Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Dejan Ognjenović

Iskanje neželenih interakcij zdravil

magistrsko delo

magistrski program računalništvo in informatika,
druga stopnja

Mentor: izr. prof. dr. Marko Robnik-Šikonja

Ljubljana 2014

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Dejan Ognjenović, z vpisno številko **63080327**, sem avtor diplomskega dela z naslovom:

Iskanje neželenih interakcij zdravil

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Marka Robnika-Šikonje.
- so elektronska oblika magistrskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki "Dela FRI".

V Ljubljani, dne 26. septembra 2014 Podpis avtorja:

Zahvala

Najprej bi se zahvalil mentorju prof. dr. Marku Robniku Šikonji za strokovno usmerjanje in nasvete ter spodbudo pri nastajanju magistrskega dela. Za uspešno sodelovanje bi se zahvalil tudi asist. mag. Andreji Čufar, mag. farm., spec. in Petri Volk Markovič mag., farm. iz Univerzitetnega kliničnega centra v Ljubljani, ki sta prispevali znanje s farmacevtskega področja, kot tudi njunim sodelavcem iz Univerzitetnega kliničnega centra v Ljubljana, ki so pomagali zbirati podatke. Zahvala gre tudi mag. Andreju Pečniku iz podjetja Wolters Kluwer, ki nam je omogočilo testni dostop do baze podatkov LexiComp.

Zahvalil se bi tudi svojim staršem in dekletu za moralno in materialno podporo med študijem.

Hvala!

Kazalo

Povzetek

Interakcije zdravil so prepletanja učinkov zdravil, ki lahko povzročijo želene ali škodljive učinke na pacientu. V nalogi smo z orodjem strojnega učenja iskali interakcije zdravil, ki lahko na zdravje bolnikov vplivajo negativno. Interakcije se lahko pojavijo med različnimi zdravili, ki jih jemlje posameznik, med zdravili in hrano ter med zdravili in specifičnimi bolezenskimi stanji. Sistematično zbiranje podatkov o kazalnikih neželenih kliničnih izidov zdravljenja v kombinaciji s predpisanimi zdravili, pridruženimi diagnozami in laboratorijskimi vrednostmi določenih parametrov nam daje možnost uporabe metod podatkovnega rudarjenja in s tem iskanja doslej še neznanih vzročnih dejavnikov za pojav neželenih kliničnih izidov zdravljenja.

Naš pristop k reševanju problema temelji na dveh algoritmih strojnega učenja za gradnjo pravil. Pri tem smo upoštevali hierarhiji zdravil (ATC) in bolezni (MKB) ter bazo interakcij LexiComp, ki med drugim vsebuje že znane interakcije parov zdravil.

Algoritem posplošenih povezovalnih pravil poskuša s pomočjo hierarhij poiskati pravila, ki poleg osnovnih elementov upoštevajo tudi njihove posplošitve v hierarhiji. Za odstranjevanje pravil smo uporabili mero R-zanimivosti.

Drugi uporabljeni algoritem je iskanje pravil s koristnostno funkcijo. Osnovni gradniki tega algoritma so v našem primeru pravila dolžine dve in tri. Določili smo funkcijo koristnosti in jo uporabili na osnovnih gradnikih. Funkcijo koristnosti smo definirali tako, da uporablja statistične informacije iz podatkov kot tudi iz hierarhij. Zaradi te lastnosti funkcija lažje odkriva redka pravila. Algoritem vrne vzorec (kombinacijo osnovnih gradnikov), ki prispeva največ glede na koristnostno

funkcijo. Na podlagi teh vzorcev lahko sestavimo nova pravila. Uporabili smo združevanje osnovnih gradnikov znotraj vzorca glede na posledico.

Algoritme smo testirali na umetno generiranih podatkih in na realnih podatkih pacientov iz Univerzitetnega kliničnega centra v Ljubljani. Baza vsebuje podatke o 150 odraslih bolnikih, ki so bili zdravljeni na kliničnih oddelkih interne klinike in klinike za infekcijske bolezni in vročinska stanja, ter so imeli visoko raven kalija v krvi.

Najdena pravila so pregledali farmacevti, ki so jih podrobno analizirali, komentirali in ovrednotili. Rezultati algoritmov so obetavni, saj smo odkrili nekaj zanimivih novih pravil in vzorcev. Največ zanimivih pravil je odkril algoritem posplošenih povezovalnih pravil.

Ključne besede: strojno učenje, podatkovno rudarjenje, povezovalna pravila, interakcije, interakcije zdravil, zdravila, hierarhije, LexiComp, generalizacija, ontologije, posplošena povezovalna pravila, povezovalna pravila s funkcijo koristnosti, funkcije koristnosti, hiperkaliemija

Abstract

Drug interactions are interweaving effects between two or more drugs that can have desirable or harmful effects on patients health. In this thesis we for searched harmful drug interactions. Interactions can occur between different drugs and foods the individual is taking, and also specific disease states. Systematic data collection of, indicators of harmful clinical outcomes of treatment in combination with prescribed medicines, associated diagnoses and laboratory values of certain parameters, gives us a possibility to use data mining techniques. With this methods we will try to find a previously unknown pattern of causal factors for adverse clinical outcomes.

Our approach is based on two machine learning algorithms for association rule mining. We use two given hierarchies, one for drugs (ATC), the other for diseases (ICD), and one proprietary interaction database LexiComp. LexiComp contains drug pairs that have well known interactions, their descriptions and classifications.

A generalized association rule algorithm tries to find rules that contain basic elements as well as elements from given hierarchies. To prune uninteresting rules we used R-interest criterion.

The second algorithm uses high-utility pattern mining. The basic building blocks for this algorithm are, in our case, rules of length two or three. We defined the utility function and used it on the building blocks. The utility function was designed to use statistical information from both the data and the hierarchies. Because of this, the function can detect rare rules. Algorithm returns a pattern (combination of building blocks), that contributes the most to the value of utility function. Based on these patterns we construct new rules by grouping building

blocks with the same consequences.

Algorithms were tested on artificial data and on a dataset from University Medical Centre Ljubljana. The database contained 150 patients that were treated in the clinical departments of in-house clinic and clinic for infectious diseases, and had a high amount of potassium in their blood.

Detected rules were reviewed, analyzed, commented and evaluated by pharmacists. The results are promising as several interesting new rules and patterns are detected. The most useful and interesting results are produced by generalized association rule mining algorithm.

Keywords: machine learning, data mining, association rules, interactions, drug interactions, drugs, hierarchy, LexiComp, generalization, ontologies, generalized association rules, high utility association rules, utility function, hypercalemia

Poglavje 1

Uvod

Mnogo ljudi jemlje več kot eno vrsto zdravil. Vpliv enega zdravila na delovanje drugega zdravila imenujemo *interakcija med zdravili*. Posledica interakcije je lahko ojačano ali zmanjšano delovanje enega ali več zdravil. Večje kot je število zdravil, ki jih posameznik jemlje, večja je tudi verjetnost, da bo med njimi prišlo do medsebojnega vpliva oz. interakcije. Na tveganje za pojav interakcije lahko do določene mere sklepamo na osnovi teoretičnega poznavanja farmakokinetike in farmakodinamike zdravil. Klinično manifestacijo interakcij se ugotavlja na podlagi kliničnih študij in na podlagi spremljanja učinkov zdravil tekom njihove uporabe v klinični praksi.

V magistrski nalogi smo poskusili s pomočjo strojnega učenja poiskati interakcije zdravil, ki bi lahko potencialno škodile zdravju pacienta. Ugotoviti, katera kombinacija povzroča neželene učinke, je zahtevno. Razlogi za to so:

- interakcije so redke,
- ljudji, ki jemljejo določeno kombinacijo zdravil, je lahko malo, zato je na voljo malo podatkov,
- posamezniki se na zdravilo lahko odzovejo različno,
- obstaja več tipov reakcij na zdravila, ki so odvisni, od različnih parametrov [6].

Primer interakcije je jemanje zdravila Aspirin in Plavix, ki podaljšata čas strjevanja krvi in ob sočasnem jemanju lahko povzročita krvavitve.

Hiperkaliemija je stanje povišane serumske koncentracije kalija v krvi nad 5,5 mmol/l. O hiperkaliemiji poročajo pri 1,1 - 10 odstotkih hospitaliziranih bolnikov [13]. V našo študijo smo vključili 150 odraslih bolnikov, ki so bili zdravljeni na kliničnih oddelkih Interne klinike in na Kliniki za infekcijske bolezni in vročinska stanja Univerzitetnega kliničnega centra Ljubljana (UKC Ljubljana). Bolniki, ki so bili vključeni v študijo, so imeli, med zdravljenjem ali ob sprejemu, povišano serumsko koncentracijo kalija. V nalogi smo želeli poiskati/identificirati še neznane dejavnike ali kombinacje dejavnikov, ki lahko vplivajo na povišano serumsko koncentracijo kalija.

Povezovalna pravila so pravila, ki so sestavljena iz pogoja in posledice in opisujejo povezavo med njima. Eden od algoritmov za gradnjo povezovalnih pravil je Apriori, ki na podlagi podpore in zaupanja iz podatkov gradi različne kombinacije pravil. Za iskanje interakcij, ki vplivajo na hiperkaliemijo, smo uporabili algoritem za iskanje posplošenih povezovalnih pravil ter algoritem iskanja povezovalnih pravil s koristnostno funkcijo. Prvi algoritem deluje podobno kot algoritem Apriori (poglavje 4), le da poskušamo odkriti povezave s pomočjo hierarhij. Na tak način lahko odkrijemo zdravilo, ki je v interakciji z določeno skupino zdravil in ne le z eno učinkovino.

Drugi algoritem poskuša odkriti pravila oziroma interakcije, ki imajo največ vpliva glede na neko koristnostno funkcijo. Funkcijo koristnosti smo definirali tako, da oceni pravilo glede na statistične informacije podatkov in hierarhij. Podatke smo testirali na umetnih podatkih in na realnih podatkih pacientov.

V 2. poglavju bomo obrazložili osnovne pojme pri gradnji povezovalnih pravil in omenili nekaj sorodnih raziskav na temo odkrivanja interakcij zdravil. V 3. poglavju opisujemo obliko in naravo naših podatkov. Opisujemo podatke, ki smo jih dobili od UKCLJ, bazo LexiComp ter hierarhije, ki so ključni del našega predznanja. Obrazložili smo, kako smo generirali umetne podatke in kakšni so.

Četrto poglavje opisuje delovanje in lastnosti algoritmov za odkrivanje povezovalnih pravil. Za vsak algoritem definiramo izrazoslovje in obrazložimo glavne dele. V 5. pogavlju komentiramo rezultate algoritmov na realnih in na umetno generiranih podatkih. V zaključku smo povzeli narejeno in podali ideje za nadaljnjo delo.

Poglavje 2

Osnovni pojmi povezovalnih pravil

V našem delu uporabljamo algoritme, ki gradijo povezovalna pravila. Podajmo najprej nekaj osnovnih pojmov. V podatkovni množici vrstice imenujemo transakcije (v našem primeru so to bolniki). Vsaka transakcija vsebuje enega ali več elementov oziroma predmetov. Povezovalna pravila so predstavljena na sledeč način, $X => Y$. X predstavlja pogoj pravila, Y pa posledico. Vsaka stran lahko vsebuje več elementov x_i oz. y_i. Pravilo beremo kot, "če imamo predmete X, lahko sklepamo na Y". Najpomembnejša informacija v pravilu je relacije med predmeti/elementi. Pravila lahko ovrednotimo tudi s spodnjimi merami. Tabela 2.1 prikazuje primer baze, iz katere jih lahko izračunamo. Za primer vzemimo pravila $P1 : A => B$, $P2 : B => A$ ter $P3 : A \wedge D \wedge \neg E => C$.

Podpora pravila (support) je definirano kot delež transakcij, ki vsebujejo elemente X, torej kot frekvenca leve strani pravila. Formalno zapišemo:

$$supp(X => Y) = P(X). \tag{2.1}$$

Iz tabele 2.1 je razvidno, da ima pravilo P1 podporo 80%, P2 podporo 60% ter P3 podporo 40%.

Transkacije	Predmet_A	Predmet_B	Predmet_C	Predmet_D	Predmet_E
1	1	1	0	0	1
2	1	1	0	0	1
3	0	1	1	0	0
4	1	1	0	1	0
5	1	0	0	0	1
6	1	1	0	0	1
7	1	0	1	1	0
8	0	0	0	1	1
9	1	0	1	1	0
10	1	1	1	1	0

Tabela 2.1: Primer baze za izračun osnovnih lastnosti pravil

Zaupanje pravila (confidence) je enako deležu transakcij z levo stranjo pravila (X), ki vsebujejo tudi desno stran (Y). To zapišemo kot:

$$conf(X => Y) = \frac{supp(X \cup Y)}{supp(X)}. \tag{2.2}$$

To lahko zapišemo kot verjetnost,

$$conf(X => Y) = P(Y|X) = \frac{P(X \cap Y)}{P(X)}. \tag{2.3}$$

To lastnost lahko interpretiramo kot točnost našega pravila. Pravilo P1 ima zaupanje 62.5% ($\frac{5}{8}$), P2 ima 83.3% ($\frac{5}{6}$), P3 pa 75% ($\frac{3}{4}$).

Dvig (Lift) je razmerje med opazovano verjetnostjo X in Y in pričakovano verjetnostjo, če bi bile X in Y neodvisni spremenljivki. Formalno zapišemo kot:

$$lift(X => Y) = \frac{supp(X \cup Y)}{supp(X) * supp(Y)}. \tag{2.4}$$

Ko je vrednost lift=1, nam to pove, da sta leva in desna stran pravila neodvisni. Takšno pravilo ne more biti zanimivo. Vrednosti, ki so večje (ali manjše) od 1, nam povedo, do katere mere sta odvisni leva in desna stran pravila. Pravilo P1 in P2 imata vrednost lift 1.04 ($\frac{0.5}{0.8*0.6}$), kar pomeni neodvisnost torej nezanimivo

pravilo. Mera lift je simetrična (preverja podporo dveh neodvisnih spremenljivk in njun vrstni red ni pomemben). P3 ima vrednost lift=1.875 ($\frac{0.3}{0.4*0.4}$), kar nakazuje na odvisnost.

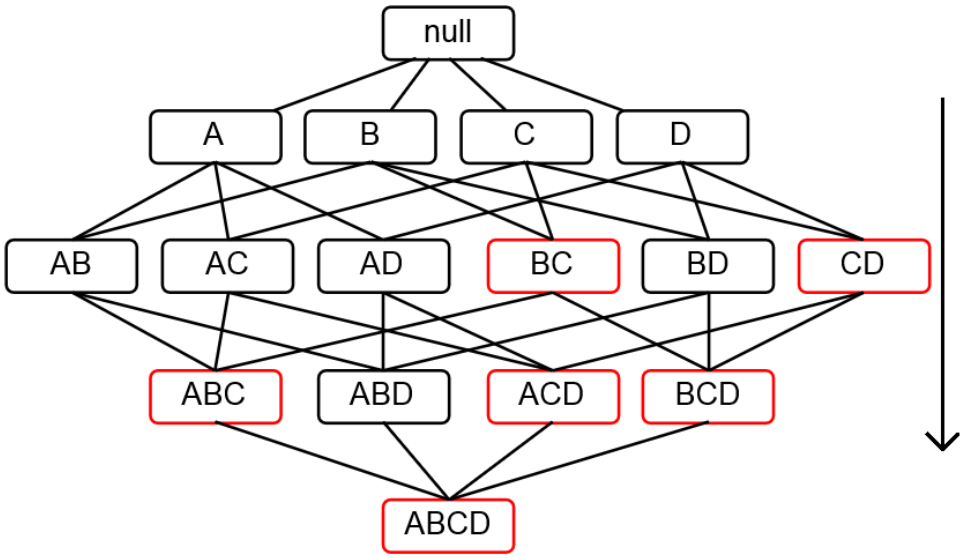
Prepričanje (Conviction) je razmerje med pričakovano verjetnostjo, da ima transakcija Y in ne X (neodvisni spremenljivki), ter verjetnostjo nepravilne predikcije. Formalno to zapišemo kot:

$$conv(X => Y) = \frac{1 - supp(Y)}{1 - conf(X => Y)}. \tag{2.5}$$

Prepričanje je uporabna mera, saj vključuje informacijo o implikaciji pravil, torej upošteva smer pravil. Če je pričakovanje=1, nam to pove, da je točnost pravila primerljiva z točnostjo, ki bi jo dosegli, če bi bili spremenljivki X in Y neodvisni. Vrednost, ki je večja (ali manjša) od 1 nam pove delež odstopanja od neodvisnosti leve in desne strani pravila. Pravilo P1 ima vrednost 1.06 ($\frac{1-0.6}{1-0.625}$). To pomeni, da je točnost enaka, kot če bi bili spremenljivki neodvisni, torej je pravilo nezanimivo. P2 ima vrednost 1.2 ($\frac{1-0.8}{1-0.833}$), kar pomeni, da bi bila točnost pravila 20% večkrat napačna, če bi predpostavili neodvisnost leve in desne strani pravila. Pravilo P3 ima vrednost 2.4 ($\frac{1-0.4}{1-0.75}$), kar pomeni, da bi točnost pravila bila 140% večkrat napačna, če bi predpostavili neodvisnost leve in desne strani pravila.

Algoritem Apriori je osnovni algoritem za iskanje povezovalnih pravil [4]. Algoritem iz elementov v podatkih (transakcijah) sestavlja različne kombinacije elementov in odstranjuje tiste, ki ne ustrezajo kriteriju minimalne podpre. Iz ohranjenih kombinacij sestavimo povezovalna pravila. Spet pregledujemo prostor pravil, le da zdaj uporabimo kriterij minimalnega zaupanja. Pravila, ki ustrezajo tudi temu kriteriju so rezultat algoritma.

Za lažje razumevanje algoritma si poglejmo sliko 2.1. Slika prikazuje celotni preiskovalni prostor štirih elementov, puščica pa prikazuje smer preiskovanja. Element, ki ni imel zadosti podpore, je obrobljen z rdečo. Tiste kombinacije elementov, ki vsebujejo elemente brez minimalne podpore, ne bodo več izpolnjevale kriterija in ne bodo preiskane (primer CD na sliki 2.1).



Slika 2.1: Primer preiskovalnega prostora štirih elementov (A, B, C in D). Rdeča obroba pomeni, da kombinacija nima zadosti podore v podatkih, zato njeni nasledniki niso preiskani.

Na področju medicine je bilo uporabljenih že mnogo algoritmov strojnega učenja tudi takšni, ki vsebujejo informacije o hierarhijah. Algoritem z večnivojsko hierarhijo (Cross-Ontology data mining) je bil uporabljen na genetskih podatkih in na genskih ontologijah [7]. Uporabljen je bil tudi pristop z FP-growth (frequent pattern growth) algoritmom, ki uporablja informacije večnivojskih hierarhij [8] s pomočjo FP-algoritma. Generiranje FP kandidatov se začne v listih hierarhije. V vsakem nivoju se generirajo vse možne kombinacije kandidatov iz elementov višje v hierarhiji. Vsi kandidati višjih nivojev, ki imajo zadosti podpore, so ohranjeni. Generiranje se izvaja, dokler ne pridemo do korena dreves. Poskus odkrivanja interakcij dveh zdravil je opisan v [5], podobno kot je to storjeno pri bazi LexiComp. Algoritmi, ki jih uporabljamo in prilagajamo v tem delu so posplošena povezovalna pravila [2] in iskanje pravil z koristnostno funkcijo s pomočjo dreves visoke donosnosti [1].

Poglavje 3

Podatki

V tem poglavju opišemo podatke o pacientih, hierarhijah zdravil in bolezni ter bazo LexiComp. V hierarhijah definiramo tudi zapis zdravil, ki ga bomo uporabljali v nalogi. Opišemo tudi postopek generiranja umetnih podatkov.

3.1 Podatki iz UKCLJ

Podatke o bolnikih so zbrali klinični farmacevti, zaposleni v lekarni Univerzitetnega kliničnega centra Ljubljana. Študijo je odobrila Komisija Republike Slovenije za medicinsko etiko. V raziskavo smo vključili 150 odraslih bolnikov, ki so bili zdravljeni na kliničnih oddelkih interne klinike ter na kliniki za infekcijske bolezni in vročinska stanja. Ob sprejemu ali med hospitalizacijo so imeli bolniki povišano plazemsko koncentracijo kalija. Izključili smo paciente glede na izključitvene kriterije:

- prejeli so manj kot tri zdravila,
- prejeli so hrano s kalijem,
- imajo bolezen, ki vpliva na kalij.

Podatki vsebujejo splošne demografske informacije o pacientih (spol, starost, telesna teža in višina), bolnikove diagnoze, vrednost plazemske koncentracije kalija

ter farmakoterapijo, oziroma zdravila, ki so jih bolniki prejemali. Hiperkaliemija je bila izbrana kot klinični kazalnik potencialnih interakcij in sicer zato, ker je jasno in nedvoumno določljiva ter razvidna iz zdravstvene dokumentacije bolnikov. Pacienti so zabeležena zdravila, dobili najmanj 24 ur pred meritvijo plazemske koncentracije kalija, v tem času so se simptomi/interakcije lahko izrazile.

V bazi so podatki o 150 bolnikih, od tega je 94 moških in 55 žensk. Povprečna starost populacije je 70 let, povprečna teža je 77.4 kg, višina pa je 168 cm. Povprečna plazemska koncentracija kalija v krvi je 5.83 mmol/l (normalna količina je 3.5 do 5 mmol/l). Bolniki so v povprečju prejeli 7.5 zdravil in bili dignosticirani s povprečno sedmimi diagnozami. Primer opisa bolnika je prikazan v tabeli 3.1.

ID	datum	spol	teža	višina	kalij	MKB	učinkovina	odmerek	ATC
6201	14.1.2014	Ž	60	175	5.9	D50.9	ciprofloksacin	800 mg	J01MA02
						K85	metronidazol	1500 mg	P01AB01
						K86.3	zelezov (II) sulfat	200 mg	B03AA07
						E11.9	zolpidem	5 mg	N05CF02
						N18.9	bisakodil	10 mg	A06AB02

Tabela 3.1: Primer bolnikovih podatkov.

Število vseh možnih kombinacij parov zdravil iz naših podatkov je 572985. Večine parov v praksi ne bomo opazili, saj vsebujejo zdravila, ki jih pacient verjetno ne bo dobil. Število kombinacij, ki se dejansko pojavijo pri pacientih je 4094. V bazi LexiComp je bilo od teh identificiranih 431 kot znane interakcije, kar je 10.5% vseh parov.

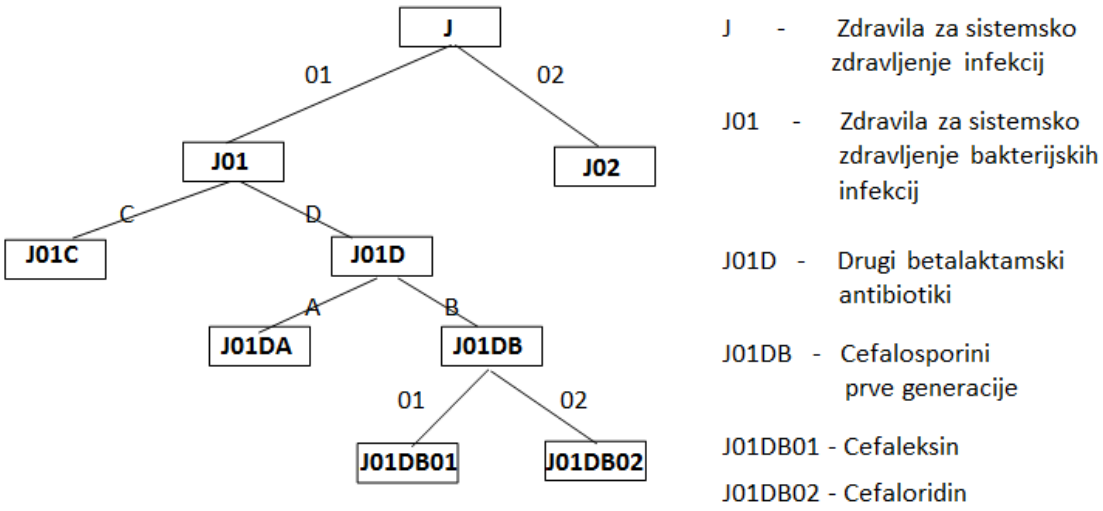
3.2 Baza LexiComp

Pridobili smo tudi dostop do komercialne baze interakcij LexiComp [25] pri podjetju Wolters Kluwer. Baza hrani med drugim pare že obstoječih znanih interakcij zdravil. Informacije smo uporabili v algoritmih, kot pomoč pri sestavi pravil. Nekatere kombinacije v naši bazi ne bi imele zadosti podpore in bi jih zavrgli. Podatki iz LexiCompa pripomorejo k temu, da določene kombinacije vseeno upoštevamo, saj vemo, da gre za znane interakcije.

V bazi LexiComp so interakcije razdeljene na pet kategorij, A, B, C, D in X. A in B so trenutno obravnavane kot neškodljive ali neznane. C so interakcije, ki potencialno lahko povzročijo zaplete, (odvisno od posameznika). D in X interakcije, ki lahko povzročijo resne zaplete in zato praksa odsvetuje rabo takšne kombinacije. Več o tem najdemo v [6]. Tabela 3.2 prikazuje, na kolikšen delež pacientov lahko določena vrsta interakcij vpliva ter kakšno podporo v algoritmu že vnaprej dodelimo vsakemu tipu interakcij.

tip interakcije	spodnja meja	zgornja meja	opis	podpora
A	0.01% <	0.01%	ni znanih interakcij	Te iščemo
B	0.01%	0.1%	posredovanje ni nujno	0.5%
C	0.1%	1%	obvezno opazovanje	1%
D	1%	10%	potrebna sprememba terapije	5%
X	10%	10% >	strogo izogibati	15%

Tabela 3.2: Tabela kategorij interakcij in njihova izbrana podpora. Spodnja in zgornja meja kažeta razpon verjetnosti, da se stranski učinek pojavi pri bolniku (verjetnost na 100 bolnikov).



Slika 3.1: Primer ATC klasifikacije za učinkovini celoksifina in cefalordina.

3.3 Hierarhije

Uporabili smo tudi podatke iz hierarhij/ontologij zdravil in bolezni. Zdravila so klasificirana po tako imenovani ATC klasifikaciji [24] (Anatomical Therapeutic Chemical), bolezni pa po MKB-10 [23] (Mednarodna statistična klasifikacija bolezni in sorodnih zdravstvenih problemov).

ATC razdeli zdravila glede na njihovo sistemsko delovanje ali na predel delovanja kot tudi po terapevtskih in kemičnih karakteristikah. Vsako zdravilo ima lahko eno ali več šifer, odvisno od namena uporabe. Hierarhija ATC je pet slojna in vsebuje 5439 različnih listov. Najvišji nivo je razdeljen na 14 skupin. Vsi listi ATC hierarhije so opisani s sedmimi znaki. Slika 3.1 prikazuje primer takšne hierarhije in zapisa.

MKB-10 ali ICD-10 (International Classification of Diseases) vsebuje šifre bolezni, simptomov, anomalij, pritožb, družbenih okoliščin in eksternih vzrokov poškodb in okužb. Hierarhija MKB je troslojna in vsebuje 18955 listov. Klasifikacija MKB-10 je po konstrukciji podobna ATC, le da ima drugačne oznake in opise. Oznaka lista je v MKB dolga 5 znakov.

Hierarhiji smo opisali kot $A|01|C|04|D$, kjer znak | predstavlja ločilo med oznakama dveh različnih nivojev (podrobnejši opis je v tabeli 3.1). Zapis lahko gledamo kot na pot, kjer koda na levi strani predstavlja višji nivo v ontologiji, koda na desni pa nižji. Npr. $A01C04D$, predstavlja pot $A|01|C|04|D$, višji nivo oziroma generalizacijo te šifre pa predstavlja pot $A|01|C$. Elementi, ki vsebujejo oznako sim predstavljajo bolezen in ne učinkovine, npr A_{sim}.

3.4 Umetno generirani podatki

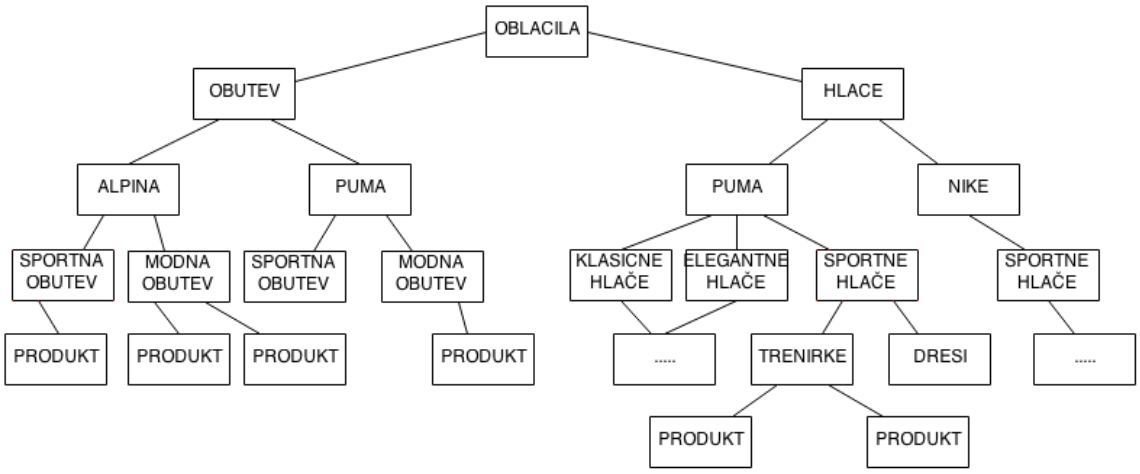
Za potrebe razvoja algoritma smo generirali podatkovne množice z znanimi lastnostmi. Da bi testirali pravilnost delovanja algoritmov, smo v podatkovne množice vstavili določeno število izmišljenih pravil. Ta so vsebovala elemente različnih nivojev ATC in MKB hierarhij. Podatke smo ustvarili tako, da smo vnaprej podali pravilo npr. $A|01|C => Bolezen1|Nivo2$ in število bolnikov, kjer se mora izražati to pravilo. Generator naključno ustvari demografske podatke o bolniku. MKB in ATC šifre generira tako, da se pomika po grafu od najvišjega nivoja proti najnižjemu in pri tem zapisuje kodo (pot). Za posamezen ATC ali MKB element je sprehod po grafu deterministično, dokler je pot opisana v vstavljenem pravilu. Ko pridemo do nivoja/poti, ki ni vsebovana v pravilu, se po hierarhiji do listov premikamo naključno (npr. $A|01|C|04|D => Bolezen1|Nivo2|Tip1|$). Generirali smo množice velikosti 100 in 10000 primerov s povprečnim številom elementov 14 (diagnoze in učinkovine).

Poglavje 4

Algoritmi

V tem poglavju razložimo, katere algoritme smo uporabili, kako delujejo, kakšne lastnosti imajo ter kako smo jih implementirali. Algoritmi, ki smo jih uporabili, iščejo vzorce ali tako imenovana asociativna oz. povezovalna pravila, ki so sestavljena iz predmetov/elementov (items), ki se pojavijo v podatkih (transakcijah). Vsak od teh predmetov spada v neko hierarhijo, ki vsebuje njegove informacije in posplošitve. Osnovni element lahko gledamo kot na list v hierarhiji, kjer je vsak višji nivo splošnejši opis skupine elementov.

Za boljše razumevanje vzemimo primer z obutvijo. Stranka kupi nek konkreten model čevljev. Vsak model čevljev lahko posplošimo z podjetjem, ki ga izdeluje, na primer Puma, Alpina in podobno. Čevlje lahko posplošimo še na kategorije, kot so športna obutev ali modna obutev vse skupaj pa lahko posplošimo kot obutev. V trgovini k temu lahko dodamo še ontologije ostalih izdelkov kot na primer majice, pokrivala, hlače in podobno. Te hierarhije lahko združimo v skupno vozlišče (oblačila). Vsako vozlišče v drevesu ima lahko eno ali več višjih nivojev, zato je smiselno, da te skupine dreves obravnavamo kot usmerjene aciklične grafe (DAG). Ta pogled nam bo prišel prav, saj imajo lahko tudi zdravila in simptomi več hierarhij. Slika 4.1 prikazuje primer hierarhije. Povezovalno pravilo vsebuje le elemente, ki so v listih drevesa (npr. $produkt_1 \wedge produkt_2 => produkt_{123}$), medtem ko posplošena povezovalna pravila lahko vsebujejo kateri koli element hierarhije (npr. $sportna\_obutev\_alpina \wedge produkt_k => hlace\_nike$).



Slika 4.1: Primer hierarhij oblačil.

4.1 Posplošena povezovalna pravila

Za iskanje povezovalnih pravil se pogosto uporablja algoritem Apriori, ki išče vzorce predmetov samo v listih usmerjenega acikličnega grafa. Nadgradnja tega je algoritem za iskanje posplošenih povezovalnih pravil [2] in uporaba zunanjih virov informacij. Algoritem išče vzorce med vsemi vozlišči grafa, saj se lahko pravila skrivajo tudi na višjih nivojih.

Algoritem lahko razbijemo na tri dele. Prvi je kreiranje množic kandidatov, ki zadostujejo minimalni podpori. Naslednji korak je uporaba pogostih elementov in sestava pravil, ki zadoščajo minimalnemu zaupanju. Nazadnje je potrebno odstraniti nezanimiva pravila. V naslednjih podpoglavjih so podane definicije in glavni koraki algoritma.

4.1.1 Izrazoslovje

Za lažje razumevanje bomo predmete oziroma osnovne elemente pisali z malimi črkami, množice elementov pa z velikimi. Naj bo $E = \{e_1, e_2, e_3, ..., e_n\}$ množica osnovnih elementov, ki se pojavijo v množici transakcij D, kjer je vsaka transakcija $T \subseteq E$. Naj bo G usmerjen aciklični graf. Vse povezave v grafu so relacije oče-

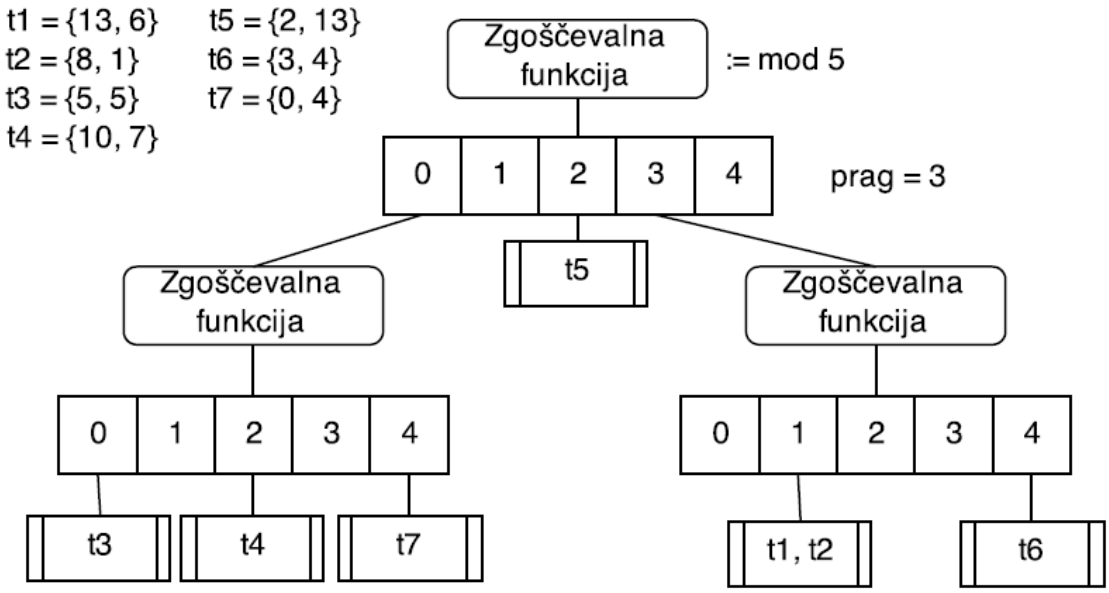
sin oziroma prednik-potomec. Vsak prednik vozlišča predstavlja generalizacijo svojih naslednikov. Element $\hat{e_i}$ imenujemo prednik elementa e_i medtem, ko e_i imenujemo potomec elementa $\hat{e_i}$. Pravila $\hat{X} => Y$, $\hat{X} => \hat{Y}$, $X => \hat{Y}$ imenujemo *predniki* pravila $X => Y$. Primer pravila $\hat{X} => Y$ v hierarhiji 4.1, bi lahko bil, $sportna\_obutev\_alpina \wedge sportna\_obutev\_puma => produkt_k$. Pravilo $\hat{X} => \hat{Y}$ imenujemo *bližnji prednik* pravila $X => Y$. Posplošeno pravilo pravilo je oblike $X => Y$, kjer je $X \subset E, Y \subset E, X \cap Y = 0$ in kjer noben element X ni prednik elementov v Y. Podpora (support) pravil je definirana kot $|Z| = |X \cup Y|$, zaupanje (confidence) pa je definirano enako kot pri običajni asociativnih pravilih, torej $conf(X => Y) = \frac{supp(X \cup Y)}{supp(X)}$. Vsa pravila, ki imajo višjo podporo in zaupanje od vnaprej določenega nastavljenega (minsupp in minconf), so potencialno zanimiva.

4.1.2 Kreiranje kandidatov in štetje podpore

V prvem koraku preverimo ali ima neka množica elementov X zadosti podpore v množici transakcij D. Glede na to, da želimo upoštevati hierarhije, bomo vsako transakcijo T razširili tako, da bomo dodali vse prednike osnovnih elementov, ki se pojavijo v T. Razširjene transakcije bomo označili s T'. T podpira X le, če $X \subseteq T'$. Prvi sprehod skozi podatke uporabimo, da izračunamo podporo vseh osnovnih elementov. V vsaki nadaljnji iteraciji moramo sestaviti vse možne kombinacije elementov, ki so imeli dovolj podpore v prejšnji iteraciji, ter izračunati njihovo podporo.

Generiranje kandidatov je kombiniranje vseh elementov, ki so imeli zadostno podporo v prejšnji iteraciji. V vsaki iteraciji na novo generiramo kandidate velikosti $l + 1$. Glede na to, da imamo tudi podatke o znanih interakcijah iz obstoječih baz (Lexicomp), lahko to znanje vpeljemo v ta korak. V primeru, ko imamo kombinacijo zdravil, ki nima zadosti podpore v podatkih, pogledamo, ali se ta kombinacija pojavi v znanih bazah. Če je temu tako, vzamemo to kombinacijo kot kandidata in ji pripišemo podporo iz baze. Če kombinacija nima podpore in je ni v bazi, jo preskočimo.

Hitro štetje podpore smo implementirali z tako imenovanimi zgoščevalnim drevesom (Hash Trees) prikazanim na sliki 4.2. Vozlišča v tej strukturi predsta-



Slika 4.2: Primer zgoščevalnega drevesa.

vljajo slovar, listi pa množico kandidatov. Vsako vozlišče ima n otrok. Vsak otrok lahko hrani k kandidatov. Pri konstrukciji drevesa vstavljamo kandidate glede na zgoščevalno funkcijo. V našem primeru je za zgoščevanje zadostoval modul. Zaradi iskanja je zaželeno, da so elementi razpršeni enakomerno po drevesu. Ko otrok hrani več kot k kandidatov, se razbije v nov slovar z n otroci in spremenjeno zgoščevalno funkcijo. Listi hranijo števce za vse kandidate v njih. Štetje izvajamo tako, da pogledamo ali se kandidat nahaja v strukturi in povečamo števec. Časovna zahtevnost iskanja oziroma štetja v strukturi je $log_n k$ pod pogojem, da zgoščevalna funkcija enakomerno razpršuje elemente po drevesu.

4.1.3 Konstrukcija pravil

Drugi korak je sestavljanje pravil iz izbranih kandidatov. Pravila sestavimo na enaki način kot v algoritmu Apriori. Kandidate kombiniramo in obdržimo tiste, ki zadoščajo minimalnemu zaupanju. Za lažje razumevanje opisanega algoritma je dodana psevdokoda 1.

4.1.4 Odstranjevanje nezanimivih pravil

Na koncu postopka dobimo mnogo neinformativnih in nezanimivih pravil. Da lažje odstranimo ta pravila, smo uporabili mero zanimivosti (interest). Pravilo $X => Y$ je R-zanimivo glede na $\hat{X} => \hat{Y}$, če je podpora ali zaupanje pravila R-krat višje od pričakovane. Če določeno pravilo nima potomcev, ga prav tako štejemo med zanimiva, saj lahko vsebuje koristne informacije.

Pričakovana podpora je definirana kot

$$E_{\hat{Z}}[Z] = \frac{sup(z_1) * sup(z_2) * ... * sup(z_i)}{sup(\hat{z_1}) * sup(\hat{z_2}) * ... * sup(\hat{z_i})} * sup(\hat{Z}) \tag{4.1}$$

kjer je $Z : X => Y$ in kjer je $sup(\hat{Z})$ definiran kot $sup(\hat{z_1}, \hat{z_2}, ..., \hat{z_j})$.

Pričakovano zaupanje je definirano kot

$$E_{\hat{X}=>\hat{Y}}[Y|X] = \frac{sup(y_1) * sup(y_2) * ... * sup(y_i)}{sup(\hat{y_1}) * sup(\hat{y_2}) * ... * sup(\hat{y_i})} * sup(\hat{Y}|\hat{X}) \tag{4.2}$$

kjer je $sup(\hat{Y}|\hat{X})$ definiran kot $sup(\hat{y_1}, \hat{y_2}, ..., \hat{y_j})$. Pričakovano zaupanje glede na nebližnjega potomca je definirana kot

$$E_{\hat{X}=>Y}[Y|X] = sup(Y|\hat{X}) \tag{4.3}$$

Algorithm 1 Psevdokoda posplošenih povezovalnih pravil.

D = sestavimo usmerjene aciklične grafe ▷ D =acikličen graf oziroma hierarhije
$L = \{\}$ ▷ L =podpora vseh elementov iz vseh generacij
$R = \{\}$ ▷ R = množica pravil
L_1 = izračunamo podporo vsakega elementa v bazi ▷ L_1 =podpora prve generacije
$k = 2$ ▷ k =trenutna generacija
while $|L_{k-1}| \neq 0$ **do**
  C_k= kreiramo nove kandidate iz množice L_{k-1} ▷ C_k =kandidati generacije k
  if $k == 2$ **then**
    odstrani kandidate, ki vsebujejo element in njegovega prednika
  end if
  for all $t \in T$ **do**
    for all $i \in t$ **do**
      transakcijo razširimo. Dodamo vse prednike i v t
    end for
    odstrani podvojene elemente v t
    povečaj števce kandidatov C_k, ki so del t
  end for
  L_k = vsi kandidati iz C_k z zadostno podporo
  ▷ če se kandidat pojavi v bazi LexiComp, ga ohranimo za bodoče iteracije.
  $k = k + 1$
end while
for all $L_k \in L$ **do**
  K = sestavi pravila iz L_k
  for all $e \in K$ **do**
    if zaupanje večje od minimalnega zaupanja **then**
      dodaj pravilo v R
    end if
  end for
end for
return L in R

4.2 Iskanje pravil s koristnostno funkcijo

Drugi algoritem, ki smo ga uporabili, je iskanje pravil s koristnostno funkcijo [1]. Večina algoritmov za iskanje povezovalnih pravil temelji na pogostosti določenih vzorcev v podatkih. To je lahko problematično, kadar so ti vzorci redki, vendar vseeno pomembni. Algoritem, predstavljen v tem razdelku, poskuša poiskati pravila, ki niso nujno najpogostejša, temveč so najmočnejša glede na neko koristnostno funkcijo.

Vzemimo prejšnji primer z obutvijo. Naj bo koristnostna funkcija v tem primeru skupni dobiček. Algoritmi, ki temeljijo na pogostosti vzorcev, bi odkrili pravila, ki opisuje pogosto prodajo izdelka kar ni nujno najbolj dobičkonosno. (na primer, tekaški čevlji in nogavice). Algoritem s koristnostno funkcijo bi odkril pravilo, ki prispeva največ k skupnemu dobičku, npr modni čevlji z obleko, ki se verjetno prodajo v manjših količinah, vendar je njihov skupen dobiček večji od prejšnjega primera.

V naši aplikaciji iščemo neželene stranske učinke interakcij zdravil, ki so redki, in jih je načeloma težko zaznati. Zaradi tega je uporaba tega algoritma smiselna, saj ne upošteva le pogostosti pravil, temveč še druge informacije, ki jih vpeljemo v koristnostno funkcijo. V nadaljevanju definiramo terminologijo, koristnostno funkcijo in obrazložimo delovanje algoritma.

4.2.1 Izrazoslovje

Naj bo $E = \{e_1, e_2, e_3, ..., e_M\}$ množica osnovnih elementov, ki se pojavijo v množici transakcij D, kjer je vsaka transakcija T_i definirana kot matrika $A_i = [a^i_{j,k}]$. Tukaj $a_{j,k}$ predstavlja vrednost j-tega atributa za e_k-ti element npr. dobiček pri nekem izdelku. Vsak osnovni element ima lahko v naprej nastavljene atribute $a_1, a_2, ..., a_L$. V našem primeru bi atributi za vsako zdravilo lahko bili količina, čas zdravljenja in podobno. Celotni *prispevek transakcije* T_i bo definiran kot

$$c_i = \sum_{j=1}^{M} c_{i,j} \tag{4.4}$$

kjer je $c_{i,j}$ prispevek elementa e_j k transakciji T_i je definiran kot

$$c_{i,j} = f(a^i_{j,1}, a^i_{j,2}, ..., a^i_{j,L}), \tag{4.5}$$

pri katerem je $f(...)$ poljubno definirana *koristnostna funkcija*.

Naj bo $P' = \{P_1, P_2, ..., P_Q\}$ množica vseh možnih vzorcev, kjer je $P_k = \{p^k_1, p^k_2, ..., p^k_M\}$ vzorec elementov. Vsak p_i lahko predstavlja vrednost 1, 0 ali -1. Če ima p_i vrednost 1, pomeni, da element e_i prispeva k koristnosti transakcije, -1 da element ne prispeva in 0 pomeni, da nas ta element ne zanima. Vzorec je v našem primeru ekvivalenten pravilu. Transakcija T_i je *vsebovana* v vzorcu P_k, če ima p_i za vsak neničelni prispevek vrednost $c_{i,j} > 0$ ter za vsako ničelno vrednost $c_{i,j} = 0$. Ali je transakcija T_i vsebovana v nekem vzorcu opišemo z sledečo notacijo $b_i = \{b_{i,1}, b_{i,2}, ..., b_{i,Q}\}$, kjer

$$b_{i,k} = \begin{cases} 1; & \text{če } T_i \text{ zadostuje vzorcu } P_k \\ 0; & \text{če } T_i \text{ ne zadostuje vzorcu } P_k \end{cases}$$

Podmnožico transakcij, ki je vsebovana v vzorcu P_k, imenujemo *naročitvena množica* P_k (subscription set) in jo označimo kot $D_k = \{T_{k1}, T_{k2}, ..., T_{kS}\}$, pri čemer je $D_k \subset D$. Definirimo *prispevek celotne podmnožice* D_k, ki ga označimo z $C(D_k)$. Prispevek je enak vsoti prispevkov vseh transakcij v tej podmnožici, torej

$$C(D_k) = \sum_{i=1}^{|D_k|} c_i \tag{4.6}$$

Definirimo *vzorčni prispevek* (in-pattern contribution), ki ga zapišemo kot $C_p(D_k)$ in definiramo kot

$$C_p(D_k) = \sum_{i=1}^{|D_k|} \sum_{j=1}^{M} p^k_j * c_{i,j} \tag{4.7}$$

Pomemben parameter je *vzorčno razmerje*, ki je definirano kot $d_k = \frac{C_p(D_k)}{C(D_k)}$. Vzorec P_k imenujemo *opredeljevalni vzorec* (defining pattern), če zadošča pogoju $d_k \geq \sigma$, kjer je σ opredeljevalni parameter. Vrednost tega parametra bo v večini primerov visoka (razpon med [0.5,1]). Pomembnost določenega vzorca določimo z

razmerjem $\frac{C(D_k)}{C(D)} \geq \gamma$, kjer je γ parameter signifikantnosti (vsebuje nizke vrednosti kot npr 0.05). Če vzorec izpolnjuje oba pogoja, ga imenujemo *značilni opredeljevalni vzorec* ali SDP (significant defining pattern). Najbolj uporabni vzorci so prav SDP, saj vsebujejo pravila, ki največ prispevajo k skupnemu prispevku.

Pokažimo definicije še na praktičnem primeru. Tabela 4.1 prikazuje transakcije (t) ter izdelke (i). Vrednost v tabeli predstavlja dobiček (koristnostna funkcija), ki ga je prinesel predmet i v neki transakciji. Iz primera lahko izpeljemo dva močna vzorca, in sicer $P_1 = [1, 0, 0, 1, 0]$ z vsebujočimi transakcijami $D_1 = \{T1, T2\}$ ter $P_2 = [-1, 0, 1, 0, 1]$ z vsebujočimi transakcijami $D_2 = \{T3, T4\}$. Prispevek celotne množice vzorca P_1 je $C(D_1) = (50 + 20 + 10 + 65) + (25 + 10 + 50 + 15) = 245$, medtem ko je vzorčni prispevek $C_p(D_1) = (50 + 65) + (25 + 50) = 190$. Vzorčno razmerje tega vzorca bi tako bilo $d_1 = \frac{190}{245} = 0.775$, kar pomeni, da bo vzorec značilen, če bo $\sigma \leq d_1$. Vzorec bo po vsej verjetnosti SDP, saj $C(D_1)$ predstavlja 43% celotnega dobička.

transakcije — izdelki	i_1	i_2	i_3	i_4	i_5
T_1	50	20	10	65	0
T_2	25	0	10	50	15
T_3	0	10	100	20	40
T_4	0	5	35	0	70
T_5	10	20	10	0	0

Tabela 4.1: Tabela prikazuje dobiček predmetov i v določeni transakciji T_j. Celotni dobičel je 565.

Ker je število trivialnih SDP vzorcev veliko, nas zanimajo samo množice vzorcev, ki so manjše in predstavljajo velik del skupnega prispevka. Iz tega sklepamo, da je kvaliteta vzorcev odvisna od *donosa* (yield), ki je razmerje $C_p(D_k)/C(D)$, ter od velikosti množice vzorcev D_k. Donos je ocena celotne strukture (drevesa).

4.2.2 Optimizacijski problem

Naš cilj lahko sedaj izrazimo kot optimizacijski problem. Poskušamo odkriti množice vzorcev PS, ki vsebujejo največji donos glede na velikost. Zanimajo nas množice vzorcev velikosti v, ki zadostujejo sledečim pogojem:

- vsi vzorci v množici morajo biti opredeljevalni vzorci,
- vsi vzorci v množici morajo biti SDP vzorci,
- vzorci se ne smejo prekrivati.

Glede na to, da je PS kombinacija vzorcev P_k, jo lahko opišemo kot $X = \{x_1, x_2, ..., x_Q\}$ kjer

$$x_k = \begin{cases} 1 & \text{če je } P_k \text{ v } PS \\ 0 & \text{če } P_k \text{ ni v } PS \end{cases}$$

Sedaj optimizacijski problem, glede na zgornje omejitve, opišemo kot

$$\sum_{k=1}^{Q} x_k \sum_{i=1}^{|D_k|} \sum_{j=1}^{M} p_j^k * c_{i,j}. \tag{4.8}$$

Zaradi visoke računske zahtevnosti je bil uporabljen algoritem za iskanje suboptimalnih rešitev, ki se imenuje drevo visoke donosnosti (High-yield partition tree).

4.2.3 Koristnostna funkcija

Koristnostna funkcija lahko vključuje različne informacije. Za osnovno funkcijo smo si izbrali prepričanje (2.5), ki meri razmerje med pričakovano pogostost X-a brez Y, če sta X in Y neodvisni spremenljivki, ter izmerjeno pogostostjo X in Y.

Omenili smo, da so interakcije zdravil redek pojav in jih je težko zaznati, saj v večini primerov le malo odstopajo od šuma. Tako metode, ki merijo le frekvenco pojavitve, niso najbolj učinkovite za ta problem. Zato smo izbrali naslednjo koristnostno funkcijo, ki se izkaže kot praktično uporabna:

$$indep(X => Y) = a * conv(X => Y) + b * abs(P(X => Y) - E_{X=\hat{>}Y}[X => Y]). \tag{4.9}$$

To funkcijo imenujemo funkcijo neodvisnosti. Prvi del funkcije vsebuje prepričanje, ki meri delež napačno klasificiranih predikcij, pod predpostavko, da sta spremenljivki X in Y neodvisni. Prepričanje ima razpon vrednosti $[0, \infty)$, kar posledično vpliva tudi na funkcijo $indep(X => Y)$. Prepričanje je uporabna mera, saj vsebuje informacijo o implikaciji pravil. Uporabnost tega je v asimetričnosti, saj pravili $X => Y$ in $Y => X$ nista enaki.

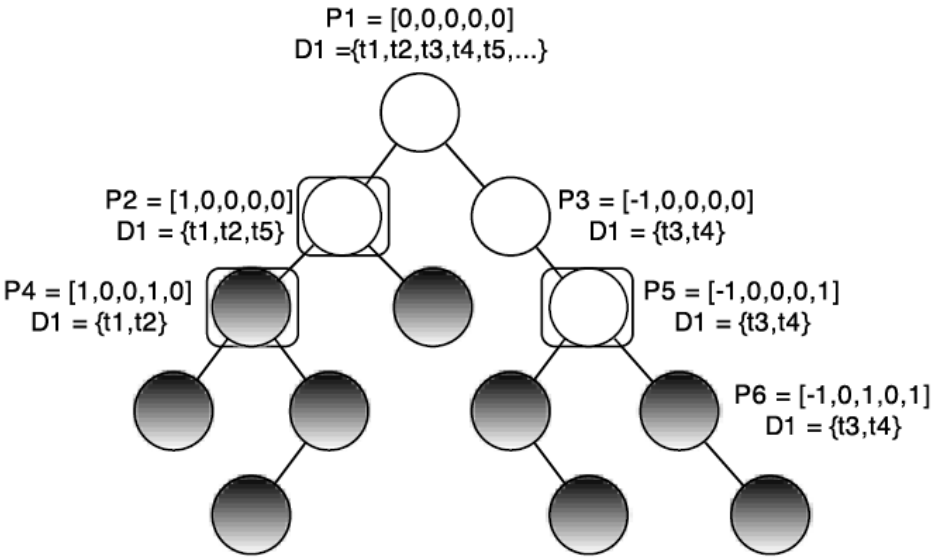
Drugi del sestavlja razlika med izmerjeno pojavitvijo pravila in pričakovano verjetnostnjo pravila glede na hierarhije. Pričakovano verjetnost pravila smo izračunali po izračunu (4.1).

S to funkcijo ocenimo pravila glede na statistične podatke. Funkcija preveri neodvisnost pravila po dveh kriterijih. Najprej preveri neodvisnost na podlagi informacij o ostalih zdravilih v bazi (prepričanje), v drugi pa neodvisnost na podlagi predhodnikov pravila (razlika med izmerjeno pojavitvijo pravila in pričakovano verjetnostnjo pravila).

Parametra a in b služita kot uteži. Z njimi lahko uravnavamo potencialno visoke vrednost prepričanja in nizke vrednosti razlike med izmerjeno pojavitvijo pravila in pričakovano verjetnostnjo pravila. V našem primeru se je kot dobra praksa izkazal sledeč izbor uteži, a v razponu med $[0.01, 0.1]$, b v razponu med $[5, 10]$.

4.2.4 Drevesa visoke donosnosti (High-yield partition tree)

Algoritem za sestavo drevesa visoke donosnosti je podoben binarnim particijskim drevesom, kjer vsako vozlišče predstavlja nek vzorec in hrani vse transakcije, ki vsebujejo ta vzorec. Slika 4.3 prikazuje primer takega drevesa z vzorci, ki smo jih dobili iz tabele 4.1.



Slika 4.3: Primer drevesa visoke donosnosti za primere iz tabele 4.1. Črni krogi predstavljajo SDP vzorec. Kvadrat označuje robna vozlišča. Oznaka P predstavlja vzorec tega vozlišča, D pa množico transakcij, ki zadostujejo vzorcu P.

Drevo zgradimo tako, da v koren vstavimo vse transakcije D. Koren predstavlja najbolj splošen vzorec $P_0 = [p_k = 0; 1 \leq k \leq M]$, kar pomeni, da nas ne zanimajo osnovni elementi. Prvotno je potrebno izbrati element i z nekim selekcijskim algoritmom (to bomo razložili kasneje). Z izbranim elementom sestavimo dva nova vzorca: P_1, ki vsebuje transakcije s prispevkom, ter P_2, ki vsebuje transakcije brez prispevka. Vzorec P_1 ostaja enak staršu, le da bo imel element p_i enak 1, vzorec P_2 pa bo imel p_i enak -1. Transakcije, ki prispevajo h koristnosti, razporedimo na levo stran drevesa, tiste, ki ne prispevajo, pa v desno stran. Takšen razcep naredimo v vsakem vozlišču drevesa. Vsak razcep ustvari bolj usmerjen in specifičen vzorec z manjšim številom transakcij D_k kot pri staršu. Prispevek podmnožice transakcij $C(D_k)$ se z vsakim vozliščem zmanjša, medtem ko vzorčno razmerje d_k raste.

Rekurzivni razcep izvajamo, dokler ni izpolnjen eden izmed dveh ustavitvenih

pogojev. Prvi pogoj je $C(D_k)/C(D) < \gamma$ kar pomeni, da je vzorec neznačilen in bodo tudi njegovi otroci neznačilni. Drugi pogoj je $C_p(D_k)/C(D_k) > \sigma$, kar pomeni, da je vzorec opredeljevalen oziroma SDP. Ob tem pogoju se rekurzija še ne ustavi, saj imajo lahko otroci tega vozlišča/vzorca še bolj specifične SDP-je oziroma višjo vzorčno razmerje. To lahko vodi do večjega donosa tega drevesa, zato je potrebno pregledati otroke tega vozlišča. Če sta oba otroka SDP vozlišča, razvijemo oba, saj je po definiciji njun vzorčni prispevek višji od očetovega. V primeru, da je samo eden otrok SDP, ga razvijemo le pod pogojem, da ima večji vzorčni prispevek od očeta. Vozlišča, ki ne ustrezajo pogoju $C_p(D_k)/C(D_k) > \sigma$ niso SDP in jih ne razvijemo. Zaradi teh lastnosti je struktura drevesa monotona in neprekinjena, kar pomeni, da bodo v listih vozlišča vedno SDP z najvišjim donosom.

Pri selekciji elementov oziroma razcepu vozlišča smo uporabili algoritem požrešnega iskanja po donosnosti (*Yield-Greedy*) in požrešnega iskanja s pogledom vnaprej (*Yield-Greedy with look ahead*). Prvi se sprehodi čez vse elemente in izbere tistega z največjim prispevkom, drugi pa razvije vozlišče po vseh kriterijih in predlaga očeta z najbolj obetavnim otrokom.

Postopek gradnje drevesa je zapisan v psevdokodi 2. Izrojeno drevo lahko še obrežemo, s čimer dobimo vzorce, ki prispevajo največ. Obrezovanje je iterativen postopek, ki je sestavljen iz dveh korakov prikazanih v psevokodi 3.

Algorithm 2 Gradnja drevesa visoke donosnosti.

function $tree\_building(\sigma, \gamma, C(D))$
  if $C(D_k)/C(D) > \gamma$ **then**
    Inicializiramo levega in desnega otroka
    e =Izberemo index najbolj obetavnega elementa za razcep v vzorcu. (Yield-Greedy)
    for all $T \in D$ **do** ▷ Za vsako transakcijo T, ki je v D.
      if $T[e] > 0$ **then**
        daj transakcijo levem sinu
      else
        daj transakcijo desnem sinu
      end if
    end for
    $left_{ratio} = C_p(D_k)/C(D_k)$ izračuna vzorčno razmerje levega sina.
    $right_{ratio} = C_p(D_k)/C(D_k)$ izračuna vzorčno razmerje desnega sina.
    if $left_{ratio} > \sigma$ **then**
      $sin_{levi}.tree_building()$
    end if
    if $right_{ratio} > \sigma$ **then**
      $sin_{desni}.tree\_building()$
    end if
  end if
end function

Algorithm 3 Obrezovanje drevesa visoke donosnosti.

function *tree_pruning*(*number_of_sdp*)
  while *number_of_sdp* < *current_number_of_sdp* **do**
    poišči robna vozlišča in nastavi operacije
    obrezovanje robnih vozlišč
    preštej število SDP vzorcev *current_number_of_sdp*.
  end while
end function

V prvem koraku je potrebno poiskati robna vozlišča. *Robno vozlišče* je vozlišče, ki vsebuje eno ali dve možni operaciji rezanja. SDP vzorec je robno vozlišče v primeru, ko ima v svojih poddrevesih točno dva lista. V tem primeru odstranimo vse potomce tega vozlišča, ta vozlišče posledično postane list. Normalno vozlišče je robno, če ima sina, ki je list, ali če ima sina, ki je SDP vzorec in ima med svojimi potomci le enega, ki je list. V tem primeru odstranimo leve ali desne potomce glede na izgubo vzorčnega razmerja najglobjih elementov. Izguba vzorčnega razmerja je enaka razliki vzorčnega razmerja najglobjega elemente in trenutnega robnega vozlišča.

Ko smo identificirali robna vozlišča in njihove operacije, jih moramo še izvesti. V vsakem robnem vozlišču izvedemo eno operacijo. Po izvršeni operaciji popravimo robno vozlišče nazaj na normalno. Zanko izvajamo dokler ne dobimo manjšega ali enakega števila želenih SDP vzorcev.

4.2.5 Prilagoditev in opis poteka delovanja algoritma

Originalni algoritem je, zaradi narave naših podatkov, potrebno prilagoditi. Na začetku poglavja smo omenili primer z obutvijo. Osnovni element je bil izdelek (določen model copatov), ki je imel atribute kot so cena, velikost, znamka in podobno. Zaradi narave teh podatkov je bilo relativno lahko sestaviti koristnostno funkcijo (npr. dobiček).

Naši podatki teh lastnosti nimajo in se je problema potrebno lotiti drugače. S

koristnostno funkcijo želimo poiskati stranske učinke zdravil. Zdravja/nezdravja ne moremo meriti, zato moramo poiskati funkcijo, ki bi lahko ujela podobno informacijo. Ker iščemo interakcije, bodo naši osnovni elementi/gradniki pravila oblike $X => Y$. V našem primeru ne bomo uporabili ostalih atributov, kot so količina zdravila, saj jih med zdravili ne znamo smiselno združevati.

Osnovna pravila lahko vsebujejo elemente iz vseh nivojev v hierarhij. Dobimo jih lahko iz algoritma posplošenih asociativnih pravil (po koraku kreiranja kandidatov in štetju podpore). Pri tem upoštevamo/dodamo že obstoječa pravila iz baze LexiComp. Transakcije so v tem primeru pacienti, predmeti pa so terapije in simptomi pacientov. Sestavimo matriko transakcij in za vsako pravilo izračunamo vrednosti s koristnostno funkcijo. To storimo zato, da lahko ocenimo celotni možni prispevek in donos podatkov.

Iz podatkov nato sestavimo drevo visoke donosnosti, s katerim dobimo množice vzorcev. Vzorci so sestavljeni iz osnovnih pravil, skupaj pa sestavljajo kompleksnejša pravila. Ta nova pravila/vzorce je potrebno nato predati specialistu, da preveri, ali so informativna. Psevdokoda 4 prikazuje celotni potek algoritma.

Algorithm 4 Psevdokoda algoritma za iskanje pravil s koristnostno funkcijo.

$\sigma = 0.85$ ▷ poljubna izbira parametrov σ, γ in $number\_of\_sdp$
$\gamma = 0.1$
$number\_of\_sdp = 5$
$rule\_lenght = 2$ ▷ Nastavimo maksimalno dovoljeno dolžino pravil, ki nam bodo služila kot osnovni gradniki.

$a = 0.01$ ▷ a in b sta parametra za koristnostno funkcijo.
$b = 10$

Izračun pravil z posplošenimi povezovalnimi pravili glede na $rule\_lenght$. Uporabimo lahko poljubni algoritem za gradnjo pravil.

Izračunamo koristnostno funkcijo pravil. Upoštevamo parametra a in b.

Seštejemo skupaj koristnostno funkcijo vseh pravil, $C(D)$.

$tree\_building(\sigma, \gamma, C(D))$. ▷ Zgradimo drevo
$tree\_pruning(number\_of\_sdp)$. ▷ Drevo obrežemo.
return *drevo*

Poglavje 5

Rezultati testiranj

Vsak algoritem smo testirali na umetnih in realnih podatkih. V tem poglavju prikažemo in komentiramo rezultate, ki smo jih dobili na različnih podatkih. Na koncu vsakega razdelka opišemo rezultate in sklepni komentar.

Na generiranih podatkih smo algoritme testirali tako, da smo v njih vstavili umetna pravila oziroma interakcije ter jih poskušali poiskati. Iskana pravila so bila:

- $P1 : B|05|X \wedge A|03|C => U_{sim}|54$
- $P2 : A|02|D \wedge A|10 \wedge B|06|A|A => H_{sim}|61|0$
- $P3 : R|06|A|D \wedge S|01|A \wedge D|07|B|A \wedge H|01 \wedge A_{sim}|22|9 => A_{sim}|57$
- $P4 : J|07|X \wedge B|01|A|D => Q_{sim}|18$

V oznaki elementa (npr. $B|06|A|A$), znak | predstavlja ločilo med dvema oznakama dveh različnih nivojev. Zapis lahko gledamo kot na pot, kjer koda na levi strani predstavlja višji nivo v ontologiji, koda na desni pa nižji. Prvi elementi z oznako $_{sim}$ pove da gre za hierarhijo bolezni. Poglejmo pravilo $P1$, oznaka $B|05|X$ predstavlja element, ki se nahaja v hierarhiji zdravil (ker prvi element ne vsebuje $_{sim}$) v korenu B. Ostali del oznake (sin 03, vnuk X) je pot, ki jo opravimo, da pridemo do elementa. Podroben opis najdete v poglavju Podatki, razdelek 3.3 in 3.4.

Pravilo P1 smo vstavili v 10% transakcij, P2 v 15%, P3 v 5% ter P4 v 1% transakcij v vsaki množici. Algoritme smo testirali na bazi 100 in 10000 primerov. V povprečju je imel vsak primer 14 predmetov. Za vsako podatkovno množico smo testirali več parametrov.

Pri posplošenih povezovalnih pravilih smo uporabili več vrednosti za minimalno podporo $(0.05, 0.07, 0.1)$, minimalno zaupanje $(0.6, 0.8, 1)$ in R-zanimivost $(1, 3, 5, 10)$. R-zanimivost pove, koliko je pravilo zanimivo, in jo določimo tako, da izmerimo pričakovano verjetnost pravila glede na hierarhijo ter dejansko frekvenco pravila. Pravilo je zanimivo, ko ima dejansko vrednost R-krat višjo od pričakovane verjetnosti.

Pri algoritmu s koristnostno funkcijo smo spreminjali parametre $\sigma = (0.5, 0.7, 0.9)$, $\gamma = (0.05, 0.1, 0.2)$ in število SDP vozlišč oziroma vzorcev $(1, 2, vsi)$. Vse vrednosti so bile izbrane empirično na podlagi predpostavk, ki jih bomo opisali kasneje.

Iz realnih podatkov generirana pravila so pregledali farmacevti, ki so pravila razvrstili v skupine in podali svoje strokovno mnenje. Pravila so bila klasificirana na *zanimiva*(Z), *potencialno zanimiva*(PZ), *pričakovana*(P), *trivialna*(T) ter *neopredeljena*(N). *Zanimiva pravila* so pravila, ki vključujejo dejavnike, ki jih običajno ne povezujemo s povečanim tveganjem za razvoj hiperkalemije. *Potencialno zanimiva* pravila bi se lahko izkazala kot zanimiva, vendar bi za vsako posamezno pravilo potrebovali temeljit pregled literature in baz. Ta pravila vključujejo zdravila, za katere v povzetkih temeljnih značilnosti zdravila ni naveden vpliv na elektrolitsko ravnovesje in plazemske koncentracijo kalija. *Pričakovana pravila* vsebujejo pravila, za katere je znano, da lahko povzročijo hiperkalemijo. *Trivialna pravila* povezujejo diagnoze in farmakoterapijo iste bolezni in so tako nezanimiva [27].

5.1 Posplošena povezovalna pravila

5.1.1 Umetno generirani podatki

Tabeli 5.1 in 5.2, ter sliki 5.1 in 5.2 prikazujejo rezultate na podatkovni množici s 10000 in 100 elementi. Vzemimo kot primer tabelo 5.1. Če pogledamo celico z

parameteri $podpora = 0.07$, $zaupanje = 0.8$ in $R = 3$ vidimo, da s to nastavitvijo dobimo 45 pravil, med katerimi smo odkrili pravilo $P1$ in $P2$. Pri nastavitvi parametrov na $podpora = 0.05$, $zaupanje = 0.6$ in $R = 5$ dobimo 1091 pravil, med katerimi so le bližnji predniki pravila $P1$ in $P2$, v celoti pa odkrijemo pravilo $P3$.

Vsaka tabela prikazuje število pravil, ki so bila odkrita in imena odkritih vstavljenih pravil. Nekatera so označena z apostrofom (') kar nakazuje, da vstavljeno prvotno pravilo ni bilo odkrito, temveč le njegov bližnji prednik. Prednik pravila je pravilo, ki namesto nekaterih originalnih elementov, vsebuje elemente višjega nivoja. Bližnji prednik pa je pravilo, ki vsebuje le elemente nivoja, ki je le za eno višji od osnovnega. Pravilo $P1 : B|05|X \wedge A|03|C => U_{sim}|54$ ima mnogo prednikov, vendar so od teh bližnji predniki le $B|05 \wedge A|03|C => U_{sim}|54$, $B|05|X \wedge A|03 => U_{sim}|54$, $B|05 \wedge A|03 => U_{sim}|54$. Zanje je razlika v nivoju le ena.

V tabeli 5.1 je bilo, ne glede na nastavljene parametre, pravilo P2 odkrito vedno, medtem ko pravilo P4 ni bilo odkrito nikoli. To je v skladu s pričakovanji, saj ima pravilo P2 minimalno podporo vedno nižjo od dejanske frekvence pravila. Enako velja za P4, le da je parameter minimalne podpore previsok. P1 se (z izjemo $R = 1$) nikoli ne pojavi pri zaupanju 1. Razlog so naključno generirana pravila, zaradi katerih to pravilo ne dobi 100% zaupanja (kar bi bilo verjetno resnično tudi pri realnih podatkih).

Najpogostejše pravilo je P2. Tega lahko v celoti odkrijemo pri parametru $R = 1$ ali $R = 3$. Višje kot se pomikamo po hierarhiji, večji je lahko prepad med izmerjeno in pričakovano verjetnostjo elementov. Zaradi tega pri vrednostih $R = 5$ ali $R = 10$ odkrijemo le bližnje prednike, saj je razlika med pričakovano in izmerjeno pojavitvijo elementov višja.

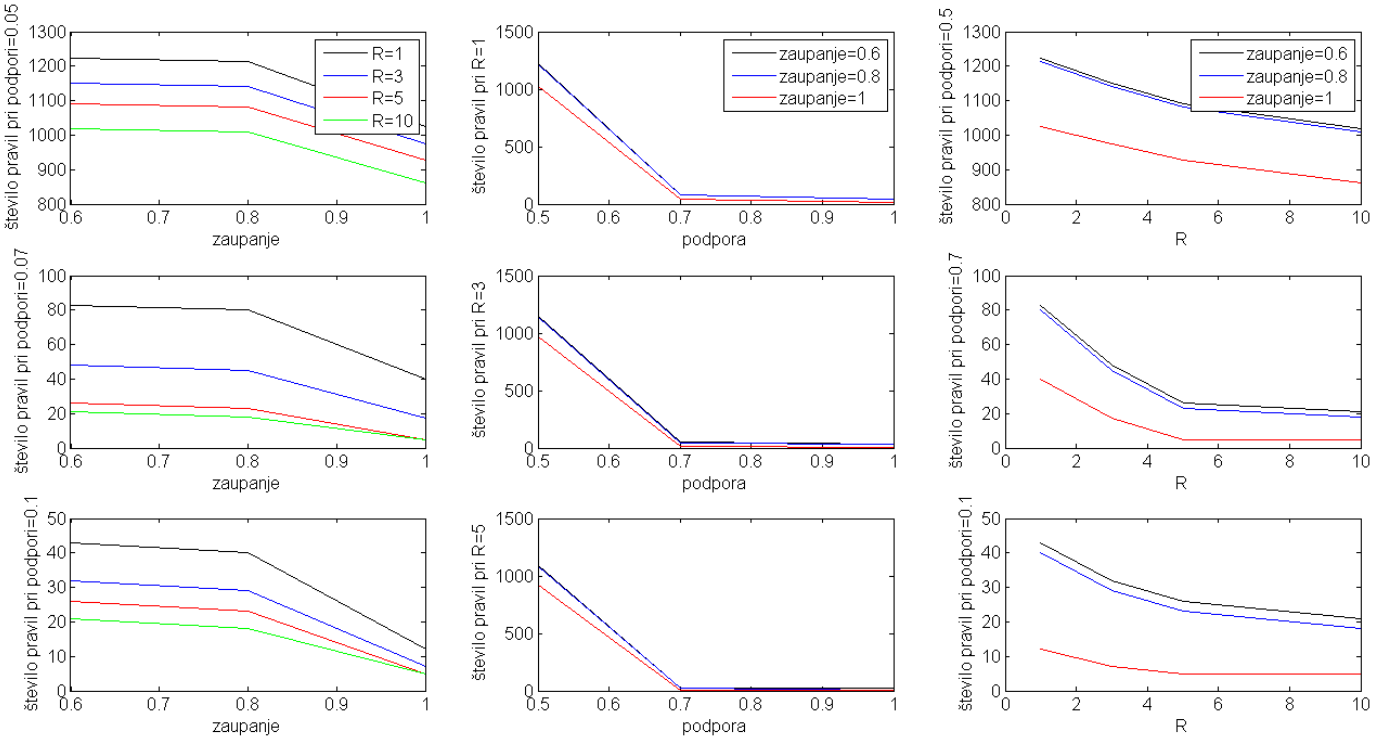
Zanimivo je pravilo P3, ki se vedno pojavi v celotni obliki. To je posledica števila elementov, ki je obratno sorazmerno s pričakovano verjetnostjo. Torej več kot imamo elementov, manjša je pričakovana verjetnost pravila. Posledica tega je, da potrebujemo visoke vrednosti R preden pravilo označimo za nezanimivo. V primeru pravila P3 je razvidno, da ga ni možno ovreči niti pri 10-kratni pričakovani

verjetnosti.

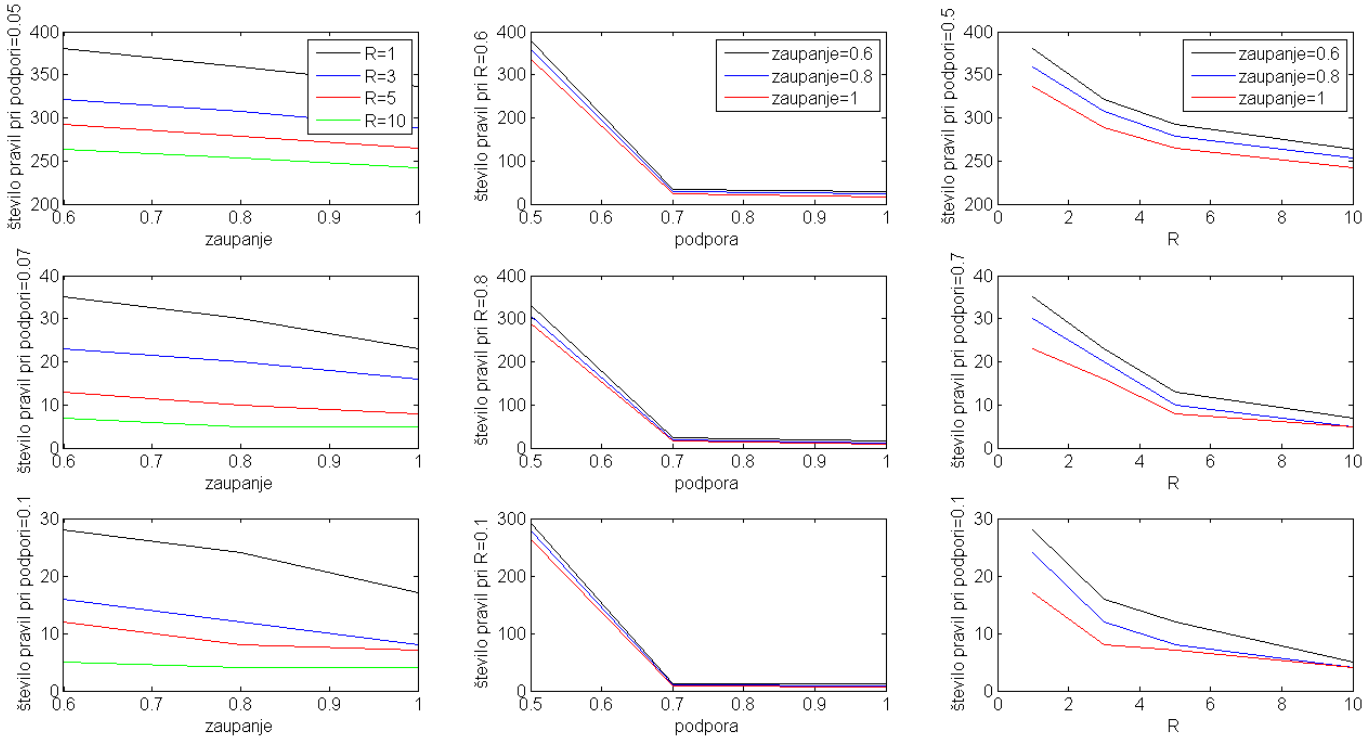
Pri parametru $R = 1$ so bila odkrita vsa pravila (P1,P2,P3), kar je v skladu s pričakovanji, glede na to, da je pri taki vrednosti parametra pravilo zanimivo že, če je frekvenca pravila višja od pričakovane verjetnosti.

Podpora		*0.05*			*0.07*			*0.1*		
R	***zaupanje***	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*
1		1222 P1,P2,P3	1212 P1,P2,P3	1026 P1,P2,P3	83 P1,P2	80 P1,P2	40 P1,P2	43 P1,P2	40 P1,P2	12 P1,P2
R	***zaupanje***	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*
3		1152 P1,P2,P3	1142 P1,P2,P3	973 P2,P3	48 P1,P2	45 P1,P2	17 P2	32 P1,P2	29 P1,P2	7 P2
R	***zaupanje***	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*
5		1091 P1',P2',P3	1082 P1',P2',P3	927 P2',P3	26 P1',P2'	23 P1',P2'	5 P2'	26 P1',P2'	23 P2'	5 P2'
R	***zaupanje***	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*
10		1018 P2',P3	1009 P2',P3	862 P2',P3	21 P2'	18 P2'	5 P2'	21 P2'	18 P2'	5 P2'

Tabela 5.1: Rezultati umetne podatkovne množice z 10000 primeri.



Slika 5.1: Grafi odvisnosti parametrov *zaupanje* (levi stran), *podpora* (sredina) in $R -$ *zanimivost* (desna stran) na število pravil na podatkovni množici z 10000 primeri (tabela 5.1).



Slika 5.2: Grafi odvisnosti parametrov *zaupanje* (levi stran), *podpora* (sredina) in R – *zanimivost* (desna stran) na število pravil na podatkovni množici z 100 primeri (tabela 5.2).

Tabela 5.2 prikazuje rezultate množice z 100 primeri. Podobno kot v tabeli 5.1 je bilo vedno odkrito pravilo P2, medtem ko pravilo P4 ni bilo nikoli odkrito. Zaradi manjšega števila podatkov se pravilo P2 v celoti odkrije le pri parametru $R = 1$ oziroma pri R vrednosti manjši od 3. Število podatkov vpliva tudi na ostala pravila, kot je na primer P3. V večji podatkovni množici smo vedno odkrili celotno pravilo, medtem ko v manjši pri $R = 5$ ali $R = 10$ odkrijemo le bližnje prednike.

Kot smo predvideli imajo naključni podatki pri manjšem številu podatkov večji vpliv. Tudi parametri imajo večji vpliv pri manjši množici podatkov. Če primerjamo rezultate med tabelam pri *podpori* = 0.07 in $R = 3$, opazimo, da s temi nastavitvami odstranimo pravilo P1. Prav tako je razvidno, da parameter R vpliva močneje, saj enake vrednosti odstranijo več pravil kot v 5.1.

Podpora		*0.05*			*0.07*			*0.1*		
R	***zaupanje***	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*
1		380 P1,P2,P3	359 P1,P2,P3	336 P1,P2,P3	35 P1,P2	30 P1,P2	23 P1,P2	28 P1,P2	24 P1,P2	17 P1,P2
R	***zaupanje***	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*
3		322 P1',P2,P3	307 P1',P2,P3	289 P1',P2,P3	23 P2'	20 P2'	16 P2'	16 P2'	12 P2'	8 P2'
R	***zaupanje***	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*
5		293 P2',P3'	279 P2',P3'	265 P2',P3'	13 P2'	10 P2'	8 P2'	12 P2'	8 P2'	7 P2'
R	***zaupanje***	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*	*0.6*	*0.8*	*1*
10		264 P2',P3'	254 P2',P3'	242 P2',P3'	7 P2'	5 P2'	5 P2'	5 P2'	4 P2'	4 P2'

Tabela 5.2: Rezultati umetne podatkovne množice z 100 primeri.

Sklepni komentar. Na umetnih podatkih smo odkrili, da so rezultati v skladu z našimi pričakovanji. Pričakovali smo, da bosta parameter minimalna podpora in zaupanje delovala enako na vstavljena pravila, kot bi v algoritmu Apriori. Pravilo z nižjo podporo od minimalne je odstranjeno, enako velja za minimalno zaupanje. Po pričakovanju se obnaša tudi R zanimivost, saj pri visokih vrednostih odstrani veliko statistično nepomembnih pravil.

Na število pravil ima največji vpliv minimalna podpora (slika 5.1 in 5.2): manjša, kot je vrednost podpore več imamo pravil in obratno. Zaupanje le šibko vpliva na število vzorcev, vendar s tem parametrom ohranimo pravila, ki so bolj prepričljiva in točna. R-zanimivost se izkaže kot dobra mera za odstranjevanje naključnih in nezanimivih pravil. Vrednost R je potrebno pazljivo nastaviti, saj velika vrednost, kot je na primer $R = 10$ odstrani močna pravila ali pa jih pretirano posploši.

Izbira parametrov je odvisna od lastnosti podatkov. V našem primeru je priporočljiva izbira parametrov za iskanje redkih pravil. Uporabili smo minimalno podporo 0.05, minimalno zaupanje okoli 0.8 ter R-zanimivost med 1 in 3. Minimalna podpora mora biti nizka, da lahko odkrijemo več redkih pravil, zaupanje mora biti visoko, da se izognemo naključnim pravilom. R mora biti relativno nizek, da ne postanejo pravila preveč splošna.

5.1.2 Realni podatki

Pri realnih podatkih smo pravila sestavili na sledeče načine:

- s hierarhijama ATC in MKB,
- s hierarhijo ATC,
- s hierarhijo MKB,
- brez hierarhij.

Razlog za takšno razdelitev je možnost za odkrivanje pravil med eno hierarhijo in listi druge. Sestavili smo tudi pravila brez hierarhij, da primerjamo naša pravila z

tistimi, ki jih odkrije algoritem Apriori. Pravila smo iskali z minimalno podporo 4%, z minimalnim zaupanjem 60% ter zanimivostjo $R = 3$. Za takšno podporo smo se odločili, ker želimo odkriti redkejša pravila in interakcije. Ker so pravila pregledali in ovrednotili farmacevti, smo želeli primerno majhno število teh pravil, po možnosti čimbolj zanimivih za ljudi. Točnost pravila mora biti vsaj 60% in pravilo mora biti 3 krat bolj pogosto od pričakovane pogostosti.

Odkrita pravila smo posredovali farmacevtom. Farmacevti so za vsako pravilo preiskali literaturo in ostale vire, ter na podlagi njihovih strokovnih argumentov določili klasifikacijo.

Tabela 5.3 prikazuje statistiko klasifikacij na vseh množicah pravil. Statistika je v skladu z našimi pričakovanji. Delež znanih in trivialnih pravil je visok, medtem ko je delež naključnih, zanimivih in potencialno zanimivih pravil nizek.

Baza | Klasifikacije	***Z***	***PZ***	***P***	***T***	***N***	***Skupaj***
ATC in MKB	18 (2.8%)	71 (11.3%)	24 (3.8%)	442 (70%)	76 (12.1%)	*630*
brez ATC	7 (10.1%)	0 (0%)	21 (31%.)	24 (35.2%)	16 (23.7%)	*68*
brez MKB	0 (0%)	3 (20%)	4 (27%)	3 (20%)	5 (33%)	*15*
brez ATC in MKB	0 (0%)	0 (0%)	0 (0%)	16 (18%)	69 (82%)	*85*

Tabela 5.3: Statistika razvrstitve pravil. Oznake klasifikacij so: Z - zanimivo pravilo, PZ - potencialno zanimivo, P - pričakovano, T - trivialno ter N - neopredeljeno pravilo.

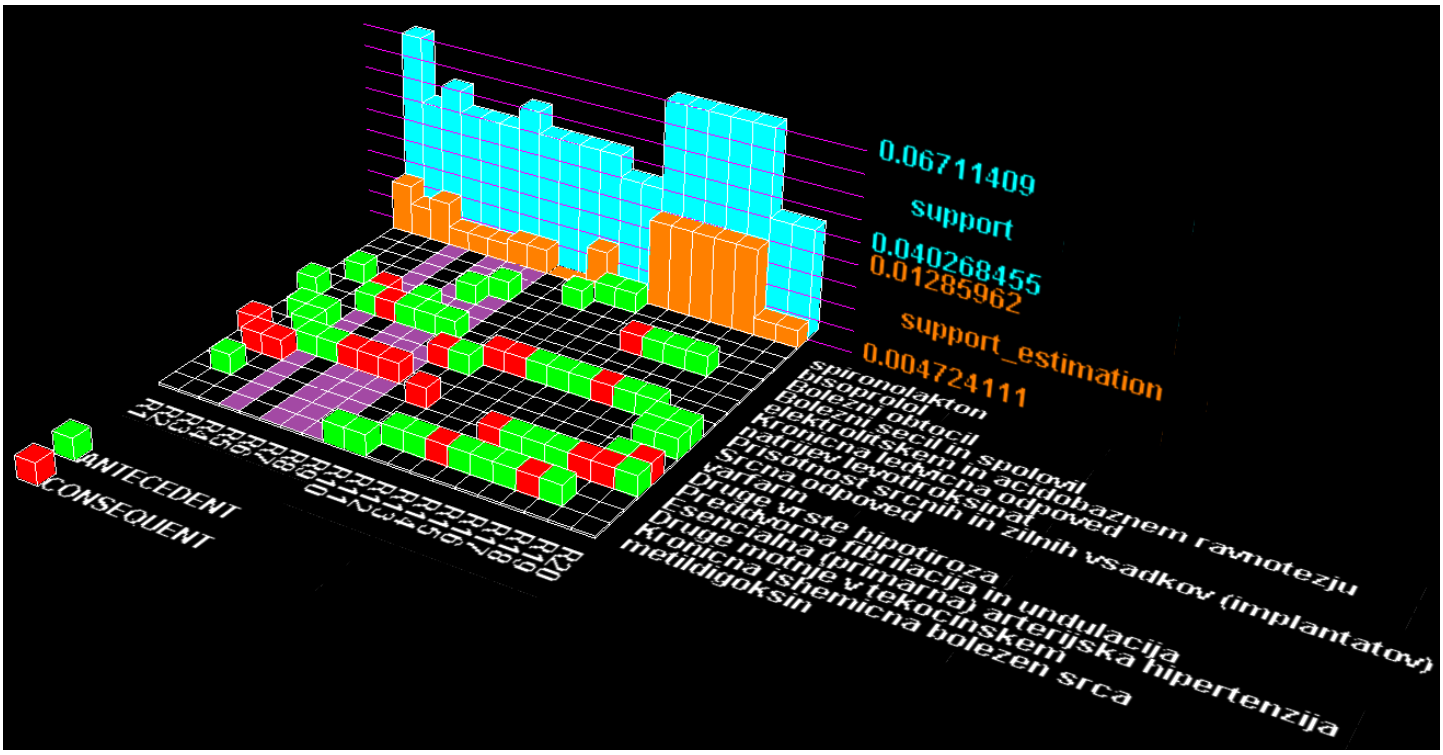
Množica pravil z ATC in MKB hierarhijami vsebuje 8283 pravil. Od tega je bilo, po zaporednem vrstnem redu pravil, pregledanih 7.6%, torej 630 pravil. Od pregledanih je bilo 2.8% (18 pravil) razvrščenih med zanimiva, 11.3% (71 pravil) med potencialno zanimiva, 3.8% (24 pravil) med pričakovana, 70% (442 pravil) kot trivialna ter 12.1% (76 pravil) kot neopredeljena. To, da odkrivamo trivialna in pričakovana pravila, nakazuje na to, da je algoritem sposoben odkrivati smiselne vzorce. Pričakujemo lahko, da algoritem zmore odkriti tudi farmacevtom neznane vzorce.

Primer odkritega zanimivega pravila je *kronična ledvična odpoved* $\wedge$ *acetilsalicilna kislina* $=>$ *kronična ishemična bolezen srca* [27], ki ima podporo 4% in zaupanje 66.6%. Zanimivo je bilo tudi pravilo *bolezni obtočil* $\wedge$ *druge vrste hipotiroza* $\wedge$ *hormonska zdravila za sistemsko zdravljenje* $=>$ *bolezni sečil in spolovil* [27], z podporo 4%, ter zaupanje 85.7%. Razlaga teh pravil pa ni v sklopu te naloge .

Neopredeljena pravila so večinoma splošna ali naključna. Če bi zmanjšali parameter R, bi naključnih pravil bilo več. R-zanimivost se izkaže kot učinkovita metoda za odstranjevanje naključnih pravil. Odkrili smo nekaj zanimivih in potencialno zanimivih pravil, skupaj 15.1% od vseh pregledanih.

Množica pravil brez ATC hierarhije je zanimiva, ker se lahko osredotočimo na vrednotenje vpliva posameznih učinkovin. Množica ima 68 pravil. Kot je opisano v tabeli 5.3 je bilo 10.1% razvrščenih kot zanimivih, 0% kot potencialno zanimivih, 31% kot pričakovanih, 35.2% trivialnih ter 23.7% neopredeljenih.

Slika 5.3 prikazuje nekaj pravil, med katerimi so z vijolično barvo označena štiri zanimiva pravila. Vsa pravila so v stolpcih matrike in so označena z R1-R15. Vrstice predstavljajo posamezne elemente. Stolpec oziroma pravilo, ki ima zeleno kocko v določeni vrstici predstavlja element, ki je na levi strani pravila. Rdeča kocka predstavlja element, ki je na desni strani pravila (posledica). Višina stolpcev v grafu prikazujejo izmerjeno pogostost pravila (support, modra barva), ter pričakovano verjetnost pravila (support_estimation, oranžna barva). V tej množici so zanimiva pravila R4, R6, R7 ter R8, ki povezujejo hipoterozo s kronično ledvično odpovedjo in boleznimi sečil. Pri hipotirozi sta ledvična okvara in elektrolitsko ne-



Slika 5.3: Prikaz pravil, ki vsebuje štiri zanimiva pravila pa tudi nezanimiva in že znana pravila. Zanimiva pravila so označena z vijolično barvo.

ravnovesje redko klinično izraženi in zato v literaturi redko opisani [27]. Izkaže se, da vizualizacija s slike 5.3, ki smo jo zgenerirali s programom ARViewer [22] ni posebej informativna, saj je interakcije težko razlikovati od šuma in je potrebna interpretacija pravil s strani strokovnjakov.

Potencialno zanimiva pravila so v tej množici tista, ki povezujejo acetilsalicilno kislino (ASA) z boleznimi srca in ožilja, endokrinimi boleznimi, boleznimi sečil in spolovil ter s perindoprilom. Algoritem je zaznal povezavo med ASA in drugimi zdravili zato, ker se to zdravilo pogosto predpisuje in je zato veliko število bolnikov s tem potencialnim neželenim učinkom oz. interakcijo [27].

Množica pravil brez MKB hierarhije vsebuje le 15 pravil, od tega so potencialno zanimiva tri pravila, štiri pričakovana, tri trivialna in pet neopredeljenih. Farmacevti so to podatkovno množico ocenili kot nezanimivo, saj pravila vsebujejo

preveč specifične posledice in simptome [27].

Množica pravil brez hierarhij vsebuje 85 pravil, pri katerih se kot posledica pojavi 5 bolezni, ki so bile najpogosteje diagnosticirane pri bolnikih, ki smo jih vključili v našo raziskavo. Od 85 pravil ima 69 pravil za posledico arterijsko hipertenzijo, ki jo je imelo 86 od vključenih 150 bolnikov. Arterijska hipertenzija je v 32 pravilih povezana le z enim dejavnikom tveganja, zato je klasificiranje teh pravil nesmiselno [27].

Sklepni komentar. Rezultate smo dobili s parametri $podpora = 0.04$, $zaupanje = 0.6$ ter zanimivost $R = 5$. Glede na dobljene rezultate lahko zaključimo, da je algoritem posplošenih povezovalnih pravil uporaben. Delovanje algoritma potrjuje tudi velik delež pričakovanih (P) in trivialnih (T) pravil. Ta pravila so sicer v breme pri interpretaciji s strani strokonjakov, vendar bi za njihovo avtomatsko identifikacijo potrebovali poglobljeno domensko znanje, ki pa ni na voljo. V primeru nedelovanja algoritma, bi bil ta delež bistveno manjši in bi imeli večji delež naključnih pravil. Algoritem je odkril zanimiva in potencialno zanimiva pravila, ki so kandidati za neznane interakcije. Farmacevti bodo ta pravila še podrobneje preučili.

Najbolj uporabna podatkovna množica je bila z ATC in MKB hierarhijo, saj vključuje vse informacije, ki so nam bile na voljo. V njej smo pregledali 7.6% pravil in zaznali 18 zanimivih in 71 potencialno zanimivih pravil. Dobre rezultate smo dobili tudi v podatkih brez ATC hierarhije, saj v njih lahko vrednotimo vpliv posameznih učinkovin. V tej množici smo odkrili sedem zanimivih pravil. Pravila iz podatkov brez MKB hierarhij so bila nezanimiva in niso vsebovala zanimivih pravil. Isto velja za pravila, ki so bila pridobljena brez vseh hierarhij, saj so vsebovala le znana dejstva.

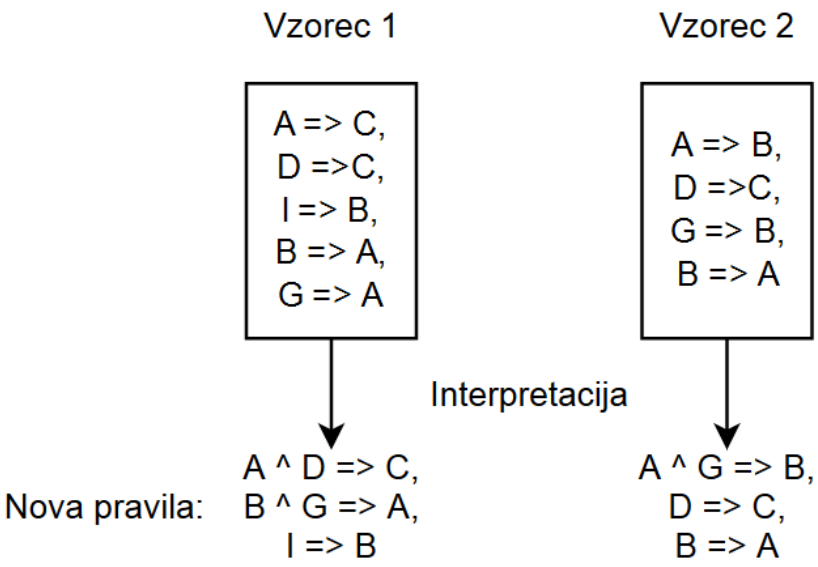
Podatki iz baze LexiComp niso vplivali na dobljene rezultate. Pravila, ki niso imela zadosti podpore v podatkih in so bila zapisana v bazi LexiComp, smo uporabili v nadaljnjih iteracijah algoritma. Izkazalo pa se je, da so bila vsa ta pravila, v nadaljnjih iteracijah gradnje pravil zavržena.

5.2 Pravila na podlagi funkcije koristnosti

5.2.1 Umetno generirani podatki

Kot osnovo smo vzeli pravila, ki so bila sestavljena z algoritmom za posplošena povezovalna pravila. Uporabili smo pravila z dolžino dva in tri (všteta je tudi posledica). To so osnovni gradniki, s katerimi sestavljamo kompleksnejša pravila. Za posplošena pravila smo nastavili minimalno podporo na 4%, minimalno zaupanje na 60% ter R-zanimivost na 1.75. Za to podporo smo se odločili, ker smo želeli dobiti več osnovnih pravil dolžine 2 ali 3. Vrednost R smo nastavili na 1.75, saj smo s tem odstranili nekaj redundantnih in neinformativnih pravil. Tako smo zmanjšali število kandidatov iz nekaj tisoč na nekaj sto.

Vsako drevo visoke koristnosti vrne vzorce pravil. Vzorce je možno interpretirati na več načinov. Eden od načinov je, da v vsakem vzorcu pregledamo vsa pravila ter jih združimo glede na posledico, pri tem pa ne združujemo elementov med vzorci. Ta interpretacija vzorcev se izkaže kot zanimiva, saj grupira le elemente, ki so v skupnem pravilu. Primer take interpretacije kaže slika 5.4. Neoptimalni parametri vplivajo na interpretacijo tako, da ta vsebuje več enakih elementov iz različnih nivojev v hierarhiji.



Slika 5.4: Primer interpretacije vzorcev.

Slabost takega pristope je relativno malo število pravil, redundantni elementi in velik delež suboptimalnih rešitev kot prikazuje primer v tabeli 5.4. Iz primera je razvidno, da je algoritem odkril pravili $P1$ in $P3$, ki smo ju definirali na začetku poglavja. Algoritem ni odkril optimalne rešitve $P1 : B|05|X \wedge A|03|C => U_{sim}|54$, temveč le bližnja soseda $R_1 : A|03 \wedge B|05 => U_{sim}$ in $R_4 : A|03|C \wedge G \wedge B \wedge B|05 \wedge A => U_{sim}$, ki sta suboptimalni rešitvi. Primer R_4 vsebuje tudi redundantne elemente npr. G, B in A, ki so preveč splošni, da bi bili uporabni.

Tabeli 5.5 in 5.6 prikazujeta nekaj osnovnih informacij o dobljenih rezultatih na umetno generiranih podatkih. Vzemimo za primer celico z določitvenim parametrom $\sigma = 0.7$, značilnostjo $\gamma = 0.1$ in s številom SDP vzorcev 2. Prva vrednost v celici prikazuje število najdenih vzorcev, druga pa prikazuje najdena pravila.

Rezultati so v skladu s pričakovanji. V tabeli 5.5 smo, z izjemo parametrov $\sigma = 0.5$, dobili vsa pravila, glede na parameter SDP. Ko je $SDP = 1$, smo dobili pravila, v kateri se nahaja pravilo $P3$, medtem ko smo za $SDP = 2$ dobili pravila, ki vsebujejo vstavljena pravila $P2$ in $P3$. Ob vrnitvi vseh vzorcev (drevo ni bilo

obrezano), smo dobili vse vzorce, kar je v skladu s pričakovanji.

Zaradi majhne količine podatkov, to ni najbolj razvidno iz tabele 5.6, ki prikazuje rezultate le za 100 transakcij. V tej množici imajo parametri večji vpliv in jih je potrebno nastaviti bolj previdno. Iz tega lahko sklepamo, da je uspešnost iskanje z našo koristnostno funkcijo pogojena tudi s številom podatkov. To bi bilo smiselno, saj naša funkcija upošteva statistične informacije o pravilih kot sta prepričanje (conviction) in razmerje med pričakovano verjetnostjo in dejansko pogostostjo pravila. Pri manjši količini podatkov je lahko ta funkcija bolj občutljiva.

Pravila	Elementi						
R_1	$A\|03$	$B\|05$				==>	U_{sim}
R_2	$H\|01$	D	$D\|07\|B$	$R\|06\|A\|D$	$S\|01\|A$	==>	$A_{sim}\|57$
R_3	$H\|01$	D	$S\|01$	$R\|06\|A\|D$	H	==>	$A_{sim}, A_{sim}\|22\|9$
R_4	$A\|03\|C$	G	B	$B\|05$	A	==>	U_{sim}

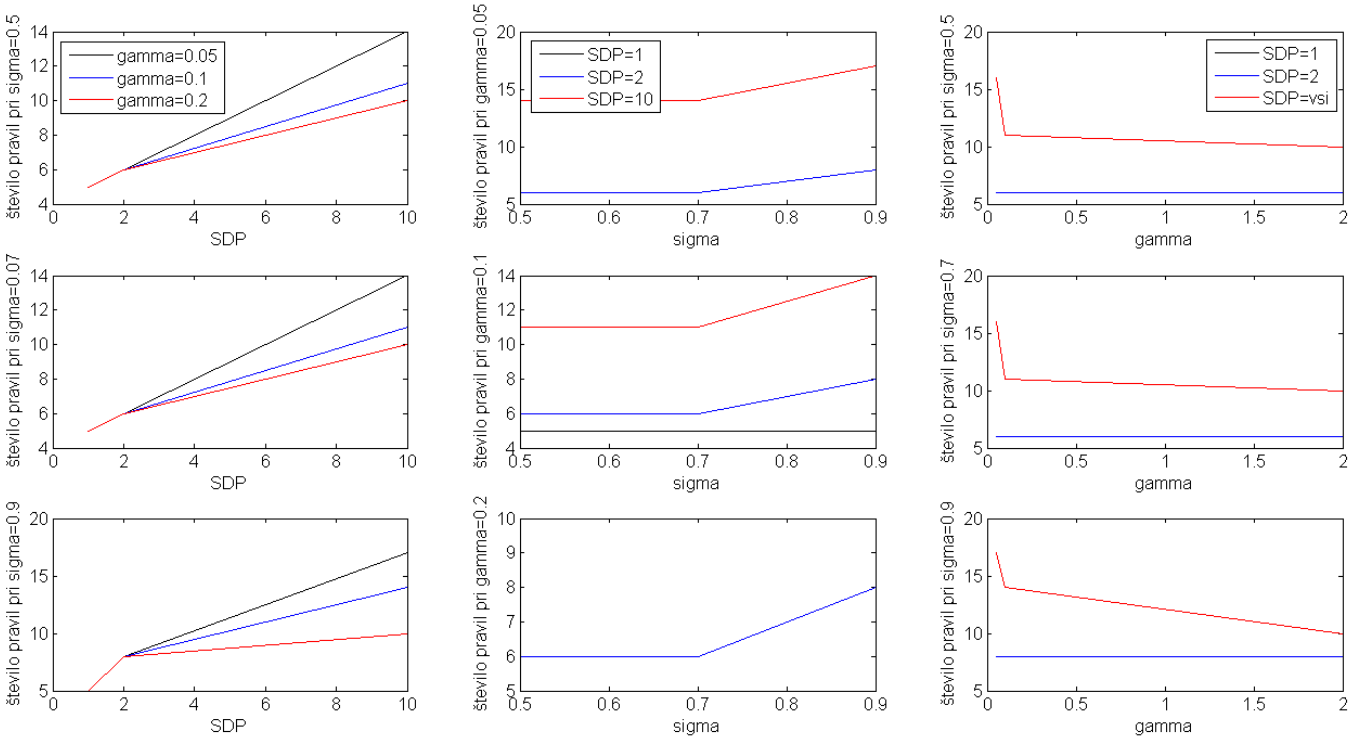
Tabela 5.4: Primer pravil, ki jih generira algoritem s koristnostno funkcijo.

σ		*0.5*			*0.7*			*0.9*		
γ	***SDP***	*1*	*2*	*vsi*	*1*	*2*	*vsi*	*1*	*2*	*vsi*
0.05		5 P3	6 P3	14 P1,P2,P3	5 P3	6 P2,P3	14 P1,P2,P3	5 P3	8 P2,P3	17 P1,P2,P3
γ	***SDP***	*1*	*2*	*vsi*	*1*	*2*	*vsi*	*1*	*2*	*vsi*
0.1		5 P3	6 P3	11 P1,P2,P3	5 P3	6 P2,P3	11 P1,P2,P3	5 P3	8 P2,P3	14 P1,P2,P3
γ	***SDP***	*1*	*2*	*vsi*	*1*	*2*	*vsi*	*1*	*2*	*vsi*
0.2		5 P3	6 P3	10 P1,P2,P3	5 P3	6 P2,P3	10 P1,P2,P3	5 P3	8 P2,P3	10 P1',P2,P3

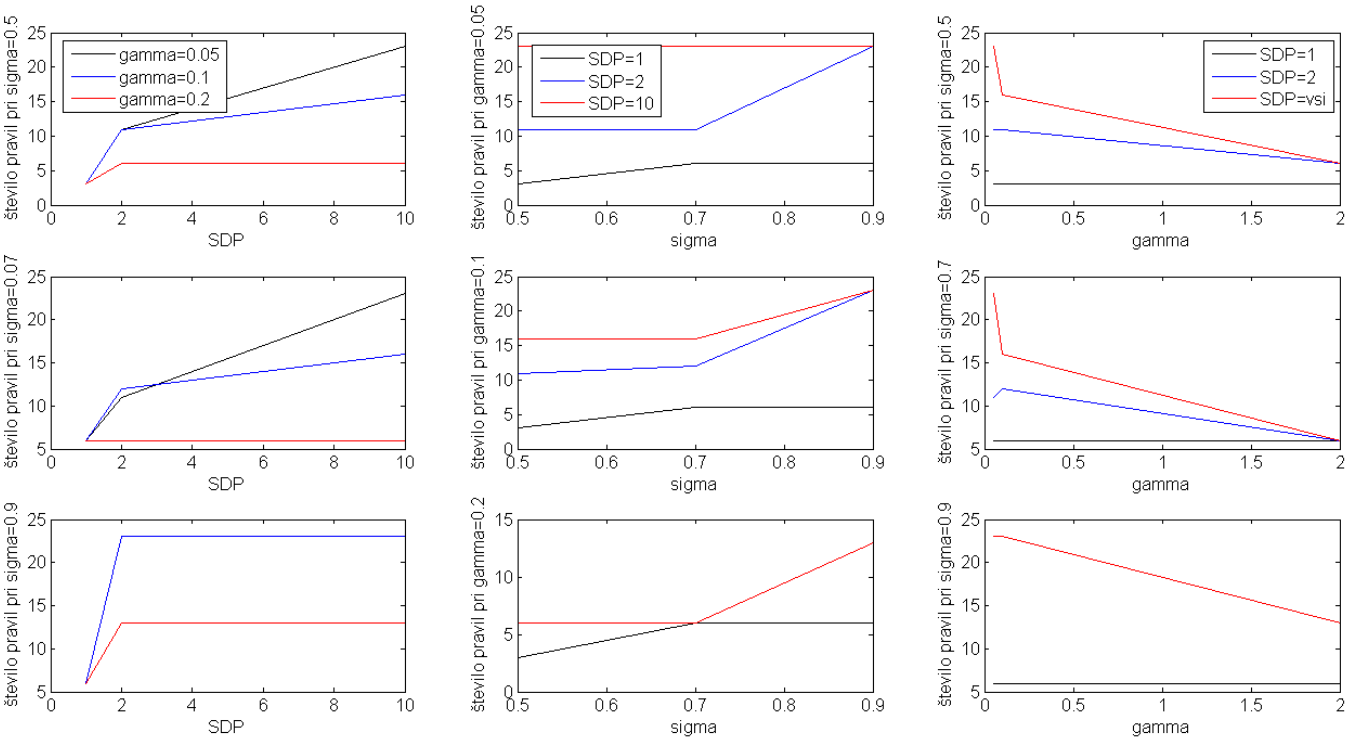
Tabela 5.5: Odkrita pravila s funkcijo koristnosti na umetni podatkovni množici s 10000 transakcijami.

σ		*0.5*			*0.7*			*0.9*		
γ	***SDP***	*1*	*2*	*vsi*	*1*	*2*	*vsi*	*1*	*2*	*vsi*
0.05		3 P2’	11 P2,P3’	23 P1’,P2,P3	6 P2’	11 P2,P3’	23 P1’,P2,P3	6 P2’	23 P1’,P2,P3	23 P1’,P2,P3
γ	***SDP***	*1*	*2*	*vsi*	*1*	*2*	*vsi*	*1*	*2*	*vsi*
0.1		3 P2’	11 P2,P3’	16 P2,P3	6 P2’	12 P2,P3	16 P1’,P2,P3	6 P2’	23 P1,P2,P3	23 P1,P2,P3
γ	***SDP***	*1*	*2*	*vsi*	*1*	*2*	*vsi*	*1*	*2*	*vsi*
0.2		3 P2’	6 P2	6 P2	6 P2’	6 P2’	6 P2’	6 P2’	13 P1’,P2’	13 P1’,P2’

Tabela 5.6: Odkrita pravila s funkcijo koristnosti na umetni podatkovni množici s 100 transakcijami.



Slika 5.5: Grafi odvisnosti parametrov SDP (levi stran), σ (sredina) in γ (desna stran) na število pravil na podatkovni množici z 10000 primeri (tabela 5.5).



Slika 5.6: Grafi odvisnosti parametrov *zaupanje* (levi stran), σ (sredina) in γ (desna stran) na število pravil na podatkovni množici z 100 primeri (tabela 5.6).

Sklepni komentar. Iz tabel 5.5 in 5.6 ter grafov 5.5 in 5.6 vidimo, da na število pravil najmočneje vpliva parameter števila SDP vzorcev. Več SDP vzorcev pomeni več posledic, po katerih združujemo elemente v pravila in posledično več pravil. SDP vzorec nastavimo glede na pričakovano število zanimivih pravil v domeni. Na število pravil vpliva tudi parameter σ. Manjša σ pomeni večje število potencialnih vzorcev in temu primerno večje število pravil. Parameter γ vpliva na kvaliteto iskanih pravil in deluje podobno kot minimalna podpora. To je najbolj razvidno iz tabele 5.6 pri $\gamma = 0.2$, kjer nismo skoraj nikoli v celoti odkrili nobenega pravila, ampak le bližnje sorodnike pravila $P2$, ki je najbolj pogosto. Ko je število podatkov majhno, je potrebno γ nastaviti bolj previdno, saj previsoka vrednost pomeni preveliko posploševanje, medtem ko prenizka pomeni večjo količino šuma, ki ga zaznamo.

Iz rezultatov je razvidno, da algoritem deluje, saj se vstavljena pravila pojavijo glede na število SDP vzorcev, ki jih želimo imeti. Algoritem pravilno združuje elemente, kar potrjuje uspešnost naše interpretacije vzorcev. Ugotovili smo tudi, da je naša funkcija koristnosti zaradi statističnih informacij bolj občutljiva na šum pri manjši količini podatkov. Zato je pri manjših podatkih potrebno testirati več parametrov, preden pridemo do smiselnega nabora pravil.

Priporočljiva kombinacija parametrov na veliki množici je $\sigma = 0.9$, γ na razponu med $[0.05, 0.1]$, ter SDP, ki naj bo enak pričakovanemu številu pravil v domeni. Pri manjših množicah je priporočljiva kombinacija parametrov $\sigma = 0.9$, $\gamma = 0.1$, ter SDP glede na pričakovano število pravil v domeni.

5.2.2 Realni podatki

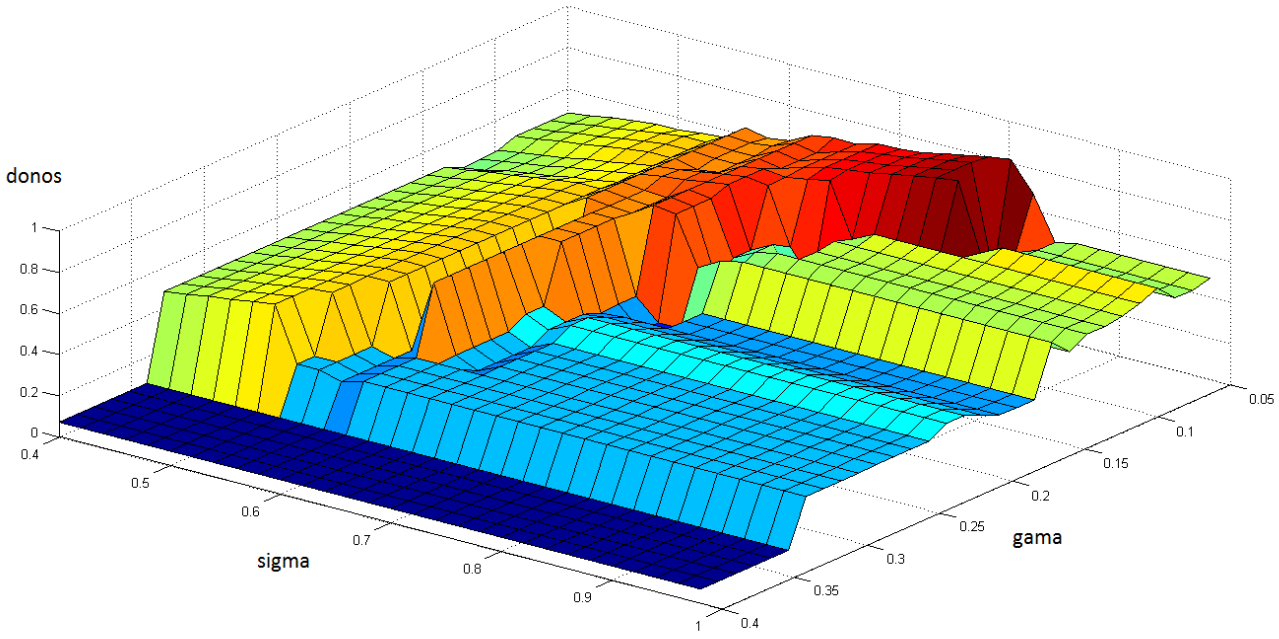
Za realne podatke smo algoritem pognali s sledečimi parametri:

- $pravila_1 : \sigma = 0.6, \gamma = 0.07, SDP = 5$
- $pravila_2 : \sigma = 0.6, \gamma = 0.08, SDP = 10$
- $pravila_3 : \sigma = 0.6, \gamma = 0.08, SDP = 20$
- $pravila_4 : \sigma = 0.8, \gamma = 0.1, SDP = 10$

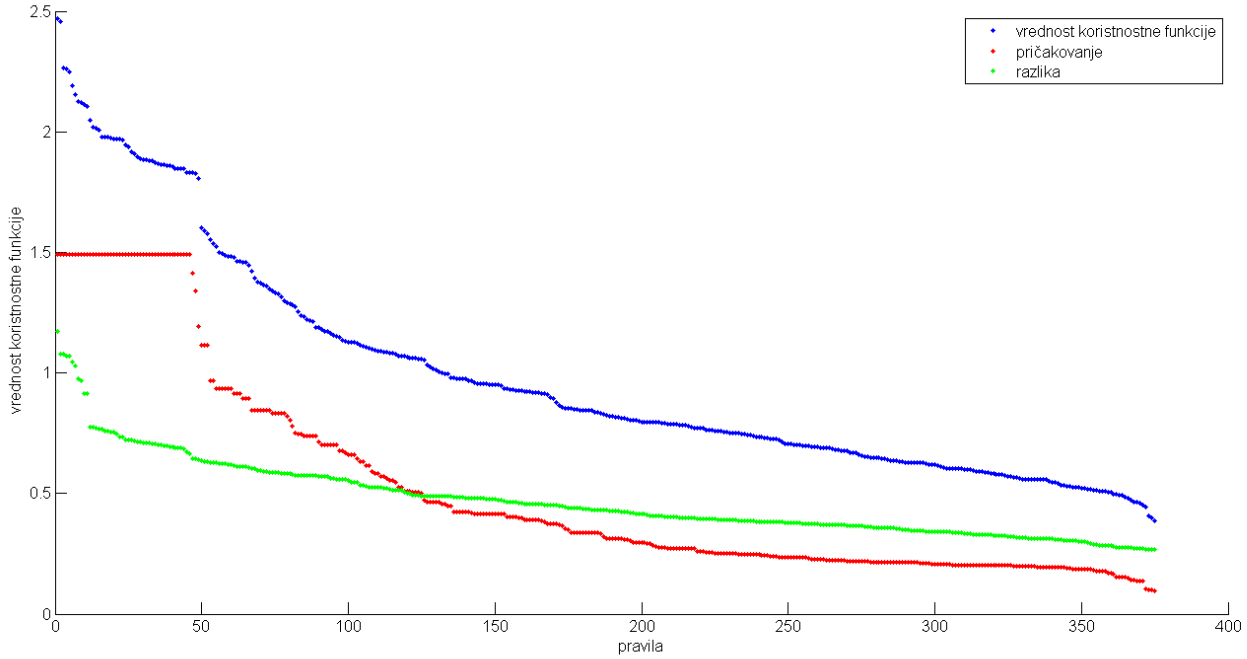
- $pravila_5 : \sigma = 0.8, \gamma = 0.07, SDP = 5$
- $pravila_6 : \sigma = 0.9, \gamma = 0.1, SDP = 5$
- $pravila_7 : \sigma = 0.9, \gamma = 0.1, SDP = 10$
- $pravila_8 : \sigma = 0.65, \gamma = 0.09, SDP = 10$

Pri računanju koristnostne funkcije smo uporabili parameter $a = 0.1$ in $b = 10$, kjer je a utež prvega dela koristnostne funkcije (prepričanje), b pa drugega (razlika med pričakovano verjetnostjo in dejansko pojavitvijo).

Parametre σ,γ in SDP smo izbrali na podlagi slike 5.7. Slika prikazuje skupno donosnost dreves za različne vrednosti σ in γ pri fiksnem parameteru SDP 10. Parameter SDP smo fiksirali, ker vpliva predvsem na vrednost donosa in manj na topologijo v sliki 5.7. Izjema je le donos z vsemi SDP vzorci, ki vsebujejo vsa pravila (glede na pogoj σ in γ) in tako dosega svoj maksimum. Iz slike opazimo, da je najvišji donos za σ v razponu med $[0.7, 0.85]$ ter γ v razponu med $[0.05, 0.1]$. Parametre smo izbrali v tem razponu.



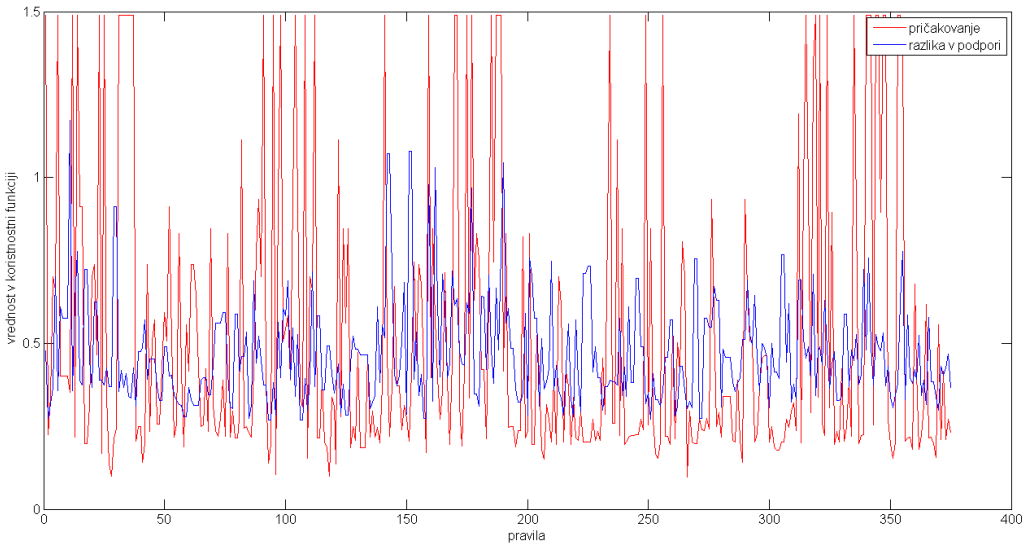
Slika 5.7: Donosnost dreves glede na različne parametre pri desetih SDP vzorcih.



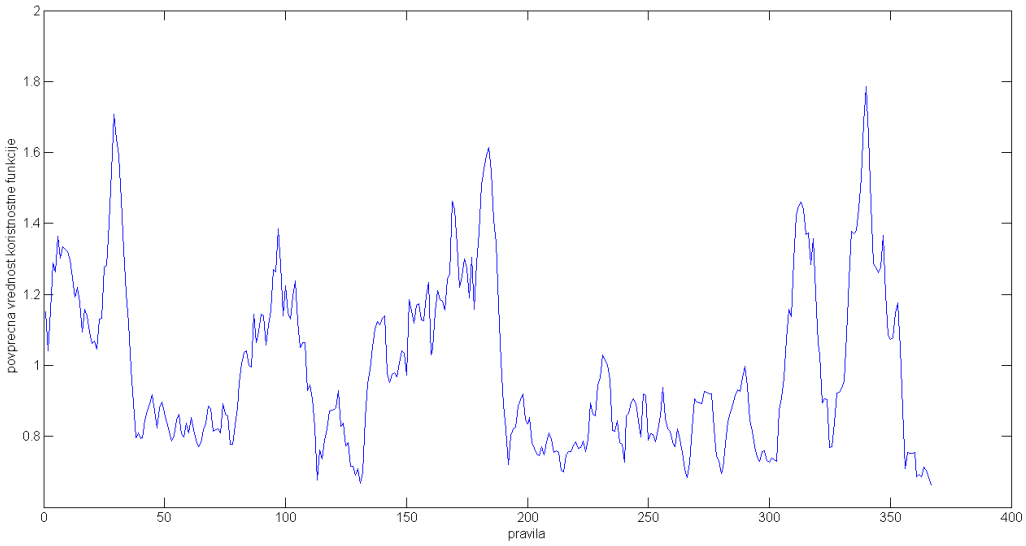
Slika 5.8: Prikaz pravil urejenih po vrednostih koristnostne funkcije (modra črta). Rdeča črta predstavlja vrednosti pričakovanja (2.5), zelena predstavlja razliko med pričakovano in dejansko podporo pravil [4.9].

Za nastavitev parametrov a in b smo izrisali graf koristnostne funkcije slikah 5.8, 5.9, in 5.10. Slike prikazujejo vrednosti in obliko koristnostne funkcije na realnih podatkih.

Parametra a in b smo nastavili tako, da imata funkcija prepričanja (conviction) in funkcija razlike med pričakovano verjetnostjo in dejansko pojavitvijo pravila podoben razpon vrednosti, kot prikazuje slika 5.9. Močna pravila bodo tako verjetno imela močno prepričanje in večjo razliko med pričakovano in dejansko podporo. Na slikah 5.9 in 5.10 takšna pravila opazimo med pravili 0 in 50, med 150 in 200 ter med 300 in 350. To je boljše razvidno iz slike 5.10, ki prikazuje povprečene vrednosti koristnostne funkcije. Za vsako pravilo smo izračunali povprečno vrednost njenih petih sosedov in jo shranili. Tako smo dobili bolj razvidna odstopanja.



Slika 5.9: Vrednosti posameznih delov koristnostne funkcije (4.9). Rdeča črta predstavlja vrednosti pričakovanja (2.5), modra predstavlja razliko med pričakovano in dejansko podporo pravil [4.9]. Za izris smo uporabili uteži $a = 0.1$ in $b = 10$.



Slika 5.10: Povprečene vrednosti koristnostne funkcije z oknom desetih pravil.

Tabela 5.7 prikazuje statistiko pravil generiranih s koristnostno funkcijo. Nekatera pravila pridobljena s tem algoritmom, so kompleksna in vključujejo do 27 elementov. Klasificiranja vseh rezultatov algoritma s koristnostno funkcijo ni bilo mogoče izvesti, saj bi zaradi kompleksnosti pravil farmacevti potrebovali več časa, da ovrednotijo ta pravila. Klasificirana so bila le pravila do dolžine šestih elementov. Pravila, ki so bila bolj kompleksna, smo označili s K [27].

Farmacevti so potrdili, da se pravila po večini prekrivajo s tistimi, ki so bila odkrita z posplošenimi pravili na podatkih z ATC in MKB hierarhijo. To lahko smatramo kot dokaz za pravilno delovanje tega algoritma. Glede na procent zanimivih in potencialno zanimivih pravil so bili najboljši parametri v množici $pravila_3$ s 29.7%, $pravila_6$ s 44% ter $pravila_6$ s 40% teh pravil.

Iskanje pravil s koristnostno funkcijo sestavlja kompleksna pravila, za katere bi bilo vrednotenje daljše. V povprečju je delež kompleksnih pravil 35%. Največ kompleksnih pravil je v tretji in sedmi množici, ki imata tudi največ pravil in veliko število SDP vzorcev. Množica $pravila_6$ ima zanimivo statistiko, saj je pri $SDP = 5$, algoritem odkril velik delež zanimivih in potencialno zanimivih pravil (v primerjavi z množico $pravila_1$ in $pravila_5$, ki imata enako število SDP vzorcev). Zanimivo je, da so parametri $pravila_5$ na najvišji točki slike 5.7 in bi pričakovali najboljšo statistiko. Iz tega lahko sklepamo, da ne moremo sklepati na dobro kombinacijo parametrov, le glede na donos drevesa. prav tako pa se izkaže, da topologija na sliki 5.7, ni primerljiva s topologijo, ki bi jo dobili ob fiksiranju parametra SDP na neko drugo vrednost npr. $SDP = 5$.

Zanimivo je, da algoritem ni odkril veliko neopredeljenih pravil. To je dobro, saj je večina tako klasificiranih pravil naključna. Delež pričakovani in trivialnih pravil je visok, saj enako kot pri generaliziranih pravilih, algoritem odkriva znane interakcije. Velik delež poznanih in trivialnih pravil pa ponovno potrjuje, da algoritem lahko odkrije informativna pravila in vzorce v realnih podatkih in ne le umetnih.

Baza* | *Klasifikacije	***Z***	***PZ***	***P***	***T***	***N***	***K***	***Skupaj***
$pravila_1$	0 (0%)	1 (14.2%)	1 (14.2%)	2 (28.5%)	0 (0%)	3 (43.1%)	*7*
$pravila_2$	0 (0%)	3 (23.3%)	2 (15%)	3 (23.3%)	0 (0%)	5 (38.4%)	*13*
$pravila_3$	1 (3.7%)	7 (26%)	2 (7%)	7 (26%)	0 (0%)	10 (37.3%)	*27*
$pravila_4$	1 (7%)	3 (21%)	4 (28%)	2 (16%)	1 (7%)	3 (21%)	*14*
$pravila_5$	1 (16.66%)	1 (16.66%)	0 (0%)	1 (16.66%)	0 (0%)	3 (50%)	*6*
$pravila_6$	1 (7%)	5 (37%)	3 (21%)	1 (7%)	1 (7%)	3 (21%)	*14*
$pravila_7$	0 (0%)	4 (16%)	2 (8%)	7 (28%)	2 (8%)	10 (40%)	*25*
$pravila_8$	0 (0%)	6 (40%)	0 (0%)	4 (26.66%)	0 (0%)	5 (33.33%)	*15*

Tabela 5.7: Statistika klasifikacije pravil za algoritem s koristnostno funkcijo. Oznake klasifikacij so: Z - zanimivo pravilo, PZ - potencialno zanimivo, P - pričakovano, T - trivialno, N - neopredeljena pravila ter K - kompleksna pravila.

Sklepni komentar rezultatov. Algoritem s koristnostno funkcijo je relativno uspešen, saj se pravila po večini prekrivajo z posplošenimi povezovalnimi pravili. Algoritem s koristnostno funkcijo pravilno izbira in združuje elemente vendar, jih težje interpretirati, saj je veliko pravil kompleksnih. Če upoštevamo rezultate glede na število najdenih zanimivih in potencialno zanimivih primerov, lahko zapišemo, da je najboljša izbira parametrov za σ v razponu med $[0.05, 0.1]$, za γ v razponu $[0.8, 0.9]$ ter SDP med $[5, 10]$.

Primerjava algoritmov. Iskanje povezovalnih pravil s koristnostno funkcijo je v primerjavi z posplošenimi povezovalnimi pravili zahtevnejše za uporabo. Število parametrov, ki jih moramo nastaviti je večje, saj je potrebno dobiti gradnike (osnovna pravila) z drugim algoritmom, nato pa je potrebno definirati še koristnostno funkcijo in jo prilagoditi za dane podatke, ter nastaviti parametre γ,

σ in število SDP vzorcev. Algoritem za iskanje posplošenih pravil ima le parametre minimalne podpore zaupanja in R-zanimivosti.

Glede kakovosti rezultatov ne moremo jasno določiti, kateri algoritem je boljši, saj za iskanje pravil s koristnostno funkcijo farmacevti niso uspeli pregledati kompleksnih pravil, ki bi lahko bila zanimiva. Ta algoritem sestavlja sicer manj pravil, vendar so ta kompleksnejša od posplošenih povezovalnih pravil, ki sestavljajo več enostavnejših pravil. Kompleksna pravila so potencialno zaželjena lastnost pri iskanju interakcij, saj ni možno predvideti, koliko dejavnikov vpliva na neko stanje, hkrati pa slabo vplivajo na razumljivost in na človeško zmožnost, da si jih razloži. V nadaljevanju bi bilo zato smiselno v koristnostno funkcijo tudi neposredno vključiti kriterij kompleksnosti.

Poudariti je potrebno, da se zanimiva in potencialno zanimiva pravila pri obeh algoritmih prekrivajo. Število odkritih pravil je večje pri iskanju posplošenih pravil (v primerjavi z iskanjem s koristnostno funkcijo), vendar to ne pomeni nujno, da gre za različna pravila, ampak so lahko le bližnji predniki drugih pravil. To lahko daje vtis, da je več zanimivih pravil, čeprav gre le za izpeljave enega samega pravila.

Ugotovili smo, da je predlagana koristnostna funkcija občutljiva na število podatkov in da se posplošena povezovalna pravila na manjših množicah podatkov izkažejo kot bolj primerna izbira.

Poglavje 6

Zaključek

V magistrski nalogi smo iskali interakcije med zdravili, ki so jih prejemali bolniki s hiperkaliemijo. S pomočjo algoritmov, ki upoštevajo informacije iz ontologij (ATC in MKB) in obstoječih baz (LexiComp), smo prišli do zanimivih rezultatov. Ugotovili smo, da se algoritma dobro izkažeta na umetno generiranih in na realnih podatkih. Oba sta odkrila nekaj zanimivih in potencialno še neznanih pravil, ki jih je bo potrebno podrobneje preučiti.

Prednost algoritma posplošenih pravil je majhno število parametrov, enostavna pravila in tako hitrejše vrednotenje in interpretacija pravil, ter manjša občutljivost na manjšo količino podatkov. Slabost algoritma je velika količina pravil, ki jih algoritem generira.

Prednost iskanja pravil s funkcijo koristnosti je majhno in obvladljivo število pravil, uporaba statističnih informacij in sestava kompleksnih pravil. Slabost algoritma je veliko število parametrov, ki jih moramo nastaviti, občutljivost na manjšo količino podatkov ter večji čas za vrednotenje kompleksnih pravil.

Vsa pravila, ki niso imela zadosti podpore, vendar so bila zabeležena v bazi LexiComp, so bila izločena med delovanjem algoritma posplošenih povezovalnih pravil. Tako se je izkazalo, da baza LexiComp ni vplivala na rezultate naše naloge.

Oba algoritma delujeta podobno in pravila, ki jih odkrijeta, se v večji meri prekrivajo. Iz farmacevtskega stališča se izkaže, da je bolj praktičen algoritem posplošenih povezovalnih pravil, saj so pravila, ki jih algoritem generira enostavnejša

in potrebujemo manj časa za pregled in ovrednotenje [27].

Iz podatkov 150 bolnikov smo s pomočjo algoritmov zaznali nove napovedne kriterije in povezave med dejavniki tveganja za hiperkaliemijo, ki jih nismo pričakovali, in so lahko v pomoč pri izdelavi postopkov, ki bodo prispeval k večji varnosti pri zdravljenju bolnikov.

Za nadaljnje delo bi bilo potrebno razmisliti, kako upoštevati in obravnavati velikosti prejetih odmerkov zdravil, ki jih zaradi uporabe hierarhij ne moremo le seštevati, saj imajo zdravila različne učinkovine, pa tudi standarden odmerek zdravil je različen. Potrebno bo razviti še drugačne koristnostne funkcije in primerjati njihovo delovanje z našo funkcijo. Funkcije bi morale temeljiti na statistiki pravil in hierarhij in ne le pogostosti interakcij. Zanimivo bi bilo uporabiti še druge hierarhije, naprimer DRON (Drug Ontology) na platformi Ontobee [26], ki vsebuje kemične lastnosti zdravil.

Literatura

[1] J. Hu, A. Mojsilovic. "High-Utility pattern mining: A Method for discovering of high-utility item sets", *Pattern Recognition*, št. 40, str. 3317–3324, 2007.

[2] R. Srikant, R. Agrawal. "Mining generalized association rules", *Future Generation Computer Systems*, št. 13, str. 161–180, 1997.

[3] R. Srikant, Q. Vu, R. Agrawal. "Mining Association Rules with Item Constraints", *Association for the Advancement of Artificial Intelligence*. KDD-97 Proceedings, 1997.

[4] R. Srikant, R. Agrawal. "Fast algorithms for mining association rules in large databases", *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, št. 20, str. 487–499, 1994.

[5] Guilherme del Fiol, B. Rocha, G. J. Kuperman, D. W. Bates, P. Nohama. *Comparison of Two Knowledge Bases on the Detection of Drug-Drug Interactions*. AMIA inc., 2000.

[6] R. Edwards, J. K. Aronson. "Adverse Drug Reactions: Definition, Diagnosis, and Management", *The Lancet*, št. 356, str. 1255–1259, 2000.

[7] P. Manda, J. Ozkan, H. Wang, F. McCarthy. "Cross-Ontology Multi-Level Association Rule Mining in the Gene Ontology", *PLoS ONE*, št. 10, zv. 7, 2012.

[8] B. Jayanthi, K. Duraiswamy. "A Novel Algorithm for Cross Level Frequent Pattern Mining in Multidata sets", *International Journal of Computer Applications*, št. 37, zv. 6, str. 30–35, 2012.

[9] P. Harrington. *Machine Learning in Action*. Manning Publications Co, 2012.

[10] B. F. Palmer. "Managing hyperkalemia caused by inhibitors of the renin-angiotensin-aldosterone system", *The New England Journal of Medicine*, št. 351, str. 585–592, 2004.

[11] K. Takaichi. "Analysis of factors causing hyperkalemia", *Internal medicine*, št. 46, str. 823–829, 2007.

[12] A. Lehnhardt, M. Kemper. "Pathogenesis, Diagnosis and management of hyperkalemija", *Pediatric Nephrology*, št. 26, str. 377–384, 2011.

[13] D. B. Mount, K. Zandi-Nejad. "Disorders in potassium balance", *Brenner and Rector's The Kidney, 9th Edition*, št. 9, str. 640–678, 2012.

[14] D. B. Mount. *Causes and evaluation of hyperkaliemia in adults*. UpToDate, 2013.

[15] M. Batlouni. "Nonsteroidal anti-inflammatory drugs: cardiovascular, cerebrovascular and renal effects", *Arquivos brasileiros de cardiologia*, št. 94, str. 556-563, 2010.

[16] Richard, Sterns. *NSAIDs: Electrolyte complications*. UpToDate, 2013.

[17] A. R. Temple. "Acute and chronic effects of aspirin toxicity and their treatment", *Archives of internal medicine*, št. 141, str. 364-369, 1981.

[18] B. Joshi, D. Jones, A. Rochford, L. Giblin. "Hypothyroidism and associated acute renal failure", *Journal of the Royal Society of Medicine*, št. 102, str. 199-200, 2009.

[19] G. Basu, A. Mohapatra. "Interactions between thyroid disorders and kidney disease", *Indian journal of endocrinology and metabolism*, št. 16, str. 204, 2012.

[20] M. Chonchol, G. Lippi, G. Salvagno, G. Zoppini, M. Muggeo, G. Targher. "Prevalence of subclinical hypothyroidism in patients with chronic kidney disease", *Clinical Journal of the American Society of Nephrology*, št. 3, str. 1296-1300, 2008.

[21] J. Kovač, D. Kovač, J. Lindič, M. Malovrh, J. Pajek "Motnje v presnovi kalija", *Bolezni ledvic. 2. izdaja*, str. 369-376, 2009.

[22] D. Delautre and S. Demay (2014) Dostopno na:
http://www2.lifl.fr/ jourdan/download/arv.html [dostop: 23.9.2014]

[23] World Health Organization (2010) Dostopno na:
http://www.who.int/classifications/icd/en/ [dostop: 24.9.2014]

[24] World Health Organization Collaborating Centre for Drug Statistics Methodology (2011) Dostopno na:
http://www.whocc.no/atc/structure_and_principles/ [dostop: 24.9.2014]

[25] LexiComp, Wolters Kluwer Health (2014) Dostopno na:
http://www.lexi.com/ [dostop: 24.9.2014]

[26] OntoBee (2014) Dostopno na:
http://www.ontobee.org [dostop: 6.9.2014]

[27] Petra Volk Markovič, Andreja Čufar: "Analiza rezultatov posplošenih povezovalnih pravil in povezovalnih pravil s funkcijo koristnosti na bazi bolnikov s hiperkaliemijo." Osebna komunikacija, avgust in september 2014.