РЕПУБЛИКА МАКЕДОНИЈА

УНИВЕРЗИТЕТ
"СВ. КИРИЛ И МЕТОДИЈ"

ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

Индуктивна околина за учење на дрва за предиктивно групирање

ДИПЛОМСКА РАБОТА

Ментор:
проф. Д-р Сузана Лошковска

Кандидат:
Драги Коцев

Коментор:
проф. Д-р Сашо Џероски

СКОПЈЕ

СОДРЖИНА

1. ВОВЕД

Количеството податоци што се чува во компјутерските документи се зголемува брзо, а корисниците на податоците очекуваат пософистицирани информации од нив. Во денешно време, маркетинг менаџерот не се задоволува со едноставен список на потрошувачката, туку бара и детална информација за минатите купувања на потрошувачите и предвидувања за тоа какви ќе бидат идните. Едноставните прашања од структурираните прашални јазици не се доволни за да се одговори на зголемената побарувачка за информации. Податочното рударење често се дефинира како барање на скриени информации во базите на податоци. Алтернативно, се нарекува и експлораторна анализа на податоците, откривање водено од податоците или дедуктивно учење.

Податочното рударење вклучува многу алгоритми со цел да оствари различни задачи. Сите алгоритми се обидуваат да добијат соодветен модел за да ги опишат податоците. Алгоритмите ги испитуваат податоците и определуваат модел кој е најблизок до карактеристиките на податоците што се испитуваат. Следните три дела ги карактеризираат алгоритмите за податочно рударење: модел (целта на алгоритмот е да даде модел кој ги опишува податоците), приоритет (мора да се употребуваат некои критериуми за да се избере најсоодветниот модел) и пребарување (сите алгоритми имаат соодветна техника за пребарување на податоците).

Податочното рударење може да даде два вида модели: предиктивни (предвидувачки) и дескриптивни (описни). Предиктивниот модел ги предвидува вредностите на податоците со употреба на веќе познатите резултати кои се добиени од различни податоци. Предиктивното моделирање може да се базира на употреба на други поранешни податоци. Податочното рударење кое враќа предиктивен модел ги употребува следните задачи: класификација, регресија, анализа на временски серии и предвидување. Дескриптивниот модел идентификува шаблони или поврзаност на податоците. За разлика од предиктивните модели, дескриптивните служат за истражување на особините на испитуваните податоци, а не за да предвидат нови особини. Дескриптивното моделирање ги има следните задачи: кластерирање, сумирање, асоцијациски правила и откривање на секвенци (прикажано на Слика 1.1).

Слика 1.1. Модели и задачи на податочното рударење

Еволуцијата на функциите на податочното рударење е резултат на долгогодишното влијание на повеќе дисциплини, што е прикажано на Слика 1.2, како бази на податоци, обработка на податоци, статистика, алгоритми и машинско учење. Други гранки од компјутерските технологии што имаат огромно влијание врз развојот на податочното рударење се мултимедијата и графиката.

Слика 1.2. Влијанија врз податочното рударење

Историските влијанија што придонеле за еволуцијата на податочното рударење се:

- **Индукција:** се употребува за добивање на општа информација од специфично знаење. Овој приод е присутен во апликации од вештачка интелигенција.
- **Компресија:** примарна цел на податочното рударење е да се опишат некои особини на множество од податоци со употреба на општ модел. Овде се извлекуваат деталните податоци од базата на податоци и се компресираат во помал опис на особини на податоците кои ги има во моделот.
- **Прашување:** процесот на податочно рударење може да се смета како еден вид на прашување на основната база на податоци. Еден правец на развој на податочното рударење е како да се дефинираат прашањата за податочно

рударење и дали прашален јазик (како SQL) може да се развие за да се опфатат различни видови прашања за податочно рударење.

- **Апроксимација:** Опишувањето на голема база на податоци може да се сфати како употреба на апроксимација за да се откријат скриени информации за податоците.
- **Пребарување:** Кога се работи за големи бази на податоци, влијанието на големината и ефикасноста од развивањето на апстрактен модел може да се сфати како проблем со пребарувањето.

Мерењето на ефикасноста и корисноста од податочното рударење не е едноставно. Од гледна точка на бизнисот и вкупна корисност, како мерка за ефикасноста на податочното рударење може да се користи враќањето на инвестицијата. Враќањето на инвестицијата ја испитува разликата од цената на податочното рударење и заштедата или заработката од употребата на податочно рударење.

Во следните години се очекува да се развијат поефикасни алгоритми кои ќе имаат подобри кориснички интерфејси, но и да се развие еден глобален модел за податочното рударење. Можеби нема да изгледа како релациониот модел, но веројатно ќе вклучува слични делови: алгоритми, податочен модел и оценка за квалитет. Сегашните алатки за податочно рударење бараат голема интеракција од човекот не само за да се дефинира барањето, туку и за да се интерпретираат податоците. Со развојот на алатките, со нивното подобрување и интегрирање, човечката интеракција се намалува. Апликациите за податочно рударење се од различен тип, па пожелен е развојот на глобален модел за податочно рударење.

2. ОСНОВНИ ЗАДАЧИ НА ПОДАТОЧНОТО РУДАРЕЊЕ

Влез во алгоритам за податочно рударење е табела која има одреден број полиња (колони) и записи (редици). Секоја редица претставува еден објект, а колоните претставуваат особини на објектите. Во терминологијата на машинското учење, редиците се нарекуваат примероци, а колоните атрибути (или особини). Атрибутите кои имаат нумеричка (реална) вредност се нарекуваат континуални атрибути, а оние кои имаат номинална вредност се нарекуваат дискретни атрибути.

Излез од алгоритам за податочно рударење е шаблон или множество од шаблони кои се валидни за дадените податоци. Шаблонот се дефинира како израз во даден јазик што ги опишува податоците во подмножество од дадените податоци и е поедноставен од набројување на сите податоци од подмножеството. Типични шаблони се равенките, класификациските и регресиските дрва и асоцијациските, класификациските и регресионите правила. Даден алгоритам за податочно рударење ќе има вградена класа на шаблони кои претпоставуваат дека разгледуваниот јазик од шаблони ќе зависи од дадените податоци (атрибутите и нивните вредности).

Многу алгоритми за податочно рударење се од полето на машинското учење и статистиката. Често гледиште во машинското учење е дека алгоритмите за машинско учење пребаруваат (најчесто хеуристички) низ просторот на хипотези (шаблони) кои ги објаснуваат податоците. Слично, на алгоритмите за податочно рударење можеме да гледаме како на пребарување, темелно или хеуристичко, низ просторот на шаблони, со цел да се најдат шаблони што се валидни за дадените податоци.

2.1 КЛАСИФИКАЦИЈА И РЕГРЕСИЈА

Задача на класификацијата и регресијата е да ја предвидат вредноста на еден атрибут од вредноста на останатите атрибути. Целниот атрибут се нарекува класа (или во статистичка терминологија-зависна променлива, а останатите атрибути се независни променливи). Ако класата е континуална, се врши регресија, а ако класата е дискретна (има конечно множество од номинални вредности), се врши класификација. Во двата случаи, влез е множество од податоци, а се генерира излезот-модел (шаблон или множество од шаблони). Добиениот модел може да се користи да се предвидат вредностите на класата за новите податоци.

Од дадено множество податоци (табела) само еден дел се користи за да се генерира (индуцира, научи) предиктивниот модел. Овој дел се нарекува множество за обучување. Останатиот дел се користи за евалуација на предвидувачките можности на наученият модел и се нарекува множество за тестирање. Множеството за тестирање се користи за да се проценат можностите на моделот на нови, неупотребувани податоци, или со други зборови, да се процени валидноста на шаблонот на нови податоци.

Препознавањето на шаблон е вид на класификација каде влезниот шаблон е класифициран во една од неколкуте класи врз основа на сличноста со предефинираните класи.

2.2 КЛАСТЕРИРАЊЕ

Кластерирањето (групирањето) ги групира објектите во класи од слични објекти. Кластер (група) е колекција од објекти кои се слични едни на други, а различни во однос на објектите од другите кластери (групи). За дадено множество од податоци, целта на кластерирањето е да ги подели примероците во подмножества (кластери). Целта е да се постигне голема сличност меѓу објектите во рамките на кластерот (интеркласна сличност), а мала сличност меѓу објектите од различни кластери (интракласна сличност).

Во статистиката кластерирањето е познато како анализа на кластери, во маркетингот и менаџментот на клиентите како сегментација на клиентите и во машинското учење како учење без надзор. Конвенционалното кластерирање се концентрира на анализата на кластери врз основа на растојание. Ознаката за растојание (или обратно, сличност) е многу важна: објектите се сметаат дека се точки во метричкиот простор (простор со мерка за растојание). Во концептуалното кластерирање, се добива и симболична репрезентација на добиените кластери, како додаток на делењето во кластери: заради ова може секој кластер да се смета за концепт (слично како кај класата во класификацијата).

Специјален тип на кластерирање е сегментацијата. Со сегментација, базата на податоци се дели на различни групи кои се нарекуваат сегменти и имаат слични торки. На сегментацијата често се гледа како на идентична со кластерирањето, други пак велат дека сегментацијата е специфичен вид на кластерирање кое се применува врз базите на податоци.

2.3 АНАЛИЗА НА АСОЦИЈАЦИИ

Анализа на асоцијации е откривање на асоцијациските правила. Силна мотивација за развој на анализата на асоцијации е анализата на потрошувачката кошничка. Асоцијациските правила ја специфицираат поврзаноста меѓу фреквентните множества (множества од продукти, како леб и путер, кои често се заедно во потрошувачката кошничка).

Анализата на асоцијации се извршува во два чекора. Прво, се наоѓаат сите фреквентни множества, каде едно множество е фреквентно ако се појавува во барем даден процент s од сите трансакции (s се нарекува поддршка). Следно, се наоѓаат асоцијациските правила во облик $X \rightarrow Y$, каде X и Y се фреквентни подмножества и довербата во правилото го поминува прагот c кој е процент од трансакциите кои содржат X, а исто така содржат и Y.

2.4 ДРУГИ ЗАДАЧИ НА ПОДАТОЧНОТО РУДАРЕЊЕ

Претходните три задачи на податочното рударење се најкористени во полето на податочното рударење и алгоритмите кои извршуваат такви задачи се вклучени во алатките за податочно рударење. На останатите задачи на податочното рударење ќе се задржиме накратко.

Сумирањето или опишувањето на податоците ги сумира главните карактеристики или особини на целната класа од податоците: ова најчесто се изведува со поставување прашања до базата на податоци. Сумарните податоци, најчесто се добиваат со употреба на основна статистика или со сумирање во OLAP (On-line Analytical Processing) и може да се претстават во различни графички форми, како графици и сл.

Откривањето на исклучоци се занимава со пронаоѓање на податочните објекти кои не се вклопуваат во генералното однесување или во моделот на податоци: овие се нарекуваат исклучоци. Исклучоците можат да бидат корисни, да речеме за откривање на измама. Тие бараат објекти кои значително се разликуваат од оние во кластерите или покажуваат големи разлики од просечните особини на објектите во групата.

Анализата на временски серии (еволуциона анализа) ги опишува правилностите или трендовите на моделите чие однесување се менува со времето. Таа вклучува откривање на девијација и промена, кое се фокусира на откривање на најзначајните промени во податоците од претходно мерените или нормативните вредности.

Откривање на секвенца се користи за да се откријат секвенцијалните шаблони во податоците. Овие шаблони се базираат на временска секвенца на акции. Шаблоните се слични на асоцијациите на поврзаните податоците (или настаните), но поврзаноста се базира на времето. За разлика од анализата на потрошувачката кошничка, која бара продуктите да бидат купени при една трансакција (во исто

време), во откривањето на секвенцата продуктите се купуваат по некој временски редослед.

За многу реални апликации од податочното рударење може да се смета дека ги предвидуваат идните состојби на податоците врз основа на минатите и тековните податоци. **Предвидувањето** може да се смета за еден тип на класификација (предвидувањето е задача на податочното рударење кое се разликува од предиктивниот модел, иако задачата на предвидувањето е дел од предиктивното моделирање). Разликата е во тоа што предвидувањето ја предвидува идната состојба, наместо сегашната. Предиктивните апликации вклучуваат препознавање на говор, машинско учење и препознавање на шаблони.

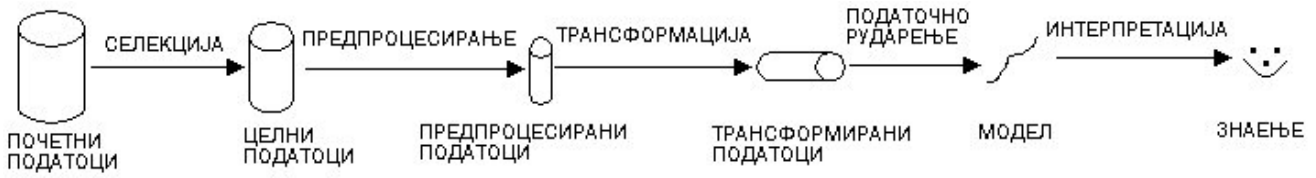
2.5 ПОДАТОЧНОТО РУДАРЕЊЕ НАСПРОТИ ОТКРИВАЊЕТО НА ЗНАЕЊЕ ВО БАЗИ НА ПОДАТОЦИ

Често пати со забуна се користат термините откривање на знаења во бази на податоци (knowledge discovery in databases - KDD) и податочно рударење. Всушност, има многу други имиња за овој процес на откривање на корисни (скриени) шаблони во податоците: екстракција на знаење, откривање на информации, експлораторна анализа на податоците, „жнеење" информации и препознавање на шаблони без надзор. Во последните неколку години терминот откривање на знаења во бази на податоци се користи за да се означи процес од повеќе чекори, а податочното рударење е само еден од нив. Значи, откривање на знаења во бази на податоци е процес на наоѓање на корисни информации и шаблони во податоците, а податочното рударење е употребата на алгоритми за да се извлечат информациите и шаблоните изведени од процесот на откривање на знаења во базите на податоци. За процесот на откривање на знаења во бази на податоци, често се вели дека е не-тривијален.

Откривањето на знаења во базите на податоци е процес кој има повеќе чекори (Слика 2.1). Влез во овој процес се податоците, а излез е информацијата која ја бараат корисниците. Меѓутоа, целта може да биде нејасна и непрецизна. Самиот процес е интерактивен и може да бара повеќе време. За да се осигура корисноста и точноста на резултатите од процесот, потребна е интеракција за време на целиот процес со експерти од конкретната област и технички експерти.

Процесот на откривање на знаења во базите на податоци се состои од следните пет чекори:

- **Селекција:** податоците кои се потребни за процесот на податочно рударење може да бидат добиени од многу различни и хетерогени извори на податоци. Овој прв чекор ги зема податоците од различни бази на податоци, документи и неелектронски извори.
- **Предпроцесирање:** Податоците кои ќе се користат може да имаат неточни вредности или да им недостасуваат вредности. Може да има податоци со аномалии како резултат на фактот дека податоците се земаат од различни извори со различни типови на податоци или метрика. Тука може да има многу различни активности. Грешните податоци може да бидат корегирани или отстранети, а исчезнатите податоци може да бидат додадени или предвидени (често со употреба на алатки за податочно рударење).
- **Трансформација:** Податоците од различните извори мора да бидат претворени во заеднички формат за да се дадат на обработка. Некои податоци може да бидат кодирани или трансформирани во покорисни формати. Може да се употреби редукција на податоците за да се намали бројот на разгледувани можни вредности на податоците.
- **Податочно рударење:** Зависно од задачата на податочно рударење која се извршува, овој чекор применува алгоритми на трансформираните податоци за да се генерираат саканите резултати.
- **Интерпретација/евалуација:** Многу е важно како се претставени резултатите од податочното рударење бидејќи корисноста на резултатите зависи од тоа. Во овој последен чекор се користат разни стратегии за визуелизација и графички-кориснички интерфејси.



Слика 2.1. Процесот на откривање на знаења од бази на податоци

Самиот процес на податочно рударење е комплексен. Постојат многу различни апликации и алгоритми за податочно рударење. Овие алгоритми мора да бидат внимателно применети за да бидат ефикасни. Откриените шаблони мора да бидат коректно интерпретирани и соодветно евалуирани за да се осигура дека добиената информација е значајна и точна.

3. ДРВА ЗА ОДЛУЧУВАЊЕ

Пристапот со дрва за одлучување најкорисен е за класификациските проблеми. Со оваа техника се конструира дрво за да се моделира класификацискиот процес. Кога ќе се изгради дрвото, се применува на секоја торка во базата на податоци и се добива класификацијата за таа торка. Постојат два основни чекори во оваа техника: градење на дрвото и примена на дрвото на базата на податоци. Најмногу од истражувањата се насочени кон изградбата на ефикасни дрва додека процесот на апликација е јасен.

Пристапот со дрвата за одлучување во класификацијата е со цел да се подели просторот за пребарување во правоаголни региони. Торка се класифицира врз основа на тоа во кој регион ќе падне.

Дефиниција за дрва за одлучување би била следната: Нека е дадена база на податоци $D = \{t_1, t_2, ..., t_n\}$ каде $t_i = \langle t_{i1}, ..., t_{ih} \rangle$ и шемата на базата на податоци се состои од следните атрибути $\{A_1, A_2, ..., A_h\}$. Исто така, дадено е и множество од класи $C = \{C_1, .., C_m\}$. Дрво за одлучување или класификациско дрво е дрво кое е поврзано со D и ги има следниве особини:

- Секој внатрешен јазел е означен со атрибут, A_i.
- Секоја гранка е означена со предикат кој може да биде применет на атрибутот кој е поврзан за родителот.
- Секој лист е означен со класа, C_j.

Решавање на проблемот на класификација со употреба на дрва за одлучување е процес со два чекори:

- Индукција на дрво за одлучување: Конструирање на дрво за одлучување со множество на податоци за обучување.
- За секоја торка $t_i \in D$, примени го дрвото за одлучување за да ја одредиш нејзината класа.

Има многу предности во употребата на дрва за одлучување за класификација. Дрвата за одлучување се лесни за употреба и се ефикасни. Може да се генерираат правила кои се лесни за интерпретација и разбирливи. Тие се добри за

големи бази на податоци, бидејќи големината на дрвото не зависи од големината на базата на податоци. Секоја торка од базата на податоци мора да помине низ дрвото. За тоа е потребно време кое е пропорционално на висината на дрвото, која е фиксна. Може да бидат конструирани дрва за податоци со многу атрибути.
Исто така, има и недостатоци кај примената на дрва за одлучување. Прво, тие не се справуваат лесно со континуални податоци. Домените на овие атрибути мора да бидат поделени на категории за да може да се справат со нив. Исто така, тешко е и справувањето со недостатокот на податоци бидејќи може да не бидат одредени точно гранките на дрвото. Бидејќи дрвата за одлучување се градат од податоците за обучување, може да дојде до преголемо нагодување. Ова може да биде надминато со поткастрување. Во процесот на градење на дрвото за одлучување се игнорира поврзаноста меѓу атрибутите во базата на податоци.

3.1 ИНДУКЦИЈА ОДГОРЕ НАДОЛУ НА ДРВА ЗА ОДЛУЧУВАЊЕ

Да се најде најмалото дрво за одлучување кое ќе се вклопува во дадено множество на податоци пресметковно е многу скапо, затоа за градењето на дрва за одлучување се користи хеуристичко пребарување. Најчесто се користи алгоритмот за индукција одгоре надолу на дрва за одлучување (Top-down Induction of Decision Trees – TDIDT) даден од Quinlan, 1986. Една негова упростена верзија е дадена со Алгоритам 1.
Атрибутите во шемата на базата на податоци кои се користат за означување на јазлите од дрвото околу кои ќе се прават поделбите се нарекуваат атрибути за поделба, а предикатите со кои се означени гранките во дрвото се нарекуваат предикати за поделба. Овој рекурзивен алгоритам го гради дрвото во насока одгоре надолу со испитување на множеството податоци за обучување. Алгоритмот го користи почетното множество на податоци за обучување и прво се избира најдобриот атрибут за поделба. Откако ќе биде определен најпогодниот атрибут за поделба се креира јазелот и неговите гранки и се додаваат на претходно добиеното дрво. Алгоритмот продолжува рекурзивно со додавање на поддрва на секоја гранка.
За дискретни атрибути се креира гранка за секоја можна вредност на атрибутот, додека за континуалните атрибути се избира праг и се креираат две гранки врз основа на прагот. Креирањето на дрвото застанува кога атрибутите во јазелот се доволно чисти (на пример, сите се во иста класа) или кога е задоволен некој друг критериум за застанување (нема добар атрибут кој би можел да се додаде).

Јазлите креирани на овој начин се нарекуваат листови и се означуваат со содветните вредности на класата.

За избор на атрибут во процесот на избирање на најдобар атрибут за поделба може да се користат најразлични мерки. Овие мерки зависат од тоа дали станува збор за класификациски дрва или регресиски дрва. За класификациските дрва, Quinlan ја употребува добивката на информации, која претставува очекувано намалување на ентропијата во класата, но може да се употребува и точноста на добиената класа. За регресиските дрва може да се употребува очекуваното намалување на варијансата на вредноста на класата.

Влез:
D //податоци за обучување
Излез:
T //Дрво за одлучување
ИзградиДрво алгоритам:
T=0;
Определи го најдобриот критериум за поделба;
T=креирај го јазелот-корен и означи го со атрибут за поделба;
T=додади гранка на јазелот-корен за секој предикат на поделба и означи;
за секоја гранка **направи**
    D=база на податоци добиена со примена на предикатот за поделба врз *D*;
    Ако е стигнато до точката за стопирање за оваа патека,
    тогаш *T'*=креирај лист и означи го со соодветна класа
    инаку *T'*=**ИзградиДрво**(*D*);
    T=додади го *T'* на гранката;

Алгоритам 1. Поедноставен алгоритам за изградба на дрво за одлучување

Поткаструваньето на дрвата е важен механизам кој се користи за спречуваньe на претерано нагодуваньe на дрвата. Поткаструваньe може да се примени за време на конструираньe на дрвото (пред-поткаструваньe) или откако ќе се конструира дрвото (пост-поткаструваньe). Обично, за пред-поткаструваньe треба минимален број на примероци во гранките, а за пост-поткаструваньe треба ниво на доверба во проценките на точноста.

3.2 ФАКТОРИ КОИ ВЛИЈААТ ВРЗ АЛГОРИТМИТЕ ЗА ГРАДЕЊЕ НА ДРВА ЗА ОДЛУЧУВАЊЕ

Најважни фактори за перформансите на алгоритмите за градење дрва за одлучуваньe се големината на множеството на податоци за обучуваньe и начинот на

кој се избира најдобриот атрибут за поделба. Следниве проблеми се среќаваат кај алгоритмите за дрва за одлучување:

- **Избор на атрибути за поделба**: изборот на атрибути за поделба влијае врз перформансите на изградбата на дрвата за одлучување. Некои атрибути се подобри од други. Изборот на атрибут не вклучува само испитување на податоците туку и информирачки влез од експерти од областа;
- **Подредување на атрибутите за поделба**: Исто така, важен е редоследот по кој се избрани атрибутите. Со добар распоред на атрибутите за поделба може да се избегнат непотребни споредби.
- **Поделби**: Бројот на поделби кои треба да се направат е поврзан со подредувањето на атрибутите. За некои атрибути доменот е мал, а бројот на поделби зависи од доменот. Ако доменот е континуален или има голем број на вредности, тогаш тешко се определува бројот на поделби;
- **Структура на дрвото**: Пожелно е балансирано дрво со помалку нивоа со цел да се подобрат перформансите на примената на дрвото за класификација. Во овој случај, можеби би било потребно повеќенасочно гранење, додека некои алгоритми градат само бинарни дрва.
- **Критериум за стопирање**: Изградбата на дрвото дефинитивно престанува кога множеството од податоци за обучување е совршено класифицирано (што е редок случај). Се стопира порано со цел да се спречи градење на огромни дрва. Ова е компромис меѓу точноста на класификацијата и перформансите. Дополнително, може да дојде до стопирање порано за да се спречи преголемо нагодување.
- **Податоци за обучување**: Структурата на креираното дрво за одлучување зависи од множеството за обучување. Ако множеството за обучување е премало, тогаш генерираното дрво може да биде премногу специфично, па да не работи со поопшти податоци. Ако множеството за обука е преголемо, тогаш креираното дрво може да биде пренагодено.
- **Поткастрување**: Кога дрвото веќе е креирано, може да дојде до некои модификации на дрвото за да се зголемат перформансите на дрвото за време на фазата на класификација. Фазата на поткастрување може да отстрани редундантни споредби или да отстрани редундантни поддрва за да се постигнат подобри перформанси.

3.3 ПЕРФОРМАНСИ НА ДРВАТА ЗА ОДЛУЧУВАЊЕ

Формата на дрвото е дефинирана од податоците за обучување и алгоритмот за индукција на дрво. Значи, пожелно е дрво кое работи задоволително добро на множеството од податоци за обучување и има најдобра форма. Некои алгоритми креираат само бинарни дрва. Бинарните дрва лесно се креираат, но, се ,,подлабоки". Перформансите кои се добиваат со примена на овие видови дрва за класификација може да бидат полоши бидејќи се потребни повеќе споредби. Меѓутоа, бидејќи овие споредби се поедноставни од оние во повеќенасочните гранења, па крајните перформанси може да бидат споредливи.

Алгоритмот за градење на дрва за одлучување може првенствено да изгради некое дрво и потоа истото да се поткастри за поефикасна класификација. Со техниките за поткастрување, делови од дрвата може да бидат отстранети или комбинирани за да се намали вкупната големина на дрвото. Делови од дрвото поврзани со класификацијата кои употребуваат неважни атрибути може да бидат отстранети. Поткастрување може да има и додека се гради дрвото, да се спречи дрвото да стане многу големо. Друг пристап е поткаструвањето да се изврши по градење на дрвото.

Временската и просторната комплексност на алгоритмите за дрвата за одлучување зависи од големината на податоците за обучување, бројот на атрибути и од формата на добиеното дрво. Во најлош случај, дрвото за одлучување што се гради може да биде доста длабоко и разгрането. Додека дрвото се гради, за секој од јазлите се испитува секој атрибут за да се најде најдобриот.

3.4 ПОПОЗНАТИ АЛГОРИТМИ ЗА ГРАДЕЊЕ НА ДРВА ЗА ОДЛУЧУВАЊЕ

Најпознати алгоритми за градење на дрва за одлучување се:

- **ID3**: оваа техника се базира на теоријата на информациите и се обидува да го минимизира очекуваниот број на споредби. Основна идеја е да се постават прашања кои обезбедуваат најмногу информација.
- **C4.5 и C5.0**: го подобруваат ID3 во однос на справувањето со недостаток на податоци, континуалните податоци, поткаструвањето, правилата и поделбите.
- **CART**: Classification And Regression Trees (CART)- Техниката на класификациски и регресиски дрва генерира бинарни дрва за одлучување.

- **Скалабилни техники за дрва за одлучување**: овие техники обезбедуваат изградба на дрва за одлучување за големи множества на податоци.

4. КЛАСТЕРИРАЊЕ

Кластерирањето е слично на класификацијата бидејќи податоците се групираат, но, за разлика од класификацијата, групите не се предефинирани. Наместо тоа, групирањето се остварува со наоѓање сличности меѓу податоците според карактеристиките на тековните податоци. Групите се нарекуваат кластери. Предложени се повеќе дефиниции за кластери:

- Множество од слични елементи. Елементите од различни кластери не се слични.
- Растојанието меѓу точките во кластерот е помало од растојанието меѓу точка во кластерот и која било точка надвор од него.

Кластерирањето се користи во многу домени, вклучувајќи ја биологијата, медицината, антропологијата, маркетингот и економијата. Апликациите за кластерирање вклучуваат класификација на растенијата и животните, класификација на болестите, обработка на слики и препознавање на шаблони. Еден од првите домени во кој е употребено кластерирање е биолошката таксономија. Најновите примени вклучуваат испитување на WEB најавување за да се откријат корисничките шаблони.

4.1 ПРОБЛЕМИ КАЈ КЛАСТЕРИРАЊЕТО

Кога се применува кластерирање на реални бази на податоци, се појавуваат доста интересни проблеми:

- Справувањето со исклучоци. Елементите природно не припаѓаат кон ниеден кластер. Тие може да бидат разгледувани како самостојни кластери. Меѓутоа, ако алгоритмот за кластерирање се обидува да најде поголеми кластери, овие исклучоци ќе бидат принудно сместени во некој кластер. Овој процес може да резултира со креирање на слаби кластери со комбинирање на два веќе постоечки кластери и ставање на исклучокот во посебен кластер.

- Динамичките податоци во базата на податоци имплицираат дека членовите во кластерот може да се менуваат со тек на времето.
- Може да биде тешка интерпретацијата на семантичкото значење. Kaj класификацијата означувањето на класите е познато однапред што не е случај со кластерирањето.
- Не постои еден точен одговор на проблем со кластерирање, всушност, постојат повеќе одговори. Не е лесно да се определи бројот на потребни кластери. Можеби ќе биде потребна помош од експерт од областа.
- Друго прашање е какви податоци треба да се користат за кластерирање. За разлика од учењето за време на класификацискиот процес, каде постои претходно знаење за тоа какви треба да бидат атрибутите за секоја класификација, во кластеририрањето немаме учење под надзор за да помогне на процесот.

4.2 ОСНОВНИ ОСОБИНИ И ДЕФИНИЦИЈА НА ПРОБЛЕМОТ НА КЛАСТЕРИРАЊЕТО

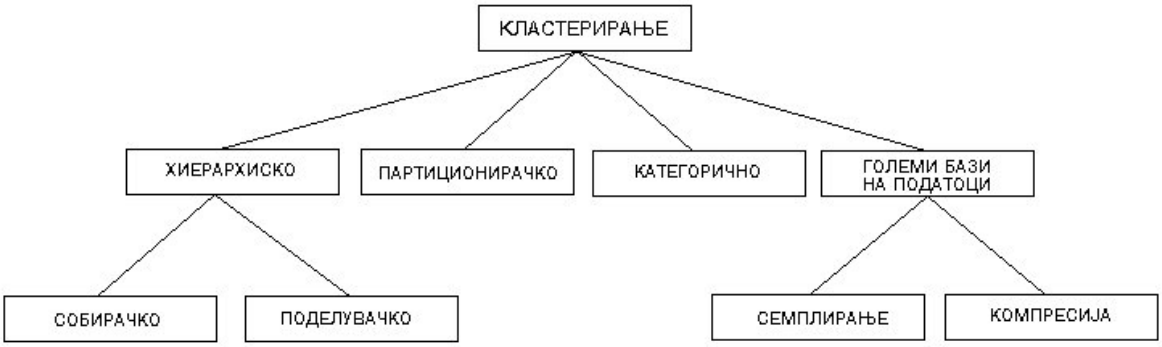
Можеме да ги сумираме основните особини на кластерирањето (кои се различни од класификацијата):

- Бројот на кластери е непознат;
- Може да не постои претходно знаење за кластерите;
- Добиените кластери се динамични.

Нека претпоставиме дека бројот на кластери кои треба да се креираат е влезна величина, k. Тековната содржина (и интерпретацијата) на секој кластер е K_j, $1 \le j \le k$, е одредена како резултат на дефиницијата на функцијата. Без да загубиме од генералноста, може да речеме дека проблем на кластерирањето е креирање на кластери: $K = \{K_1, K_2, ..., K_n\}$. Нека е дадена база на податоци $D = \{t_1, t_2, ..., t_n\}$ од торки и целобројна вредност k, проблемот на кластерирање е да се дефинира пресликување $f : D \to \{1, ..., k\}$ каде секој t_i се доделува на еден кластер K_j, $1 \le j \le k$. Кластерот K_j, ги содржи само оние торки кои се пресликани во него; односно $K_j = \{t_i | f(t_i) = K_j,\ 1 \le i \le k.$, и $t_i \in D\}$.

4.3 АЛГОРИТМИ ЗА КЛАСТЕРИРАЊЕ

На слика 4.1 е даден приказ на алгоритмите за кластерирање. Алгоритмите за кластерирање може да се хиерархиски или партиционирачки. При хиерархиското кластерирање се создаваат вгнездени множества од кластери. Секое ниво во хиерархијата има посебно множество од кластери. На најниско ниво, секој запис е во својот уникатен кластер. На највисоко ниво, сите записи припаѓаат на истиот кластер. При хиерархиското кластерирање влез не е саканиот број на кластери. При партиционирачкото кластерирање алгоритмот создава само едно множество од кластери. Ваквиот пристап за влез го има саканиот број на кластери. Традиционалните алгоритми за кластерирање се наменети за мали бази на податоци кои се сместени во меморијата. Но, поновите алгоритми за кластерирање може да работат со поголеми бази на податоци и со динамички бази на податоци. Алгоритмите за големи бази на податоци може да имаат мемориски ограничувања кои ќе ги исполнат со земање на примероци на податоците или со употреба на некои податочни структури, кои може да бидат компресирани или поткастрени за да се вклопат во големината на меморијата. Кластерирачките алгоритми се разликуваат и во тоа дали создаваат препокривачки или непрепокривачки кластери. Иако од интерес се само непрепокривачките кластери, можно е да се стави еден запис во повеќе кластери, па непрепокривачките кластери може да бидат надворешни и внатрешни. Надворешните техники користат означување на записите за да му помогнат на класификацискиот процес. Овие алгоритми ги вклучуваат традиционалните класификациски алгоритми за учење со надзор за кои се употребува специјален влез за обучување. Внатрешните алгоритми не употребуваат некоја претходна ознака за категорија, но зависат само од матрицата на близина која го содржи растојанието меѓу објектите.



Слика 4.1. Преглед на алгоритмите за кластерирање

Типовите на алгоритми за кластерирање може да бидат поделени во однос на имплементацискатa техника која ја користат. Хиерархиските алгоритми може да бидат категоризирани како собирачки или разделувачки. Кај собирачките, кластерите се градат оддолу нагоре, а кај разделувачките се работи во насока одгоре надолу. Иако и хиерархиските и партиционирачките алгоритми може да се опишат со поделбата на собирачки и поделувачки, сепак оваа поделба повеќе е сврзана со хиерархиските алгоритми. Алгоритмите може да се поделат во зависост од тоа дали секој елемент се обработува посебно (овие алгоритми се наречени сериски или инкрементални) или сите елементи се обработуваат истовремено (алгоритмите се наречени симултани). Ако одредена торка има вредности за сите атрибути од шемата, тогаш кластерирачките алгоритми се разликуваат по тоа како вредностите на атрибутите се испитуваат. Некои алгоритми ги испитуваат вредностите на алгоритмите една по една и се нарекуваат монотетични (вообичаено кај класификациските техники кај дрвата за одлучување). Политетичните алгоритми ги земаат предвид вредностите на сите атрибути одеднаш. Кластерирачките алгоритми уште може да се поделат според математичката формулација на алгоритмот: теорија на графови или линеарна алгебра (матрична алгебра).

4.4 МЕРКИ ЗА СЛИЧНОСТ И РАСТОЈАНИЕ

Една од најважните особини за кластерите е дека торка во рамките на еден кластер е послична со торките од истиот кластер отколку со торките од друг кластер. Затоа мора да дефинираме мерка за сличност $sim(t_i, t_l)$, која се дефинира меѓу торките t_i , t_l ϵ D. Од ова произлегува различна и поцврста дефиниција за проблемот на кластерирање.

Нека е дадена база на податоци $D = \{t_1, t_2, ..., t_n\}$ од торки, мерка за сличност дефинирана меѓу кои било две торки $t_i, t_l \in D$, $sim(t_i, t_l)$ и целобројна вредност k, проблемот на кластерирање претставува дефинирање на пресликување $f : D \to \{1, ..., k\}$ каде секој t_i се доделува на еден кластер K_j, $1 \leq j \leq k$.. За даден кластер, $K_j, \forall t_{jl}, t_{jm} \in K_j$ и $t_{jm} \notin K_j$, $sim(t_{jl}, t_{jm}) > sim(t_{jl}, t_i)$.

Мерката за растојание (оддалеченост), $dis(t_i, t_j)$, е спротивна на сличноста и често се користи при кластерирањето. Значи кластерирањето го има следното својство: за даден кластер $K_j, \forall t_{jl}, t_{jm} \in K_j$ и $t_{jm} \notin K_j, dis(t_{jl}, t_{jm}) \leq dis(t_{jl}, t_i)$.

Некои алгоритми за кластерирање ги земаат предвид само нумеричките податоци претпоставувајќи метрички податочни точки. Кластерите може да бидат опишани со употреба на неколку карактеристични точки. Нека е даден кластер K_m со N точки $\{t_{m1}, t_{m2}, ..., t_{mN}\}$, за кој може да дефинира:

- $centroid = C_m = \frac{\sum_{i=1}^{N}(t_{mi})}{N}$
- $radius = R_m = \sqrt{\frac{\sum_{i=1}^{N}(t_{mi} - C_m)^2}{N}}$
- $diameter = D_m = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}(t_{mi} - t_{mj})^2}{N(N-1)}}$

Центроидот е средината на кластерот и може да не е точка од кластерот. Некои алгоритми за кластерирање претпоставуваат дека има некој централно лоциран објект во кластерот кој се нарекува медоид. Радиусот е квадратен корен на просечното средно квадратно растојание од секоја точка до центроидот, а дијаметарот е квадратен корен од просечното средно квадратно растојание меѓу сите парови точки од кластерот. Ознаката M_m се користи за да се означи медоидот на кластерот K_m.

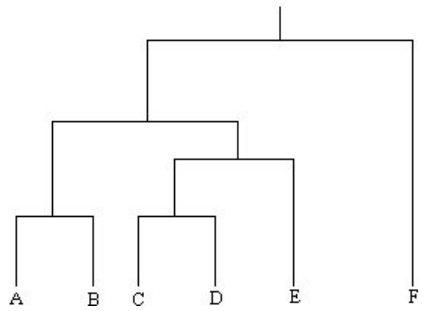
Многу алгоритми за кластерирање бараат да се одреди растојанието меѓу кластерите, а не растојанието меѓу точките од кластерите. Тоа не е лесна задача бидејќи растојанието меѓу кластерите може различно да се дефинира. Ако се дадени кластерите K_i и K_j, тогаш дефиниции за растојанието меѓу нив би биле:

- **Единствена врска**: најмалото растојание меѓу елемент од едниот кластер и елемент од другиот кластер. Тогаш имаме $dis(K_i, K_j) = \min(dis(t_{il}, t_{jm}))$ $\forall t_{il} \in K_i \notin K_j$ и $\forall t_{jm} \in K_j \notin K_i$.

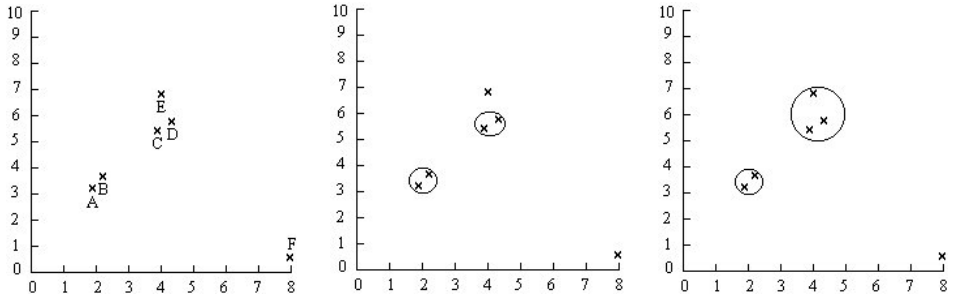
- **Целосна врска**: најголемото растојание меѓу елемент од еден кластер и елемент во друг кластер. Тогаш имаме $dis(K_i, K_j) = \max(dis(t_{il}, t_{jm}))$ $\forall t_{il} \in K_i \notin K_j$ и $\forall t_{jm} \in K_j \notin K_i$.
- **Просечна врска**: просечното растојание меѓу елементите од еден кластер и елементите од друг кластер. Тогаш имаме $dis(K_i, K_j) = mean(dis(t_{il}, t_{jm}))$ $\forall t_{il} \in K_i \notin K_j$ и $\forall t_{jm} \in K_j \notin K_i$.
- **Центроид**: Ако кластерите имаат центроид, тогаш центроидното растојание е дефинирано како растојание меѓу центроидите. Тогаш имаме: $dis(K_i, K_j) = dis(C_i, C_j)$, каде C_i е центроид на K_i и слично за C_j.
- **Медоид**: се употребува медоид за да се претстави секој кластер, растојанието меѓу кластерите може да се претстави со растојанието меѓу медоидите: $dis(K_i, K_j) = dis(M_i, M_j)$.

4.5 ХИЕРАРХИСКИ АЛГОРИТМИ

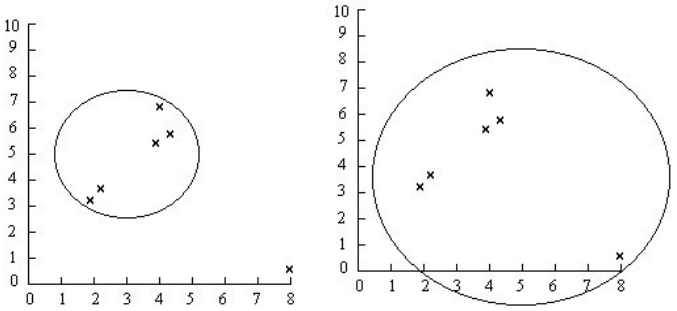
Хиерархиските алгоритми за кластерирање создаваат множества од кластери. Тие се разликуваат по начинот на создавање на множествата. За да се илустрира техниката на хиерархиското кластерирање и множествата од различни кластери може да се употреби податочната структура-дрво, наречена дендограм (Слика 4.2). Коренот на дендограмот се состои од еден кластер во кој се сите елементи. Листовите во дендограмот се состојат од кластери со еден елемент. Внатрешните јазли во дендограмот претставуваат нови кластери кои се формираат со спојување на кластерите кои се негови деца во дрвото. Секое ниво на дрвото е поврзано со мерката за растојание која е користена за да се спојат кластерите. Сите креирани кластери на одредено ниво се комбинираат бидејќи децата кластери се на помало растојание од вредноста на растојанието која е поврзана со ова ниво во дрвото (сликовито е прикажано на Слика 4.3).



Слика 4.2. Пример за дендограм



(а) шест кластери (б) четири кластери (в) три кластери



(г) два кластери (д) еден кластер

Слика 4.3. Пет нивоа на кластерирање

Просторната комплексност на хиерархиските алгоритми зависи од просторот кој е потребен за матрицата на близина и просторот за дендограмот.

Хиерархиските техники се многу погодни и за многу апликации со кластерирање кои природно покажуваат вгнездена поврзаност меѓу кластерите. На пример, во биологијата, таксономијата на растенијата и животните е еден вид хиерархија на кластери.

4.5.1 СОБИРАЧКИ АЛГОРИТМИ

Собирачките алгоритми почнуваат со тоа што секој запис е во свој сопствен кластер и потоа, итеративно се зголемуваат кластерите се додека сите записи не припаѓаат во еден кластер (Слика 4.3). Собирачки алгоритми се разликуваат по тоа

како се спојуваат сите кластери на секое ниво. Алгоритмот 2 прикажува типичен собирачки алгоритам за кластерирање. Претпоставува дека за влез има множество од елементи и растојанијата меѓу нив во квадратна матрица на близина, A. Матрицата на близина ги содржи растојанијата како вредности, а не како Булови вредности: $A[i,j]=dis(t_i,t_j)$. Излез од алгоритмот е дендограм, DE, кој претставува множество од подредени тројки $\langle d,k,K\rangle$ каде d е праговото растојание, k е бројот на кластери, а K е множество од кластери. Дендограмот е всушност множество од кластери, и корисникот може да избере (врз основа на праговото растојание) кои кластери ќе ги користи.

Влез:
$D=\{t_1,t_2,...,t_n\}$ //множество од елементи
A //матрица на близина која се состои од растојанијата меѓу елементите
Излез:
DE //Дендограм, претставен како множество од тројки
Собери алгоритам:
$d=0;$
$k=n;$
$K=\{\{t_1\},\{t_2\},...,\{t_n\}\};$
$DE=\{\langle d,k,K\rangle\};$ //почетно дендограмот се состои од кластери од по еден елемент
повторувај
$oldk=k;$
$d=d+1;$
A_d =максимална матрица на близина за графот со прагово растојание d
$\langle k,K\rangle=NewClusters(A_d,D);$
ако $oldk\neq k$ **тогаш**
$DE=DE\cup\langle d,k,K\rangle;$//се додава ново множество од кластери на дендограмот
додека $k=1$

Алгоритам 2. Типичен собирачки алгоритам за кластерирање

Овој алгоритам употребува процедура $NewClusters$ за да реши како да го креира следното ниво на кластери од претходното ниво. Овде се разликуваат разните типови на собирачки алгоритми. Можно е да се спојат два или повеќе кластери од претходното ниво. Алгоритмите, исто така, се разликуваат и кои кластери се спојуваат кога има повеќе кластери со исто растојание. Дополнително, техниката за одредување на растојанието може да е различна. Најпознати собирачки техники

кои се базираат на теоријата на графови се: единствена врска, целосна врска и просечна врска.

Сите собирачки техники имаат големи временски и просторни ограничувања кои зависат од просторот кој е потребен за матрицата на близина. Бидејќи алгоритмот има итеративна природа мора на матрицата (или на дел од неа) да се пристапи повеќе пати. Комплексноста на *NewClusters* може да влијае врз ограничувањата. Друг проблем со собирачките алгоритми е дека не се инкрементални. Значи, кога се додаваат нови елементи или старите се отстранети или променети, мора целиот алгоритам да се изврши од почеток.

4.5.1.1 Техниката на единствена врска

Техниката на единствена врска се базира на идејата на пронаоѓање на максимално поврзаните компоненти во еден граф. Поврзана компонента е граф во кој постои патека меѓу кои било два елементи. Со пристапот на единствена врска, два кластери се спојуваат ако има барем една контура која ги поврзува двата кластери, т.е. два кластери се спојуваат ако минималното растојание меѓу кои било две точки е помало или еднакво од даденото прагово растојание. Заради ова, оваа техника се наречува и техника на најблизок сосед.

4.5.1.2 Техниката на целосна врска

Иако техниката на целосна врска е слична со техниката на единствена врска, бара групи наместо поврзани компоненти. Група е максимален граф во кој има контура меѓу кои било два елементи. Овде се користи процедура за да се најде максималното растојание меѓу кои било кластери, и два кластери се спојуваат ако максималното растојание е помало или еднакво од праговото растојание.

Кластерите кои се добиени со оваа техника се покомпактни од оние кои се добиени со употреба на техниката на единствена врска. Една варијација на оваа техника се нарекува техника на најдалечен сосед.

4.5.1.3 Техника на просечна врска

Техниката на просечна врска спојува два кластери ако просечното растојание меѓу две точки во двата целни кластери е помало од праговото растојание.

4.5.2 РАЗДЕЛУВАЧКО КЛАСТЕРИРАЊЕ

Кога имаме разделувачко кластерирање, сите записи почетно се сместени во еден кластер и кластерите се делат се додека сите записи не се во нивниот сопствен кластер. Идејата е да се делат кластерите каде некои елементи не се доволно блиски до другите елементи.

4.6 ПАРТИЦИОНИРАЧКИ АЛГОРИТМИ

Нехиерархиското или партиционирачкото кластерирање создава кластери во еден чекор, а не во повеќе. Се создава само едно множество од кластери, иако кај некои алгоритми може да се создадат повеќе внатрешни множества од кластери. Бидејќи едно множество од кластери е излез, корисникот мора да го внесе бараниот број на кластери, k. Дополнително, се употребува и некоја метрика или критериум функција за да се одреди колку е добро предложеното решение. Оваа мерка за квалитет може да биде просечното растојание меѓу кластерите или некоја друга метрика. Една вообичаена мерка е метриката на квадратна грешка, која го мери квадратното растојание од секоја точка до центроидот на соодветниот кластер:

$$\sum_{m=1}^{k} \sum_{t_{mi} \in K_m} dis\left(C_m, t_{mi}\right)^2$$

Проблем кај партиционирачките алгоритми е тоа што тие страдаат од преголемиот број на можни решенија. Очигледно е дека не е изводливо да се најдат сите можни алтернативи за кластерирање. На пример, со даден критериум за мерки, наивен пристап е да се погледнат сите можни множества од k кластери. Има $S(n,k)$ комбинации кои треба да се испитат. Според

$$S(n,k) = \frac{1}{k!} \sum_{i=1}^{k} (-1)^{k-1} \binom{k}{i} (i)^n$$

има 12259666000 различни начини да се кластерираат 19 записи во 4 кластери. Затоа, најголем дел од алгоритмите земаат мало подмножество од сите кластери со употреба на некоја стратегија да се идентификуваат осетливите кластери.

4.6.1 МИНИМАЛНО ПОКРИВАЧКО ДРВО

Бидејќи кластерирачкиот проблем е да се дефинира пресликување, излезот од овој алгоритам ги дава кластерите како множество од подредени парови $\langle t_i, j \rangle$ каде $f(t_i) = K_j$.

Влез:

$D = \{t_1, t_2, ..., t_n\}$ //множество од елементи

A //матрица на близина која се состои од растојанијата меѓу елементите

k //број на сакани кластери

Излез:

f //Пресликување, претставено како множество од подредени парови

ПартиционирачкиMST алгоритам:

$M = MST(A);$

идентификувај ги неконзистентните контури во M;

отстрани $k-1$ неконзистентни контури;

креирај излез;

Алгоритам 3. MST партиционирачки алгоритам

Проблем е како да се дефинира неконзистентноста. Едно решение MST да ги отстранува контурите од најголемата кон најмалата. Ова би ги отстранила најголемите $k-1$ контури од почетниот целосно поврзан граф. Постојат и други мерки за неконзистентност. Една од нив се базира на тежината (растојанието) на контурата во однос на оние кои се блиску до неа.

4.6.2 КЛАСТЕРИРАЧКИ АЛГОРИТАМ СО КВАДРАТНА ГРЕШКА

Кластерирачкиот алгоритам со квадратна грешка ја минимизира квадратната грешка. Квадратна грешка за еден кластер е сума на квадратните Евклидови растојанија меѓу секој елемент во кластерот и центроидот на кластерот, C_k. Нека е даден кластер K_i и множеството од записи кое се пресликува во тој кластер е $\{t_{i1}, t_{i2}, ..., t_{im}\}$. Квадратната грешка се дефинира како:

$$se_{K_i} = \sum_{j=1}^{m} \left\| t_{ij} - C_k \right\|^2$$

За дадено множество од кластери $K = \{K_1, K_2, ..., K_n\}$, квадратната грешка за K е дефинирана на следниот начин:

$$se_K = \sum_{j=1}^{k} se_{K_j}$$

Постојат повеќе алгоритми за кластерирање со квадратна грешка. Нивната основна структура е дадена со Алгоритам 4.

Влез:

$D = \{t_1, t_2, ..., t_n\}$ //множество од елементи

k //број на сакани кластери

Излез:

K //Множество од кластери

КвадратнаГрешка алгоритам:

придружи го секој запис t_i на кластер;

пресметај го центарот за секој кластер;

повторувај

придружи го секој запис t_i на кластерот со најблизок центар;

пресметај го новиот центар за секој кластер;

пресметај ја квадратната грешка;

додека разликата меѓу последователните квадратни грешки е под прагот;

Алгоритам 4. Кластерирачки алгоритам со квадратна грешка

4.6.3 КЛАСТЕРИРАЊЕ СО K-УСРЕДНУВАЊА

K-усреднување е итеративен алгоритам за кластерирање кај кој записите се преместуваат низ множествата од кластери додека не се постигне бараното множество. Се добива голема сличност меѓу елементите од еден кластер, а истовремено се добива голема различност меѓу елементите од различни кластери. Средина на кластерот $K_i = \{t_{i1}, t_{i2}, ..., t_{im}\}$ се дефинира како

$$m_i = \frac{1}{m}\sum_{j=1}^{m} t_{ij}$$

Оваа дефиниција претпоставува дека секоја торка има само една нумеричка вредност, додека постојат торки со повеќе нумерички вредности од атрибутите. Алгоритмот за кластерирање со K-усреднувања бара да постои некоја дефиниција за средина на кластерот, но не мора да е претходната. Овде средината е дефинирана исто како центроидот. Основната структура на ваквите алгоритми е дадена на Алгоритам 5.

Влез:

$D = \{t_1, t_2, ..., t_n\}$ //множество од елементи

k //број на сакани кластери

Излез:

K //Множество од кластери

K-Усреднувања алгоритам:

Придружи почетни вредности за средините $m_1, m_2, ..., m_k$;

//овие може да бидат избрани случајно или вредностите од првите k влеза

повторувај

придружи го секој запис t_i на кластерот со најблиска средина;

пресметај ја новата средина за секој кластер;

додека не се исполни критериумот за конвергенција;

//може да биде квадратна грешка, број на торки доделени на кластер, максимален број на итерации ...

Алгоритам 5. Кластерирање со K-усреднувања

4.6.4 АЛГОРИТАМ ЗА НАЈБЛИЗОК СОСЕД

Алгоритмот за најблизок сосед е сличен со техниката на единствена врска (Алгоритам 6). Со овој алгоритам, записите итеративно се додаваат на најблиските веќе постоечки кластери. За овој алгоритам се дава праг, t, со кој се одлучува дали записите се додаваат на постоечките кластери или се креира нов кластер.

Влез:

$D = \{t_1, t_2, ..., t_n\}$ //множество од елементи

A //матрица на близина која се состои од растојанијата меѓу елементите

Излез:

K //Множество од кластери

НајблизокСосед алгоритам:

$K_1 = \{t_1\};$

$K = \{K_1\};$

$k = 1;$

за $i = 2$ **до** n **направи**

најди го t_m во некој кластер K_m од K таков што $dis(t_i, t_m)$ е најмало;

ако $dis(t_i, t_m) \leq t$ **тогаш**

$K_m = K_m \cup t_i$

инаку

$k = k + 1;$

$K_k = \{t_i\};$

Алгоритам 6. Алгоритам за најблизок сосед

4.6.5 PAM АЛГОРИТАМ

PAM (partitioning around medoids) алгоритмот (партиционирање околу медоидите), уште наречен и K-медоид алгоритам, го претставува кластерот со медоид. Со употребата на медоидот, овој пристап многу добро се справува со исклучоците. Почетно, се зема случајно множество од k записи да биде множество од медоиди. Тогаш, при секој чекор, се испитуваат записите од влезното множество од податоци еден по еден да се види дали тие би требало да се медоиди, т.е. алгоритмот пресметува дали има запис кој треба да замени некој од постоечките медоиди. Со гледање на паровите од медоид и не-медоид, алгоритмот го избира парот кој најмногу го подобрува квалитетот на кластерирањето и ги менува. Квалитетот овде се мери со сумата на сите растојанија од не-медоиден објект до медоидот во кластерот во кој се наоѓа. Запис се придружува на кластерот чиј медоид е најблиску (минимално растојание).

4.6.6 ОСТАНАТИ ПАРТИЦИОНИРАЧКИ АЛГОРИТМИ

Други партиционирачки алгоритми кои се користат за кластерирање се: алгоритмот со поврзувачка енергија, генетските алгоритми и невронските мрежи. Самите имиња на последните два кажуваат која техника ќе ја користат. Алгоритмите со поврзувачка енергија ја користат поврзаноста (афинитет) на два атрибути за да ги сместат во ист кластер.

4.7 ДРВА ЗА ПРЕДИКТИВНО КЛАСТЕРИРАЊЕ

Дрвата за одлучување често пати се користат за класификација или регресија на еден атрибут т.е. тие претставуваат модел во кој се предвидува вредноста на една променлива. Дрвото за одлучување идентификува делови од податоците, меѓутоа може да го разгледуваме и како хиерархија од кластери. Добра хиерархија од кластери е онаа кај која записите кои се во ист кластер се слични меѓусебе во однос на даден број од набљудуваните особини.

Ова води кон едноставен метод за градење на дрва кои дозволуваат предвидување на повеќе целни атрибути одеднаш. Ако се дефинираат мерки за растојание меѓу торки од целни атрибути, може да се изградат дрва за предвидување на повеќе атрибути. Може да се употреби стандардниот алгоритам за индукција на дрва за одлучување, а како хеуристика за избор на тестови кои ќе се вклучат во дрвото може да се користи минимизација на интра-кластерната варијанса (и максимизација на интер-кластерната варијанса).

Дрво за кластерирање е дрво за одлучување кај кое листовите не содржат класи, а секој јазел и секој лист одговара на еден кластер. За да се изградат дрва за кластерирање се применуваат принципи од учење со примери и градење на дрва за одлучување, при што се претпоставува дека е дадена мерката за растојание која го пресметува растојанието меѓу два примероци. За да се пресмета растојанието меѓу два кластери (т.е. множества од примероци) се применува функција која пресметува прототип на множество од примероци. Прототипот се смета за примерок, и тоа дозволува растојанието меѓу два кластери да се дефинира како растојание меѓу нивните прототипови.

Во зависност од примероците и од мерката за растојание разликуваме два начина за градење на дрва за кластерирање:

- Учење со надзор: кај учењето со надзор (како кај класичниот метод за градење на дрва за одлучување) мерката за растојание предвид ја зема

само класната информација на секој примерок. Исто така, регресионите дрва може да се сметаат за учење со надзор.

- Учење без надзор: кај учењето без надзор примероците може да не бидат класифицирани и мерката за растојание не зема предвид никаква класна информација. Овде во мерката за растојание се земаат предвид сите атрибути или особини на примероците.

Постојат различни мерки за квалитет на дрвата за кластерирање кои зависат од апликацијата. За класификациони цели најчесто се користи точноста на предвидувањето на новите примероци. За регресиони цели најчесто се користи релативната грешка, за која треба да се познати мерката за растојание и прототип функцијата. Релативната грешка е дадена со следната равенка.

$$RE = \frac{\sum_{i=1}^{n} d\left(e_i, \hat{e}_i\right)^2}{\sum_{i=1}^{n} d\left(e_i, p\right)^2}$$

каде e_i се примероците, $\hat{e}_i$ се предвидувањата и p е прототипот.

Нека е даден кластер C и тест T кој ќе го подели C на два кластери C_1 и C_2. Го пресметуваме растојанието меѓу прототиповите на двата кластери $d\left(p_{C_1}, p_{C_2}\right)$, каде p_{C_1} и p_{C_2} се прототиповите на кластерите C_1 и C_2, соодветно. Од сите тестови го избираме оној кој го дава максималното растојание. Ова одговара на принципот за максимизација на интер-кластерното растојание и минимизација на интра-кластерното растојание.

Критериумот за стопирање со градењето на дрвото често пати се базира на тестови за значајност. Бидејќи кластерирањето ја користи варијансата за избор на најдобрата поделба, добра хеуристика за критериум за стопирање би бил F-тестот, кој се пресметува со

$$F = \frac{\frac{SS}{n-1}}{\frac{SS_L + SS_R}{n-2}}$$

каде SS е сума на квадратни разлики до средината на множеството примероци, SS_L и SS_R се истото за креираните подмножества од примероци и n е вкупниот број на примероци.

Ако множество од примероци се подели на две подмножества, тогаш варијансата треба значајно да опадне, т.е. F треба да биде значајно голема.

Предвидувањето на повеќе целни атрибут одеднаш е еден вид предвидување на структурата, каде целната променлива е множество од класи. Овој проблем може да биде надминат со учење на посебни модели за секоја класа, но учењето на еден модел за сите класи има предност во тоа што добиената предиктивна теорија е многу помала и може да се земат предвид зависностите меѓу различни класи, и дури и овие зависности може да бидат разјаснети. Предвидувањето на повеќе целни атрибути се среќава често.

5. ИНДУКТИВНИ БАЗИ НА ПОДАТОЦИ

Најголемо внимание во истражувањата поврзани со податочно рударење се посветува на развој на нови методи кои ќе можат поточно и побрзо да извлекуваат знаење од големите количини на податоци. На пример, се посветува големо внимание на пронаоѓање техники и алгоритми за откривање на шаблони, класификација, кластерирање итн.

Меѓутоа, иако општата цел на податочното рударење не е добро дефинирана, многу помалку работа се посветува на откривање на тоа што значи цел на податочно рударење т.е. дали постои ,,теорија на податочно рударење" која ги опфаќа централните задачи, но, не е премногу широка. Во последно време ова прашање добива се поголемо внимание во истражувањата од областа на податочно рударење.

Постојат две главни барања кои треба да ги исполни теоријата на податочно рударење. Прво, треба да ги опфаќа повеќето од задачите на податочното рударење. Второ, треба да ги разјасни и унифицира различните задачи на податочното рударење. Значи, теоријата на податочното рударење треба да биде корисна во практиката, но треба и да кажува нешто за унифицираноста на огромните методи кои се познати во податочното рударење и да го поддржи процесот на откривање на знаење.

Индуктивните бази на податоци може да послужат како една можна опција за теоријата на податочното рударење. Основна идеја на индуктивните бази на податоци е да се прошират технологиите за бази на податоци на таков начин да овозможат базите на податоци да го поддржуваат податочното рударење и процесот на откривање на знаење.

Индуктивните бази на податоци се бази на податоци кои покрај самите податоци содржат индуктивни генерализации за податоците. Овие индуктивни генерализации може да бидат кластерирања, класификации или шаблони. Индуктивните бази на податоци може да ја поддржат експлораторната анализа на податоци во смисла на ad-hoc поставување на прашања. Претставата за индуктивни бази на податоци на високо ниво ги опфаќа сите главни задачи на податочното рударење, како што се откривање на шаблони, кластерирање и предвидување. Меѓутоа не е јасно како овие задачи треба да бидат изразени во рамката на индуктивните бази на податоци и каква треба да биде рамката на индуктивните бази на податоци за да ги изрази овие задачи на корисен начин. Корените на најголем дел од постоечките истражувања за индуктивните бази на податоци се во откривањето на шаблони, а предложените индуктивни бази на податоци главно се продолженија на вообичаените прашални јазици за базите на податоци. Моделот на индуктивни бази на податоци се состои од дел со податоци и дел со шаблони и на двата дела може да се поставуваат прашања.

Шема на индуктивните бази на податоци е парот $R = (\mathbf{R},(\mathrm{L},\mathcal{E},\mathcal{V}))$ каде $\mathbf{R}$ е шемата на базата на податоци, L е пребројиво множество од шаблони, $\mathcal{V}$ е множество од резултати и $\mathcal{E}$ е евалуациона функција која ги карактеризира шаблоните. Ако е дадена база на податоци $\mathbf{r}$ над $\mathbf{R}$ и шаблон $\theta \in L$, евалуационата функција го пресликува парот (r,θ) во елемент од $\mathcal{V}$. Инстанца на индуктивна база на податоци $(\mathbf{r},s)$ над R се состои база на податоци $\mathbf{r}$ над шемата $\mathbf{R}$ и подмножеството $s \subseteq L$.

Типичен процес на откривање на знаења од бази на податоци работи над двете компоненти на индуктивните бази на податоци. Корисникот може да селектира подмножество од записи или поопшто да селектира податоци од базата на податоци. Во тој случај, делот со шаблони останува непроменет. Корисникот, исто така, може да селектира и подмножества од шаблони, тогаш делот со податоци останува непроменет. Прашањата кои се однесуваат на делот со шаблони ја специфицираат задачата на рударењето и се нарекуваат индуктивни прашања.

Нека е дадена инстанца на индуктивна база на податоци ($\mathbf{r}$,s) чија шема е ($\mathbf{R}$,(L,ε,V)), индуктивното прашање се означува како $\sigma_C(s)$ и специфицира делови од s (шаблони) кои се од интерес. C е спој од ограничувања кои мора да бидат исполнети од специфицираните шаблони. Проверките на некои вакви споеви може да бидат потребни за евалуација на ε над $\mathbf{r}$ и може да вклучува пребарување на податоците.

Процесот на откривање на знаења од бази на податоци може да се опише како секвенца од прашања до индуктивна база на податоци.

За индуктивните бази може да се донесат следните заклучоци:

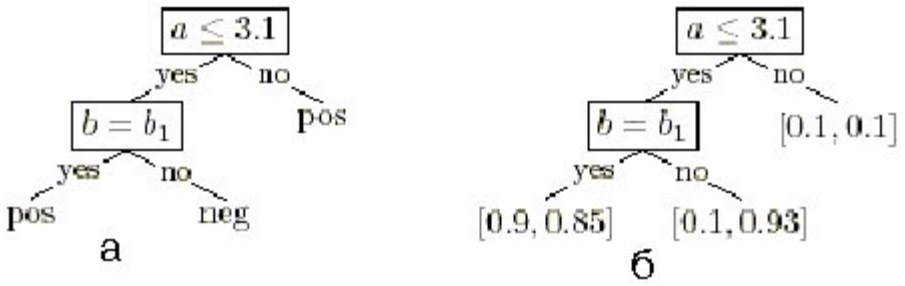
- Индуктивната база на податоци се состои од обична (нормална) база на податоци на која е додадено подмножество од шаблони од некоја класа на шаблони и евалуациона функција која кажува како се појавуваат шаблоните во податоците.
- До индуктивна база на податоци може да се поставуваат прашања (во принцип) само со употреба на обична релациска алгебра или SQL, со додадена можност прашањата да се упатуваат и до вредностите што се добиваат од евалуационата функција применета врз шаблоните.
- Моделирањето на процесите на откривање на податоци во базите на податоци како секвенца на прашања врз индуктивна база на податоци дава можност за уточнување и оптимизирање на овие процеси.

6. ОПИС НА СИСТЕМОТ CLUS

6.1 ВОВЕД

Clus е генерички систем за градење на дрва за одлучување кои често се користат во машинското учење и податочното рударење, најмногу за предвидување. *Clus* може да се користи за градење на повеќе видови дрва. Најчесто тоа се класификациски дрва кај кои се предвидува една целна променлива, а може да се користат и за градење на регресиски дрва, за предвидување на вредноста на нумерички атрибут. Со *Clus* може да се градат и дрва за кластерирање или дрва со повеќе целни променливи, за предвидување на вредноста на повеќе атрибути одеднаш. Примери за дрва за одлучување се дадени на слика 6.1. На слика 6.1.а)

е прикажано класификациско дрво за бинарен класификацики проблем со класи pos и neg. Секој запис е опишан со нумерички атрибут a и номинален атрибут b. На слика 6.2.б) е прикажано регресиско дрво со повеќе целни променливи за предвидување на вредноста на две целни променливи.



Слика 6.1. Примери за дрва за одлучување

Внатрешните јазли на дрвото за одлучување се состојат од тестови врз атрибутите кои се употребени во описот на податоците, а листовите ги чуваат предвидувањата. Предвидување за даден запис се добива кога тој запис ќе го поминеме низ дрвото.

Кога имаме дрва со повеќе целни променливи (кластерирање без надзор) влезните податоци ги делиме на кластери, со цел минимизација на интра-кластерното растојание и максимизација на интер-кластерното растојание. На дрвата за одлучување може да се гледа како на хиерархиско кластерирање на податоците: кореновиот јазел одговара на кластер кој ги содржи сите записи, кој се дели рекурзивно на помали кластери кога се поминува дрвото. На пример, кореновиот јазел на слика 6.1, го дели множеството податоци на два кластери: едниот ги содржи сите записи за кои $a \leq 3.1$ а вториот ги содржи сите записи за кои $a > 3.1$. Листовите кај дрвата за кластерирање, наместо предвидување, содржат јасен опис на кластерот, т.е. во форма на прототип на кластерот.

6.2 CLUS СИСТЕМОТ

Како влез *Clus* прима два документи: документ со податоците за обучување кои се претставени како табела со еден ред за секој запис и една колона за секој атрибут (во .arff формат, т.е. форматот на WEKA системот за податочно рударење) и документ за поставување на некои параметри и специфицирање на задачата на учењето. Специфицирањето на задачата на учењето се постигнува со

распоредување на атрибутите во три подмножества: описни (дескриптивни) атрибути D, кластерирачки атрибути C и целни атрибути T. Описните атрибути се употребуваат во внатрешните јазли на дрвото, кластерирачките атрибути се употребуваат при изградба на дрвото (за пресметување на хеуристиката), а целните атрибути се употребуваат за пресметување на вредностите кои се зачувуваат во листовите. За дадените поставување, важи дека $C = T$.

За да се градат дрва за одлучување со *Clus*, се поставува D да е множество од влезни атрибути и T е множество од атрибути кои треба да се предвидат. На овој начин, *Clus* може да биде конфигуриран да гради класификациски, регресиски дрва и дрва со повеќе целни атрибути (со можност за мешање на нумерички и номинални целни атрибути). На пример, за градење на дрвото до слика 6.1.а) важи $D = \{a, b\}$ и $T = \{c\}$, каде c е класен атрибут, а за дрвото од слика 6.1.б) важи $D = \{a, b\}$ и $T = \{d, e\}$, каде d и e се нумерички целни атрибути.

За да се градат кластерирачки дрва со *Clus*, се поставуваат D и T да се множество од сите атрибути A. Како резултат на ова, ќе биде изградено дрво за кластерирање кое во тестовите на внатрешните јазли може да го употреби кој било атрибут (бидејќи D=A) и кое во секој лист зачувува јасен опис на кластерот. Овој опис е вектор со по една компонента за секој атрибут (бидејќи T=A). За нумерички атрибути, се зачувува средната вредност на кластерот, а за номинални атрибути се зачувува најчестата вредност.

6.3 АЛГОРИТАМ ЗА ГРАДЕЊЕ ДРВА КАЈ CLUS

Clus употребува стандарден рекурзивен одгоре надолу алгоритам за градење на дрва за одлучување, сличен на оној од C4.5 и CART. Во основа одгоре надолу алгоритмот го гради дрвото почнувајќи од врвниот јазел. Тестот за овој јазел се одредува врз основа на резултатот од хеуристичката евалуација (сите можни тестови се нумерираат и рангираат по резултатот од хеуристиката кој кажува колку добро даден тест го дели множеството податоци за обука). Се избира најдобриот и се зачувува во јазелот. Последователно, множеството податоци за обука се дели согласно избраниот тест и алгоритмот се повикува рекурзивно за да се конструира поддрво за секое подмножество во поделбата. Процесот на делење продолжува се додека не се исполни критериумот за прекин. Откако ќе се изгради дрвото, секој

лист се означува врз основа на записите од множеството за обука кои се сместени во него.

Clus го употребува истиот алгоритам за градење на сите типови на дрва за одлучување. Главната разлика е во тоа која функција за евалуација на хеуристиката е употребена. За класификациски дрва, функцијата ја користи информацискатa добивка и хеуристиката со степен на добивка. За градење на класификациски дрва со повеќе целни атрибути, *Clus* употребува тривијално проширување на иформацискатa добивка, во која мерката за ентропија е заменета со сума од ентропиите за различните целни атрибути.

За регресиските дрва, регресиските дрва со повеќе целни атрибути и дрвата за кластерирање, *Clus* користи хеуристика која се базира на интра-кластерната варијанса.

Обично, дрвата за одлучување се поткаструваат откако ќе бидат изградени. Методот за пост-поткаструване ги заменува поддрвата од дрвото за одлучување со листови, за на пример, да се подобри точноста на предвидувањето. *Clus* вклучува повеќе методи за поткаструване.

6.4 ГРАДЕЊЕ ДРВА СО КОРИСНИЧКИ ОГРАНИЧУВАЊА

Clus системот поддржува два вида на кориснички ограничувања: синтактички ограничувања (за структурата на дрвото) и ограничувања кои влијаат врз големината и точноста.

Синтактичките ограничувања кажуваат дека добиеното дрво мора да содржи одредено поддрво почнувајќи од кореновиот јазел. Ова поддрво може да биде специфицирано со документот за подесување на параметри. *Clus* ги спроведува ограничувањата на начин што го иницијализира алгоритмот за градење на дрвото со даденото дрво и за да го дограде дрвото користи обичен алгоритам за градење одгоре надолу. Можноста за употреба на синтактички ограничувања овозможува поголема вмешаност на корисникот во градењето на дрвото за одлучување и поголемо влијание на корисникот врз конечниот резултат. Синтактичките ограничувања се посебно корисни за да се вгради знаењето од областа во дрвото.

Втор вид на ограничувања кои се можни со *Clus* се ограничувања на големината на дрвото (најмногу даден број јазли) и на точноста на дрвото (најмногу дадена грешка). Овој вид на ограничувања се спроведува или со употреба на метод за

пост-поткаструване или со употреба на beam-search алгоритмот кој е вклучен во *Clus*.

Ако *Clus* се изврши во режимот beam-search, го зема предвид множеството од к-најдобри дрва кои се најдени до сега (beam-от). Овој beam се иницијализира со едно дрво кое се состои само од лист. Во секоја итерација на beam-search, beam-от се променува со конструирање на сите прочистувања на сите дрва од beam-от. Прочистување се добива кога листот се заменува со внатрешен јазел. Ова се повторува за секој можен тест кој може да се употреби во внатрешен јазел. Ако прочистувањето е подобро од најлошото дрво во beam-от, тогаш ова дрво се заменува со новото прочистување. Додека beam-от содржи помалку од k дрва, му се додаваат сите прочистувања. Критериумот за подредување на дрвата во beam-от може да се базира на точноста или на ентропијата (во случајот со класификациски дрва). Алгоритмот завршува ако не се најдат подобри прочистувања и се враќа beam-от со k-те најдобри дрва.

7. КОРИСНИЧКИ ИНТЕРФЕЈС НА СИСТЕМОТ CLUS

7.1 ВОВЕД

Развојот на податочното рударење ќе се одвива во насока на развој на теоријата на податочното рударење, усовршување на постоечките и креирање нови алгоритми за податочно рударење и креирање на интерфејси за алгоритмите со цел да се олесни работата на корисникот со истите. Добро познати се предностите што ги нуди интерфејсот за некоја програма: полесно управување со програмата, полесно задавање на влезни податоци, полесно разбирање на излезот од програмата и максимално искористување на можностите на програмата.

Како влез системот *Clus* користи два документа: првиот е базата на податоци (во .arff формат кој е компатибилен со WEKA) и документ за поставување на параметри (со екстензија .s). Во документот за поставување на параметри може да се постават сите можни параметри со кои може да работи системот: целни атрибути, атрибути кои не се земаат предвид при градењето на дрвото (disabled), грешка на дрвото (кога имаме дрва со повеќе целни атрибути има можност за поставување максимална грешка по сите целни атрибути поединечно или вкупно),

максимална големина (вкупен број на внатрешни јазли и листови), поставување на синтактички ограничувања, избор на алгоритам за поткаструване, избор на вредност за критериумот за запирање (F-тест) и многу други параметри кои можат потесно да го специфицираат она што треба системот да го заврши. Ако некој параметар не е специфициран, *Clus* користи предефинирани почетни вредности.

Излез од работата на ситемот *Clus* се два документа: еден со екстензија .out (тука во вид на текстуален документ се запишува дрвото што е добиено, кои се параметрите со кои е добиено дрвото, големината, точноста-грешката, како е извршено поткаструването и слично) и друг со екстензија .model (во кој се запишани моделите кои се добиени со обработката на податоците. Овој документ всушност служи за приказ на дрвата во графичка форма).

Добиените модели може да се видат со извршување на една од опциите на системот *Clus*, а тоа е -showTree и како аргумент се дава .model документот. Системот дава приказ на моделите и дава можност за визуелно поткаструване или проширување на дрвото.

Значи, не може да стане збор за конзистентност на моделите. При ново стартување на системот со истата база на податоци, претходно добиените модели се пребришуваат. Една можност за зачувување на моделите е преименување на .model документите, ама тоа не е практично ако сакаме да правиме споредба меѓу моделите или да пребаруваме низ претходно добиените модели.

Поставувањето на параметрите со кои работи *Clus* се прави со креирање на .s документот. Таму се запишуваат сите ограничувања со кои сакаме да работиме.

Претходните објаснувања се однесуваат за системот *Clus* пред да се направи интерфејсот.

Со креирањето на интерфејсот се овозможува полесно оперирање со системот. Со интерфејсот се овозможува:

- избор на целни и оневозможени атрибути;
- внесување на некои ограничувања: ограничување на грешката, ограничување на големината на дрвата и синтактички ограничувања;
- конзистентност на моделите и можност да се пребарува низ нив;
- визуелна манипулација со синтактичките ограничувања
- добиените модели може да се преработат и упатат како нови синтактички ограничувања.

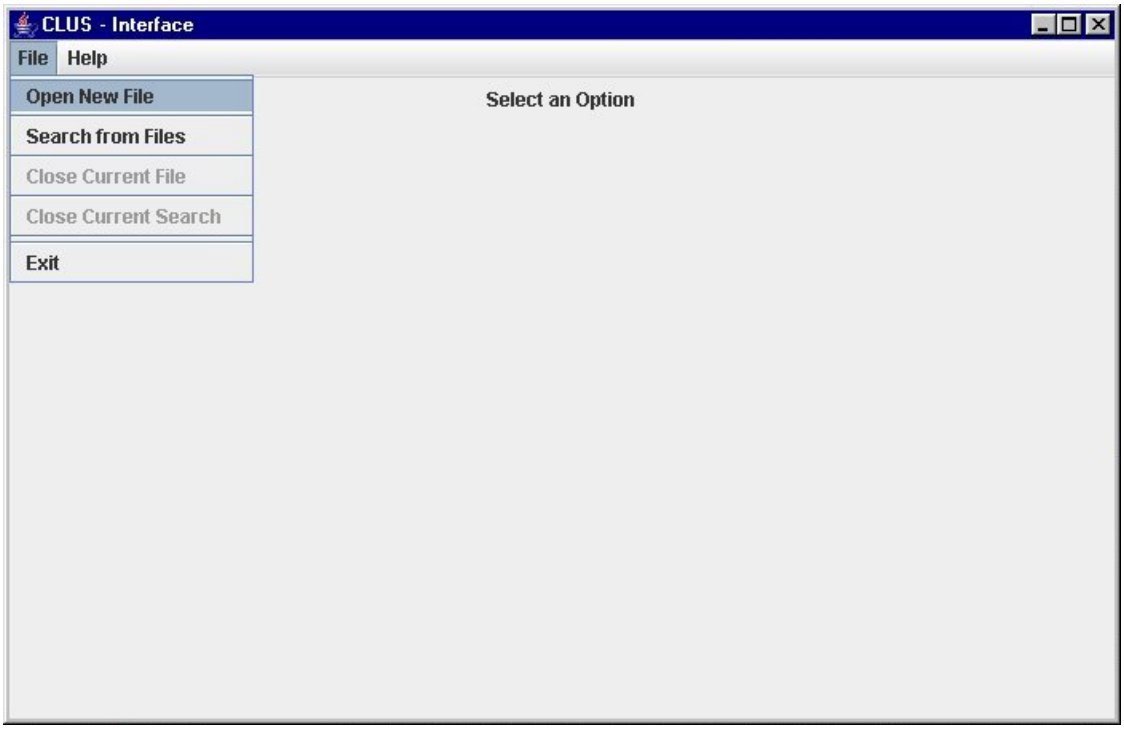
Интерфејсот може да се подели на два дела:

- учење нови модели и
- пребарување низ порано научените модели.

Уште со самото стратување на интерфејсот имаме избор во која насока ќе се движиме: ќе учиме нови модели или сакаме да пребаруваме низ веќе научените модели (слика 7.1).

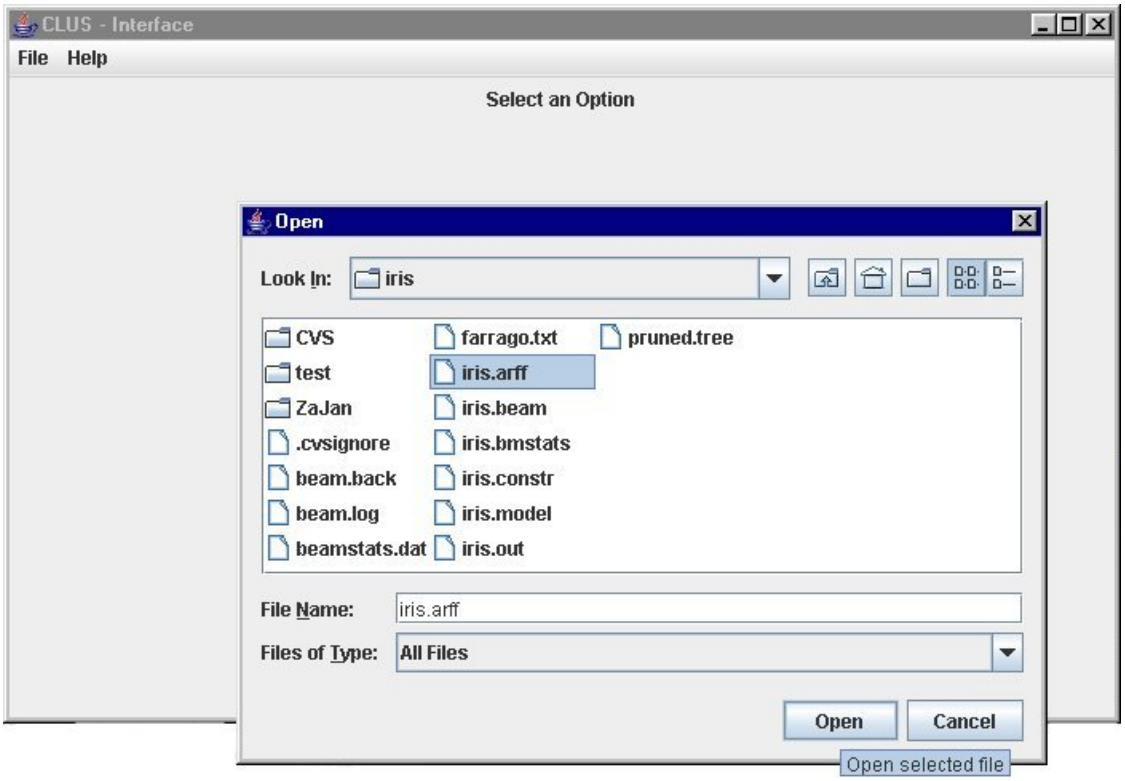
7.2 УЧЕЊЕ НОВИ МОДЕЛИ

Почнуваме со учењето нови модели ако од менито избереме File→Open New File како што е прикажано на слика 7.1.



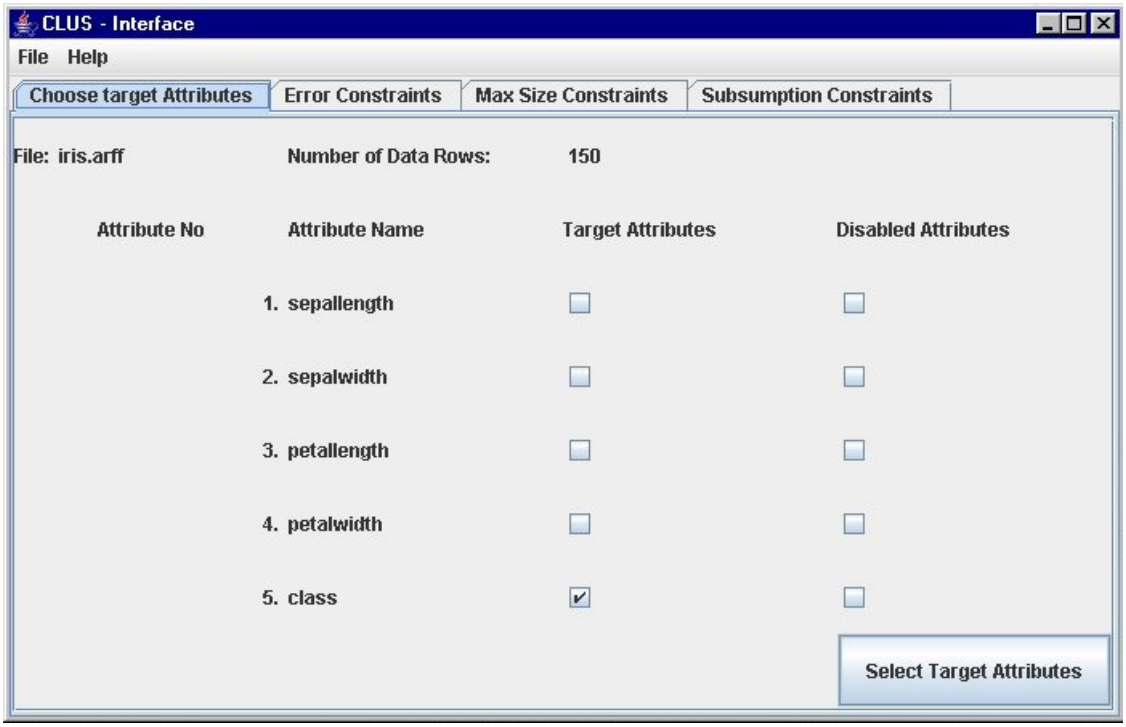
Слика 7.1. Почетен изглед на интерфејсот

По ова се појавува прозорец (слика 7.2) за избор на .arff документот (базата на податоци) со кој ќе работиме. Доколку избереме документ кој не е со екстензија .arff интерфејсот ќе пријави грешка и имаме можност повторно да се обидеме од почеток. Доколку избереме .arff документ, следно ни се прикажуваат сите атрибути и имаме можност да избереме кои се целните атрибути, а кои се оневозможени. Ако дојде до случај истовремено еден ист атрибут да е означен како целен атрибут и оневозможен атрибут, тогаш тој понатаму се третира како целен атрибут (слика 7.3).



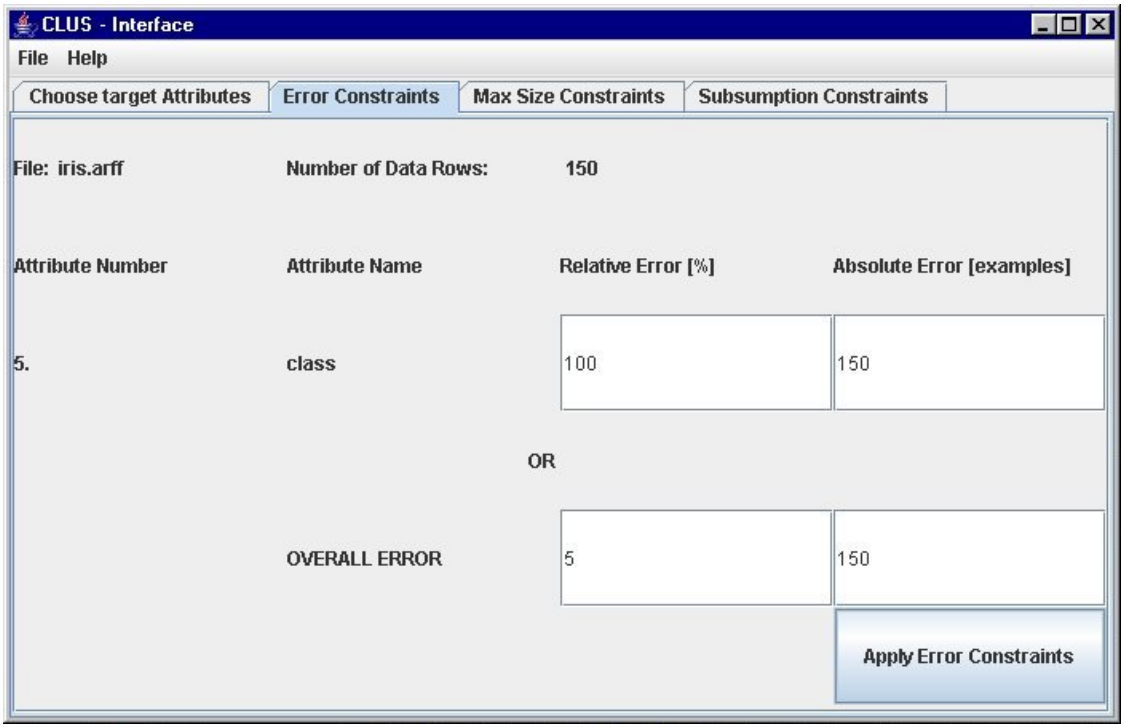
Слика 7.2. Избор на документ со кој ќе работиме

Ако сакаме да продолжиме понатаму со работа мораме да избереме целен атрибут или група целни атрибути.

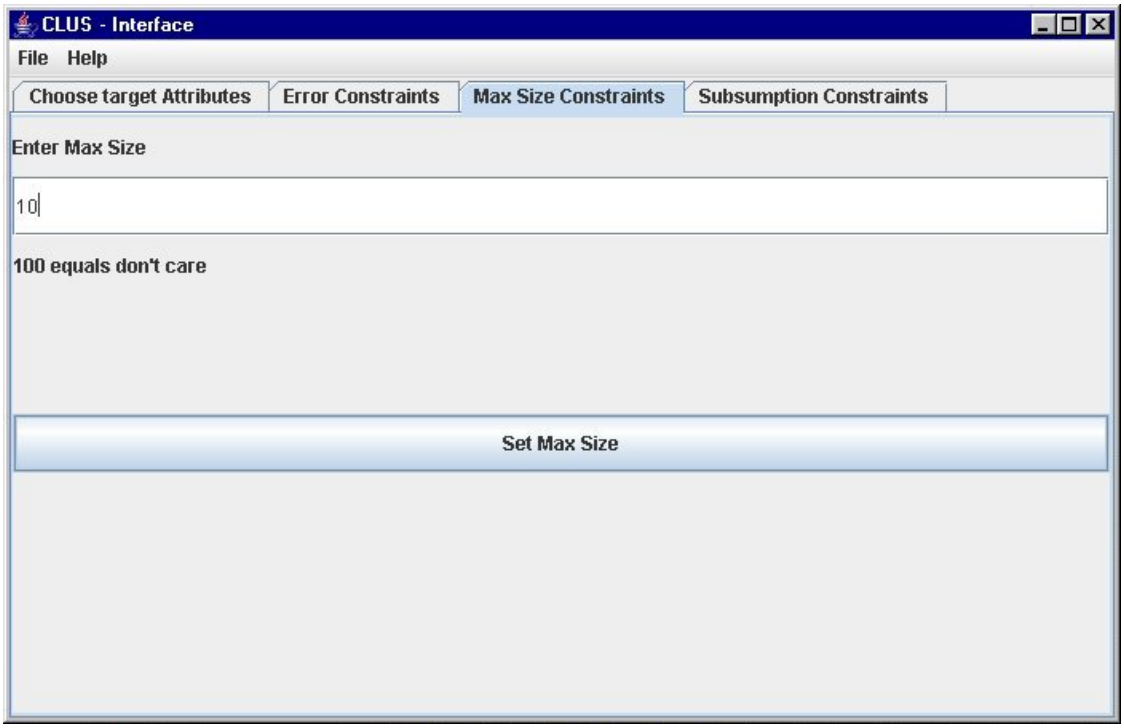


Слика 7.3. Избор на целни и оневозможени атрибути

По ова можеме да поставуваме ограничувања. Треба да се напомне дека мора се кликнат сите копчиња за ограничувања. Доколку сакаме ограничувањето за грешката да не влијае врз резултатот тогаш само го кликаме копчето Apply Error Constraints (слика 7.4) и за максимално дозволена грешка се запишува 100% (што значи дека добиеното дрво може да има колкава и да е грешка). Истото важи и при поставувањето на ограничувањето на големината на дрвата. Ако само се кликне копчето Set Max Size (слика 7.5) за максимално дозволена големина дрвото ќе биде поставена 100, што е доволно за реални случаи на бази на податоци.



Слика 7.4. Поставување на ограничувањата на грешката



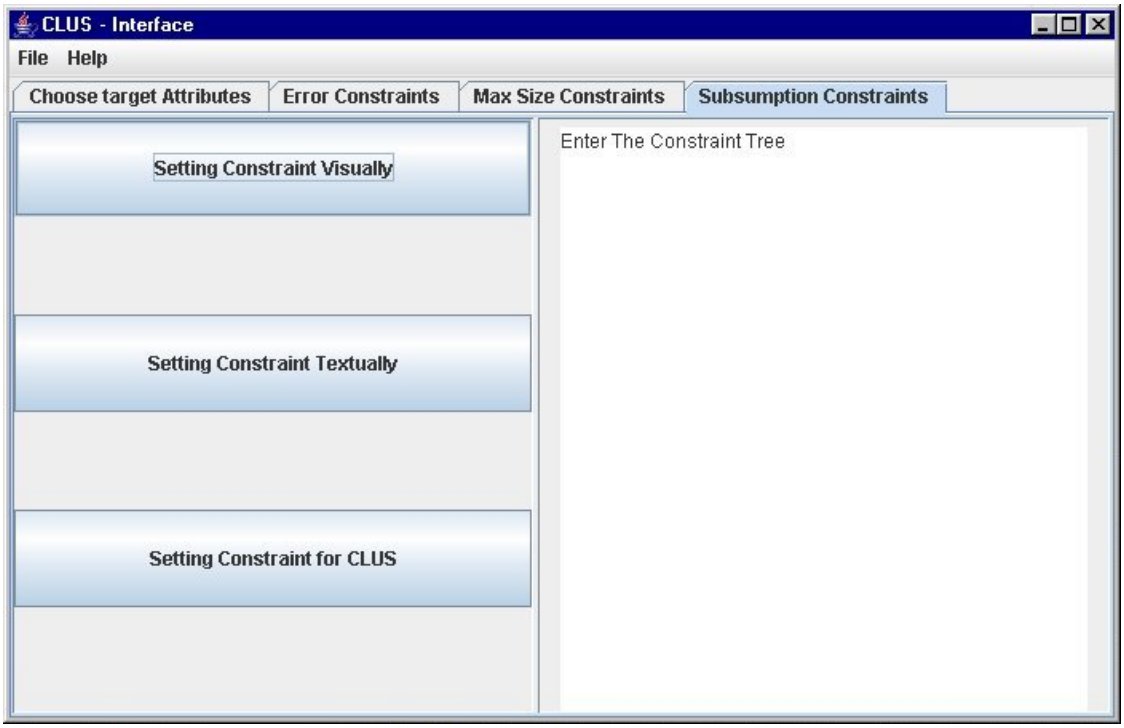
Слика 7.5. Поставување на ограничување на големината

Синтактичките ограничувања може да се внесуваат или визуелно или текстуално (слика 7.7). Ако не сакаме да внесеме синтактички ограничувања кликаме на копчето Setting Constraint for CLUS и предвид ќе бидат земени ограничувањата за грешка и големина на дрвото. Притоа, при текстуалното внесување на ограничувањето треба многу да се внимава на начинот на кој се внесува дрвото. Тоа мора да е во формат кој веќе го разбира системот и е прикажан на слика 7.6. Во овој формат, всушност *Clus* го прикажува дрвото во .out документот.

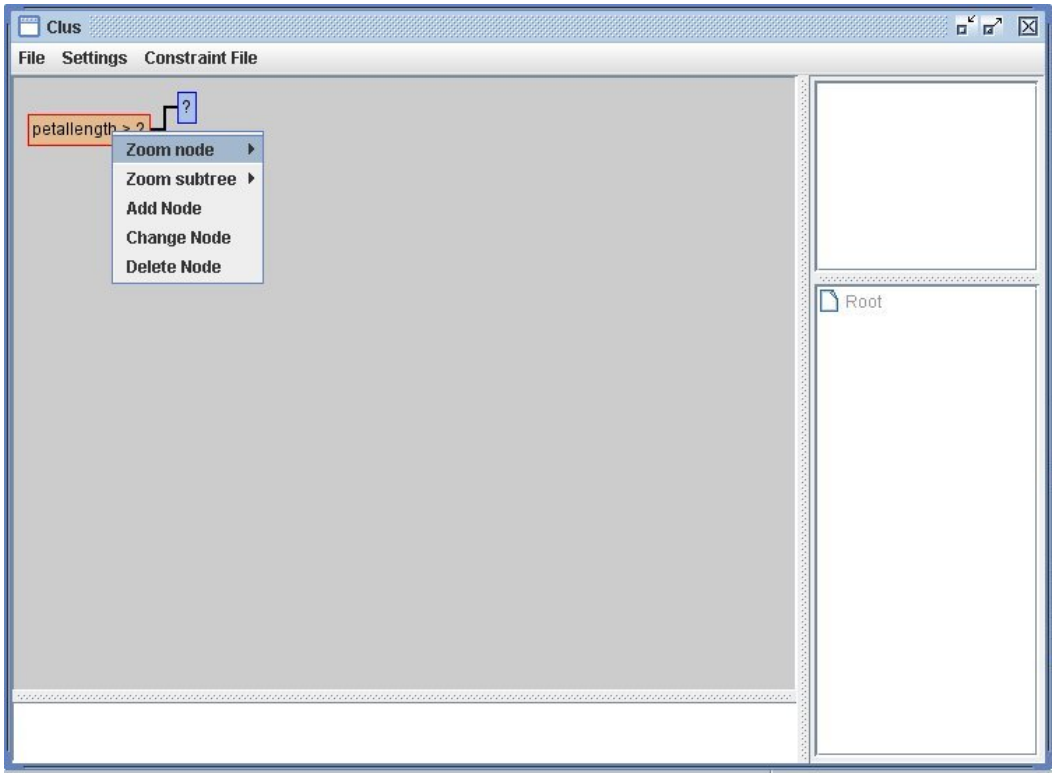
```
attribute > ?
+--yes: ?
+--no:  ?
```

Слика 7.6. Формат за текстуално внесување на синтактички ограничувања

- Доколку, избереме синтактичкото ограничување да го внесеме визуелно, тогаш се бара од нас да внесеме израз кој ќе биде корен на дрвото (слика 7.14). И овде мора да се внимава на имињата на атрибутите. Потоа се отвара нова рамка на која дрвото е прикажано графички (слика 7.8).



Слика 7.7. Избор за внесување синтактичко ограничување

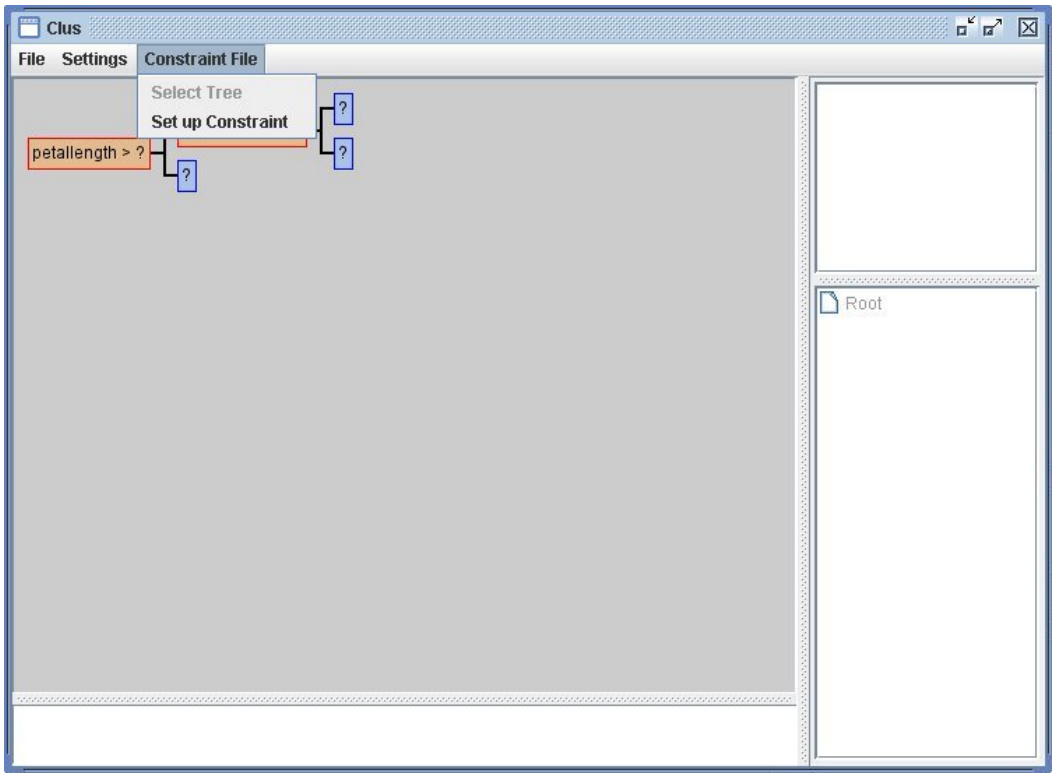


Слика 7.8. Визуелна манипулација со дрвото

На дрвото можеме да ги правиме следните манипулации:

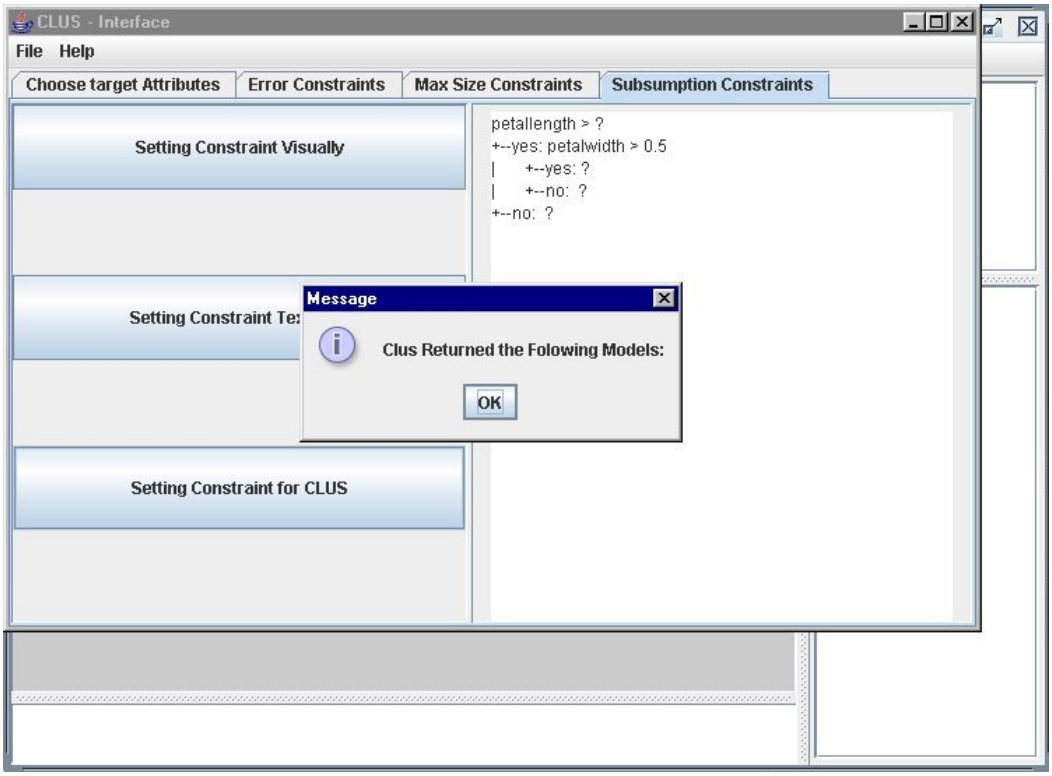
- додади јазел: се додава јазел по оној од кој сме ја избрале таа опција. Се внесува вредноста на јазелот и кажуваме во која гранка да го стави новиот јазел;
- измени јазел: се променува вредноста која е запишана во јазелот;
- избриши јазел: се брише јазелот кај кој сме ја избрале таа опција и сите негови деца.

Откако ќе завршиме со сите манипулации со дрвото од менито избираме Constraint File→Set up Constraint (слика 7.9) и со тоа дрвото кое сме го креирале се поставува за синтактичко ограничување со кое ќе работи системот.



Слика 7.9. Визуелно поставување на синтактичко ограничување

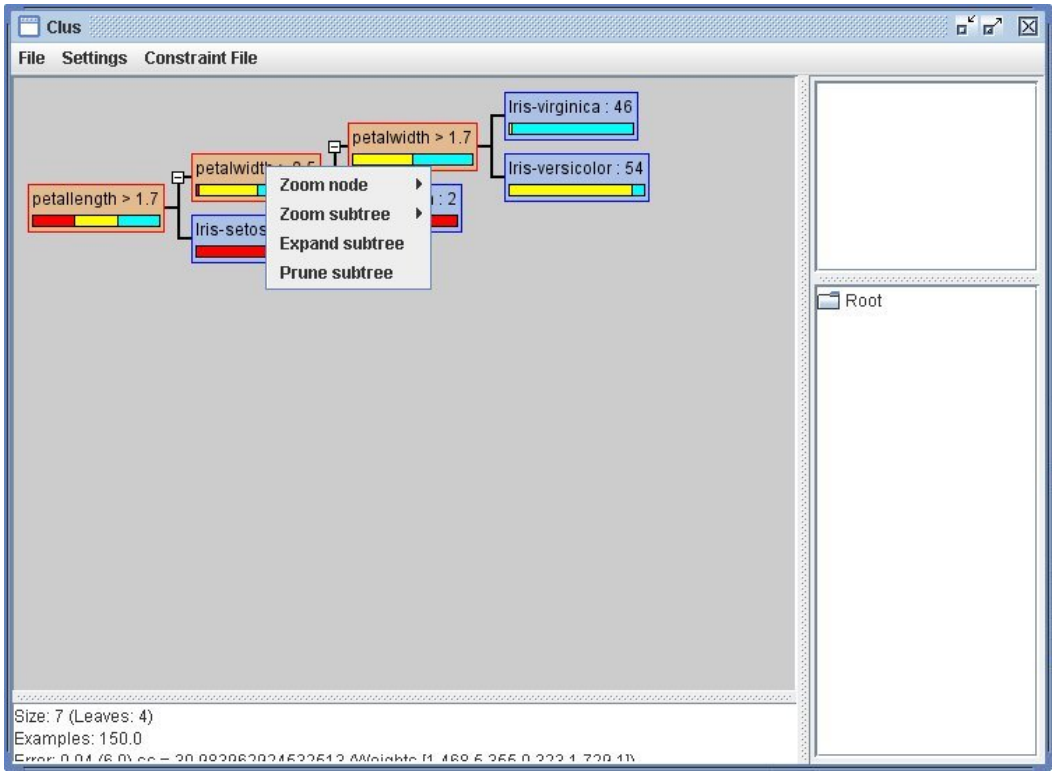
Сите ограничувања и .arff документот се предаваат на *Clus* да ги обработи. По обработката се пријавува завршаната работа (Слика 7.10).



Слика 7.10. Обработка на податоците со избраните ограничувања

Со кликање на OK ни се прикажуваат добиените модели. При овој приказ врз моделите може да се прави поткастрување и проширување (Слика 7.11). Ако сакаме некој од добиените модели да го искористиме како ново синтактичко ограничување тогаш од менито избираме Constraint File→Select tree. Притоа ќе треба да избереме дали сакаме да правиме некои промени во ограничувањето на грешката или големината на дрвото.

Притоа, сите модели кои се добиваат од обработката на податоците се зачувуваат за подоцна да можеме да пребаруваме низ нив. Инаку, клучно при запишувањето на научените модели е името на базата на податоци која се обработува. На пример, ако се обработува базата на податоци xyz.arff, тогаш сите модели се запишуваат во xyz.model. Кога сакаме да пребаруваме низ научени модели, прво избираме низ кои модели сакаме да пребаруваме. Значи, претходно научените модели се запишуваат во документи кои се чуваат во рамките на проектот.



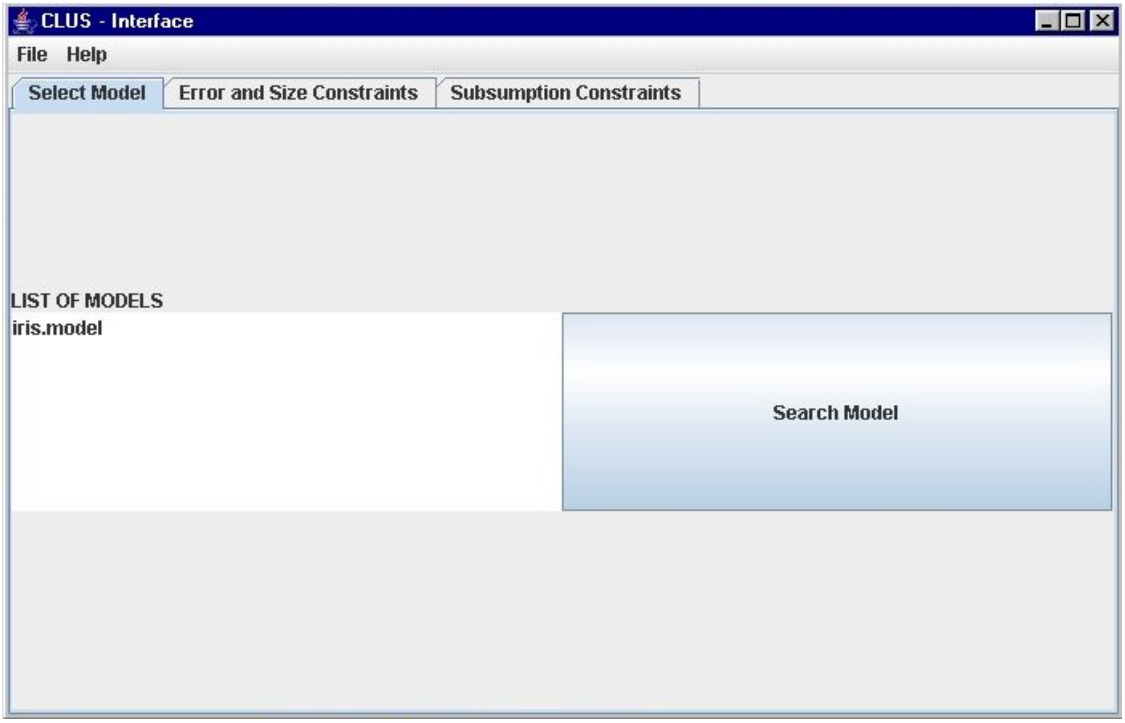
Слика 7.11. Приказ на добиените модели

Кога ќе завршиме со работа на базата на податоци избираме File→Close Current File. Тоа ни овозможува да избираме дали ќе работиме со некоја нова база на податоци или пак ќе пребаруваме низ веќе научените модели.

7.3 ПРЕБАРУВАЊЕ НИЗ ВЕЌЕ НАУЧЕНИ МОДЕЛИ

Почнуваме со пребарување низ научените модели ако од менито избереме File→Search from Files (прикажано на слика 7.1).

Потоа мора да избереме низ кои модели ќе пребаруваме. Се појавува листа од која треба да избереме (Слика 7.12). Додека не избереме не можеме да задаваме ограничувања (критериуми) за пребарување.

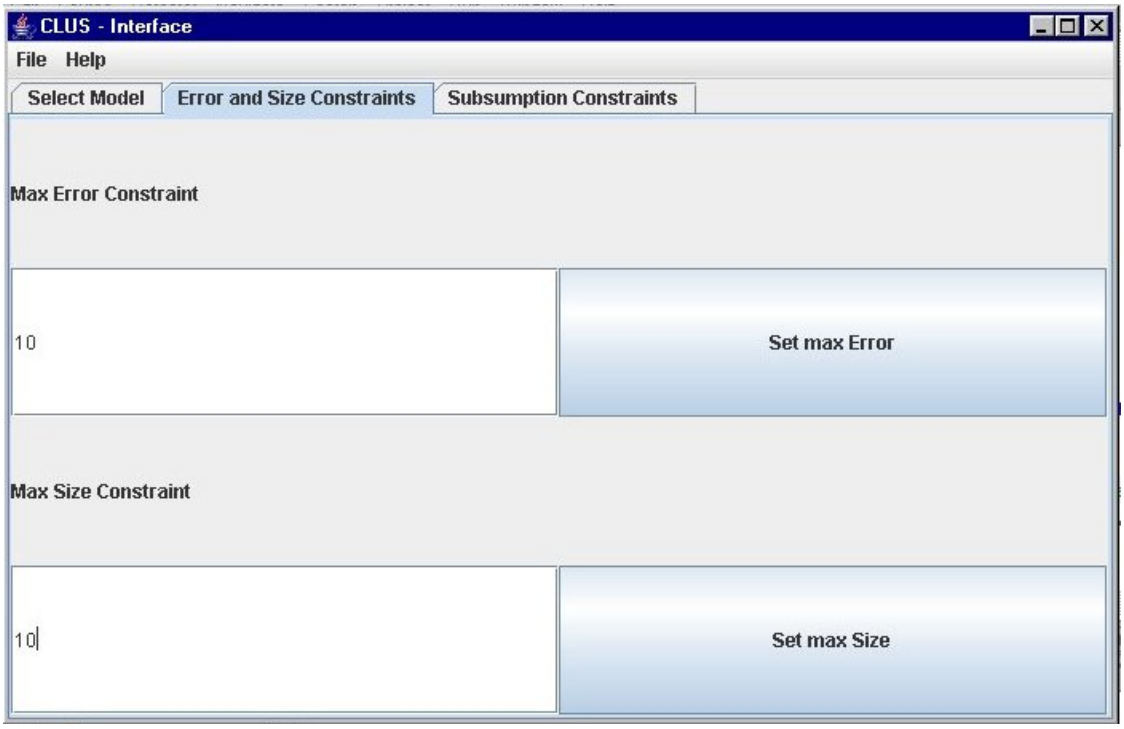


Слика 7.12. Избор на модели низ кои ќе пребаруваме

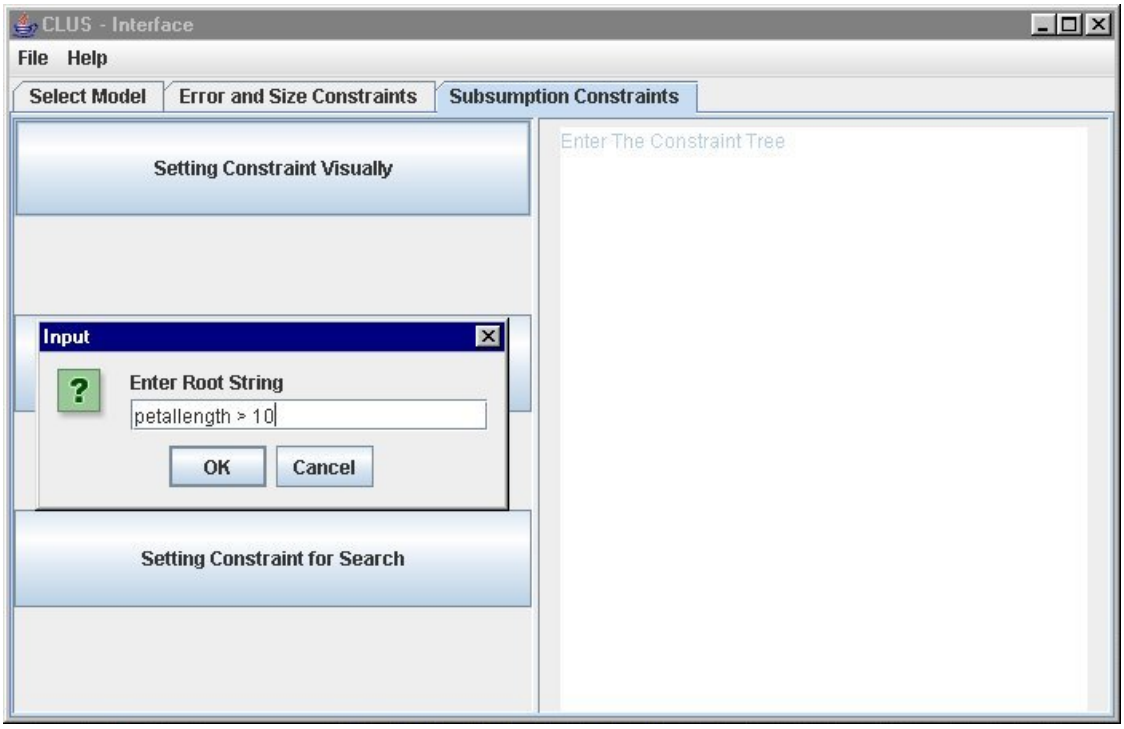
Можеме да даваме ограничувања за вкупната грешка на дрвото, вкупната големина на дрвото и синтактичките ограничувања. Меѓу овие ограничувања може да се направат сите логички поврзувања (ова е прикажано на слика 7.15). Притоа, важат истите работи како кај учењето на нови модели. Ако сакаме грешката да не се зема предвид, оставаме максимално дозволената грешка да биде 100%. Слично и за максималната големина на дрвото. Секако, можеме да избереме да не ставиме синтактички ограничувања. На слика 7.13 прикажано е поставувањето на максимално дозволената грешка и максималната големина на дрвото.

Кога се пребарува низ моделите за да се најдат оние кои ги исполнуваат бараните услови и кога треба да се споредува максимално дозволената грешка, всушност се споредува:

- средна квадратна грешка (RMSE) во случајот кога имаме регресиски дрва за предвидување на вредноста на еден атрибут и дрва за предвидување на вредноста на повеќе атрибути и
- (1-точност) во случајот кога имаме предвидување на вредноста на номинален атрибут.



Слика 7.13. Поставување на ограничувањата за грешка и големина

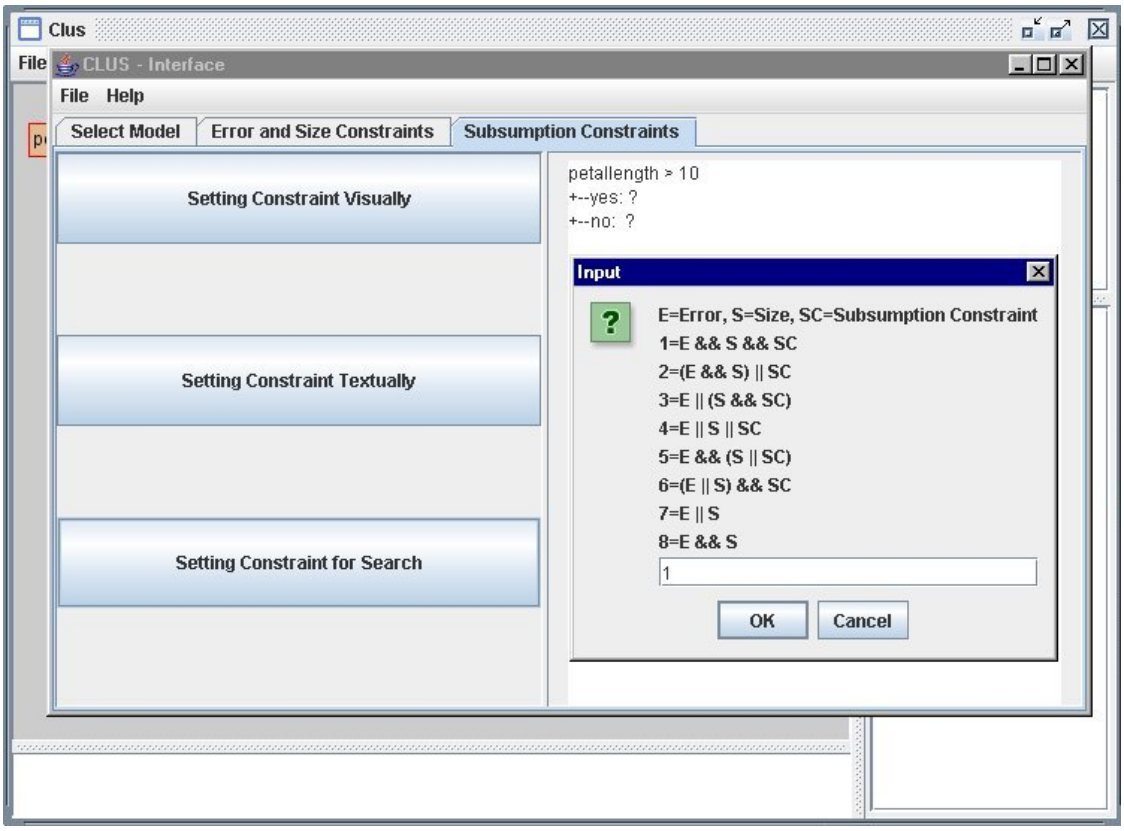


Слика 7.14. Поставување на синтактички ограничувања

Ако избереме синтактичките ограничувања да ги поставиме визуелно, прво ја внесуваме вредноста на коренот. Ако избереме овие ограничувања да ги внесеме текстуално тогаш едноставно ги запишуваме во текстуалната област оддесно.

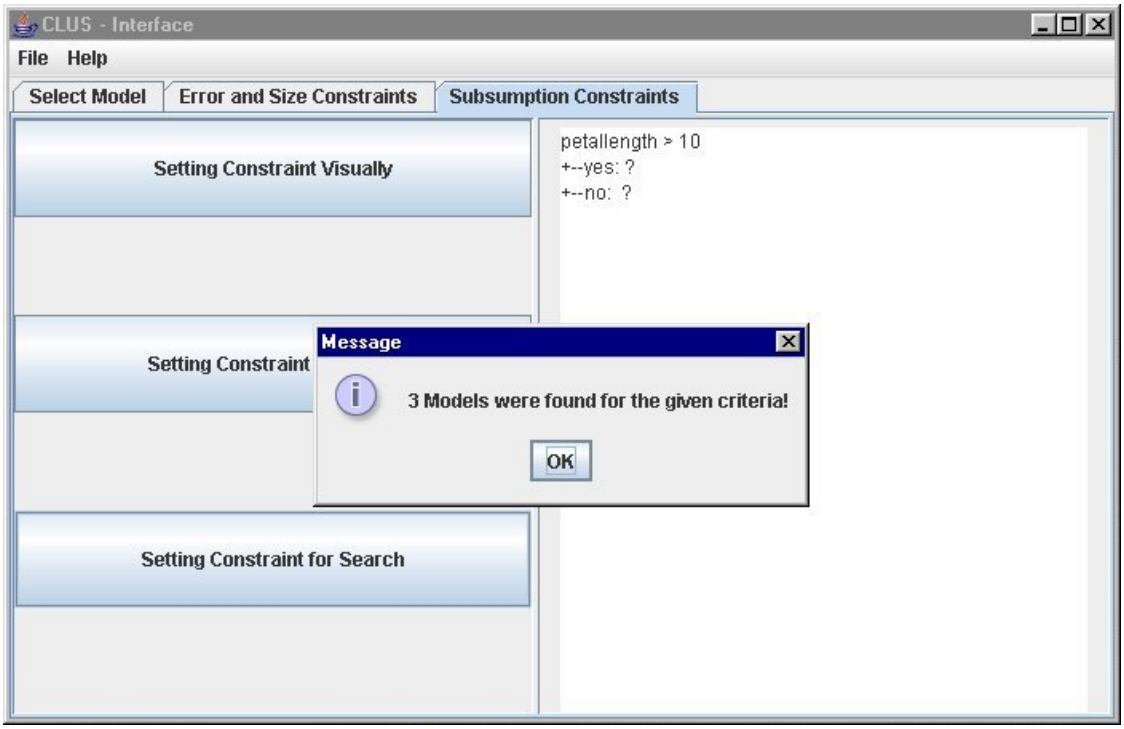
Притоа, важат правилата за формат на внесеното дрво кои беа спомнати претходно.

Правилата за визуелна манипулација со дрвото се исти како оние од претходно. За да дрвото биде поставено како синтактичко ограничување, мора да избереме Constraint File→Set up Constraint (Слика 7.9). Откако дрвото ќе се постави за ограничување, тогаш избираме како ќе бидат логички поврзани ограничувањата (Слика 7.15).



Слика 7.15. Логичко поврзување на ограничувањата

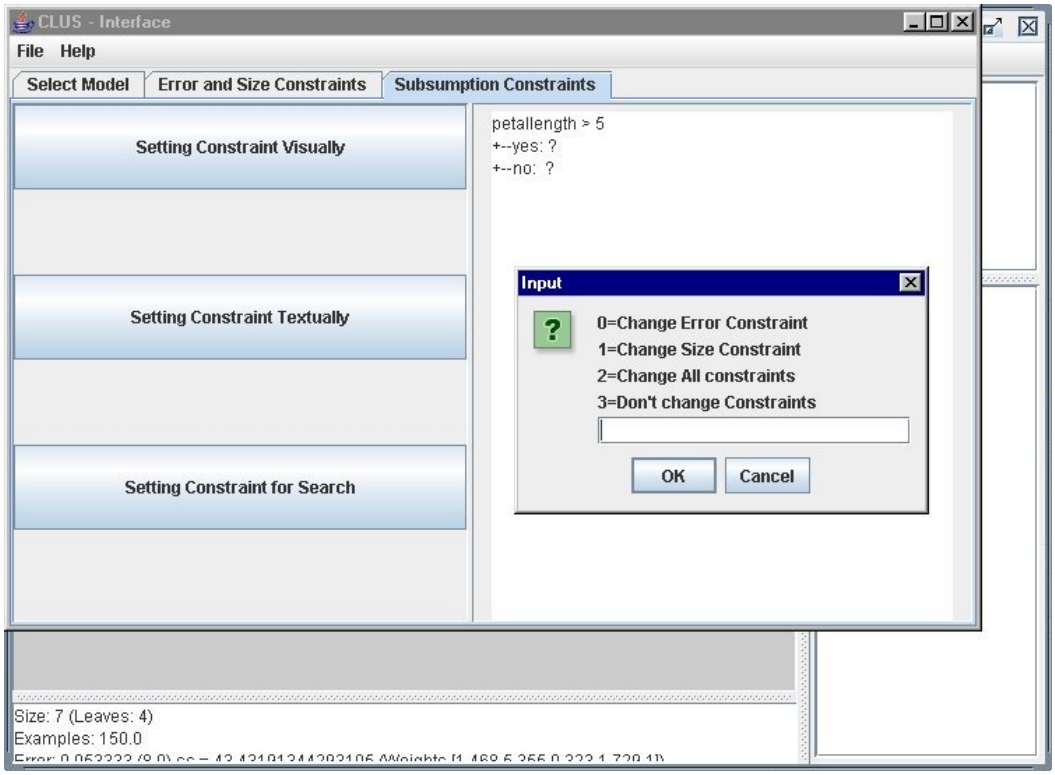
Откако ќе завршиме со ова, се пребарува низ моделите и се наоѓаат оние кои ги задоволуваат поставените услови. Се прикажува колку такви модели се пронајдени (Слика 7.16) и потоа ги прикажува моделите.



Слика 7.16. Приказ на состојбата по завршување на пребарувањето

Добиените модели може повторно да ги искористиме како синтактички ограничувања и да им правиме разни манипулации (додади, промени и избриши јазел), а тоа го правиме со избор на Constraint File→Select tree од менито.

Кога ќе завршиме со манипулација треба уште само да кажеме дали сакаме да правиме промени во останатите ограничувања (Слика 7.17).



Слика 7.17. Повторна употреба на добиените модели

На крај, кога ќе завршиме со пребарувањето низ веќе добиените модели избираме File→Close Current Search.

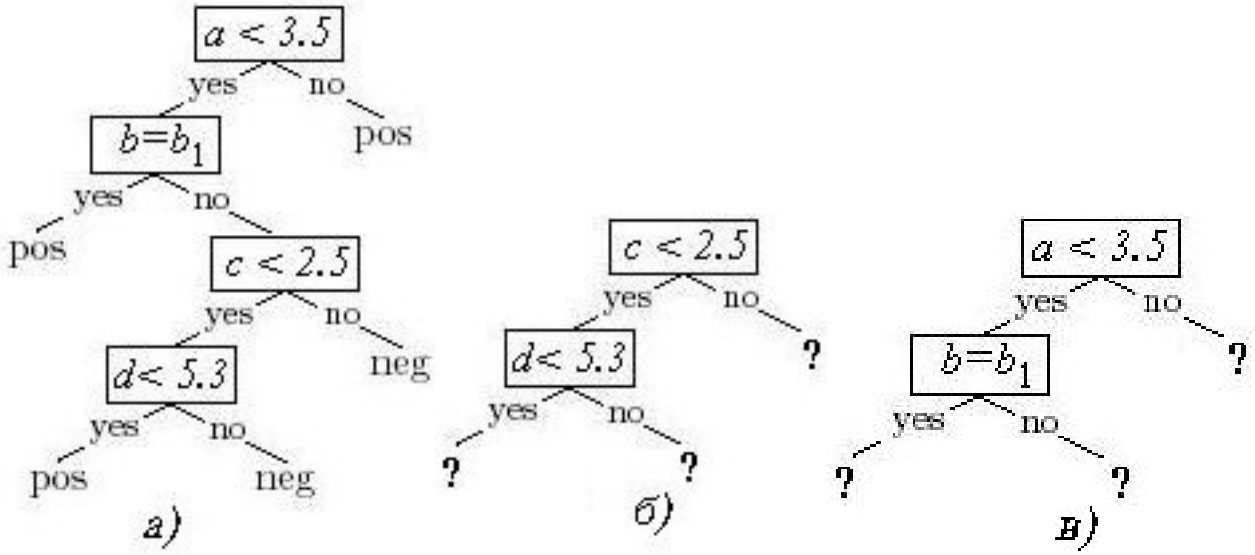
Кога сакаме целосно да завршиме со работата избираме File→Exit.

7.4 ПОМОШНИ АЛГОРИТМИ

Интерфејсот и системот *Clus* тесно се поврзани бидејќи интерфејсот користи некои алгоритми кои веќе се имплементирани во системот. Во прва мера, интерфејсот го користи алгоритмот за запишување на дрво во документ, за читање на дрво од документ и го користи механизмот за визуелен приказ на дрвото. Визуелизација што ја има оригиналниот систем, мораше да претрпи промени со цел да се овозможат новите манипулации врз дрвата (додавање, измена и бришење јазел). Овие манипулации се овозможени со додавање на методи во класата за јазел (*ClusNode*).

Дополнително, интерфејсот има посебен парсер за .arff документи, кој го враќа бројот на атрибути, имињата на атрибутите и бројот на редови со податоци.

Интересен е и алгоритмот за проверка дали едно дрво е поддрво на дрво. Поточно, алгоритмот проверува дали едно дрво е подмножество од друго дрво. Ова е илустрирано на слика 7.18.



Слика 7.18. Приказ на дрва

Алгоритмот ќе врати дека дрвата на слика 7.18.б и слика 7.18.в се поддрва на дрвото од слика 7.18.а. Значи, во кој било дел од дрвото да се наоѓа поддрвото, алгоритмот ќе врати потврден одговор (концептот на подмножество-subsumption).
Исто така, имплементиран е алгоритам за проверка низ моделите кој ги враќа оние модели кои ги задоволуваат бараните критериуми.

7.5 ОГРАНИЧУВАЊА НА ИНТЕРФЕЈСОТ

Ограничувањата на интерфејсот се во фактот дека нема некоја голема контрола на грешка. Значи може некои грешни внесувања да поминат незабележано од него, а подоцна *Clus* системот да прави грешка или пак воопшто да не работи. Ова е бидејќи се претпоставува дека со софтверот ќе работи некој што има основни познавања од областа и податочното рударење.
Ограничување може да се смета и тоа што кога пребаруваме низ моделите, одбираме низ кои модели (кои се добиени од некое множество на податоци) ќе пребаруваме, а не можеме да пребаруваме низ сите модели, на пример, да ги најдеме сите модели кои имаат грешка помала од 5% и/или големина помала од 10.

Ограничување е тоа што не може да се каже да го даде моделот со најмала или најголема грешка (во случај на повеќе исти да го даде оној со минимална големина или максимална) и не може да се каже да се најде моделот со минимална или максимална големина (во случај на повеќе исти да го даде оној со минимална или максимална грешка).
Невозможно би било да се постават синтактички ограничувања за сите модели бидејќи тие се изградени од различни множества на податоци.
Како ограничување може да се смета и дека во научените модели може да се сретнат два или повеќе исти модела (нема отстранување на исти модели).
Голем дел од овие ограничувања би можело да бидат надминати во некоја следна верзија на интерфесот.

7.6 ДИЗАЈН НА ИНТЕРФЕЈСОТ И ОРГАНИЗАЦИЈА НА ПОДАТОЦИТЕ

Интерфејсот како и *Clus* системот се напишани во JAVA. Системот *Clus* е напишан од Jan Struyf и Hendrik Blockeel од Katholieke Universiteit Leuven, Belgium. За пишување на интерфејсот користена е платформата ECLIPSE 3.0.
За градбата на интерфејсот користен е swing пакетот од JAVA кој има многу компоненти за визуелен приказ. Сите компоненти може да се забележат на сликите од слика 7.1 до слика 7.17. Користени се JFrame, JPanel, JLabel, JTextField, JTextArea, JSplitPane, JScrollPane, JTabbedPane, JButton, JCheckBox, JMenu, JList и други.
Контролата врз работата на интерфејсот се изведува со логички променливи.
Сите податоци што требаат се запишуваат во документи. Овде се наметнува прашањето: зошто да не се употреби некој сервер за бази на податоци? Одговорот е дека употребата на сервер за бази на податоци само би го искомплицирал решението на проблемот. Ова е заради тоа што самото дрво носи со себе многу други информации (најразлични статистики), поради што неговото запишување во база не е толку тривијално. Тоа претставува донекаде ограничување во смисла дека не може да се пребарува низ резултатите кои ги добиваат повеќе луѓе кои работат со истото множество на податоци.
Во текот на работата интерфејсот креира неколку помошни документи кои ги брише кога ќе се заврши со работа.
Синтактичките ограничувања кои се даваат при учење на нови модели остануваат запишани во .constr документот во директориумот со изворниот .arff документ. Ова

е согласно со потребите на системот *Clus* кој го вчитува синтактичкото ограничување од документ со екстензија .constr.

8. ЗАКЛУЧОК

Количеството на податоци се зголемува двојно секоја година, ама се чини дека количество на корисна информација опаѓа. За да се помогне во решавањето наовој проблем се развива податочното рударење. Податочното рударење не е само интересна област за истражување, туку има и голема реална примена. Доста јасна е потребата од развивање на побрзи и поефикасни алгоритми за податочно рударење кои ќе имаат добар кориснички интерфејс.

Системот *Clus* немаше никаков кориснички интерфејс. Единствено имаше можност графички да се прикажат добиените резултати. Овој интерфејс за системот *Clus* е прв. Паралелно со еволуирањето на *Clus* би требало да еволуира и неговиот интерфејс, зошто тие се тесно поврзани. Новите верзии на интерфејсот би требало да ги надминат ограничувањата дадени во 7.5 и да воведат нови делови, така што би ја опфатиле целосната манипулација (поставување на сите можни параметри) со *Clus* и со тоа би развиле еден современ систем за податочно рударење.

9.ЛИТЕРАТУРА

1. Margaret Dunham - DATA MINING Introductory and Advanced Topics, Pearson Education 2003, ISBN 0-13-088892-3
2. Saso Dzeroski - Data Mining in a Nutshell
3. Hendrik Blockeel, Luc De Raedt and Jan Ramon – Top-down induction of clustering trees
4. Saso Dzeroski, Ljupco Todorovski and Hendrik Blockeel – Relational Ranking with Predictive Clustering Trees
5. Hendrik Blockeel, Maurice Bruynooghe, Saso Dzeroski, Jan Ramon and Jan Struyf – Hierarchical multi-classification
6. Jan Struyf – Description of CLUS system (draft)
7. Jan Struyf and Saso Dzeroski – Constraint Based Induction of Multi-Objective Regression Trees
8. Taneli Mielikäinen – Inductive Databases as Ranking
9. Jean-François Boulicaut, Mika Klemettinen and Heikki Mannila – Modeling KDD Processes within the Inductive Database Framework
10. Jean-François Boulicaut and Baptiste Jeudy – Using Constraints During Set Mining: Should We Prune or not?
11. http://java.sun.com
12. JAVA tutorials