
VO1E	95.5(0.7)	6.7(1.4)	0.84(0.03)
Z2RM	54.6(7.2)	18.2(3.5)	0.50(0.09)

Slika 4.5: Rezultati algoritma LFC pri konjunktivnih konstruktih

Poskusi so dali dobre rezultate v nekaterih domenah. (npr. LYMP). Očitno se odvisnosti v teh domenah lažje izražajo z disjunkcijo kot s konjunkcijo.

Čeprav so disjunktivni konstrukti težje razumljivi, lahko večja natančnost klasifikacije morda to odteha. Rezultati so na sliki 4.6.

4.5 Poljubni konstrukti

Če pustimo algoritmu LFC, da sam izbere obliko konstrukta, dobimo rezultate s slike 4.7. Rezultati so pri isti širini in globini iskanja večinoma boljši od dreves s samo konjunktivnimi ali samo disjunktivnimi konstrukti. To ni presenetljivo, saj konjunkcijo in disjunkcijo precej dobro izražajo že drevesa brez konstruktov. Poljubna kombinacija konjunkcije, disjunkcije in negacije predstavlja zadosti drugačen zapis, da se pokažejo tudi drugačne odvisnosti med atributi.

Na žalost so dobljeni konstrukti včasih precej zapleteni in človeku težko doumljivi.

4.6 Težavnost problemov

Opazimo lahko, da so izboljšanja natančnosti, ki jih prinese LFC v primerjavi z algoritmom brez konstruktivne indukcije, pri različnih domenah različna. To si razlagam z dejstvom, da pridejo prednosti algoritma LFC do izraza pri problemih z visoko stopnjo interakcije med atributi, prav takšni problemi pa so za algoritme brez konstrukcije tudi najtežji.

Pojavi se vprašanje, kako oceniti težavnost problema. Obstajajo različne mere težavnosti, prilagojene različnim namenom, na primer statistična korelacija, variacija in število vrhov funkcije pripadnosti razredu, entropija, itd. Merilo, prirejeno posebej za algoritem LFC, se imenuje zamegljenost koncepta.

označba domene	natančnost (%)	število listov	inf. vsebina
ANKE	67.5(4.1)	6.4(3.8)	0.32(0.07)
ASIS	74.9(5.9)	18.3(2.6)	0.45(0.12)
BAYS	70.3(4.5)	19.1(5.6)	0.37(0.07)
BO1L	89.0(1.6)	11.8(2.5)	0.56(0.03)
BREA	78.5(5.2)	2.7(1.9)	-0.05(0.07)
DIAB	71.6(2.7)	43.4(10.5)	0.30(0.04)
E1PI	84.7(2.3)	3.8(2.1)	0.58(0.03)
HEPA	78.3(5.2)	2.6(1.0)	0.13(0.10)
IRIS	95.9(2.0)	5.4(0.8)	1.48(0.05)
K1RK	98.5(1.1)	7.1(0.7)	0.86(0.04)
K2RK	88.0(2.9)	22.9(7.3)	0.56(0.05)
LE1D	71.1(9.5)	24.9(2.6)	2.14(0.10)
LYMP	82.3(2.8)	9.1(0.8)	0.75(0.06)
P27Y	100(0.0)	3.7(0.5)	1.00(0.00)
P37Y	99.3(2.0)	9.5(3.2)	0.99(0.04)
P2AR	94.4(2.7)	4.3(1.1)	0.84(0.04)
P3AR	72.2(13.8)	18.1(6.1)	0.41(0.27)
P400	56.8(6.4)	49.0(6.2)	0.13(0.12)
PRIM	37.5(3.8)	53.2(3.7)	1.06(0.14)
SCIT	67.9(2.3)	11.1(4.9)	0.47(0.06)
SOYB	81.8(5.9)	35.3(1.7)	2.99(0.12)
VO1E	95.4(1.0)	2.6(1.5)	0.81(0.02)
Z2RM	59.0(7.2)	25.9(4.2)	0.65(0.07)

Slika 4.6: Rezultati algoritma LFC omejenega na tvorbo le disjunktivnih konstruktov

označba domene	natančnost (%)	število listov	inf. vsebina
ANKE	65.7(2.8)	5.9(4.2)	0.30(0.04)
ASIS	74.8(4.3)	20.2(5.2)	0.45(0.09)
BAYS	74.8(3.5)	16.1(2.5)	0.43(0.06)
BO1L	89.0(1.6)	7.1(2.5)	0.56(0.03)
BREA	76.3(4.2)	35.4(7.6)	0.01(0.07)
DIAB	72.6(3.3)	8.7(4.4)	0.26(0.05)
E1PI	82.7(3.4)	21.7(2.5)	0.65(0.06)
HEPA	81.8(2.8)	3.9(0.8)	0.27(0.09)
IRIS	96.2(1.6)	3.8(0.4)	1.49(0.04)
K1RK	99.1(0.9)	5.2(1.0)	0.88(0.03)
K2RK	81.7(2.3)	49.6(3.3)	0.50(0.05)
LE1D	72.0(2.8)	28.8(2.0)	2.19(0.07)
LYMP	83.0(3.3)	7.7(1.2)	0.78(0.09)
P27Y	99.2(2.3)	3.4(0.8)	0.99(0.05)
P37Y	94.9(8.6)	8.5(2.3)	0.90(0.18)
P2AR	95.6(2.1)	3.0(0.0)	0.86(0.04)
P3AR	77.3(17.9)	12.6(5.6)	0.52(0.35)
P400	61.3(13.9)	11.0(5.9)	0.19(0.27)
PRIM	36.4(4.5)	59.4(6.9)	1.00(0.14)
SCIT	57.8(3.4)	15.8(3.9)	0.33(0.03)
SOYB	81.7(5.3)	31.3(2.3)	2.95(0.07)
VO1E	95.3(1.3)	7.1(1.8)	0.84(0.04)
Z2RM	59.5(2.9)	22.0(3.8)	0.68(0.07)

Slika 4.7: Rezultati algoritma LFC, če dovolimo poljubne konstrukte.

4.6.1 Zamegljenost koncepta

Zamegljenost koncepta $\triangle$ (blurring) *(Ragavan in Rendell, 1993; 1993a)* je definirana kot pogojna entropija za problem z dvema razredoma, povprečena prek vseh pomembnih atributov, oziroma kot projekcija na enodimenzionalen podprostor prostora učnih primerov. S to definicijo v bistvu predpostavljamo neodvisne in neredundantne atribute.

$$\triangle = \frac{1}{R}\sum_{i=1}^{R} H(Y|X_i) \tag{4.4}$$

Izraza 2.2 in 2.3 opisujeta entropijo oz. pogojno entropijo za binarne atribute, pri zamegljenosti koncepta pa atribut ni nujno binaren:

$$\triangle = \frac{1}{R}\sum_{i=1}^{R}\sum_{l=1}^{V_i} -p(v_{i,l})[p(y|v_{i,l})\log_2 p(y|v_{i,l}) + p(\overline{y}|v_{i,l})\log_2 p(\overline{y}|v_{i,l})] \tag{4.5}$$

Vrednosti zamegljenosti se nahajajo na intervalu od 0 do 1. Visoka vrednost zamegljenosti pomeni težek koncept, nizka vrednost zamegljenosti pa klasificira problem kot lahek. Pozor, težavnost ni v linearni odvisnosti z zamegljenostjo. Ko se vrednost $\triangle$ približa in preseže 0.9, se težavnost problema, kot ga vidijo standardni algoritmi, močno poveča.

Če sumimo, da pri opisu koncepta niso pomembni vsi atributi, povprečimo pogojne entropije le preko vseh pomembnih atributov; dodatni nepomebni atributi namreč dvignejo vrednost zamegljenosti.

Vrednosti zamegljenosti za domene, ki sem jih testiral, so prikazane na sliki 4.8 v prvih treh stolpcih. Prvi stolpec prikazuje zamegljenost upoštevajoč vse atribute, drugi polovico najboljših (tistih z najnižjimi vrednostmi $H(Y|X_i)$), tretji upošteva samo najboljši atribut. Pri problemih z več kot dvema razredoma je večinski razred upoštevan kot prvi razred, vsi ostali razredi pa tvorijo drugi razred. Takšna omejitev na žalost skrije mnogo težavnosti domen z mnogo razredi.

Ponuja se posplošitev na več razredov. Podaja jo izraz:

$$\triangle' = \frac{1}{R}\sum_{i=1}^{R}\sum_{j=1}^{V_i} p(v_{i,j})\sum_{k=1}^{C} -p(c_k|v_{i,j})\log_2 p(c_k|v_{i,j}) \tag{4.6}$$

domena	$\triangle$(vsi)	$\triangle(\frac{vsi}{2})$	$\triangle(1)$	$\triangle'$(vsi)	$\triangle'(\frac{vsi}{2})$	$\triangle'(1)$	$\bigtriangledown$(vsi)	$\bigtriangledown(\frac{vsi}{2})$	$\bigtriangledown(1)$
ANKE	0.91	0.90	0.82	1.72	1.71	1.64	0.01	0.02	0.09
ASIS	0.97	0.95	0.85	0.97	0.95	0.85	0.02	0.04	0.14
BAYS	0.94	0.89	0.88	0.94	0.89	0.88	0.05	0.10	0.11
BO1L	0.90	0.89	0.89	0.90	0.89	0.89	0.01	0.02	0.02
BREA	0.72	0.71	0.70	0.72	0.71	0.70	0.01	0.02	0.03
DIAB	0.87	0.84	0.76	0.87	0.84	0.76	0.06	0.10	0.17
E1PI	0.94	0.88	0.76	0.94	0.88	0.76	0.06	0.12	0.24
HEPA	0.70	0.66	0.62	0.70	0.66	0.62	0.04	0.08	0.12
IRIS	0.45	0.26	0.20	0.98	0.71	0.58	0.60	0.88	1.00
K1RK	0.88	0.85	0.70	0.88	0.85	0.70	0.03	0.06	0.21
K2RK	0.91	0.91	0.91	0.91	0.91	0.91	0.00	0.00	0.00
LE1D	0.49	0.47	0.44	2.93	2.86	2.84	0.38	0.46	0.47
LYMP	0.93	0.88	0.72	1.24	1.20	1.07	0.04	0.08	0.21
P27Y	1.00	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00
P37Y	1.00	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00
P2AR	0.99	0.98	0.96	0.99	0.98	0.96	0.01	0.01	0.03
P3AR	0.99	0.98	0.97	0.99	0.98	0.97	0.00	0.01	0.02
P400	1.00	0.99	0.99	1.00	0.99	0.99	0.00	0.01	0.01
PRIM	0.78	0.74	0.55	3.71	3.66	3.52	0.05	0.10	0.23
SCIT	0.96	0.92	0.79	1.55	1.49	1.32	0.05	0.11	0.27
SOYB	0.59	0.56	0.51	3.26	3.05	2.66	0.32	0.53	0.92
VO1E	0.71	0.56	0.23	0.71	0.56	0.23	0.25	0.40	0.73
Z2RM	0.85	0.73	0.58	1.68	1.59	1.43	0.10	0.19	0.35

Slika 4.8: Povprečne vrednosti pogojne entropije ($\triangle$ - samo dva razreda, $\triangle'$ - poljubno število razredov) in vzajemne infomacije ($\bigtriangledown$)

Pogojna entropija, ki jo tako dobimo, je prikazana v drugih treh stolpičih na sliki 4.8. Vrednosti zdaj niso več omejene na interval med 0 in 1.

Tudi posplošitev pokaže svoje težave, saj so problemi z mnogo razredi ocenjeni kot pretirano težavni (npr. SOYB). To pomanjkljivost skušamo odpraviti tako, da izračunamo povprečno zmanjšanje nedoločenosti $\bigtriangledown$. Njene vrednosti so predstavljene v zadnjih treh stolpcih na sliki 4.4, definirana pa je z izrazom:

$$\bigtriangledown = -\sum_{k=1}^{C} p(c_k)\log_2 p(c_k) + \frac{1}{R}\sum_{i=1}^{R}\sum_{j=1}^{V_i} p(v_{i,j})\sum_{k=1}^{C} p(c_k|v_{i,j})\log_2 p(c_k|v_{i,j})$$

V nasprotju s prvimi šestimi stolpci predstavlja večja vrednost $\bigtriangledown$ lažje probleme.

Nobeno od zgornjih meril ni idealno za določanje težavnosti problemov, tako na primer nobeno ne upošteva števila primerov in projekcij v večdimenzionalne prostore.

Povežemo lahko uspešnost klasifikacije na neki domeni z algoritmom LFC in zamegljenost domene. Čeprav odvisnost ni povsod prisotna, lahko vendarle opazimo določeno zakonitost: večja je zamegljenost domene, slabša je uspešnost algoritma brez konstruktivne indukcije in večja je razlika v uspešnosti med LFC in tem algoritmom.

Naj omenim tudi, da spadajo skoraj vsi problemi, ki sem jih testiral, glede na zamegljenost med težke probleme. Če bi hoteli dobiti pravo primerjavo med zamegljenostjo in uspešnostjo klasifikatorjev, bi morali upoštevati tudi lažje.

Iz primerjave natančnosti klasifikacije na sliki 4.2 je tako razvidno, da je največja razlika v natančnosti pri domenah parnosti (P27Y, P37Y, P2AR, P3AR) in šahovski domeni K2RK. Kot vidimo na sliki 4.8, so prav to domene, ki imajo najvišje vrednosti $\bigtriangleup$ in najnižje vrednosti $\bigtriangledown$ za različne izbire relevantnih atributov.

4.7 Razumljivost dreves s konstruktivno indukcijo

Mera za kompleksnost odločitvenega drevesa je lahko pri običajnih odločitvenih drevesih kar število vozlišč ali listov *(Gams in Lavrač, 1987)*. Pri drevesih s konstrukti ta primerjava ni čisto poštena, saj imamo lahko npr. drevesi z enakim številom listov, eno ima v vozliščih le po en atribut, pri drugem pa so konstrukti sestavljeni iz zapletenih izrazov. Jasno je, da je drugo drevo kompleksnejše in slabše razumljivo.

Slika 4.3 prikazuje povprečno število listov v drevesih, ki smo jih dobili za posamezne domene. Vidimo, da ima LFC številko povsod precej nižjo. Na isti sliki najdemo tudi podatke za standardni odklon števila listov.

Koristni so tisti konstrukti, ki zajamejo kakšno od zakonitosti v problemski domeni in s tem opisujejo koncept na višjem nivoju. Ali to uspeva algoritmu LFC, je težko reči; konstrukte bi moral oceniti strokovnjak z vsakega od področij. Potrdim lahko kvaliteto konstruktov za umetne domene, kjer je LFC pravilno identificiral pomembne zakonitosti (pri parnosti je našel posamezne člene, ki definirajo koncept). Podrobnejšo analizo zgrajenih konstruktov za šahovsko domeno K2RK najdemo v dodatku.

V nekaterih domenah skorajda ni razlik med natančnostjo klasifikacije s konstruktivno indukcijo in brez nje, in čeprav ima drevo s konstrukti manj listov, bi se z vidika razumljivosti morda vendarle rajši odločili za običajno drevo brez konstruktov.

4.8 Kompleksnost izračunov

Izračun odločitvenega drevesa s konstrukti je računsko bistveno bolj zahtevna naloga kot izračun navadnega drevesa.

Časovno najzahtevnejši del algoritma LFC je izgradnja konstruktov. Kompleksnost je tu odvisna od števila zgrajenih konstruktov. V najslabšem primeru moramo zgraditi:

$$\#konstruktov = maxGlobina \cdot maxŠirina \cdot AV \cdot W \cdot 4 \qquad (4.7)$$

kjer predstavlja $maxGlobina$ globino pogleda vnaprej, $maxŠirina$ širino prostora (beam), ki ga preiskujemo, AV skupno število vrednosti atributov, s katerimi poskušamo razširiti konstrukt, in W širino okna. Za upoštevanje tako konjunktivnih kot disjunktivnih razširitev moramo dodati faktor 2; faktor 2 pa dodamo tudi za upoštevanje zanikanih konstruktov.

V praksi se pokaže, da je kompleksnost vendarle manjša. Geometrijska omejitev nam pri lažjih problemih zmanjša problemski prostor tudi za 90 in več odstotkov, skoraj pri vseh pa za polovico. Pri večini problemov, lahko z oknom zmanjšamo število kandidatov za širitev do 70%. Za primerjavo naj povem, da traja izračun enega drevesa za povprečno domeno (npr. K2RK, 1000 primerov, 48 vrednostmi atributov, pri širini iskanja 15 in globini 3), približno pet minut na računalniku z operacijskim sistemom MS-DOS in procesorjem Intel 80486 (33MHz); izračun drevesa brez konstruktov traja približno minuto. Del potrebnega časa izračuna v primerjavi z algoritmom brez konstruktivne indukcije si LFC prihrani tudi z manjšim številom vozlišč.

5.

Sklep

Naloga potrjuje pomembnost konstruktivne indukcije v domenah, ki so opisane z med seboj močno odvisnimi atributi. Kjer so osnovni atributi premalo informativni, lahko s konstruktivno indukcijo zgradimo konstrukte, ki identificirajo pomembne zakonitosti in s tem pripomorejo k razumevanju na višjem nivoju in večji natančnosti klasifikacije.

Na podlagi geometrijske predstavitve kriterijske funkcije LFC dokaj uspešno omejuje problemski prostor konstruktov. To nam omogoči pogled vnaprej za bistveno manjšo ceno v primerjavi s pregledom vseh možnosti, obenem pa ne izgubimo pomembnih konstruktov.

LFC je namenjen učenju v težkih domenah, kjer so atributi med seboj močno odvisni. Medicinske domene, ki sem jih testiral, na žalost niso takšne, saj so atributi po prepričanju strokovnjakov precej neodvisni. Rezultati te trditve potrjujejo, saj konstruktivna indukcija ni prinesla statistično pomembnih izboljšav v klasifikacijski natančnosti. Izjema je domena LYMP, kjer je LFC dosegel 5% izboljšanje, analiza dreves pa je pokazala, da se pri vseh zgrajenih drevesih pojavljajo nekateri enaki konstrukti. Iz tega sklepam, da izražajo dejanske zakonitosti in niso le plod slučajnosti.

Najboljše rezultate je pokazal LFC v šahovski domeni K2RK, kjer tudi zgrajeni konstrukti pričajo o zajetju pomembnih zakonitosti. Rezultati in analiza konstruktov na tej domeni nas opozarjajo, da potrebujemo za izgradnjo dobrih konstruktov iz neinformativnih atributov zadosti primerov.

Pri domenah parnosti, se kljub pogledu naprej pokaže kratkovidnost kriterijske funkcije. Vsi atributi so navidezno enakovredni, vgrajene omejitve pa problemskega prostora ne zmanjšajo. Koristne konstrukte najdemo le zaradi iskanja v širino. S tem lahko razložimo razmeroma majhno natančnost klasifikacije na domeni P400, kjer tudi širina iskanja 100 ni zadoščala za izbor pomembnih konstruktov.

Opisan pristop k šumnim in nepopolnim podatkom se je v praksi izkazal kot ustrezen. Ocenjevanje verjetnosti z m-oceno povečuje zaupanje tudi v liste z majhnim številom primerov.

Za uspešno rezanje z ocenjevanjem natančnosti na neodvisnem testnem vzorcu, bi potrebovali domene z več primeri. To rezanje je kljub vsemu dalo ponekod kar dobre rezultate. Drevesa so bila precej manjša kot pri rezanju z m-oceno verjetnosti, natančnost klasifikacije približno enaka, zmanjšala pa se je informacijska vsebina.

Opisane mere težavnosti so se razkrile kot nezanesljive. Iz njih ne moremo sklepati na pravo težavnost koncepta, zanimive so le kot dodatna razlaga nekaterih rezultatov.

Kljub uspešnosti omejevanja problemskega prostora konstruktov se zdi, da je prav gradnja uspešnih konstruktov področje, kjer so možne največje izboljšave in spremembe.

Pri gradnji konstruktov v nekem vozlišču se rado zgodi, da so konstrukti, ki jih najdemo, zgolj zelo prilagojeni primerom in ne izražajo dejanskih zakonitosti koncepta. Temu bi se lahko izognili tako, da bi konstrukte gradili le v korenu drevesa ali njemu bližnjih vozliščih, v spodnjih nivojih drevesa pa bi poskušali uporabiti zgoraj zgrajene konstrukte.

Z beleženjem pogostosti pojavljanja konstruktov v drevesih, bi lahko morda odkrili tiste konstrukte, ki v resnici opisujejo zakonitosti domene. Program bi te konstrukte spremenil v nove atribute in s tem postopoma dvignil nivo predstavitve koncepta.

LFC ocenjuje pomembnost atributov z informativnostjo. Znane so slabosti te funkcije in v zadnjem času se pojavljajo predlogi, kako preseči njeno kratkovidnost. Obetaven se zdi algoritem RELIEF in njegova uporaba v kombinaciji s konstruktivno indukcijo, bi gotovo dala zanimive rezultate.

Zamenjava omejitve z oknom v LFC z ustrezno omejitvijo z RELIEF-om se zdi privlačna razširitev.

Nadaljnje delo bi bilo potrebno opraviti tudi pri identificiranju konstruktov, ki na višjem nivoju opisujejo iskani koncept. Predvsem v domenah, kjer je sedanja predstavitev problema z atributi nezadostna, bi lahko marsikaj storili. S pomočjo veščaka za takšno področje, ki bi ocenil pomen ponavljajočih se konstruktov, bi morda našli zakonitosti, skrite v interakcijah osnovnih atributov.

Zahvala

Mentor dr. Igor Kononenko mi je nudil vso potrebno pomoč ter se mi posvetil ob vsaki priložnosti.

Mnoge nejasnosti s področja strojnega učenja sva si v pogovorih razjasnila z Edijem Šimcem.

Člani Laboratorija za umetno inteligenco so z naklonjenostjo gledali na moje delo, Uroš Pompe pa mi je pomagal tudi pri tehničnih težavah.

Vsem se iskreno zahvaljujem.

A. Dodatek

Analiza konstruktov

V primerjavi z algoritmom brez konstruktivne indukcije, je dosegel LFC najboljše rezultate v domenah LYMP, K2RK in domenah parnosti.

LYMP je medicinski problem, ki opisuje limfografijo. Dobljene konstrukte bi zato moral oceniti zdravnik specialist. Zanimivo je, da je pri vseh desetih drevesih, ki smo jih zgradili, v korenu drevesa enak konstrukt:
(CHANGES IN NODE=lac. margin) $\vee$ *(NO. OF NODES IN=0-9)* $\vee$ *(SPECIAL FORMS=chalices)*.

Večkrat se ponovijo tudi nekateri drugi konstrukti v poddrevesih. To kaže, da konstrukti verjetno izražajo dejanske zakonitosti.

V domenah parnosti (P27Y, P37Y, P2AR, P3AR, P400) najdemo zaradi preiskovanja v širino konstrukte, ki ustrezajo konceptu parnosti, npr. $X_1\overline{X_2}$ za parnost drugega reda in $X_1X_2\overline{X_3}$ pri parnosti tretjega reda.

Zanimivi so konstrukti v šahovski domeni K2RK. Gre za problem ugotavljanja ilegalnih pozicij v končnici belega kralja in trdnjave proti črnemu kralju, kjer je na potezi beli. Problem je opisan v *(Muggleton in sod., 1989)* citirano v *(Džeroski, 1991)*.

Atributi opisujejo položaj figur na šahovnici. Po dva atributa opisujeta položaj ene figure in sicer eden vrstico, drugi pa stolpec, kjer se figura nahaja.

Skupaj imamo 6 atributov, vsak lahko zavzame 8 različnih vrednosti.

Na voljo je bilo 1000 primerov, 669 pozicij je bilo legalnih, 331 pa ilegalnih. Ločimo lahko tri vrste ilegalnih pozicij:

1. Bela trdnjava in črni kralj sta v isti vrsti ali stolpcu na šahovnici in črni kralj je v šahu. Če je na potezi beli, to ni legalna pozicija. Takšnih primerov je bilo 237.
2. Beli in črni kralj se nahajata na sosednjih poljih (74 primerov).
3. Dve figuri se nahajata na istem polju (20 primerov).

Drevesa, ki jih je gradil LFC, so bila povečini verige konstruktov. Konstrukti so si zaporedoma sledili, vsak je pokril nekaj primerov in jih klasificiral kot ilegalne. Od korena drevesa najbolj oddaljen list je vseboval preostale legalne primere.

Program je v povprečju za eno drevo zgradil 22 konstruktov (s standardnim odklonom 7.5), ki so opisovali naslednje zakonitosti:

- **bela trdnjava in črni kralj v isti vrsti:** 7.8 konstruktov (standardni odklon 0.6) npr.: *(vrsta bele trdnajve = 4)* $\wedge$ *(vrsta črnega kralja = 4)*
- **bela trdnjava in črni kralj v istem stolpcu:** 7.4 konstruktov (standardni odklon 0.8), npr.: *(stolpec bele trdnajve = 2)* $\wedge$ *(stolpec črnega kralja = 2)*
- **kralja sta na sosednjih poljih:** 3.3 konstrukte (standardni odklon 4.3). LFC ni zgradil popolnega opisa kraljev na sosednjih poljih, saj za kaj takega ni imel zadostne statistične podpore, eno ali dve koordinati je pustil nedoločeni, npr.: *(vrsta belega kralja = 5)* $\wedge$ *(vrsta črnega kralja = 5)* $\wedge$ *(stolpec črnega kralja = 6)*, ali pa (pogosteje): *(stolpec črnega kralja = 7)* $\wedge$ *(stolpec črnega kralja = 7)*

Konstruktov, ki bi opisovali primere, ko se dve figuri nahajata na istem polju, LFC ni zmogel zgraditi. To ni presenetljivo, saj takšne primere opisuje mnogo premalo primerov.

Povprečno 3.5 konstruktov (standardni odklon 3.6) ni opisovalo nobenih zakonitosti v domeni, ampak so bili le prilagojeni primerom v vozlišču.

Vidimo, da je LFC s konstrukti dejansko pokril skoraj vse primere, kjer sta črni kralj in bela trdnjava v isti vrsti ali stolpcu. Bližino kraljev je sicer

skušal opisati, vendar mu je to uspelo le deloma. To ne preseneča, saj vemo, da je v primerjavi z razsežnostjo problemskega prostora na voljo razmeroma malo primerov s to značilnostjo. Isto velja tudi za primere, ki opisujejo dve figuri na istem mestu.

Natančnost klasifikacije na domeni K2RK je 88%. Na domeni K1RK, ki opisuje iste primere z bolj informativnimi atributi (primerjava pozicij figur z operatorji $\{>, <, =\}$), dosežemo natančnost klasifikacije 99% že z algoritmom brez konstruktivne indukcije. LFC na domeni K2RK deloma nadomesti neinformativnost atributov s konstrukti in tako opiše tiste zakonitosti, ki so zadosti statistično podprte.

B. Dodatek

Navodila za uporabo programa

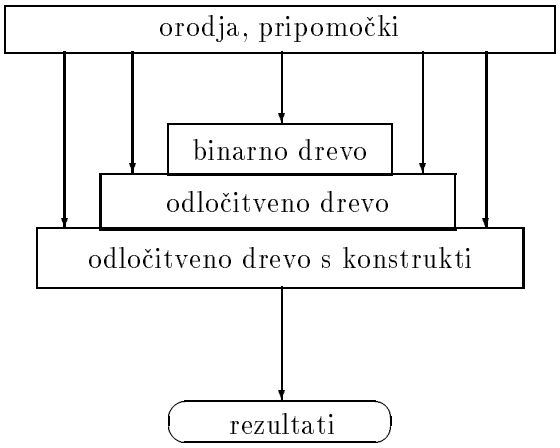
V tem dodatku navajam kratek opis programa in njegove uporabe.

B.1 Opis logične strukture programa

Program je napisan v jeziku C++ in deluje pod operacijskima sistemoma MS-DOS in OS/2. Pod MS-DOS-om sem uporabljal prevajalnik Borland C++ 3.1, pod OS/2 pa C Set ++ 2.0. Program vsebuje približno 6000 vrstic izvorne kode.

Logično je program organiziran v tristopenjsko hierarhijo, ki jo prikazuje slika B.1. Na prvem nivoju je binarno drevo in funkcije v zvezi z njim (dodajanje, izpis, brisanje), drugi nivo predstavlja klasično binarno odločitveno drevo in njegove funkcije (branje podatkov z datotek, rezanje drevesa, izpisi poročil), na tretjem nivoju pa se nahaja odločitveno drevo s konstrukti (gradnja drevesa in konstruktov, določitev parametrov, izpis konstruktov).

Povezanost modulov v projektu sledi logični zgradbi programa. Dodani so še moduli za javljanje napak, uporabniški vmesnik in modul, ki združuje različne pripomočke (sortiranja, izračun povprečja in standardnega odklona, dvodimenzionalna tabela, ...). Vse module povezuje glavni modul, ki vodi interakcijo z uporabnikom in kliče podrejene module.



Slika B.1: Logična struktura programa

B.2 Uporabniški vmesnik

Uporabniški vmesnik predstavla trd oreh s stališča prenosljivosti programa, kajti standardi so še nedorečeni. Ker sem želel napraviti program prenosljiv med različnimi sistemi, sem vzel njihov skupni imenovalec. Uporabniški vmesnik je zato precej špartanski.

B.2.1 Zagon programa

Program poženemo iz ukazne vrstice z ukazom:

```
lfc [datoteka_s_parametri [daj]]
```

Če poženemo program brez parametrov, se pojavi interaktivni vmesnik. V primeru, da navedemo ime datoteke z nastavitvami, se bodo parametri programa avtomatsko nastavili na vrednosti v datoteki. Če dodamo še en poljuben parameter, bo program začel testiranje celotne domene s parametri iz datoteke. Ta način je ugoden za testiranje več domen hkrati s paketno obdelavo.

Opišimo sedaj možnosti, ki nam jih ponuja program.

B.2.2 Osnovni izbirni zaslon

Ko poženemo program brez parametrov, se pojavi izbirni zaslon z več možnostmi. Ko se odločimo, kaj bomo počeli, vtipkamo številko, ki se nahaja pred našim izborom. Opišimo možnosti:

1. **Nalaganje podatkov o domeni:** V pomnilnik se naložijo podatki o domeni, ki jo nameravamo testirati. Program nas opozori na morebitne napake v opisu domene ali če domena še ni določena.
2. **Posamezno drevo:** Začnemo graditi drevo s trenutnimi parametri. Za gradnjo vzamemo tisto razbitje podatkov, ki ga predpisuje datoteka z izbranim indeksom (0-9). Ko je drevo zgrajeno, ugotovimo natančnost klasifikacije z njim, informacijsko vsebino klasifikacije in število listov. Drevo obrežemo s predpisano metodo in ponovno opravimo meritve. Rezultate izpišemo na zaslon in na datoteko. Ime datoteke je določeno z imenom domene in indeksom razbitja (npr. `LYMP.3`). Datoteka z rezultati se zapiše na direktorij z rezultati.
3. **Testiranje celotne domene:** Za dano domeno zgradimo in testiramo 10 dreves (isti postopek kot pri posameznem drevesu), vsako na svojem razbitju podatkov. Na koncu izračunamo povprečja in standardne odklone. Vse rezultate zapišemo na direktorij z rezultati v datoteko z imenom domene in končnico `out`, npr.: `LYMP.OUT`
4. **Podatki o domeni:** Na zaslon in na datoteko s končnico `.rep` izpišemo značilne podatke za izbrano domeno: število primerov, število atributov, število vrednosti atributov, število razredov, delež večinskega razreda in mere težavnosti (povprečna pogojna entropija z dvema razredoma, povprečna pogojna entropija s poljubnim številom razredov in povprečna medsebojna informacija). Program nas vpraša, koliko najboljših atributov naj upošteva pri izračunu težavnosti. Če želimo izračun za vse atribute, lahko vnesemo tudi 0.
5. **Določitev parametrov programa:** Pojavi se nov izbirni zaslon, kjer vnašamo željene nastavitve za program. Opisane so v naslednjem razdelku.
6. **Nalaganje parametrov:** Če večkrat testiramo domene s podobnimi parametri in se želimo izogniti vsakokratnemu določanju parametrov, si lahko trenutne parametre shranimo na datoteko in jih po potrebi naložimo. Potrebno je vnesti ime datoteke.

7. **Shranjevanje parametrov:** Trenutne nastavitve programa shranimo na datoteko, katere ime vtipkamo.
8. **Izhod iz programa:** Program sprosti zasedeni pomnilnik in se ustavi.

B.2.3 Nastavitev parametrov

V glavnem meniju izberemo opcijo *določitev parametrov programa* in pojavi se izbirni zaslon z vsemi nastavitvami programa. Vsaka izbira ima v oklepaju trenutno vrednost parametra, ki ga predstavlja. Vrednosti spreminjamo tako, da vnesemo številko pred izbiro. Pojavi se zahteva po vnosu nove vrednosti. Vtipkamo željeno vrednost, ki jo program preveri. Če je nepravilna, bo ostala v veljavi stara vrednost, sicer pa bomo v oklepaju pri izbiri opazili novo vrednost. Spreminjamo lahko naslednje parametre:

1. **Ime domene:** Vnesemo ime domene (4 črke), s katero želimo delati, npr. `LYMP`. Imena ostalih datotek (opis podatkov, podatki, razbitje, rezultati) bo program generiral na podlagi tega imena.
2. **Indeks razbitja:** Razbitja podatkov na učne in testne primere se nahajajo na datotekah z indeksi od 0 do 9. Če želimo delati s posameznimi drevesi, vnesemo željeni indeks. V primeru, da bomo testirali celotno domeno, nam indeksa ni potrebno vnašati.
3. **Direktorij s podatki:** Določimo direktorij, na katerem se nahajajo podatki z opisi domen in razbitji, npr.: `D:\DATA`. Podamo lahko celotno pot do podatkov, mogoče pa je vnesti tudi pot glede na trenutni direktorij.
4. **Direktorij z rezultati:** Podobno kot za podatke, lahko tudi za izpise rezultatov določimo poseben direktorij. Če želimo izpis na trenuten direktorij, vnesemo prazen niz.
5. **Splošna oblika konstruktov:** Splošna oblika konstruktov pomeni, da bomo pri gradnji konstruktov poskušali konstrukt razširiti tako s konjunkcijo kot z disjunkcijo. Če se odločimo za splošno obliko konstruktov, so nastavitve širine iskanja disjunktov in maksimalnega števila disjunktov nepomembne, sicer pa na način iskanja vplivata tudi opciji za disjunkcijo.

6. **Širina iskanja:** S širino iskanja določimo, koliko konstruktov bomo poskušali razširiti na neki globini iskanja. Če ne želimo iskanja v širino, postavimo to vrednost na 1 (0 je prepovedana).
7. **Globina pogleda vnaprej:** Omejimo globino pogleda vnaprej, oziroma število členov v konstruktu. Vrednost 0 pomeni, da ni pogleda vnaprej, dobimo torej običajna odločitvena drevesa brez konstruktov.
8. **Širina okna:** V okno okoli konstrukta izberemo tiste vrednosti atributov, ki so mu v geometrijski predstavitvi najbližje. Širina okna (vrednost med 0 in 1) pomeni delež vrednosti atributov, ki so kandidati za razširitev. Širina okna 1 pomeni, da bomo poskušali konstrukt razširiti z vsemi možnimi vrednostmi atributov, vrednost 0.5 pa pomeni, da bomo širitev poskušali s polovico najboljših (najbližjih).
9. **Širina iskanja disjunktov:** Če smo se odločili, da ne bomo gradili konstruktov v splošni obliki, potem gradimo najprej samo konjunktivne konstrukte, nato pa najboljše zgrajene konjunkte povezujemo še disjunktivno. V tem primeru opcija *širina iskanja* določa širino iskanja konjunktov, opcija *širina iskanja disjunktov* pa širino iskanja disjunktov. Opciji *globina pogleda naprej* oz. *maksimalno število disjunktov* določata globini iskanja naprej pri izračunu konjunktov oz. disjunktov.

   Če želimo graditi samo konjunktivne konstrukte, izklopimo splošno obliko konstruktov in postavimo širino iskanja disjunktov na 0 (opcija *maksimalno število disjunktov* je zdaj nepomembna). Širino in globino iskanja reguliramo z opcijama *širina iskanja* in *globina pogleda naprej*.

   V primeru, da želimo graditi samo disjunktivne konstrukte, izklopimo splošno obliko konstruktov, postavimo širino iskanja na 1 in globino pogleda naprej na 0. Širino in globino iskanja reguliramo z opcijama *širina iskanja disjunktov* in *maksimalno število disjunktov*.
10. **Ustavitveni pogoji:** Določimo lahko več pogojev, kdaj naj se gradnja drevesa ustavi. Gradnja se ustavi, če je izpolnjen katerikoli izmed pogojev. Če hočemo nek pogoj izključiti, ga moramo postaviti na vrednost, kjer je neučinkovit. Na voljo imamo:
    - **Minimalna teža vozlišča:** Določimo, koliko sme biti najmanjša teža primerov v vozlišču, da nadaljujemo z delitvijo (če v domeni ni manjkajočih vrednosti, je teža primerov enaka številu primerov).

Tipične vrednosti parametra so med 1 in 5. Parameter onesposobimo, če ga postavimo na 0.

- **Minimalen delež primerov v vozlišču:** Podobno kot zgoraj, le da ne gre za dejansko število primerov, ampak za odstotek vseh primerov.
- **Delež večinskega razreda v vozlišču:** Gradnjo ustavimo, če delež večinskega razreda v nekem vozlišču preseže odstotek, ki ga določa parameter. Ustavitveni pogoj onesposobimo, če ga postavimo na 100%.
- **Prikladnost atributa:** *(Cestnik in sod., 1987)* Je hevristično merilo, ki ocenjuje kvaliteto delitve, ki smo jo izbrali. Tipična učinkovita vrednost tega parametra je 3; parameter onesposobimo, če ga postavimo na 0.
- **Minimalna teža lista:** Določimo, koliko najmanj mora znašati teža (število) primerov v listu. Smiselno je postaviti to vrednost na 1 ali več. Opcijo onesposobimo, če jo postavimo na 0.

11. **Minimalna verjetnost primera:** Če v domeni nastopajo primeri z manjkajočimi vrednostmi atributov, jih obravnavamo verjetnostno. Ko delimo primere v levo in desno vejo glede na nek konstrukt, se zgodi, da pripadnosti nekega primera ne moremo določiti zaradi manjkajočih vrednosti atributov. Tak primer razdelimo v obe veji z verjetnostma, ki smo ju ocenili z m-oceno verjetnosti. Pripeti se, da so v nekaterih vejah primeri z verjetnostjo blizu 0. Takšne primere lahko zanemarimo, saj nam samo otežujejo računanje. Običajno postavljamo parameter na vrednost 0.01. Parameter izključimo, če ga postavimo na 0.
12. **Binarizacija atributov:** Atribute z več vrednostmi je mogoče pred uporabo binarizirati. Vrednosti atributa razdelimo v dve množici. Če je pri nekem primeru vrednost atributa element prve množice, mu pripišemo pozitivno vrednost tega atributa sicer negativno. Dani atribut lahko tako uporabljamo, kot da bi imel samo dve vrednosti. Binarizacija je koristna, če ne uporabljamo konstruktivne indukcije (širino iskanja postavimo na 1, globino pa na 0), vključena konstruktivna indukcija pa že sama bolje opravlja to nalogo in je binarizacija večkrat celo škodljiva.
13. **Parameter m-ocene verjetnosti za konstrukcijo:** Običajno dosegamo najboljše rezultate s postavitvijo parametra $m = 2$. Na ocenjevanje

z relativno frekvenco preidemo, če postavimo $m = 0$.

14. **Parameter m-ocene verjetnosti za rezanje:** Velja enako kot zgoraj.
15. **Niblett-Bratkovo rezanje z m-oceno verjetnosti:** Ta način rezanja lahko vključimo ali izključimo. Obseg rezanja je odvisen od parametra m-ocene verjetnosti za rezanje.
16. **Rezanje z ocenjevanjem natančnosti na testnem vzorcu:** Rezanje vključimo ali izključimo. Za smiselne rezultate potrebujemo zadosti velik testni vzorec. Velikost testnega vzorca določimo s parametrom *delež učnih primerov za ocenjevanje napak.*
17. **Rezanje konstruktov:** Tudi za rezanje konstruktov potrebujemo zadosti velik testni vzorec, ki ga določimo s naslednjim parametrom.
18. **Delež učnih primerov za ocenjevanje napak:** Za ocenitev napake, ki jo napravi drevo ali konstrukt, potrebujemo zadosti velik testni vzorec. Primeri v testni množici za ocenjevanje napak morajo biti neodvisni od učnih primerov. Tipično žrtvujemo 10 do 15% učnih primerov. Če napake ne želimo ocenjevati s testnim vzorcem, postavimo vrednost na 0.
19. **Minimalna teža testnega vzorca za ocenjevanje napak:** Da bi realno ocenili napako, ki jo napravimo z rezanjem drevesa ali konstrukta v nekem vozlišču, potrebujemo v tem vozlišču zadosti primerov, s katerimi bomo ocenjevali napako. Če jih nimamo na voljo, je ocena nezanesljiva in bolje je, da ne režemo. Primerna vrednost tega parametra je 4, parameter onemogočimo s postavitvijo na 0.

Nastavitve lahko spreminjamo interaktivno ali pa tako, da naložimo datoteko s parametri. Poglejmo si drugo možnost.

B.3 Datoteka s parametri

Datoteka s parametri vsebuje vse nastavitve, ki smo jih opisali v prejšnjem razdelku in to v istem vrstnem redu. Vsaka nastavitev se nahaja v svoji vrstici. Do znaka = se nahaja opis parametra, ki ga pri branju preskočimo, sledi vrednost parametra.

Če želimo uravnavati parametre z datoteko, je najbolje, če zapišemo trenutno nastavitev parametrov na datoteko in si tako ustvarimo vzorec, ki ga lahko kasneje spreminjamo z vsakim urejevalnikom teksta.

Predstavimo primer datoteke s parametri:

Direktorij s podatki =podatki
Direktorij z rezultati =c:\drevesa
Indeks razbitja =0
Ime domene (4 znake) =lymp
Splošna oblika konstruktov (y/n) ? =y
Širina iskanja =10
Globina pogleda vnaprej =3
Širina okna =0.3
Širina iskanja disjunktov =0
Največje število disjunktov v konstruktu =0
Minimalna teža vozlišča, da ga še delimo =5.0
Minimalni delež primerov v vozlišču, da ga še delimo =0.04
Delež večinskega razreda, da ustavimo gradnjo drevesa =0.95
Prikladnost atributa =3
Minimalna teža lista =0.99
Minimalna verjetnost primera, da ga še upoštevamo =0.01
Binarizacija atributov (y/n) ? =n
Parameter m-ocene za gradnjo =0.0
Parameter m-ocene za rezanje =2
Rezanje z Niblett-Bratkovo metodo z m-oceno (y/n) ? =y
Rezanje drevesa z oceno napake na testnem vzorcu (y/n) ? =n
Rezanje konstruktov (y/n) ? =n
Delež učnih primerov za ocenjevanje napak =0.00
Najmanjša teža učnih primerov za ocenjevanje napak =3.0

Literatura

[1] I. Bratko: *Prolog Programming for Artificial Intelligence, second edition.* Addison-Wesley, Reading, Massachusetts, 1990

[2] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone: *Classification and regression trees.* Wadsworth Inc., Belmont, California, 1984

[3] B. Cestnik, I. Kononenko, I. Bratko: *ASSISTANT 86: A Knowledge-Elicitation Tool for Sophisticated Users.* Published in I. Bratko, N. Lavrač (editors): *Progress in Machine Learning, Proceedings of EWSL 87.* Sigma Press, Wilmslow, 1987

[4] B. Cestnik: *Ocenjevanje verjetnosti v avtomatskem učenju.* Doktorska disertacija. Univerza v Ljubljani, Fakulteta za elektrotehniko in računalništvo, Ljubljana, 1991

[5] S. Džeroski: *Induktivno učenje relacij iz nezanesljivih podatkov.* Magistersko delo. Univerza v Ljubljani, Fakulteta za elektrotehniko in računalništvo, Ljubljana, 1991

[6] M. Gams, N. Lavrač: *Review of Five Empirical Learning Systems Within a Proposed Schemata.* Published in I. Bratko, N. Lavrač (editors): *Progress in Machine Learning, Proceedings of EWSL 87.* Sigma Press, Wilmslow, 1987

[7] L. Gyergyék: *Teorija o informacijah.* Fakulteta za elektrotehniko, Ljubljana, 1988

[8] E.B. Hunt, J.Martin, P.J. Stone: *Expiriments in induction.* Academic Press, New York, 1966

[9] R. Jamnik: *Matematična statistika.* Državna založba Slovenije, Ljubljana, 1980

[10] K. Kira, L. Rendell: *A practical approach to feature selection.* In *Proceedings of the Machine Learning Conference.* Aberdeen, 1992, pp. 250-256

[11] I. Kononenko: *Combining decisions of multiple rules.* Published in V. Sgorev, B. du Boulay (editors): *AIMSA'92.* Elsevier Science Publications, 1992

[12] I. Kononenko: *Estimating attributes: analysis and extensions of RELIEF.* Predloženo na European Conference on Machine Learning, 1994

[13] I. Kononenko, I. Bratko: *Information based evaluation criterion for classifier's performance.* Machine Learning, Vol. 6, No. 1, pp. 67-80, 1991

[14] S.H. Muggleton, M. Bain, J. Hayes-Michie, D. Michie: *An experimental comparison of human and machine learning formalisms.* Sixth International Workshop on Machine Learning. Ithaca, New York, Mogran Kaufmann, 1989

[15] J.R. Quinlan: *Discovering rules by induction from large collection of examples.* In D.Michie (editor): *Expert Systems in Microelectronic Age.* Edinburgh University Press, Edinburgh, 1979

[16] J.R. Quinlan: *Induction of decision trees.* Machine Learning, Vol. 1, No. 1, pp. 81-106, 1986

[17] *Računalniški slovarček, tretja izdaja.* Cankarjeva založba, Ljubljana, 1993

[18] H. Ragavan, L. Rendell: *Lookahead Feature Construction for Learning Hard Concepts.* Proceedings of the Tenth International Machine Learning Conference, 1993, pp. 252-259

[19] H. Ragavan, L. Rendell: *Improving the Design of Induction Methods by Analyzing Algorithm Functionality and Data-Based Concept Complexity.* Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Chambery, France, 1993, pp. 952-958

[20] H. Ragavan, L. Rendell, M. Shaw, A. Tessmer: *Learning Complex Real-World Concepts through Feature Construction.* Technical Report. The Beckman Institute, University of Illinois, Urbana, 1993

[21] H. Ragavan, L. Rendell, M. Shaw, A. Tessmer: *Complex Concept Acquisition through Directed Search and Feature Caching.* Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Chambery, France, 1993, pp. 946-951

[22] P. Smyth, R. M. Goodman: *An Information Theoretic Approach to Rule Induction from Databases.* IEEE Transactions on Knowledge and Data Engineering, April 1990

[23] B. Stroustrup: *The C++ Programming Language, second edition.* Addison-Wesley, Reading, Massachusetts, 1991

[24] C.J. Thornton: *Techniques in Computational Learning, an Introduction.* Chapman & Hall, London, 1992

[25] P. H. Winston: *Artificial Intelligence, third edition.* Addisson-Wesley, Reading, Massachusetts, 1992

[26] D. S. Yang, L. Rendell, G. Blix: *A Scheme for Feature Construction and a Comparison of Empirical Methods.* Proceedings of the Twelft International Joint Conference on Artificial Intelligence, Sydney, Australia, 1991, pp. 699-704

Izjavljam, da sem diplomsko delo samostojno izdelal pod vodstvom mentorja dr. Igorja Kononenka. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.