UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Andrej Oblak

Kombiniranje sistemov HYPER in Alchemy za učenje iz šumnih podatkov

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Ivan Bratko

Ljubljana, junij 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01557/2009
Datum: 05.04.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ANDREJ OBLAK**

Naslov: **KOMBINACIJA SISTEMOV HYPER IN ALCHEMY ZA UČENJE IZ ŠUMNIH PODATKOV**

COMBINING SYSTEMS HYPER AND ALCHEMY FOR LEARNING FROM NOISY DATA

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Induktivno logično programiranje (ILP) je definirano kot strojno učenje v logiki. ILP program HYPER, ki izhaja iz te logične definicije, zato ni primeren za učenje iz šumnih podatkov. Naloga je spremeniti ILP program HYPER tako, da bo primeren tudi za učenje iz šumnih podatkov. Tako izboljšani HYPER naj bo povezan s sistemom za statistično relacijsko učenje Alchemy, tako da bo Alchemy določal verjetnostne ocene hipotez, ki jih generira HYPER. Razviti sistem naj bo preizkušen na zašumljenih standardnih učnih domenah za ILP ter na realnih vremenskih podatkih.

Mentor:

akad. prof. dr. Ivan Bratko

Dekan:

prof. dr. Franc Solina

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Andrej Oblak,

z vpisno številko 63040114,

sem avtor diplomskega dela z naslovom:

Kombiniranje sistemov HYPER in Alchemy za učenje iz šumnih podatkov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  prof. dr. Ivana Bratka
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 30.6.2009 Podpis avtorja:

Zahvala

Najprej bi se zahvalil prof. dr. Ivanu Bratku za mentorstvo pri diplomi. Prav tako bi se zahvalil Martinu Možini in ostalim zaposlenim na XMEDIA projektu za vse nasvete in uporabne ideje, ki so bile v veliko pomoč pri izdelavi te diplomske naloge.

Posebna zahvala gre tudi Luki Šepetavcu za pomoč in nasvete pri pisanju v LaTeX-u ter Mojci Žefran za lektoriranje angleškega povzetka.

Kazalo

Seznam uporabljenih kratic in simbolov

ILP – induktivno logično programiranje

BK – predznanje (background knowledge)

MLN – markovske logične mreže (Markov logic networks)

WPLL – utežena psevdo-log-verjetnost (weighted pseudo-log-likelihood)

Povzetek

To diplomsko delo se nanaša na področje strojnega učenja, bolj specifično na uporabo induktivnega logičnega programiranja (ILP) za učenje hipotez iz šumnih podatkov. V ta namen smo uporabili ILP program HYPER, kateri pa ni sposoben učenja iz šumnih podatkov, zato smo ga morali najprej ustrezno predelati, predelano verzijo pa smo poimenovali HYPER/N. Ker pa se HYPER/N lahko nauči tudi hipoteze, ki predstavljajo neke naključne odvisnosti v učnih podatkih, smo vse naučene hipoteze ocenili še s sistemom za statistično relacijsko učenje Alchemy, ter obdržali le hipoteze, ki so dobile najvišje ocene. Delovanje naveze HYPER/N - Alchemy smo nato testirali na sintetičnih učnih podatkih (standardni ILP učni problemi in vremenska domena) z dodanim šumom in na realnih vremenskih podatkih. Z dobljenimi rezultati smo bili zadovoljni, kljub temu, da smo pri tem odkrili nekaj pomanjkljivosti, katere moramo v prihodnje še odpraviti.

V tem delu najprej podajamo osnove induktivnega logičnega programiranja z opisom ILP programa HYPER. Nato sledijo osnove markovskih logičnih mrež, katere so potrebne za razumevanje delovanja Alchemyja, katerega opis tudi podajamo, vključno z rezultati testiranj učenja hipotez z Alchemyjem na standardnih ILP učnih problemih. V zadnjem poglavju pa opisujemo program HYPER/N, povezavo le-tega z učenjem uteži v Alchemyju in rezultate učenja na sintetičnih učnih domenah z dodanim šumom ter na realnih vremenskih podatkih.

Ključne besede:

induktivno logično programiranje, šum, ocenjevanje hipotez, vremenski podatki

Abstract

This thesis refers to the field of machine learning. It concentrates on the use of inductive logic programming (ILP) for learning hypotheses from noisy data. We used ILP program HYPER, which is not actually able to learn from noisy data, so for this purpose we had to modify it accordingly. We named this modified version HYPER/N. But since HYPER/N also learns hypotheses, which are a result of some random dependencies in learning data, we had to evaluate learned hypotheses with Alchemy - a system for statistical relational learning. We kept only the hypotheses with highest estimates and discarded all the remaining ones. We tested learning with HYPER/N and Alchemy on synthetic data (standard ILP learning problems and weather domain) with added noise and on real weather data. We were satisfied with the results despite some problems, which we have to solve in the future.

This work is organized as follows. First we describe the basics of inductive logic programming and ILP program HYPER. Then we describe the basics of Markov logic networks, which are important for the understanding of Alchemy, the description of which follows next. We also describe results of learning with Alchemy on standard ILP learning problems. In the last chapter, we describe HYPER/N, its connection with Alchemy's weight learning and results of learning on synthetic data with added noise and also on real weather data.

Key words:

inductive logic programming, noise, hypothesis evaluation, weather data

Poglavje 1

Uvod

Metode strojnega učenja običajno uporabljamo za učenje pravil, ki nam pomagajo pri razumevanju delovanja nekega (realnega) sistema, ki je v večini primerov za nas črna škatla. O njemu vemo samo to, kakšne podatke smo mu podali na vhod in podatke, ki so bili pri danem vhodu rezultat delovanja opazovanega sistema. Obojim v kontekstu strojnega učenja pravimo učni podatki. Na podlagi teh učnih podatkov se je algoritem strojnega učenja sposoben naučiti pravil, ki veljajo za pretvarjanje vhodnih podatkov v izhodne, posledično lastnosti opazovanega sistema. Želimo si seveda, da so ti podatki čisti, saj v njih ni dvoma o odvisnostih in pravilih, ki veljajo za njih. Vendar pri opazovanju realnih sistemov le redko dobimo čiste učne podatke; podatki so običajno šumni. To pomeni da v njih obstajajo neke dvoumnosti in nekonsistence, katere učnemu algoritmu bistveno otežijo učenje pravil in odvisnosti za take sisteme. Učni algoritem mora biti zato robusten za take nepravilnosti v podatkih.

Včasih pa imamo o opazovanem sistemu že določeno predznanje; sistem torej ni več črna, temveč siva škatla. V učnih podatkih lahko veljajo določene relacije med podatki, delovanje sistema pa je lahko rekurzivno. Če je naš učni algoritem sposoben sprejemati in pri učenju upoštevati to predznanje in relacije v podatkih, sposoben pa je tudi učenja rekurzije, so lahko rezultati učenja bistveno boljši. Pristop k strojnemu učenju, ki omogoča tako učenje, se imenuje induktivno logično programiranje. Učnih algoritmov, ki implementirajo induktivno logično programiranje, poznamo več (PROGOL, Foil, Claudien, ...) mi pa smo se osredotočili na HYPER [1]. HYPER omogoča vse opisane prednosti (podajanje predznanja, relacijsko učenje, učenje rekurzije), njegova največja hiba pa je ta, da ne obvlada učenja na šumnih podatkih.

V tem diplomskem delu smo se zato osredotočili na izboljšavo HYPER-

ja, da se bo lahko učil tudi iz šumnih podatkov. Ker pa se zaradi načina obvladovanja šuma v učnih podatkih nauči tudi določene naključne odvisnosti, ki veljajo v podatkih, smo potrebovali še nek algoritem za ocenjevanje naučenih hipotez. Tega pa smo našli v algoritmu za učenje uteži hipotez v programskem paketu za statistično relacijsko učenje in verjetnostno logično sklepanje Alchemy. Učenje z navezo HYPER - Alchemy smo nato preizkusili in ocenili na nekaterih standardnih ILP učnih problemih ter na realnih vremenskih podatkih, kjer smo se hoteli naučiti hipotezo za napovedovanje gibanja padavin.

Poglavje 2

Induktivno logično programiranje in HYPER

To poglavje opisuje principe induktivnega logičnega programiranja, predstavljen pa je tudi ILP program HYPER. V celoti je povzeto po [1].

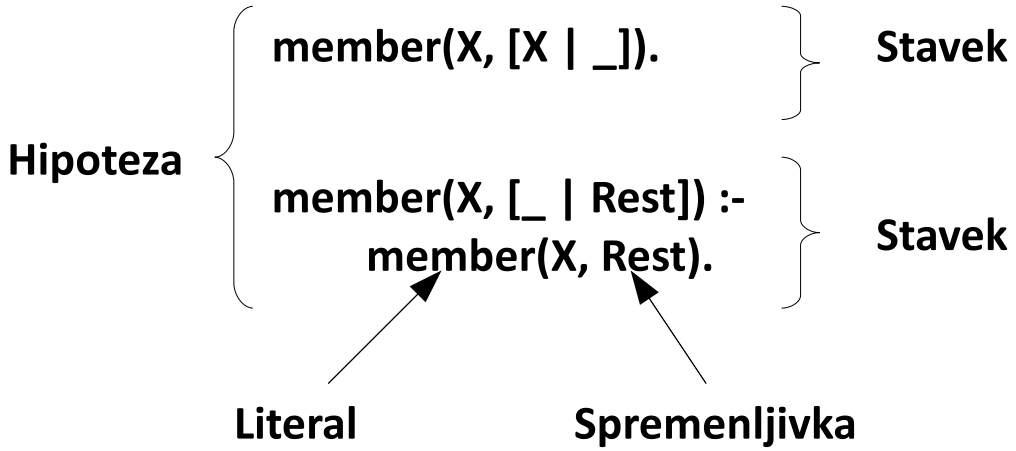
2.1 Induktivno logično programiranje

Induktivno logično programiranje (v nadaljevanju ILP) je pristop k strojnemu učenju, kjer se učimo relacije iz podanih učnih primerov. Naučenim relacijam pravimo hipoteze, le-te pa so zapisane v logiki prvega reda. Najpogosteje so hipoteze podane kar kot Prologov program.

Hipoteza je definirana kot množica enega ali več konjuktivno povezanih stavkov. Posamezen stavek v hipotezi je definiran kot Hornov stavek (disjunkcija literalov). Literali pa lahko vsebujejo 0 ali več spremenljivk. Literali opisujejo lastnosti spremenljivk ali relacije med spremenljivkami. Za lažjo predstavo je na sliki 2.1 prikazan preprost Prologov program, ki preverja, ali se element `X` nahaja v seznamu. Ta program bi zlahka predstavljal hipotezo, katero se je, ob ustrezno podanih učnih primerih, naučil nek ILP program.

Pri ILP-ju uporabnik ne piše programov direktno, temveč poda učne primere, ki predpisujejo kaj naj bi želeni program delal. Pozitivni učni primeri predpisujejo kaj želeni program mora delati, negativni učni primeri pa česa ne sme delati. Poleg učnih primerov uporabnik lahko poda tudi predznanje o učnem problemu, katerega ILP program lahko uporabi pri konstruiranju želenega programa.

Poglejmo si preprost primer avtomatsko konstruiranega programa (hipoteze) v Prologu. Predpostavimo, da imamo podane literale `parent(X,Y)`, `male(X)`



Slika 2.1: Struktura hipoteze

in `female(X)`, kateri definirajo neke družinske relacije, kot je to razvidno iz slike 2.2. Pozitivni učni primeri so podani z literalom `ex(has_daughter(X))`, negativni pa z literalom `nex(has_daughter(X))`. Učni problem je poiskati tako definicijo literala `has_daughter(X)` s pomočjo literalov `parent(X,Y)`, `male(X)` in `female(X)`, da velja za vse podane pozitivne učne primere in za nobenega negativnega.

ILP program lahko najde sledečo hipotezo:

```
has_daughter(X) :-
  parent(X,Y),
  female(Y).
```

Ta hipoteza lahko pojasni vse podane učne primere. Pri njej lahko opazimo tipično lastnost relacijskega učenja v primerjavi z atributnim. Namreč lastnost `has_daughter` spremenljivke X ni definirana le z atributi X-a, temveč je odvisna tudi od lastnosti spremenljivke Y, ki je v relaciji z X.

ILP problem lahko formuliramo tudi bolj formalno.

Podano imamo:

- Množico pozitivnih učnih primerov E_+ in množico negativnih učnih primerov E_-
- Predznanje BK (background knowledge), podano kot množica logičnih formul, iz katerih ne sme biti možno izpeljati pozitivnih učnih primerov E_+.

```
parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).
parent(pat,eve).

female(pam).
female(liz).
female(ann).
female(pat).
female(eve).

male(tom).
male(bob).
male(jim).

ex(has_daughter(tom)).
ex(has_daughter(bob)).
ex(has_daughter(pat)).

nex(has_daughter(pam)).
nex(has_daughter(jim)).
```

Slika 2.2: Učna domena za učenje družinskih relacij.

Poiskati moramo hipotezo H, predstavljeno z množico logičnih formul, za katere velja:

- Vse pozitivne učne primere E_+ je možno izpeljati s pomočjo BK in H.
- Nobenega negativnega učnega primera E_- ni možno izpeljati iz BK in H.

Rečemo lahko tudi, da mora hipoteza H skupaj s predznanjem BK pokriti vse pozitivne učne primere in nobenega negativnega. Taka hipoteza je popolna (pokriva vse pozitivne učne primere) in konsistentna (ne pokriva nobenega negativnega učnega primera).

Prednosti ILP-ja v primerjavi z atributnim učenjem so v jeziku hipotez, s katerim lahko opišemo relacije med posameznimi spremenljivkami, dovoljuje pa tudi rekurzijo, kar običajno ni možno pri ostalih pristopih k strojnemu učenju. Poleg tega lahko uporabnik poda ILP programu predznanje. To predznanje je lahko praktično katerikoli Prologov program, kar uporabniku omogoča podajanje specifičnega predznanja o učni domeni v zelo naravni obliki. S tem predznanjem lahko uporabnik zelo dobro predstavi učni problem in v učni proces vpelje omejitve, značilne za učno domeno. Pri atributnem učenju pa je za spremembo od ILP-ja podajanje predznanja zelo omejeno. Vendar vse te prednosti ILP-ja imajo tudi svojo ceno. Povzročajo namreč visoko kombinatorično kompleksnost, zato je atributno učenje (npr. odločitvena drevesa) mnogo bolj učinkovito kot ILP in kot tako v večini učnih primerov, kjer nam predstavitev problema z atributi zadošča, bolj priporočljivo.

2.2 Konstruiranje hipotez iz primerov

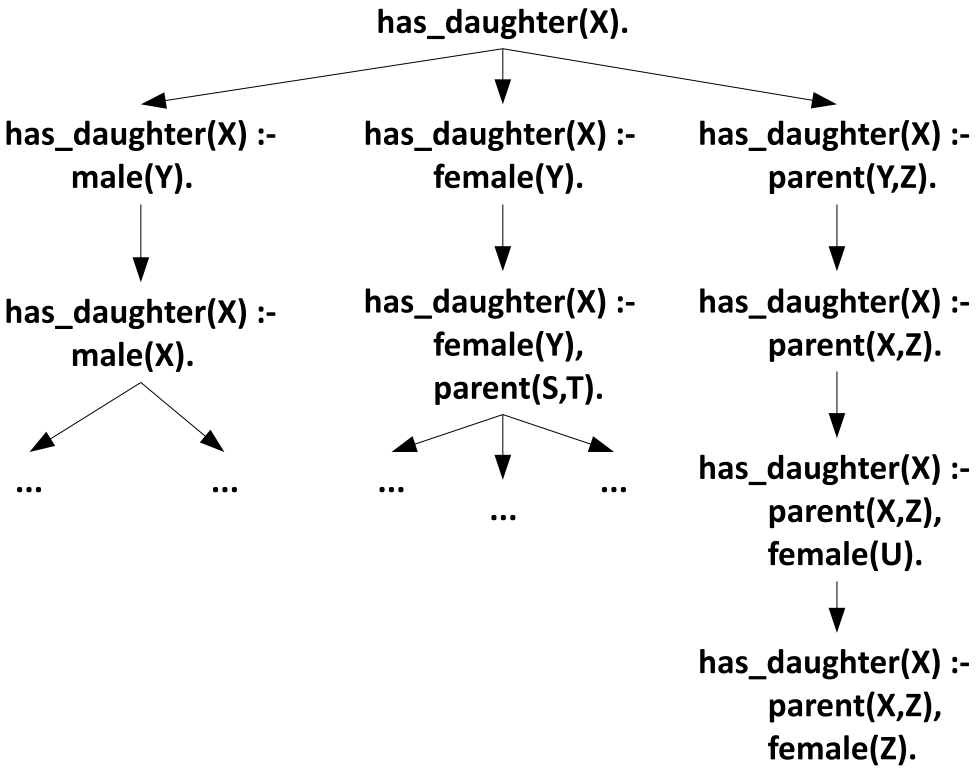
Sedaj si poglejmo, kako ILP program konstruira popolno in konsistentno hipotezo iz podanih učnih primerov. To si bomo pogledali kar na preprostem učnem primeru iz prejšnjega podpoglavja, kateri je podan na sliki 2.2. Podano imamo predznanje o družinskih relacijah `parent(X,Y)`, kar pomeni, da je oseba, predstavljena s spremenjivko `X`, starš osebi, predstavljeni s spremenljivko `Y`. Poleg teh relacij imamo podano tudi predznanje v obliki literalov `male(X)` in `female(X)`, s katerimi je določen spol osebe, predstavljene s spremenljivko `X`. Naučiti se želimo definicije literala `has_daughter(X)`. *Definiciji literala* običajno rečemo kar *hipoteza*, zato bo v nadaljevanju uporabljen kar ta pojem.

Kaj naj velja za to hipotezo, uporabnik pove ILP programu s podajanjem pozitivnih in negativnih učnih primerov. V našem primeru imamo podane 3 pozitivne (`ex(has_daughter(tom))`, `ex(has_daughter(bob))`, `ex(has_daughter(pat))`) in 2 negativna učna primera (`nex(has_daughter(pam))`, `nex(has_daughter(jim))`). Kot je omenjeno že v prejšnjem podpoglavju pri formalni definiciji ILP problema, mora naučena hipoteza pokriti vse pozitivne učne primere (popolna hipoteza) in nobenega negativnega (konsistentna hipoteza).

ILP program začne iskanje popolne in konsistentne hipoteze z neko splošno hipotezo (v našem primeru bo to `has_daughter(X).`), katera sicer je popolna (pokriva vse pozitivne učne primere), vendar je tudi nekonsistentna (pokriva tudi negativne učne primere). To hipotezo mora ILP program ustrezno specializirati (izostriti), tako da ohrani njeno popolnost, obenem pa stremi k njeni

konsistentnosti. Vsaka izostritev vzame neko hipotezo H1 in proizvede bolj specifično hipotezo H2, tako, da H2 pokriva podmnožico učnih primerov pokritih s hipotezo H1.

Takemu prostoru hipotez in njihovih izostritev pravimo izostritveni graf. Del izostritvenega grafa za naš učni primer iz slike 2.2 vidimo na sliki 2.3. Vozlišča izostritvenega grafa predstavljajo hipoteze, povezave med njimi pa izostritve. Ko imamo izostritveni graf, je naš učni problem le še iskanje popolne in konsistentne hipoteze v tem grafu.



Slika 2.3: Izostritveni graf za učni problem iz slike 2.2

Za realizacijo tega pristopa sta torej potrebna:

- izostritveni operator, kateri generira izostritve posameznih hipotez
- iskalna procedura za iskanje hipotez v grafu izostritev.

Iz izostritvenega grafa na sliki 2.3 vidimo, da se uporabljata dve vrsti izostritev. Hipotezo izostrimo z:

- izenačenjem dveh spremenljivk
- dodajanjem novega literala (iz predznanja) v hipotezo.

Primer prvega tipa izostritev je:

```
has_daughter(X) :- parent(Y,Z).
```

izostrimo v

```
has_daughter(X) :- parent(X,Z).
```

z izenačenjem `Y=X`. Primer drugega tipa izostritev pa:

```
has_daughter(X).
```

izostrimo v

```
has_daughter(X) :- parent(Y,Z).
```

z dodajanjem literala `parent(Y,Z)` iz predznanja.

Poleg tega pa se izkaže, da obstaja še ena vrsta izostritev in sicer pri učenju hipotez, ki vsebujejo sezname (npr. učenje vstavljanja elementa v seznam, učenje sortiranja seznama, ...). V tem primeru se lahko hipoteza

```
member(X1,L1) :- member(X1,L3).
```

izostri v

```
member(X1,[X2|L2]) :- member(X1,L3).
```

kjer se spremenljivka `L1` zamenja s strukturo `[X2|L2]`.

Pri vsem tem pa je potrebno opozoriti na dve pomembni lastnosti izostritev:

- Vsaka izostritev je specializacija. To pomeni, da naslednik hipoteze H v izostritvenem grafu pokriva podmnožico primerov, pokritih s hipotezo H. Zato zadostuje, če pri iskanju popolne in konsistente hipoteze upoštevamo le popolne hipoteze. Namreč nepopolno hipotezo ne moremo nikoli izostriti v popolno.
- Izostritveni operator mora delati dovolj 'majhne' izostritvene korake. Namreč pregrob izostritveni operator lahko skoči iz popolne in nekonsistente hipoteze v nepopolno in konsistentno ter vmes preskoči popolno in konsistentno.

2.3 HYPER

V tem podpoglavju je predstavljen ILP program HYPER (hypothesis refiner). Napisan je v Prologu, vendar je v tem diplomskem delu zaradi preglednosti predstavljen le v psevdokodi, opisani pa so tudi vsi pomembnejši sklopi. Natančnejši opis HYPERja s celotno izvorno kodo pa se nahaja v [1].

Pomemben del HYPER-ja je interpreter za generirane hipoteze, ki deluje podobno kot Prolog. Ob podani hipotezi in cilju (pozitiven ali negativen učni primer) nam interpreter odgovori, če je cilj možno logično izpeljati iz podane hipoteze. Interpreter nam bi načeloma lahko odgovoril le z *yes* ali *no*, vendar težava nastopi pri dokazovanju rekurzivnih hipotez tipa

```
p(X) :- p(X).
```

kjer obstaja nevarnost neskončnih zank. Zaradi tega moramo v interpreterju omejiti dolžino dokaza. Če pri dokazovanju cilja število klicev doseže mejo dovoljene dolžine dokaza, se dokazovanje prekine, interpreter pa vrne odgovor *maybe*. Pri tovrstnem odgovoru se HYPER odzove različno glede na cilj, katerega dokazuje:

- Če dokazuje pozitiven učni primer, se odgovor *maybe* smatra kot *primer ni pokrit*.
- Če dokazuje negativen učni primer, se odgovor *maybe* smatra kot *primer je pokrit*.

Razlog za tako interpretacijo odgovora *maybe* je, da med vsemi možnimi popolnimi hipotezami favoriziramo računsko učinkovite. Odgovor *maybe* v najboljšem primeru pomeni, da je hipoteza računsko neučinkovita, v najslabšem primeru pa, da je hipoteza nepopolna.

Na sliki 2.4 lahko vidimo glavno HYPER-jevo zanko. Izvajanje programa se začne s seznamom začetnih hipotez, katere uporabnik poda HYPER-ju. Ta seznam običajno zajema le glave hipotez, tj. literale, katerih definicije se hočemo naučiti. V učnem primeru, predstavljenem na sliki 2.2 (učenje družinskih relacij) bi tako kot začetno hipotezo podali `has_daughter(X).` Uporabnik lahko poda eno ali več začetnih hipotez. Za vse hipoteze se nato izračuna cena (več o računanju cene hipotez v naslednjem odstavku), hipoteze pa se nato sortira naraščujoče po njihovi ceni. Za prvo izostritev se nato izbere prva hipoteza iz seznama, torej hipoteza z najnižjo ceno.

Hyper bo izostreval hipoteze dokler ne naleti na hipotezo s ceno 0. Kako se izračuna cena hipoteze, je razvidno iz slike 2.5. Število pokritih negativnih

```
Hyps = seznam začetnih hipotez;
Hyp = vrniPrvo(Hyps);

WHILE Hyp.cena >= 0
  IF Hyp.cena = 0 AND popolna(Hyp)
    vrni Hyp kot rezultat;
    IF uporabnik zadovoljen z rezultatom
      zaključi z izvajanjem glavne zanke;
    ELSE
      nadaljuj z izvajanjem glavne zanke;

  Hyps0 = vseIzostritve(Hyp);
  Hyps = dodajVseznam(Hyps0,Hyps);
  Hyp = vrniPrvo(Hyps);
```

Slika 2.4: Glavna HYPERjeva zanka

učnih primerov se izračuna s pomočjo HYPER-jevega interpreterja hipotez in sicer tako, da preštejemo vse negativne učne primere, za katere HYPER-jev interpreter odgovori z *yes* ali *maybe*. Število pokritih pozitivnih učnih primerov se v računanju cene ne upošteva, ker morajo biti vse hipoteze popolne, zato število pokritih učnih primerov ne vpliva na oceno.

```
IF število pokritih negativnih primerov = 0
  Cena = 0;
ELSE
  Velikost = k1 * število literalov v hipotezi +
             + k2 * število spremenljivk v hipotezi;
  Cena = Velikost + 10 * število pokritih negativnih primerov;
```

Slika 2.5: Računanje cene hipoteze

V izračunu cene se uporabljata tudi dve konstanti, kateri imata privzeti vrednosti `k1 = 10` in `k2 = 1`, uporabnik pa ju lahko poljubno nastavi. Če naprimer povečamo vrednost uteži `k1`, bomo favorizirali krajše hipoteze (z manj literali) in obratno, če vrednost uteži zmanjšamo. Podobno velja za utež `k2`, le da ta utež vpliva na število spremenljivk v hipotezi.

Predpostavimo, da imamo v nekem trenutku izvajanja HYPER-ja hipotezo `Hyp` s ceno višjo od 0. HYPER bo za to hipotezo poiskal vse njene izostritve po pravilih, opisanih v prejšnjem podpoglavju, s tem, da pri tem upošteva nekaj pomembnih detajlov, ki so opisani v naslednjih treh odstavkih.

Uporabnik mora pri definiranju literalov določiti vrsto in tip spremenljivk, ki nastopajo v literalu. Tipi posameznih spremenljivk so lahko med seboj enaki, priporočljivo (za pohitritev učenja) pa je, da se spremenljivkam smiselno določi tipe. V literalu `member(X,L)` (preverjanje ali se element `X` nahaja v seznamu `L`) bi tako lahko določili obema spremenljivkama isti tip, vendar bo učenje bistveno hitrejše, če spremenljivki `X` določimo tip `item`, spremenljivki `L` pa tip `list`. Po vrsti pa se spremenljivke delijo na vhodne in izhodne. Če je spremenljivka vhodna, se mora pri dodajanju literala v hipotezo obvezno izenačiti s spremenljivko enakega tipa, ki že nastopa v hipotezi. Če pa je spremenljivka izhodna, pa se spremenljivka *lahko* kasneje izenači z neko drugo spremenljivko v hipotezi.

Če je hipoteza sestavljena iz enega samega stavka, potem ni dvoma pri izbiranju stavka za nadaljnjo izostritev. Če pa je hipoteza sestavljena iz več stavkov, HYPER poišče prvi stavek, ki pokriva vsaj en negativen učni primer (če pri dokazovanju negativnih učnih primerov z izbranim stavkom interpreter na vsaj en dokaz vrne odgovor *yes* ali *maybe*) in izostri samo ta stavek. Tako v hipotezi najprej izostrujemo le prvi stavek do konsistentnosti, ko je ta konsistenten, izostrujemo drugega, itd. Razlog za to, da v hipotezi z več stavki posamezne stavke izostrujemo samo do konsistentnosti in ne do popolnosti, je ta, da so stavki med sabo disjunktno povezani. Tako lahko gledamo na množico pokritih pozitivnih učnih primerov hipoteze kot na unijo množic pozitivnih učnih primerov, pokritih s posameznimi stavki v hipotezi. To pomeni, da ni nujno, da posamezen stavek v hipotezi pokriva vse pozitivne učne primere. Pomembno je le, da unija vseh množic pozitivnih učnih primerov, pokritih s posameznimi stavki, predstavlja celotno množico pozitivnih učnih primerov. Enako velja za množico pokritih negativnih učnih primerov. Torej če hočemo, da je hipoteza konsistentna, morajo biti tudi posamezni stavki konsistentni.

Poleg tega HYPER zavrže redundantne stavke (to so stavki, ki vsebujejo več kopij istega literala) in stavke, katerih telo ni mogoče izpeljati s pomočjo HYPERjevega interpreterja iz trenutne hipoteze.

Upoštevajoč vse te detajle HYPER lahko izostri hipotezo z:

- Izenačenjem dveh spremenljivk istega tipa v stavku.
- Zamenjavo spremenljivke s seznamom ali konstanto.

- Dodajanjem novega literala; ob tem se vse vhodne spremenljivke v literalu izenačijo z obstoječimi spremenljivkami v stavku.

Če izostrena hipoteza ni popolna, jo HYPER v tem koraku zavrže. Prav tako se v tem koraku izračunajo tudi cene popolnih hipotez.

V naslednjem koraku glavne HYPER-jeve zanke (slika 2.4) imamo v spremenljivki `Hyps0` shranjen seznam izostrenih, popolnih hipotez ter njihovih cen. Ta seznam se nato sortira po ceni od najnižje do najvišje, zatem pa se vstavi v seznam hipotez `Hyp`, čakajočih na nadaljnjo izostritev in sicer tako, da je na koncu tudi ta seznam sortiran po naraščujoči ceni. Iz tega seznama se nato vzame prva hipoteza (z najnižjo ceno) in celotna zanka se ponovi.

Če HYPER pri izvajanju te zanke naleti na hipotezo s ceno 0 (popolna in konsistentna hipoteza), to hipotezo ponudi uporabniku kot rešitev. Če je ta s ponujeno rešitvijo zadovoljen, se izvajanje HYPER-ja zaključi, če pa ni zadovoljen, pa se izvajanje glavne zanke HYPER-ja nadaljuje. Če HYPER iz kateregakoli razloga ne najde popolne in konsistentne hipoteze, bo njegova glavna zanka tekla v neskončnost.

Poglavje 3

Markovske logične mreže in Alchemy

V tem poglavju so najprej predstavljene osnove markovskih logičnih mrež, nato pa je predstavljen še program za statistično relacijsko učenje in verjetnostno napovedovanje Alchemy, kateri se v celoti opira na teorijo markovskih logičnih mrež.

3.1 Markovske logične mreže

Sledeče podpoglavje je povzeto po [5].

Kombiniranje verjetnosti in logike prvega reda je bilo dolgo cilj umetne inteligence. Logika prvega reda nam namreč omogoča kompaktno in naravno predstavitev znanja, verjetnost pa učinkovito obvladovanje negotovosti. Večina realnih problemov od nas zahteva uporabo obojega. V ta namen sta avtorja Richardson in Domingos v [5] predlagala združitev in uporabo markovskih mrež in logike prvega reda v markovske logične mreže (*Markov logic networks* oz. s kratico *MLN*). Markovska logična mreža je množica hipotez, zapisanih v logiki prvega reda, s pripadajočimi utežmi in nam služi kot predloga za izgradnjo markovske mreže. Z vidika logike prvega reda nam markovske logične mreže ponujajo kompakten jezik za definiranje velikih markovskih mrež in možnost, da vanje fleksibilno in modularno vpeljemo predznanje o učni domeni. Z vidika verjetnosti pa nam markovske logične mreže omogočajo učinkovito obravnavanje negotovosti, nepravilnosti in nasprotujočih si dejstev v podatkih.

3.1.1 Markovske mreže

Markovska mreža je model za predstavitev multivariatne verjetnostne porazdelitve množice spremenljivk $X = (X_1, X_2, ..., X_n) \in \chi$. Multivariatna verjetnostna porazdelitev je podana z enačbo:

$$P(X = x) = \frac{1}{Z} exp\left(\sum_i w_i f_i(x)\right) \tag{3.1}$$

kjer je w_i utež, f_i pa značilka stanja spremenljivke x. Ta ima lahko katerokoli realno vrednost, vendar kot bomo videli kasneje, se osredotočimo le na binarne značilke, $f_i(x) \in \{0, 1\}$. Z pa je particijska funkcija, podana z enačbo:

$$Z = \sum_{x \in \chi} exp\left(\sum_i w_i f_i(x)\right) \tag{3.2}$$

3.1.2 Logika prvega reda

Hipoteze, zapisane v logiki prvega reda, sestavljajo:

- konstante
- spremenljivke
- funkcije (preslikave n-teric objektov v objekte)
- literali (relacije med objekti)

Osnovno hipotezo dobimo tako, da z literalom povežemo n objektov, kateri so lahko konstante, spremenljivke ali funkcije. Hipotezo pa sestavimo tako, da osnovne hipoteze z uporabo logičnih simbolov in kvantifikatorjev (negacija, konjunkcija, disjunkcija, implikacija, ekvivalenca, univerzalni in eksistencialni kvantifikator) povežemo skupaj. Množici hipotez pravimo *model*.

Osnovni atom je literal ali funkcija, ki ne vsebuje spremenljivk, le konstante. Primer: recimo, da imamo konstante `bob`, `john` in `mary`, ter literal `person(X)`. Vsi osnovni atomi tega literala so: `person(bob)`, `person(john)` in `person(mary)`. *Osnova hipoteze* je hipoteza, sestavljena iz osnovnih atomov.

Svet je predstavljen z množico konkretnih vrednosti objektov, funkcij in relacij med objekti.

Logika prvega reda postavlja strogo omejitev. Namreč če vsaj ena hipoteza iz modela, predstavljenega v logiki prvega reda, krši nek svet, potem je verjetnost tega sveta enaka 0.

3.1.3 Markovske logične mreže

Ideja markovskih logičnih mrež je omiliti omejitev, katero postavlja logika prvega reda. Če v markovskih logičnih mrežah hipoteza iz modela krši nek svet, ta svet še ni nemogoč, le manj verjeten. Njegova verjetnost pa je odvisna od uteži hipoteze, katera krši svet. Višja kot je utež hipoteze, manj bo verjeten svet, ki ga hipoteza s to utežjo krši. Če pa je utež hipoteze negativna, bo svet, ki ga taka hipoteza krši, imel celo višjo verjetnost, kot svet, ki ga hipoteza izpolnjuje.

Poglejmo si formalno definicijo markovskih logičnih mrež:
Markovska logična mreža L je množica parov (F_i, w_i), kjer F_i predstavlja hipotezo, $w_i \in \Re$ pa utež te hipoteze. Skupaj s končno množico konstant $C = \left\{c_1, c_2, ..., c_{|C|}\right\}$ definira markovsko mrežo $M_{L,C}$, za katero velja:

1. $M_{L,C}$ vsebuje binarno vozlišče za vsak možni osnovni atom, ki se pojavlja v L. Vrednost vozlišča je 1, če se osnovni atom, predstavljen s tem vozliščem, tudi dejansko pojavlja v opazovanem svetu, sicer pa ima vrednost 0.

2. $M_{L,C}$ vsebuje značilko f_i za vsako možno osnovo hipoteze F_i, ki se pojavlja v L. Vrednost te značilke je 1, če značilka predstavlja osnovo hipoteze z logično vrednostjo *true*, sicer pa je vrednost značilke 0. Utež te značilke je w_i.

Za nas je pomembna le druga točka te definicije. Zaradi nje v enačbah 3.1 in 3.2 za računanje multivariatne verjetnostne porazdelitve uporabljamo le vrednosti značilk $f_i \in \{0, 1\}$. Ker pa so za računanje enačb 3.1 in 3.2 pomembne pravzaprav le značilke, z vrednostjo $f_i = 1$, lahko ti dve enačbi zapišemo kot:

$$P(X = x) = \frac{1}{Z} exp\left(\sum_i w_i n_i(x)\right) \tag{3.3}$$

$$Z = \sum_{x \in \chi} exp\left(\sum_i w_i n_i(x)\right) \tag{3.4}$$

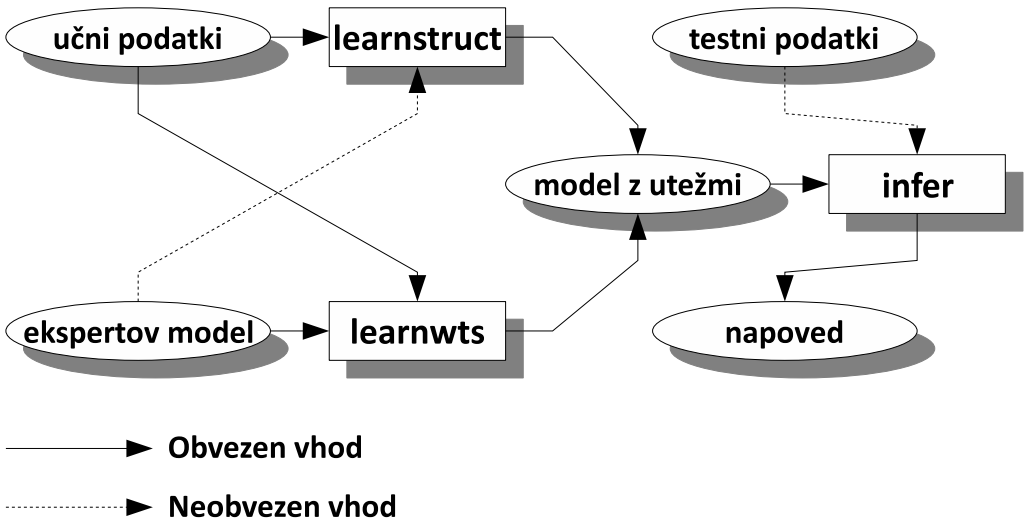
kjer je:

- $n_i(x)$ število osnov hipoteze F_i z logično vrednostjo *true*
- w_i pa utež hipoteze F_i.

3.2 Alchemy

Alchemy je programski paket algoritmov za statistično relacijsko učenje in verjetnostno logično sklepanje, ki se v celoti opirajo na idejo markovskih logičnih mrež. Sestavljajo ga trije programi:

- learnwts: algoritmi za učenje uteži za posamezne hipoteze
- learnstruct: algoritmi za učenje hipotez iz učnih podatkov
- infer: algoritmi za verjetnostno logično sklepanje.



Slika 3.1: Struktura Alchemyja s tokom vhodnih in izhodnih podatkov

Kako se ti programi medsebojno povezujejo, kakšne podatke potrebujejo in kakšne rezultate vračajo, je razvidno iz slike 3.1. Pravokotniki predstavljajo zgoraj omenjene programe, elipse predstavljajo vhodne podatke oziroma podatke, ki so rezultat izvajanja enega od programov (izhodni podatki), puščice pa tok podatkov. V primeru, ko puščice tečejo od podatkov k nekemu Alchemyjevemu programu, puščica z neprekinjeno črto pomeni, da so ti vhodni podatki obvezni, puščica s prekinjeno črto pa, da so ti vhodni podatki opcijski.

V nadaljevanju sledi podrobnejši opis Alchemyjevih programov, vhodnih podatkov in rezultatov.

3.2.1 Učenje uteži

Učenje uteži se izvaja v programu *learnwts*. Kot je razvidno iz slike 3.1, za to potrebuje ekspertov model, ki je množica hipotez, katere po ekspertovem mnenju dobro opisujejo problemsko domeno. Nato pa na podlagi učnih domen (učnih podatkov), katere moramo prav tako podati na vhod, izračuna uteži posameznih hipotez v ekspertovem modelu. Kot rezultat pa vrne ekspertov model z dodanimi utežmi za posamezne hipoteze.

Učenje uteži v Alchemyju se deli na generativno [5] in diskriminativno [6]. Razlika med njima je v tem, da pri diskriminativnem učenju vnaprej vemo, katere literale bomo podali kot predznanje in katere bomo ob tem podanem predznanju napovedovali. [6]. Učenje se lahko s tem bistveno pohitri.

Za oba pristopa k učenju uteži obstaja veliko algoritmov, kateri se opirajo na kompleksne verjetnostne teorije in kot taki presegajo okvire tega diplomskega dela. Povečini prinašajo le izboljšave v hitrosti, medtem ko so naučene uteži približno enake. Zato si bomo v tem razdelku pogledali le enega od preprostejših algoritmov, algoritem *voted perceptron* [5],[4].

Voted perceptron algoritem deluje po principu gradientnega spuščanja, prikazanega z enačbo 3.5:

$$\vec{w_{t+1}} = \vec{w}_t + \eta\vec{g} \tag{3.5}$$

kjer $\vec{w}_t$ predstavlja vektor uteži v iteraciji t, η predstavlja parameter za nastavljanje hitrosti učenja, $\vec{g}$ pa predstavlja vektor gradientov uteži. Za $\vec{w_0}$ lahko vzamemo praktično katerikoli vektor uteži, vendar običajno vzamemo kar vektor ničel.

i-ti element vektorja gradientov uteži lahko izračunamo po enačbi 3.6:

$$g_i(x) \;=\; n_i(x) - \sum_{x'} P_w(X = x')n_i(x') \tag{3.6}$$

kjer:

- $x = (x_1, ..., x_i, ..., x_n)$ predstavlja množico vseh možnih osnovnih atomov. $x_i = 1$, če se osnovni atom pojavlja tudi v učni oziroma testni domeni, sicer pa je $x_i = 0$
- $g_i(x)$ predstavlja i-ti element vektorja gradientov uteži
- $P_w(X = x)$ predstavlja verjetnostno porazdelitev za množico osnovnih atomov x, izračunano po enačbi 3.3, pri vektorju uteži w

- $n_i(x)$ predstavlja število osnov hipoteze F_i z logično vrednostjo *true* pri dani množici osnovnih atomov x
- $\sum_{x'}$ predstavlja vsoto po vseh možnih množicah osnovnih atomov.

Učenje uteži se ustavi, ko je gradient manjši od predpisanega ϵ. Da se izognemo prevelikemu prilagajanju učnim podatkom, je končni vektor uteži aritmetično povprečje vseh prejšnjih vektorjev uteži.

Opisana metoda je sicer preprosta, vendar v praksi neučinkovita, saj spada med #P-polne probleme. [5] #P-poln problem je povezan z NP-polnim problemom; če nas pri NP-polnem problemu zanima npr. če se v grafu nahaja Hamiltonov cikel, nas pri #P-polnem problemu zanima, koliko Hamiltonovih ciklov se nahaja v grafu. Zaradi #P-polnosti tega problema se pri računanju enačbe 3.6 uporabljajo določene aproksimacije, katere pa za razumevanje učenja uteži v Alchemyju niso bistvene.

Za lažjo predstavo si poglejmo učenje uteži na preprostem primeru goljufive kocke. Kocka je prilagojena tako, da pade vedno 3 ali 6 (z enako verjetnostjo za oba izida). S to kocko naredimo 20 poizkusov, rezultati teh poizkusov so predstavljeni na sliki 3.2. Ta slika predstavlja učno domeno za naš učni primer. Izid posameznega meta je definiran z literalom `Outcome(throw,face)`, kjer `throw` označuje zaporedno številko poizkusa, `face` pa izid meta.

Kot ekspertov model podamo sledečih 6 hipotez:

```
Outcome(t,1)
Outcome(t,2)
Outcome(t,3)
Outcome(t,4)
Outcome(t,5)
Outcome(t,6)
```

naučiti pa se hočemo uteži za vsak možen izid meta kocke (tj. vsako hipotezo v podanem modelu). Množica osnovnih atomov $x = (x_1, ..., x_i, ..., x_n)$ bo definirana kot vse možne osnove literala `Outcome(t,f)`, kjer $t \in \{1, 2, ..., 20\}$ in $f \in \{1, 2, ..., 6\}$. $x_i = 1$, če se ta osnovni atom pojavlja tudi v učni domeni (slika 3.2), sicer pa $x_i = 0$.

V prvi iteraciji imamo vektor uteži enak $\vec{w_0} = [0, 0, 0, 0, 0, 0]$, izračunati pa moramo vektor gradientov $\vec{g}$. Posamezne elemente $g_i(x)$ tega vektorja lahko izračunamo po enačbi 3.6. Preden se spustimo v računanje gradienta, izračunajmo vseh 6 vrednosti $n_i(x)$ (za vsako hipotezo v ekspertovem modelu ena):

```
Outcome(1,3)
Outcome(2,6)
Outcome(3,3)
Outcome(4,6)
Outcome(5,3)
Outcome(6,6)
Outcome(7,3)
Outcome(8,6)
Outcome(9,3)
Outcome(10,6)
Outcome(11,3)
Outcome(12,6)
Outcome(13,3)
Outcome(14,6)
Outcome(15,3)
Outcome(16,6)
Outcome(17,3)
Outcome(18,6)
Outcome(19,3)
Outcome(20,6)
```

Slika 3.2: Rezultati poizkusov z goljufivo kocko

$n_1(x) = 0$
$n_2(x) = 0$
$n_3(x) = 10$
$n_4(x) = 0$
$n_5(x) = 0$
$n_6(x) = 10$

Za primer, kako smo izračunali zgornje rezultate: $n_3(x)$ smo izračunali tako, da smo naredili vse osnove hipoteze `Outcome(t,3)` pri $t \in \{1, 2, ..., 20\}$. Te so: `Outcome(1,3)`, `Outcome(2,3)`, `Outcome(3,3)`, ..., `Outcome(20,3)`. Nato pa smo prešteli koliko teh osnov se je dejansko pojavilo v učnih podatkih (slika 3.2). Le-teh je bilo 10 (`Outcome(1,3)`, `Outcome(3,3)`, `Outcome(5,3)`, ..., `Outcome(19,3)`), zato je $n_3(x) = 10$.
Sedaj pa lahko po enačbi 3.6 izračunamo posamezne elemente vektorja gradientov:

$g_1(x) = 0 - \sum_{x'} P_w(X = x')n_1(x')$
$g_2(x) = 0 - \sum_{x'} P_w(X = x')n_2(x')$
$g_3(x) = 10 - \sum_{x'} P_w(X = x')n_3(x')$
$g_4(x) = 0 - \sum_{x'} P_w(X = x')n_4(x')$
$g_5(x) = 0 - \sum_{x'} P_w(X = x')n_5(x')$
$g_6(x) = 10 - \sum_{x'} P_w(X = x')n_6(x')$

Vendar tukaj naletimo na težavo. Računanje vsote $\sum_{x'} P_w(X = x')n_i(x')$ je namreč zelo kompleksen problem. Število osnovnih atomov literala `Outcome(t,f)`, pri $t \in \{1, 2, ..., 20\}$ in $f \in \{1, 2, ..., 6\}$ je 120. Glede na to, da ima lahko osnovni atom v množici x' vrednost 0 ali 1, lahko ugotovimo, da je vseh možnih množic 2^{120}! Zato moramo za praktično računanje nujno uporabiti aproksimacije. Če poženemo učenje uteži v Alchemyju za ta primer, se s pomočjo aproksimacijskega algoritma naučimo sledečih uteži:

$\vec{w} = [-1.30, -1.25, 2.11, -1.25, -1.24, 2.12]$

Kot vidimo sta 3. in 6. utež bistveno višji od ostalih in približno enaki med sabo. Ti dve uteži ustrezata hipotezama v ekspertovem modelu, ki opisujeta izida 3 in 6 na goljufivi kocki. Uteži sta višji od ostalih, ker sta povezani s hipotezama, ki opisujeta edina izida pri poizkusih z goljufivo kocko. Med sabo pa sta (približno - zaradi aproksimacije) enaki, ker je število poizkusov, kjer je bil izid na kocki 3, enako številu poizkusov z izidom 6.

3.2.2 Učenje hipotez

Učenje hipotez se dogaja v programu *learnstruct* in pravzaprav v sebi vključuje tudi učenje uteži za množico naučenih hipotez. Ta program bomo za razliko prejšnjega (*learnwts*) uporabili, kadar nimamo na voljo ekspertovega predznanja o domeni, temveč le podatke o dogajanju v nekem sistemu, okolju, ... oziroma ko je ekspertovo znanje pomanjkljivo ali mogoče celo napačno.

Učenje hipotez na vhodu zahteva učne podatke, opcijsko pa tudi ekspertov model. Če podamo na vhod poleg učnih podatkov tudi ekspertov model, bo Alchemy preveril pravilnost posameznih hipotez in jih po potrebi dodatno izostril ter dodal manjkajoče hipoteze. [5],[3] To je uporabno, če imamo na voljo le pomanjkljivo (ali mogoče celo napačno) znanje o učni domeni.

Za učenje hipotez se je v Alchemyju sprva uporabljal ILP program CLAUDIEN [5], kasneje pa so bili razviti tudi lastni algoritmi za učenje hipotez. Osnove enega od teh algoritmov opisujemo v nadaljevanju.

Pri učenju se uporablja hevristična funkcija, poimenovana *utežena psevdo-log-verjetnost* (*weighted pseudo-log-likelihood* ali s kratico *WPLL*) [3]:

$$logP_w^{\bullet}(X = x) \;=\; \sum_{r \in R} c_r \sum_{k=1}^{g_r} logP_w(X_{r,k} = x_{r,k} | MB_x(X_{r,k})) \qquad (3.7)$$

kjer je:

- R množica literalov prvega reda
- g_r število osnov literala r
- c_r poljubna utež; če želimo, da so literali obravnavani enako, ne glede na njihovo številnost (število spremenljivk v literalu), nastavimo $c_r = \frac{1}{g_r}$
- $x_{r,k} = 1$, če ima k-ta osnova r-ja logično vrednost *true*, sicer pa je $x_{r,k} = 0$
- $MB_x(X_{r,k})$ stanje *markovske odeje* (*Markov blanket*) spremenljivke $X_{r,k}$. To je stanje vseh vozlišč, ki so v markovski mreži neposredno povezana z vozliščem, označenim s spremenljivko $X_{r,k}$.
- $P_w^{\bullet}(X = x)$ pa označuje, da računamo aproksimacijo enačbe 3.3 pri vektorju uteži w.

Pri konstruiranju hipotez sta najpogostejši operaciji dodajanja novih literalov in odstranjevanja obstoječih (v primeru, ko na vhod podamo tudi ekspertov model, v katerem mora Alchemy odpraviti napake). Ker je pogosta napaka v ekspertnih modelih napačna smer implikacije, obstaja tudi operacija zamenjave vrstnega reda literalov. Pri dodajanju novega literala v hipotezo se naredijo tudi vse možne (in smiselne - upoštevanje tipov spremenljivk) kombinacije spremenljivk v literalu s spremenljivkami, ki že nastopajo v hipotezi.

Za iskanje najboljših hipotez sta na voljo dve iskalni strategiji. Prva (hitrejša) uporablja iskanje v snopu, hipoteze pa dodaja v model eno po eno, pri tem dolžina hipoteze ni pomembna. Z zgoraj opisanimi operacijami naredi izostritve hipoteze, nato pa obdrži b (širina snopa) najboljših. To počne, dokler nobena nova hipoteza ne izboljša WPLL, podan z enačbo 3.7. Končni rezultat je hipoteza z najvišjim WPLL v katerikoli iteraciji izvajanja algoritma. Druga (popolnejša) strategija pa uporablja iterativno poglabljanje. Za spremembo od prve strategije ta dodaja k hipotez naenkrat v model. Zaradi iterativnega poglabljanja v model doda najprej vse „dobre“ hipoteze dolžine l, preden se sploh loti iskanja hipotez dolžine $l + 1$. [3]

3.2.3 Verjetnostno logično sklepanje

Verjetnostno logično sklepanje se izvaja v programu *infer*. Za to potrebuje model z utežmi in opcijsko podatke, na podlagi katerih želimo delati napovedi. Če podatkov ne podamo, Alchemy izračuna verjetnosti vseh možnih osnov literalov v modelu.

Najverjetnejše stanje sveta bi načeloma lahko iskali s pomočjo enačb 3.3 in 3.4, vendar kot smo že omenili, je računanje teh enačb #P-poln problem. V praksi se zato uporabljajo stohastični algoritmi (MaxWalkSat, LazyWalkSat), ki vrnejo približek rešitve, katero bi dobili z računanjem enačb 3.3 in 3.4. [2] Razumevanje delovanja teh algoritmov ni bistveno za razumevanje delovanja Alchemyja, zato opis ni podan.

Za lažje razumevanje pa si poglejmo primer goljufive kocke, katera je prilagojena tako, da na njej pade vedno 3 ali 6, oba izida z enako verjetnostjo. Pri opisu programa za učenje uteži *learnwts* smo se že naučili uteži za hipoteze, ki opisujejo posamezne izide. Te so:

```
-1.30 Outcome(t,1)
-1.25 Outcome(t,2)
 2.11 Outcome(t,3)
-1.25 Outcome(t,4)
-1.24 Outcome(t,5)
 2.12 Outcome(t,6)
```

sedaj pa si poglejmo še, kako po enačbi 3.3 izračunamo verjetnosti posameznih izidov. Računali bomo verjetnosti samo za en met (spremenljivka `t` v literalu `Outcome` se bo nastavila na 1). Najprej izračunajmo particijsko funkcijo po enačbi 3.4:

$$Z = e^{-1.30 \cdot 1} + e^{-1.25 \cdot 1} + e^{2.11 \cdot 1} + e^{-1.25 \cdot 1} + e^{-1.24 \cdot 1} + e^{2.12 \cdot 1} = 17.71$$

V tej enačbi smo vzeli $n_i(x) = 1$, saj za i-ti izid obstaja samo en možen osnovni atom `Outcome` in sicer `Outcome(1,i)` ($i \in \{1, 2, ..., 6\}$). Sedaj se lahko lotimo računanja verjetnosti posameznih izidov po enačbi 3.3:

$$P(X = 1) = \frac{1}{17.71} e^{-1.30 \cdot 1} = 0.015$$

$$P(X = 2) = \frac{1}{17.71} e^{-1.25 \cdot 1} = 0.016$$

$$P(X = 3) = \frac{1}{17.71} e^{2.11 \cdot 1} = 0.466$$

$P(X = 4) = \frac{1}{17.71} e^{-1.25 \cdot 1} = 0.016$

$P(X = 5) = \frac{1}{17.71} e^{-1.24 \cdot 1} = 0.016$

$P(X = 6) = \frac{1}{17.71} e^{2.12 \cdot 1} = 0.47$

Pri teh rezultatih se jasno izrazi ideja markovskih logičnih mrež - izidi 1, 2, 4 in 5 se pri poizkusih z goljufivo kocko (slika 3.2) niso pojavili, vendar jih Alchemy zaradi tega še ne smatra kot nemogoče, temveč le kot zelo malo verjetne. Po drugi strani pa sta verjetnosti za izida 3 in 6 med sabo približno enaki in bistveno višji od ostalih, kar se je odrazilo tudi pri poizkusih z goljufivo kocko.

3.3 Primerjava učenja standardnih ILP problemov s HYPER-jem in Alchemyjem

V tem podpoglavju si oglejmo primerjavo med HYPER-jem in Alchemyjem na standardnih ILP problemih, opisanih v [1].

Prvo primerjavo smo naredili z učenjem družinskih relacij (slika 2.2). HYPER se tukaj nauči hipotezo:

```
has_daughter(X) :-
  parent(X,Y),
  female(Y).
```

katera se interpretira kot: *če je X starš Y in je Y ženskega spola, potem ima X hčer.*

Alchemy pa se nauči sledečo hipotezo:

$parent(X, Y) \Rightarrow has\_daughter(X) \vee has\_daughter(Y)$

katera se interpretira kot: *če je X starš Y, potem ima vsaj eden od njiju hčer.* Ta interpretacija je sicer pravilna v kontekstu učnih podatkov iz slike 2.2, vendar je očitno, da v splošnem ne velja. Če smo Alchemyju podali preproste, nedvoumne podatke, prikazane s shemo na sliki A.1, s podanimi vsemi možnimi pozitivnimi in negativnimi učnimi primeri, se je iz teh podatkov naučil boljšo hipotezo:

$parent(A, B) \wedge female(B) \Rightarrow has\_daughter(A)$

katera se interpretira kot: *če je A starš B in je B ženskega spola, potem ima A hčer.* Naučena hipoteza je tokrat pravilna.

Testirali smo še na večji učni domeni družinskih relacij s 23 osebami, prikazani na sliki A.2, s podanimi vsemi možnimi pozitivnimi in negativnimi učnimi primeri. Alchemy se nauči sledeče hipoteze, katera pa ima negativno, zelo nizko utež:

$$parent(A, B) \wedge female(B) \Rightarrow \neg has\_daughter(A)$$

Ta se interpretira kot: *če je A starš B, in je B ženskega spola, potem A nima hčere.* Naučena hipoteza je ravno obratna, kot bi pričakovali, to se odrazi tudi na njeni nizki negativni uteži. Torej nek svet, ki ga ta hipoteza krši, bo imel večjo verjetnost, kot pa tisti, ki ga hipoteza izpolnjuje. Čeprav je naučena hipoteza v navezi z naučeno negativno utežjo tehnično pravilna, bi vseeno raje videli, da se Alchemy nauči hipotezo, ki ni v popolnem nasprotju z našimi intuitivnimi pričakovanji.

Naslednji test je bil test učenja rekurzije. Za to smo izbrali problem učenja prednika, s predznanjem podanim na sliki 2.2 in učnimi primeri podanimi na sliki A.3. HYPER se nauči za ta problem sledeče hipoteze:

```
predecessor(A,B) :-
  parent(A,B).

predecessor(A,B) :-
  parent(A,C),
  predecessor(C,B).
```

katera se interpretira kot: *A je prednik B-ja, če je A B-jev starš, oziroma če ima A otroka C (torej je A C-jev starš), C pa je B-jev prednik.*

Alchemy pa se iz teh podatkov nauči sledeče hipoteze:

$$parent(A, B) \Rightarrow predecessor(C, B) \vee \neg predecessor(C, D) \vee \neg predecessor(E, B) \vee \neg predecessor(F, E)$$

Ta hipoteza nima neke smiselne interpretacije. Očitno je, da se literali `prececessor` le verižijo, kar pa z učenjem rekurzije nima nič skupnega. Da bi se prepričali, da do tega ni prišlo le po naključju zaradi nesrečno izbranih vhodnih podatkov, smo sestavili preprostejši učni primer za učenje pravila prednika, prikazan s shemo na sliki A.4, s podanimi vsemi možnimi pozitivnimi in negativnimi učnimi primeri. Alchemy se nauči sledeče hipoteze:

$parent(A, B) \Rightarrow predecessor(A, C) \vee predecessor(B, C) \vee \neg predecessor(A, B)$

Enako kot pri prejšnji hipotezi gre tudi tukaj le za veriženje literalov `prececessor`, hipoteza pa nima smiselne interpretacije. Sklepamo lahko, da Alchemy učenja rekurzije ne obvlada.

Še zadnji primer, na katerem smo izvedli primerjavo med HYPER-jem in Alchemyjem, pa je učni primer, kateri je priložen Alchemyju, skupaj z navodili za poganjanje učenja. S tem primerom lahko odstranimo dvom, da je bil HYPER v dosedanjih primerih boljši od Alchemyja zaradi napačno nastavljenih učnih parametrov pri slednjem. Učni primer je predstavljen na slikah A.5 in A.6. HYPER se nauči sledeče hipoteze:

```
professor(A) :-
  hasPosition(A,B).

student(A) :-
  inPhase(A,B).

advisedBy(A,B) :-
  publication(C,A),
  publication(C,B),
  inPhase(A,D),
  hasPosition(B,E).
```

Ta hipoteza se interpretira kot: *A je profesor, če je zaposlen na fakulteti. A je študent, če je v fazi študija. B je A-jev mentor, če sta oba avtorja publikacije C, obenem pa je A v fazi študija (A je študent), B pa je zaposlen na fakulteti (B je profesor).*

Alchemy pa se nauči hipoteze:

$professor(A) \Rightarrow \neg student(A)$

katera se interpretira kot: *če je A profesor, ne more biti študent.* Hipoteza je sicer pravilna, vendar z njeno pomočjo ne moremo napovedovati relacije mentorstva ter kdaj je oseba študent in kdaj profesor (ne pozabimo, da v predznanju nimamo podanih literalov `professor(X)` in `student(X)`).

Poglejmo si še, kako se odreže učenje uteži z Alchemyjem, če mu podamo eno pravilno in dve napačni hipotezi. Testirali smo na standardnem ILP učnem primeru za učenje družinskih relacij, Alchemy pa se je za sledeče tri hipoteze (prva je pravilna, drugi dve napačni) naučil naslednje uteži:

1.40628 $parent(a, b) \wedge female(b) \Rightarrow hasdaughter(a)$

-0.210452 $parent(a, b) \wedge male(b) \Rightarrow hasdaughter(a)$

0.303093 $parent(a, b) \Rightarrow hasdaughter(a)$

Kot vidimo je prva (pravilna) hipoteza dobila bistveno višjo utež od ostalih dveh (napačni). Druga hipoteza, ki je v bistvu ravno nasprotje prve, pa dobi celo negativno utež. Svet, ki ga taka hipoteza izpolni, bo imel zato zelo nizko verjetnost.

Glede na to, da se HYPER nauči popolnejše hipoteze tudi na primeru, ki je priložen Alchemyju (vključno z navodili za poganjanje učenja), lahko sklepamo, da je pri učenju hipotez iz čistih učnih podatkov HYPER boljši od Alchemyja. Slednji pa se sicer dobro odreže pri učenju uteži, saj je znal dobro ugotoviti, katere hipoteze so pravilne in katere napačne.

Poglavje 4

HYPER/N in povezava z Alchemyjem

Kot smo videli iz primerjave učenja hipotez s HYPER-jem in Alchemyjem v prejšnjem poglavju, se HYPER na čistih učnih podatkih nauči boljših hipotez kot Alchemy. Težava je le v tem, da se HYPER ne zna učiti iz šumnih podatkov, zato ni primeren za učenje iz realnih podatkov, kateri praviloma vedno vsebujejo določeno stopnjo šuma. V ta namen smo razvili HYPER/N - prilagojeno verzijo HYPER-ja, ki je zmožna tudi učenja na šumnih podatkih. Nato pa smo hipoteze, katere se HYPER/N nauči, podali algoritmu za učenje uteži v Alchemyju, za katerega smo v prejšnjem poglavju tudi ugotovili, da se dobro odreže pri uteževanju hipotez. Naučene uteži smo nato uporabili kot ocene hipotez; obdržali smo le hipoteze z najvišjimi utežmi, medtem ko smo hipoteze z nizkimi in negativnimi utežmi zavrgli. Natančnejši postopek je opisan v sledečih podpoglavjih.

4.1 HYPER/N

Preden začnemo opisovati HYPER/N, definirajmo šum, katerega HYPER/N obvlada. Recimo, da imamo učni problem, katerega želimo klasificirati v n razredov, podanih z množico $C = \{C_1, C_2, ..., C_n\}$. i-ti ($1 \leq i \leq n$) klasifikator (hipoteza), označimo ga s H_i, klasificira primere v 2 množici:

- če primer zadosti pogojem, ki jih postavlja hipoteza H_i, ga klasificiramo v razred C_i
- če primer ne zadosti pogojem, ki jih postavlja hipoteza H_i, ga klasificiramo v enega izmed $C \setminus C_i$ razredov; v katerega, pa s hipotezo H_i ne

moremo odločiti, za to potrebujemo neko drugo hipotezo H_j, $1 \leq j \leq n \wedge j \neq i$

Šum pa smatramo kot napačno klasificiran primer. Šumen je torej primer, ki se klasificira v razred C_i, čeprav spada v razred C_j, $j \neq i$.

Tako pojmovanje šuma smo izbrali zato, ker je bil HYPER/N prvotno namenjen učenju v vremenski domeni, s katerim se ukvarjamo na projektu XMEDIA (*http://www.x-media-project.org/*). V tej domeni želimo napovedovati radarske slike padavin, katere vsebujejo 5 razredov: *brez dežja*, *rahel dež*, *srednji dež*, *močan dež* in *toča*. Do šuma pride v tej domeni zaradi napak pri preverjanju okolice z radarjem pa tudi zaradi diskretizacije pri združevanju manjših celic s padavinami v večje. Odrazi pa se tako, da je celica označena z napačnim razredom (primer: celica je označena kot celica s srednjim dežjem, kljub temu, da je v njej padal rahel dež).

S šumom definiranim se lahko lotimo opisovanja HYPER/N-a. HYPER in HYPER/N sta v večini identična, tukaj bomo izpostavili le bistvene razlike med njima.

Prvi problem, katerega smo morali razrešiti, je bilo seveda obvladovanje šumnih primerov. To najlažje razložimo s pomočjo zgornje definicije šuma. Ko se učimo hipotezo H_i, imamo zanjo podani množici pozitivnih in negativnih učnih primerov. Če so vsi učni primeri čisti (nimamo šumnih primerov), bodo vsi pozitivni učni primeri spadali v razred C_i, vsi negativni pa v nek drug razred iz množice razredov $C \setminus C_i$; v katerega, pa nas pri učenju hipoteze H_i ne zanima, pomembno je le, da ne spadajo v razred C_i. Zaradi večje preglednosti označimo C_i s C_{poz} in $C \setminus C_i$ s C_{neg}.

Če pa so pozitivni in negativni učni primeri šumni, se zgodi, da je nekaj učnih primerov, kateri bi sicer spadali v množico C_{neg}, predstavljenih kot primeri iz množice C_{poz} in obratno. Pri učenju z originalnim HYPER-jem bi se v takem primeru zgodilo, da bi v nekem koraku HYPER generiral hipotezo, katera bi sicer pokrila vse čiste pozitivne učne primere, ne bi pa pokrila šumnih pozitivnih učnih primerov. Ker s tem ne bi bil izpolnjen pogoj popolnosti (pokriti vsi pozitivni učni primeri), bi se ta hipoteza zavrgla, čeprav bi se ista hipoteza, v primeru, da bi imeli na voljo čiste učne podatke, sprejela. HYPER/N ima zato pogoja popolnosti in konsistentnosti olajšana.

Uporabnik pred začetkom izvajanja poda pričakovano stopnjo šuma z literalom `expected_noise_level(X)`, kjer je $X \in [0.0, 1.0]$ in predstavlja pričakovan odstotek šumnih učnih primerov - 0.0 pomeni, da pričakujemo 0%, 1.0 pa, da pričakujemo 100% šuma. Nato pa se izračuna število pričakovanih šumnih primerov N_{noise} po enačbi 4.1:

$$N_{noise} = (N_{poz} + N_{neg}) \cdot X \tag{4.1}$$

kjer sta:

- N_{poz} število pozitivnih učnih primerov in
- N_{neg} število negativnih učnih primerov

Sedaj lahko definiramo pojma *približne popolnosti* in *približne konsistentnosti*:

- Hipoteza je približno popolna, če pokriva $N \geq N_{poz} - N_{noise}$ pozitivnih učnih primerov.
- Hipoteza je približno konsistentna, če pokriva $N \leq N_{noise}$ negativnih učnih primerov.

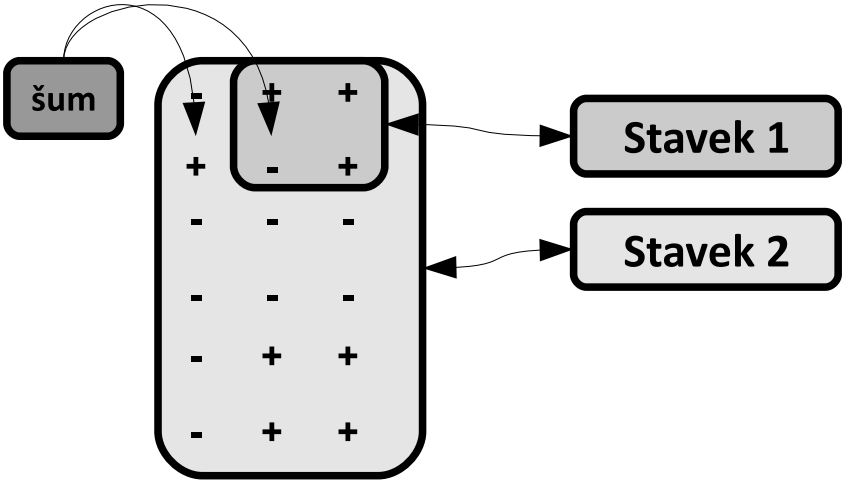
Če v HYPER/N-u nastavimo `expected_noise_level(0)` (torej je X v enačbi 4.1 enak 0; s tem HYPER/N-u povemo, da v podatkih ne pričakujemo šuma), sta definiciji približno popolne in približno konsistentne hipoteze enaki definicijama popolne in konsistentne hipoteze.

Če HYPER/N generira hipotezo, ki ni popolna, je pa približno popolna, je ne bo zavrgel, temveč jo bo obravnaval enako kot popolne hipoteze. Če za neko generirano hipotezo velja, da je približno popolna in približno konsistentna, jo bo ponudil kot končno rešitev.

Prvi izmed problemov, ki nastane pri tej spremembi, je poraba pomnilnika. Namreč že pri HYPER-ju pri učenju kompleksnih učnih problemov poraba pomnilnika močno naraste, v HYPER/N-u, kjer dovolimo tudi shranjevanje približno popolnih hipotez, pa se poraba pomnilnika še poveča. Zato smo v HYPER/N vgradili iskanje v snopu. Le-to se vklopi s pomočjo literala `use_beam_search(X)`. Če je $X = 0$, se uporabi neomejen prostor (v bistvu je še vedno omejen s količino fizičnega pomnilnika), če pa je $X = 1$, pa se uporabi iskanje v snopu. Velikost snopa se nastavlja z literalom `beam_size(Y)`. Privzeto se uporabi velikost snopa $Y = 200$, uporabnik pa po potrebi lahko ustrezno poveča velikost snopa.

Naslednji problem, na katerega naletimo, je povezan s hipotezami, ki so sestavljene iz več stavkov. Ta problem najlažje ilustriramo s sliko 4.1. Pozitivni učni primeri na tej sliki so označeni z znakom +, negativni pa z znakom -. Hipoteza je sestavljena iz dveh stavkov; prvi stavek je že izostren do te mere, da pokriva 3 pozitivne učne primere in enega negativnega, drugi stavek pa še ni izostren, zato pokriva vse učne primere. V učnih primerih sta šumna dva

podatka in sicer en pozitiven (zaradi šuma se v podatkih predstavlja kot negativen) in en negativen (zaradi šuma se v podatkih predstavlja kot pozitiven). Predpostavimo še, da v predznanju ni na voljo literalov, s pomočjo katerih bi lahko šumni negativni učni primer izločili iz množice pokritih primerov.



Slika 4.1: Primer težave, ki nastane pri izbiranju prvega stavka, ki pokrije negativen učni primer, za nadaljnjo izostritev.

HYPER pri izbiri stavka za izostritev izbere prvi stavek, ki pokriva vsaj en negativni primer. V našem primeru je to prvi stavek, ki pokriva šumni negativni učni primer, zato bi HYPER za izostritev izbral tega, kljub temu, da je očitno, da je drugi stavek obetavnejši za izostrevanje kot prvi. V praksi se ta princip izbiranja stavkov za izostritev pri učenju s šumnimi podatki izkaže za neuporabnega, saj je v določenih situacijah HYPER/N za izostrevanje neprestano izbiral prvi stavek, zato se nikoli ni naučil celotne hipoteze.

HYPER/N zato pri izbiranju stavka za izostritev raje ubere malce drugačen pristop. Za vsak stavek v hipotezi najprej pogleda število pozitivnih (p) in negativnih (n) učnih primerov, pokritih *samo* z opazovanim stavkom, nato pa izračuna oceno vsakega stavka po enačbi 4.2:

$$ocena = \frac{n}{p+1} \tag{4.2}$$

$p+1$ v imenovalcu potrebujemo zato, da se izognemo deljenju z ničlo v primeru, ko opazovani stavek ne pokrije nobenega pozitivnega učnega primera.

Za izostritev nato izbere stavek z najvišjo oceno, če pa imajo vsi enako oceno, pa izbere prvega. Prvi stavek v primeru na sliki 4.1 bi tako dobil oceno $\frac{1}{4}$, drugi pa $\frac{10}{9}$, zato bi se za izostritev izbral slednji. Zaradi tega se učenje bistveno pohitri, HYPER/N pa se nauči tudi hipoteze, katerih se prej ni mogel (predvsem iz šumnih podatkov).

Vendar ta izboljšava ne pride brez svoje cene. Posledica takega izbiranja stavkov za izostritev je ta, da se HYPER/N ne more naučiti rekurzije. Zakaj je temu tako, najlažje ilustriramo na enem izmed standardnih ILP problemov - iskanja poti v grafu [1]. Zanj imamo podano predznanje in učne primere, prikazane na sliki 4.2.

```
link(a,b).
link(a,c).
link(b,c).
link(b,d).
link(d,e).

ex(path(a,a,[a])).
ex(path(b,b,[b])).
ex(path(e,e,[e])).
ex(path(f,f,[f])).
ex(path(a,c,[a,c])).
ex(path(b,e,[b,d,e])).
ex(path(a,e,[a,b,d,e])).

nex(path(a,a,[])).
nex(path(a,a,[b])).
nex(path(a,a,[b,b])).
nex(path(e,d,[e,d])).
nex(path(a,d,[a,b,c])).
nex(path(a,c,[a])).
nex(path(a,c,[a,c,a,c])).
nex(path(a,d,[a,d])).
```

Slika 4.2: Učna domena za učenje poti v grafu

Rekurzivna hipoteza, ki se jo HYPER nauči iz učne domene na sliki 4.2 je:

```
path(A,A,[A]).
```

```
path(A,C,[A,B|E]) :-
  link(A,B),
  path(B,C,[B|E]).
```

HYPER/N pa po 20 izostritvah pride do hipoteze:

```
path(A,A,[A]).

path(B,A,[B,D|C]) :-
  link(B,D),
  path(D,E,F).
```

Prvi stavek v hipotezi je že pravilno izostren robni pogoj. Pokriva 4 pozitivne učne primere in nobenega negativnega, zato po enačbi 4.2 dobi oceno 0. Ker pa se pri ocenjevanju posameznega stavka uporabi le opazovani stavek in ne cela hipoteza, dokazovanje drugega stavka nikoli ne uspe, saj je njegov dokaz odvisen tudi od robnega pogoja, katerega pa ne podamo v dokaz. Zaradi tega drugi stavek ne pokrije nobenega pozitivnega in nobenega negativnega primera, posledično dobi oceno 0. Sedaj pa HYPER/N izbira med dvema stavkoma z enako oceno (oba 0), izbere prvega in s tem pokvari robni pogoj. Posledično se nikoli ne nauči iskane rekurzivne hipoteze.

Možna rešitev tega problema je ta, da pri izbiranju med stavki z enako oceno izberemo tistega, ki pokriva manj pozitivnih učnih primerov. V zgornjem primeru prvi stavek v hipotezi (robni pogoj) pokriva 4 pozitivne in nobenega negativnega učnega primera, medtem ko drugi stavek sam ne pokrije nobenega negativnega niti pozitivnega učnega primera. Zato bi za izostritev izbrali drugi stavek namesto prvega. Te ideje nam zaradi pomanjkanja časa še ni uspelo preizkusiti, zato to obstaja delo za v prihodnje.

Pri tem opazimo pomembno razliko s HYPER-jem. HYPER namreč obravnava hipotezo kot celoto (to je še posebej pomembno pri rekurziji), pri HYPER/N-u pa posamezni stavki niso odvisni od ostalih stavkov v hipotezi; če iz hipoteze odstranimo ostale stavke, bo opazovani stavek še vedno pokril enako množico učnih primerov.

Naslednji problem, katerega smo morali razrešiti, je bil pogoj za ustavljanje učenja. HYPER namreč preneha z učenjem, ko najde popolno in konsistentno hipotezo. Če pa se učimo iz šumnih podatkov, pa učenja ne smemo ustaviti pri prvi približno popolni in približno konsistentni hipotezi, saj ne vemo, ali je naučena hipoteza najboljša možna, ali pa bi se z nadaljnjim učenjem lahko naučili boljšo ali celo popolno in konsistentno hipotezo. HYPER/N ima zato ustavitveni pogoj sestavljen iz dveh pogojev:

- Če najde popolno in konsistentno hipotezo, ustavi učenje in to hipotezo ponudi kot rešitev. Če uporabnik z njo ni zadovoljen, se učenje lahko nadaljuje.

- Pred izvajanjem uporabnik poda maksimalno število dovoljenih izostritev in ko HYPER/N doseže to mejo, ustavi učenje in izpiše vse najdene hipoteze.

Kot smo nakazali že v prejšnjem odstavku, se HYPER/N lahko nauči več hipotez. Vse te hipoteze shranjuje v seznam potencialno dobrih hipotez. Ko pri učenju najde približno popolno in približno konsistentno hipotezo H_i, pogleda množico pozitivnih učnih primerov, pokritih s to hipotezo. Označimo to množico s P_{H_i}. Nato to množico primerja z množicami ostalih hipotez, katere so že shranjene v seznamu potencialno dobrih hipotez. Naj bodo te množice predstavljene s $P_H = \{P_{H_1}, P_{H_2}, ..., P_{H_n}\}; n < i$. Hipoteza H_i se bo dodala v seznam potencialno dobrih hipotez, če je izpolnjen pogoj iz enačbe 4.3:

$$\forall P_{H_k} : P_{H_i} \setminus P_{H_k} \neq \emptyset; P_{H_k} \in P_H \tag{4.3}$$

Hipoteza H_i se torej doda v seznam potencialno dobrih hipotez le, če njena množica pokritih pozitivnih učnih primerov P_{H_i} ni podmnožica ali enaka množici P_{H_k} nobene hipoteze H_k, ki se že nahaja v seznamu potencialno dobrih hipotez. V primeru, da se hipoteza H_i doda v seznam potencialno dobrih hipotez, moramo iz seznama odstraniti vse hipoteze H_k ($k \neq i$), katerih množice pokritih pozitivnih učnih primerov P_{H_k} so podmnožice ali enake množici P_{H_i}. S tem dosežemo, da je seznam potencialno dobrih hipotez kratek, pokrije pa kar se da veliko pozitivnih učnih primerov.

4.2 Ocenjevanje naučenih hipotez z Alchemyjem

Če HYPER/N ni našel popolne in konsistentne hipoteze, nam (ob dovolj visoko nastavljenem pričakovanem odstotku šuma) izpiše seznam potencialno dobrih hipotez. Iz tega seznama bi radi izločili vse slabe in obdržali vse dobre hipoteze, da pa bi to storili, potrebujemo ocene teh hipotez. Za ocenjevanje hipotez lahko uporabimo učenje uteži v Alchemyju, nato pa obdržimo hipoteze z najvišjimi utežmi, tiste z najnižjimi pa zavržemo.

Za učenje uteži z Alchemyjem moramo hipoteze, zapisane v Prologovi sintaksi, najprej pretvoriti v Alchemyjevo sintakso. Ker so vse hipoteze, naučene s HYPER/N-om, oblike:

```
goal :-
  pred1,
  pred2,
  ...
  predn.
```

jih preprosto pretvorimo v Alchemyjevo sintakso z uporabo enačbe 4.4:

$$pred_1 \wedge pred_2 \wedge \cdots \wedge pred_n \Rightarrow goal \tag{4.4}$$

Pri tem se moramo zavedati, da Alchemy ne pozna seznamov, zato moramo vse sezname, ki se pojavljajo kot argumenti literalov v telesu hipoteze, razbiti na posamezne spremenljivke. Primer: pri učenju v vremenski domeni smo za pohitritev učenja koordinate X,Y celic pisali v obliki seznama (`rain([X,Y],Time)`), pri učenju uteži z Alchemyjem, pa smo morali ta seznam razbiti na dve ločeni spremenljivki ($rain(X, Y, Time)$).

Pri pretvarjanju ravno tako razbijemo hipoteze, sestavljene iz n stavkov v n hipotez, sestavljenih iz enega stavka. Namreč pri testiranju na praktičnih primerih se je izkazalo, da se HYPER/N velikokrat nauči hipoteze sestavljene iz večih stavkov, kjer je en stavek dober, ostali pa slabi. S kombiniranjem stavkov iz različnih hipotez lahko tako dobimo veliko boljšo hipotezo. Prav tako zna Alchemy združiti enake stavke in se za njih naučiti enotno utež, zato je izhodna datoteka veliko bolj pregledna. Končni rezultat takega početja je teoretično lahko slab, saj je lahko en učni primer pokrit z večimi stavki (HYPER/N namreč teži k temu, da se množice pokritih učnih primerov s posameznimi stavki v hipotezi ne prekrivajo med sabo), vendar kakor se je izkazalo v praksi, tega problema načeloma nismo imeli.

Za hitrejše pretvarjanje hipotez iz Prologove v Alchemyjevo sintakso smo napisali program v Javi, ki obsega približno 700 vrstic. V tem programu moramo definirati literale in tipe spremenljivk, ki nastopajo v teh literalih, program pa nato avtomatsko pretvori izhodne datoteke iz HYPER/N-a v učne datoteke za Alchemy.

Ko imamo učno datoteko, jo podamo Alchemyjevem programu za učenje uteži (*learnwts*) skupaj z enakimi učnimi podatki, kot smo jih podali že v HYPER/N-u (le da so seveda v Alchemyjevi sintaksi). Ko se Alchemy nauči uteži za posamezne hipoteze, pa moramo na podlagi teh uteži odločiti, katere

hipoteze bomo obdržali in katere zavrgli. Ena možna varianta, kako to narediti, je ta, da poiščemo najvišjo utež u, nato pa obdržimo vse hipoteze H_i z utežjo u_i, za katere velja pogoj iz enačbe 4.5:

$$u_i \geq u - 1.0 \tag{4.5}$$

Kot bomo videli, se je ta postopek v večini praktičnih primerov izkazal za dobrega, nekaj težav je bilo le pri učenju na realnih podatkih. Seveda pa je možnih postopkov še več, odvisni pa so predvsem od predznanja, ki ga imamo o učni domeni. Če naprimer vemo, da bo končna hipoteza sestavljena iz n stavkov, potem izmed uteženih hipotez obdržimo le n hipotez z najvišjimi utežmi ipd.

4.3 Testiranje naveze HYPER/N - Alchemy na sintetičnih podatkih

Ker HYPER/N in Alchemy ne obvladata učenja rekurzivnih hipotez, smo bili primorani izpustiti testiranje na večini standardnih ILP problemov. Zato smo testirali samo na dveh sintetičnih učnih domenah - družinske relacije in vremenska domena. Testiranje učenja na sintetičnih podatkih ima to prednost, da zlahka ocenimo, če je naučena hipoteza pravilna. Pravilna je namreč v primeru, da je identična hipotezi, s katero smo podatke generirali. Poleg pravilnosti naučenih hipotez smo želeli tudi preveriti, kako je HYPER/N občutljiv na previsoko nastavljeno pričakovano stopnjo šuma. Za sintetične podatke imamo namreč natančen podatek o številu šumnih primerov, zato ni težko ugotoviti koliko lahko pretiravamo pri nastavljanju pričakovane stopnje šuma, da se HYPER/N še nauči pravilno hipotezo.

Za generiranje sintetičnih podatkov smo napisali program v Javi, ki obsega približno 1000 vrstic, vanj pa podamo definicije literalov in spremenljivk, ki nastopajo v njih, hipotezo, po kateri želimo generirati podatke ter odstotek šumnih podatkov, katere naj program generira. Učni primer pa zašumimo tako, da naključno zamenjamo njegov razred - pozitivni učni primer označimo kot negativnega in obratno.

Najprej smo testirali učenje družinskih relacij. V ta namen smo s hipotezo

```
hasdaughter(X) :-
  parent(X,Y),
  female(Y).
```

generirali domene s 1000 učnimi primeri in različnimi stopnjami šuma.

Najprej smo testirali učenje na učni domeni brez šuma, kjer smo postopoma zviševali pričakovano stopnjo šuma. HYPER/N se je za vsako pričakovano stopnjo šuma (od 0 do 100%) naučil pravilno hipotezo. Razlog za to je ta, da popolno in konsistentno hipotezo najde zelo hitro (po dveh izostritvah), še preden se iskanje zaradi previsoko nastavljene pričakovane stopnje šuma usmeri v napačno vejo drevesa izostritev.

Nato smo generirali učno domeno z 8.3% šuma. Rezultati učenja so bili sledeči:

- Če smo pričakovano stopnjo šuma nastavili na 8.5%, se je HYPER/N naučil le eno hipotezo in ta je bila tudi pravilna.
- Če smo pričakovano stopnjo šuma nastavili na 10%, se je HYPER/N naučil 3 hipoteze, vsaka je bila sestavljena iz dveh stavkov, od tega je prvi stavek predstavljal pravilno hipotezo, drugi stavek pa neko naključno odvisnost. Pri uteževanju z Alchemyjem je dobil pravilni stavek utež 1.7781, stavek z drugo najvišjo utežjo pa 0.0729448. Z upoštevanjem pogoja iz enačbe 4.5, zavržemo tako vse stavke, razen najboljšega.
- Če smo pričakovano stopnjo šuma nastavili na 15%, se je HYPER/N naučil dve hipotezi. Prva je sestavljena iz dveh stavkov, od njiju prvi predstavlja pravilno hipotezo, drugi stavek in druga hipoteza, ki vsebuje samo en stavek, pa predstavljata neke naključne odvisnosti. Pri uteževanju z Alchemyjem je dobil pravilni stavek utež 1.68991, stavek z drugo najvišjo utežjo pa 0.0994732. Z upoštevanjem pogoja iz enačbe 4.5 ponovno obdržimo le najboljši stavek, ostala dva pa zavržemo.

Če smo z enakimi podatki testirali še učenje hipoteze z Alchemyjem, pa se je le-ta naučil sledečo hipotezo:

$$parent(b, a) \wedge female(a) \wedge \neg parent(a, a) \Rightarrow hasdaughter(b)$$

Naučena hipoteza je načeloma pravilna, vendar nas moti pogoj $\neg parent(a, a)$. Z njim namreč zahtevamo da oseba ni sama sebi starš. Sicer je res, da je tehnično ta hipoteza pravilna, vendar ni ravno najbolj intuitivna.

Kot vidimo se je pri učenju družinskih relacij naveza HYPER/N in Alchemy odrezala zelo dobro (medtem ko se učenje samo z Alchemyjem ni tako dobro). Poglejmo si, kako se je obnesla na sintetičnih vremenskih podatkih. S hipotezo

```
rainT1([X,Y],Time) :-
```

```
  rain([X1,Y1],Time),
  n_neighbor([X1,Y1],[X,Y]),
  n_wind(Time).

rainT1([X,Y],Time) :-
  rain([X1,Y1],Time),
  e_neighbor([X1,Y1],[X,Y]),
  e_wind(Time).
```

smo generirali domene s 128 učnimi primeri (64 učnih primerov za vsak stavek v hipotezi) in različnimi stopnjami šuma. Podajmo najprej interpretacijo zgornje hipoteze. Prvi stavek se interpretira kot: *če se v celici s koordinatami $X1$,$Y1$ v času $Time$ nahaja dež, obenem pa v tistem času piha severni veter, se bo dež v naslednji časovni enoti nahajal v celici s koordinatami X,Y, katera je severni sosed celice s koordinatami $X1$,$Y1$.* Podobno se interpretira tudi drugi stavek, le da v času $Time$ piha vzhodni veter, padavine pa se premaknejo v vzhodno celico.

Opomba: literal `rainT1([X,Y],Time)` se interpretira kot `rain([X,Y],Time+1)`. Tako predstavitev smo izbrali z namenom pohitritve učenja, ker bi sicer kot predznanje morali podati še relacijo naslednika:

```
succ(Time,Time1) :-
  Time1 is Time + 1.
```

Z vsakim dodanim literalom v množico predznanja pa se učenje s HYPER/N-om upočasni. Literala `rainT1` pa ne podamo kot predznanje, zato ga HYPER/N pri učenju ne uporabi.

Iz predznanja smo izločili tudi polovico relacij soseda, zaradi ekvivalenc, ki veljajo med njimi:

```
n_neighbor(A,B) = s_neighbor(B,A)
ne_neighbor(A,B) = sw_neighbor(B,A)
e_neighbor(A,B) = w_neighbor(B,A)
se_neighbor(A,B) = nw_neighbor(B,A)
```

Učenje se je s tem bistveno pohitrilo.

Najprej smo testirali učenje na učni domeni brez šuma, kjer smo postopoma zviševali pričakovano stopnjo šuma. Rezultati učenja so bili sledeči:

- Če smo pričakovano stopnjo šuma nastavili na 0%, se je HYPER/N naučil pravilno hipotezo.

- Če smo pričakovano stopnjo šuma nastavili na 5%, se je HYPER/N naučil pravilno hipotezo.

- Če smo pričakovano stopnjo šuma nastavili na 10%, se je HYPER/N naučil pravilno hipotezo.

- Če smo pričakovano stopnjo šuma nastavili na 15%, se HYPER/N pri 2000 dovoljenih izostritvah in velikosti snopa 300 ni naučil pravilne hipoteze.

Iz teh rezultatov lahko predpostavimo, da se lahko pri določanju pričakovane stopnje šuma zmotimo za približno 10%, da se bo HYPER/N še vedno sposoben relativno hitro naučiti dobro hipotezo.

Nato smo generirali učno domeno s približno 6% šuma. Rezultati učenja so bili sledeči:

- Če smo pričakovano stopnjo šuma nastavili na 6%, se je HYPER/N naučil le eno hipotezo in ta je bila tudi pravilna.

- Če smo pričakovano stopnjo šuma nastavili na 10%, se HYPER/N sicer ni naučil hipoteze, ki bi bila dobra kot celota, vendar bi se s kombiniranjem stavkov iz posameznih naučenih hipotez dalo sestaviti pravilno hipotezo. Pri uteževanju sta tudi oba stavka, ki sestavljata pravilno hipotezo, dobila najvišje uteži - 3.89983 in 3.5338. Stavek s tretjo najvišjo utežjo pa je dobil utež 2.15365. Z upoštevanjem pogoja iz enačbe 4.5 bi tako obdržali pravilna stavka, nepravilne pa bi zavrgli.

- Če smo pričakovano stopnjo šuma nastavili na 15%, se HYPER/N ponovno ni naučil hipoteze, ki bi bila dobra kot celota, toda tokrat povsem pravilne hipoteze nismo mogli sestaviti niti s kombiniranjem stavkov iz posameznih naučenih hipotez. Pri uteževanju z Alchemyjem sta dobila dva stavka najvišje uteži - 2.87707 in 2.48348. Stavek z drugo najvišjo utežjo je bil pravilen, vendar stavek z najvišjo pa ne povsem; manjkal mu je namreč podatek o smeri vetra. Stavek s tretjo najvišjo utežjo pa je dobil utež 1.58649, zato smo ga z upoštevanjem pogoja iz enačbe 4.5 že zavrgli. Hipoteza, ki jo sestavimo iz preostalih dveh stavkov, tako ni povsem pravilna, vendar je vseeno zadovoljiva.

S temi rezultati se tako potrdi zgornja predpostavka. HYPER/N se bo v navezi z Alchemyjem relativno hitro naučil dobre hipoteze, če se pri določanju pričakovane stopnje šuma zmotimo za približno 10%.

4.4 Testiranje naveze HYPER/N - Alchemy na realnih podatkih

Testirali smo tudi učenje z navezo HYPER/N - Alchemy na realnih vremenskih podatkih, katere uporabljamo v sklopu projekta XMEDIA. Vremenski podatki so predstavljeni z radarskimi slikami padavin (zajete na 10 minut) in sinoptičnimi podatki (temperatura, vlaga, hitrost in smer vetra, ... v najboljšem primeru zajeti na eno uro) iz vremenskih merilnih postaj. Za testiranje učenja s HYPER/N in Alchemyjem smo uporabili le radarske slike padavin in podatek o smeri in hitrosti gibana padavin, učili pa smo se enakih hipotez, kot v prejšnjem podpoglavju s sintetičnimi podatki. Naš (dolgoročni) cilj je ta, da bi se naučili hipotezo, ki bi, ob podani radarski sliki in sinoptičnih podatkih, kar se da točno napovedala naslednjo radarsko sliko. Nato pa bi z več zaporednimi napovedmi (kjer bi vsaka napovedana radarska slika služila kot predznanje za napoved naslednje) simulirali gibanje dežja in predvsem nastajanje nevihtnih celic v roku dveh ur.

Pri podatkih o smeri in hitrosti padavin velja opomniti, da to niso enaki podatki kot smer in hitrost vetra, pridobljeni iz sinoptičnih podatkov. Namreč, izkazalo se je, da se smer in hitrost vetra le redko ujemata z dejansko smerjo in hitrostjo gibanja padavin, zato smo le-te izračunali iz zaporedja radarskih slik.

Radarske slike imajo resolucijo 301 * 401 piksel, vsak piksel pa predstavlja področje v naravi z dimenzijami 1 km * 1 km. Vsak piksel na radarski sliki je lahko ene izmed šestih barv, katere pomenijo sledeče:

- črna barva: območje izven dosega radarja
- bela barva: ni padavin
- zelena barva: rahel dež
- rumena barva: srednji dež
- oranžna barva: močan dež
- rdeča barva: nevihta s točo

V našem učnem problemu nismo razlikovali med posameznimi jakostmi padavin, vse jakosti so bile predstavljene z enim samim literalom - `rain([X,Y],Time)` oziroma `rainT1([X,Y],Time)`, ko gre za napoved.

Če bi pri učenju obravnavali vsak piksel na radarski sliki posebej, bi s tem dobili več kot 100.000 učnih primerov, HYPER/N pa postane neučinkovit že pri učnih domenah z nekaj 100 učnimi primeri. Zaradi tega smo celotno radarsko sliko razdelili na celice dimenzij 5*5 pikslov, iz te mreže pa smo vzeli le podmnožico podatkov - mrežo 10*10 celic, tako da smo dobili 100 učnih primerov. Posamezno celico smo proglasili za deževno, če je bila večina pikslov v njej „deževnih“ (tj. bili so zelene, rumene, oranžne ali rdeče barve), sicer pa smo jo proglasili za celico brez dežja.

Zaradi neučinkovitosti HYPER/N pri velikih učnih domenah smo se naenkrat lahko učili hipoteze le iz enega para radarskih slik (prva slika služi kot predznanje, druga pa kot pozitivni in negativni učni primeri), zato smo najboljše hipoteze določali z glasovanjem; za vsak par radarskih slik smo se s HYPER/N naučili hipoteze, z Alchemyjem uteži za te hipoteze, nato pa smo z upoštevanjem pogoja iz enačbe 4.5 obdržali le hipoteze z najvišjimi utežmi. To smo ponovili za več parov, za najboljšo hipotezo pa smo proglasili tisto, katero smo se z navezo HYPER/N - Alchemy največkrat naučili. Pri izenačenjih smo za boljšo vzeli hipotezo, kateri je Alchemy večkrat določil najvišjo utež. V rezultatih so najvišje uteži podčrtane.

Slika 4.3: Par radarskih slik, uporabljenih kot učni primer. Z rdečim kvadratom je označena podmnožica celic, uporabljenih za učenje. Z leve slike dobimo podatke za predznanje, z desne pa pozitivne in negativne učne primere.

Najprej smo testirali učenje hipotez na 9 učnih primerih - vremensko dogajanje 13.5.2004 od 15:00 do 16:30 - ko je bilo gibanje padavin izrazito jugozahodno. Primer učnega primera (par radarskih slik), uporabljenega za učenje, vidimo na sliki 4.3. Ker v realnih podatkih nimamo natančnega podatka o stopnji šuma, smo morali le-tega ugibati. Za ta konkretni učni primer

smo nastavili pričakovano stopnjo šuma na 15%. Najpogosteje smo se naučili sledeče hipoteze:

```
rainT1([X,Y],Time) :-
  rain([X1,Y1],Time),
  ne_neighbor([X,Y],[X1,Y1]).
```

Naučili smo se 6x, uteži pa so: **5.49916**, **2.23667**, **2.97956**, **2.73231**, **1.48521**, **3.96011**. Interpretira se kot: *padavine se premaknejo jugozahodno.*

```
rainT1([X,Y],Time) :-
  rain([X,Y],Time).
```

Naučili smo se 6x, uteži pa so: **4.77962**, **4.91444**, **1.90331**, **2.86057**, **1.3751**, **3.02375**. Interpretira se kot: *padavine ostanejo na istem mestu.*

```
rainT1([X,Y],Time) :-
  rain([X1,Y1],Time),
  e_neighbor([X,Y],[X1,Y1]).
```

Naučili smo se 5x, uteži pa so: **3.2182**, **3.95952**, **4.00705**, **2.03303**, **1.95526**. Interpretira se kot: *padavine se premaknejo zahodno.*

```
rainT1([X,Y],Time) :-
  rain([X1,Y1],Time),
  n_neighbor([X,Y],[X1,Y1]).
```

Naučili smo se 3x, uteži pa so: **4.07123**, **1.66927**, **1.60898**. Interpretira se kot: *padavine se premaknejo južno.*

Te interpretacije hipotez mogoče niso povsem intuitivne; prvo hipotezo smo namreč interpretirali kot premik padavin jugozahodno, kljub temu, da je bil uporabljen literal `ne_neighbor`, vendar naj ponovno opozorimo, da smo polovico relacij soseda opustili iz modela zaradi ekvivalence med njimi, zato ima prva hipoteza enak pomen, kot hipoteza:

```
rainT1([X,Y],Time) :-
  rain([X1,Y1],Time),
  sw_neighbor([X1,Y1],[X,Y]).
```

Opazimo tudi, da v hipotezah ni uporabljena smer vetra. Razlog za to je ta, da za vse učne podatke velja enaka smer vetra, zato je uporaba tega podatka (vsaj v kontekstu učnih podatkov) povsem odveč.

Prva hipoteza opisuje premikanje padavin kot bi ga opisal tudi človek, če bi videl zaporedje učnih radarskih slik. Ostale naučene hipoteze pa so bolj posledica tega, da smo za učenje vzeli le podmnožico vseh celic na radarski sliki vključno z relativno visoko nastavljeno pričakovano stopnjo šuma. HYPER/N in Alchemy sta imela zato precej lokalen pogled na celotno vremensko dogajanje; predpostavljamo da se z bolj globalnim pogledom (celotno radarsko sliko) 2., 3. in 4. hipoteze ne bi naučili tako pogosto.

Nato smo enak test ponovili na 9 učnih primerih dne 28.5.2004 od 16:00 do 17:30. Na tem zaporedju radarskih slik se padavine premikajo sicer severovzhodno vendar tako počasi, da je med dvema radarskima slikama premik komajda opazen. Namesto smeri vetra zato uporabimo literal `no_wind(Time)`. Pričakovano stopnjo šuma smo tudi tukaj nastavili na 15%. Naučimo se sledečih hipotez:

```
rainT1([X,Y],Time) :-
  rain([X,Y],Time).
```

Naučili smo se 8x, uteži pa so: **4.64113**, **4.76356**, **2.72921**, **5.61354**, **6.80215**, **2.44811**, **3.92627**, **5.32996**. Interpretira se kot: *padavine ostanejo na istem mestu.*

```
rainT1([X,Y],Time) :-
  rain([X1,Y1],Time),
  n_neighbor([X1,Y1],[X,Y]).
```

Naučili smo se 4x, uteži pa so: **4.25445**, **3.63786**, **3.26237**, **3.99355**. Interpretira se kot: *padavine se premaknejo severno.*

```
rainT1([X,Y],Time) :-
  rain([X1,Y1],Time),
  e_neighbor([X1,Y1],[X,Y]).
```

Naučili smo se 2x, uteži pa sta: **2.91087**, **3.69755**. Interpretira se kot: *padavine se premaknejo vzhodno.*

```
rainT1([X,Y],Time) :-
  rain([X1,Y1],Time),
  ne_neighbor([X1,Y1],[X,Y]).
```

Naučili smo se 1x, utež pa je: **3.29301**. Interpretira se kot: *padavine se premaknejo severovzhodno.*

Z rezultati smo zadovoljni, saj sta HYPER/N in Alchemy v večini primerov ugotovila, da padavine praktično ostanejo na mestu.

Enak test smo ponovili še na 6 učnih primerih dne 13.6.2004 od 14:30 do 15:30, kjer gre, tako kot v prejšnjem testu, za premikanje padavin severovzhodno, le da je v tem primeru premikanje hitrejše in izrazitejše, zato pričakujemo, da se bosta HYPER/N in Alchemy največkrat naučila hipotezo, ki opisuje premikanje padavin severovzhodno. Rezultati so sledeči:

```
rainT1([X,Y],Time) :-
  rain([X1,Y1],Time),
  ne_neighbor([X1,Y1],[X,Y]).
```

Naučili smo se 6x, uteži pa so: **3.18348**, **2.91399**, **3.25174**, **4.23608**, **2.5554**, **4.90089**. Interpretira se kot: *padavine se premaknejo severovzhodno.*

```
rainT1([X,Y],Time) :-
  rain([X1,Y1],Time),
  n_neighbor([X1,Y1],[X,Y]).
```

Naučili smo se 2x, uteži pa sta: **3.93904**, **1.69173**. Interpretira se kot: *padavine se premaknejo severno.*

```
rainT1([X,Y],Time) :-
  rain([X1,Y1],Time),
  e_neighbor([X1,Y1],[X,Y]).
```

Naučili smo se 2x, uteži pa sta: **2.12455**, **3.53767**. Interpretira se kot: *padavine se premaknejo vzhodno.*

Tudi tokrat smo bili z rezultati zadovoljni, saj sta se HYPER/N in Alchemy za vse učne primere naučila pričakovano hipotezo.

Za zadnji test pa smo vzeli po en učni primer iz prejšnjih treh testov in jih združili v en sam učni primer. Ker smo s tem učno domeno povečali na 300 učnih primerov, je bilo učenje s HYPER/N-om zelo počasno (izostritev določenih stavkov je trajala več kot 1 uro). Pohitrili smo ga lahko s tem, da smo omejili največjo velikost hipotez na 4 literale. HYPER/N in Alchemy naj bi se naučila hipotezo, sestavljeno iz treh stavkov, kjer bi vsak stavek opisoval enega od premikov padavin (premik jugozahodno, premik severovzhodno in padavine ostanejo na mestu). Tudi v tem primeru smo pričakovano stopnjo šuma nastavili na 15%. Rezultat učenja na tej učni domeni je sledeč:

```
rainT1([X,Y],Time) :-
  sw_wind(Time),
  rain([X1,Y1],Time),
  ne_neighbor([X,Y],[X1,Y1]).
```

Utež: **5.20565**. Interpretira se kot: *če piha jugozahodni veter, se padavine premaknejo jugozahodno.*

```
rainT1([X,Y],Time) :-
  no_wind(Time),
  rain([X,Y],Time).
```

Utež: **2.70404**. Interpretira se kot: *če veter ne piha, se padavine ne premaknejo.*

```
rainT1([X,Y],Time) :-
  rain([X,Y],Time).
```

Utež: **2.69797**. Interpretira se kot: *padavine ostanejo na istem mestu.*

```
rainT1([X,Y],Time) :-
  ne_wind(Time),
  rain([X1,Y1],Time),
  ne_neighbor([X1,Y1],[X,Y]).
```

Utež: **2.54816**. Interpretira se kot: *če piha severovzhodni veter, se padavine premaknejo severovzhodno.*

Najprej opazimo, da je za razliko od prejšnjih testov tokrat HYPER/N v hipoteze vstavil tudi podatke o smeri vetra. Poleg tega pa žal naletimo tudi na težavo. Namreč če smo pozorni na naučene uteži, vidimo da bi z upoštevanjem enačbe 4.5 zavrgli 2., 3. in 4. hipotezo in obdržali samo prvo, kljub temu, da za opis vremenskega dogajanja v učnih primerih nujno potrebujemo vsaj še 2. in 4. hipotezo. Namesto uporabe pogoja iz enačbe 4.5 je zato pri filtriranju bolje uporabiti predznanje, če nam je le-to na voljo. V tem primeru recimo vemo, da se hočemo naučiti tri hipoteze, poleg tega vemo tudi, da mora vsaka hipoteza vsebovati podatek o smeri vetra, zato bi obdržali le 3 hipoteze z najvišjimi utežmi, katere vsebujejo podatek o smeri vetra. Vendar težava je v tem, da v večini učnih problemov takega predznanja običajno nimamo, zato bi nam bolj koristil nek robusten algoritem za filtriranje hipotez. Le-tega trenutno še ne moremo podati; razvoj takega algoritma ostaja delo za v bodoče.

Poglavje 5

Sklepne ugotovitve

S HYPER/N-om, povezavo le-tega z učenjem uteži v Alchemyju in rezultati, ki smo jih s tem dobili, smo bili zadovoljni kljub temu, da smo pri tem naleteli na nekaj težav.

Prva težava je ta, da se HYPER/N za razliko od HYPER-ja ne more več naučiti rekurzije. Idejo za možno rešitev že imamo, vendar tudi če ta ne bo delovala, izguba zmožnosti učenja rekurzije ni tako huda. Predpostavljamo namreč, da se pri učenju v vremenski domeni, za kar je bil HYPER/N prvotno namenjen, ne bi naučili rekurzivnih hipotez niti, če bi bil HYPER/N tega sposoben. Poleg tega Alchemy ravno tako ne obvlada rekurzivnih hipotez, zato bi imeli težave tudi pri učenju uteži za take hipoteze. Druga, malce večja težava pa je ta, da HYPER/N postane neučinkovit, če mu podamo učno domeno z velikim številom učnih primerov. Neučinkovitost lahko obvladujemo s predznanjem o učni domeni (recimo omejitev maksimalne dolžine hipoteze), prav tako pa so možne še konkretne optimizacije postopka za dokazovanje hipotez. Tretja težava, katero moramo tudi še razrešiti, pa je filtriranje hipotez na podlagi naučenih uteži. Filtriranje hipotez s pomočjo preprostega pogoja iz enačbe 4.5 se je na realnih podatkih izkazalo za pomanjkljivo, zato moramo razviti še robusten algoritem za filtriranje hipotez. Če imamo na voljo ustrezno predznanje o naučenih hipotezah (recimo dolžina hipoteze, pojavljanje določenega literala v hipotezi, ...), je seveda smiselno uporabiti tudi to.

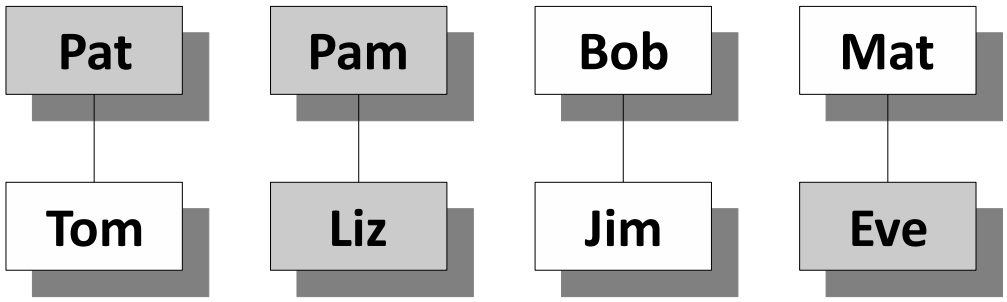
Navkljub tem težavam pa HYPER/N še vedno obdrži dve pomembni prednosti v primerjavi z ostalimi metodami strojnega učenja. Prva prednost je možnost relacijskega učenja, ki se izkaže še za posebej pomembno v vremenski domeni, saj v njej obstaja veliko relacij med podatki. Druga prednost pa je možnost podajanja specifičnega predznanja v naravni obliki, česar pri ostalih metodah strojnega učenja ne moremo narediti oziroma lahko naredimo

le v zelo omejeni obliki. Tudi ta prednost se pri učenju v vremenski domeni izkaže za zelo uporabno, saj se s podajanjem ustreznega predznanja učenje bistveno pohitri, naučene hipoteze pa so dosti boljše, kot če tega predznanja ne bi uporabili.

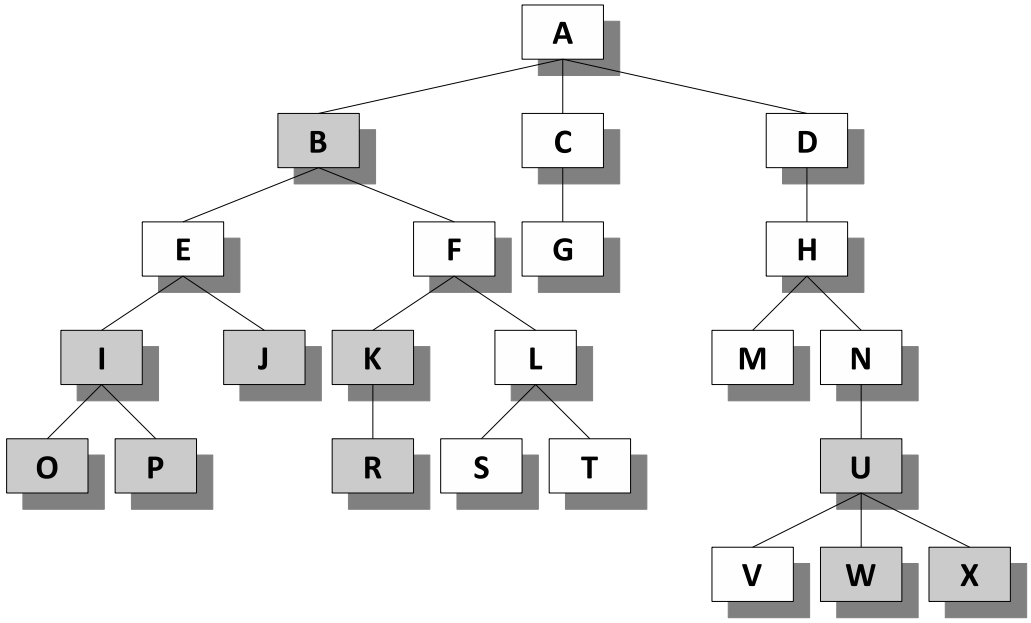
Glede na te prednosti, katere ima HYPER/N in glede na to, da imamo vse težave s HYPER/N-om in Alchemyjem identificirane ter za večino tudi že ideje, kako jih razrešiti, smo mnenja, da je nadaljnji razvoj HYPER/N-a obetaven.

Dodatek A

Učne domene



Slika A.1: Shema preprostega učne domene družinskih relacij. Sivi pravokotniki predstavljajo osebe ženskega, beli osebe moškega spola, povezave med njimi pa relacijo parent(A,B).

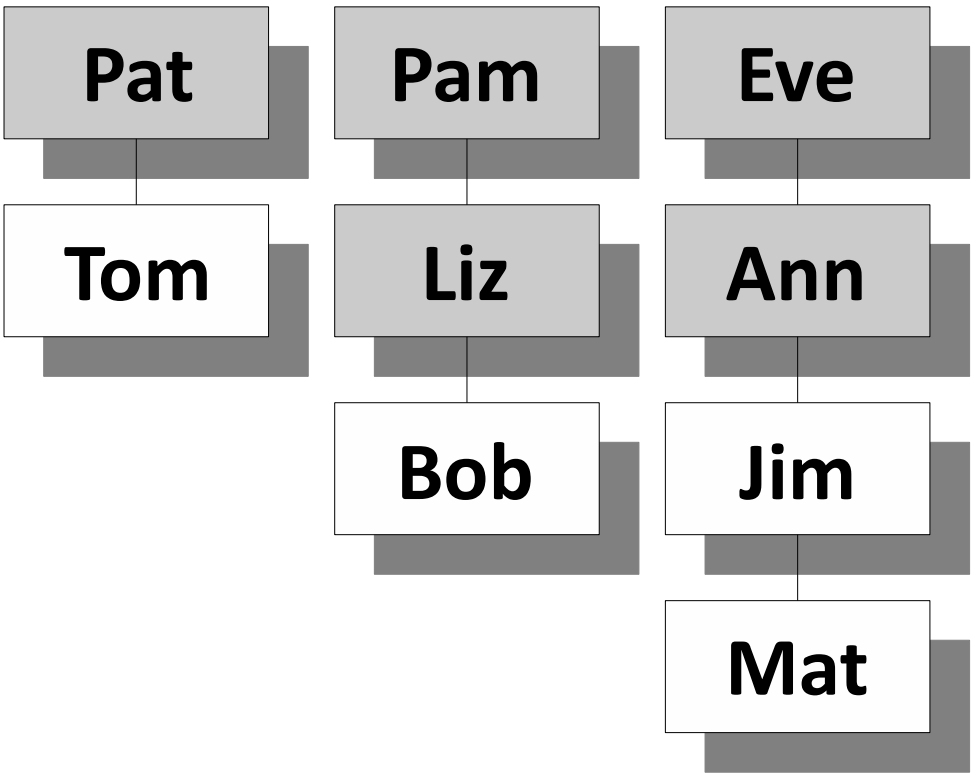


Slika A.2: Shema večje učne domene družinskih relacij. Sivi pravokotniki predstavljajo osebe ženskega, beli osebe moškega spola, povezave med njimi pa relacijo parent(A,B).

```
ex(predecessor(pam,bob)).
ex(predecessor(pam,jim)).
ex(predecessor(tom,ann)).
ex(predecessor(tom,jim)).
ex(predecessor(tom,liz)).

nex(predecessor(liz,bob)).
nex(predecessor(pat,bob)).
nex(predecessor(pam,liz)).
nex(predecessor(liz,jim)).
nex(predecessor(liz,liz)).
```

Slika A.3: Učni primeri za problem učenja prednika, podani v Prologovi sintaksi. Predznanje je podano na sliki 2.2



Slika A.4: Shema preproste učne domene za učenje prednika. Sivi pravokotniki predstavljajo osebe ženskega, beli osebe moškega spola, povezave med njimi pa relacijo parent(A,B).

```
publication(title1,bart).
publication(title1,ada).
publication(title2,becca).
publication(title2,ada).
publication(title3,betty).
publication(title3,alan).
publication(title4,bill).
publication(title4,alan).
publication(title5,bob).
publication(title5,alex).
publication(title6,carl).
publication(title6,alex).
publication(title7,carol).
publication(title7,alice).
publication(title8,cathy).
publication(title8,alice).
publication(title9,charles).
publication(title9,andy).
publication(title10,claire).
publication(title10,andy).

inPhase(bart,pre_quals).
inPhase(becca,pre_quals).
inPhase(betty,post_quals).
inPhase(bill,post_quals).
inPhase(bob,post_quals).
inPhase(carl,post_quals).
inPhase(carol,post_quals).
inPhase(cathy,post_quals).
inPhase(charles,post_quals).
inPhase(claire,post_quals).

hasPosition(ada,faculty).
hasPosition(alan,faculty).
hasPosition(alex,faculty).
hasPosition(alice,faculty).
hasPosition(andy,faculty_emeritus).
```

Slika A.5: Učni primer, ki je priložen Alchemyju, podan v Prologovi sintaksi - predznanje.

```
ex(advisedBy(bart,ada)).
ex(advisedBy(becca,ada)).
ex(advisedBy(betty,alan)).
ex(advisedBy(bill,alan)).
ex(advisedBy(bob,alex)).
ex(advisedBy(carl,alex)).
ex(advisedBy(carol,alice)).
ex(advisedBy(cathy,alice)).
ex(advisedBy(charles,andy)).
ex(advisedBy(claire,andy)).
nex(advisedBy(bart,cathy)).
nex(advisedBy(alice,andy)).
nex(advisedBy(bill,bill)).

ex(professor(ada)).
ex(professor(alan)).
ex(professor(alex)).
ex(professor(alice)).
ex(professor(andy)).
nex(professor(betty)).
nex(professor(charles)).

ex(student(bart)).
ex(student(becca)).
ex(student(betty)).
ex(student(bill)).
ex(student(bob)).
ex(student(carl)).
ex(student(carol)).
ex(student(cathy)).
ex(student(charles)).
ex(student(claire)).
nex(student(alan)).
```

Slika A.6: Učni primer, ki je priložen Alchemyju, podan v Prologovi sintaksi - učni primeri.

Literatura

[1] I. Bratko, "Inductive Logic Programming" v *Prolog Programming for Artificial Intelligence*, Pearson Education, Addison-Wessley, 2001, str. 484-519.

[2] P. Domingos, M. Richardson, "Markov Logic" v *Introduction to Statistical Relational Learning*, MIT Press, 2007, str. 339-371.

[3] S. Kok, P. Domingos, "Learning the Structure of Markov Logic Networks" v *Proceedings of the Twenty-Second International Conference on Machine Learning*, ACM Press, 2005, str. 441-448.

[4] D. Lowd, P. Domingos, "Efficient Weight Learning for Markov Logic Networks" v *Proceedings of the Eleventh European Conference on Principles and Practice of Knowledge Discovery in Databases*, Springer, 2007, str. 200-211.

[5] M. Richardson, P. Domingos, "Markov Logic Networks" v *Machine Learning, 62*, AAAI Press, 2006, str. 107-136.

[6] P. Singla, P. Domingos, "Discriminative Training of Markov Logic Networks" v *Proceedings of the Twentieth National Conference on Artificial Intelligence*, 2005, str. 868-873.