Strojové učenie

Princípy a algoritmy

Kristína Machová

Košice
2002

Ing. Kristína Machová, CSc.
Katedra kybernetiky a umelej inteligencie
Fakulta elektrotechniky a informatiky
Technická univerzita v Košiciach
machova@tuke.sk

Táto práca bola vydaná s finančnou podporou Fakulty elektrotechniky a informatiky TU Košice. Publikácia vznikla aj za podpory VEGA grantu MŠ SR č. 1/8131/01 „Znalostné technológie pre získavanie a sprístupňovanie informácií“.

Lektorovali: doc. Ing. Marián Mach CSc., FEI TU Košice
Ing. Vojtech Svátek PhD., EU Praha
Ing. Ján Paralič PhD., FEI TU Košice

ISBN

PREDSLOV

Strojové učenie nepatrí medzi najnovšie oblasti výskumu. Napriek tomu sa mu venuje aj dnes veľa pozornosti a to v takých oblastiach ako: objavovanie znalostí (Knowledge Discovery), ILP (Inductive Logic Programming) a indukcia množín klasifikátorov (bagging, boosting, ...).

Predkladaný učebný text si nečiní nároky na vyčerpávajúci prehľad všetkých tém strojového učenia, čo by ho robilo príliš rozsiahlym. Podáva základy strojového učenia, pričom sa zameriava na princípy a algoritmy. Je určený predovšetkým poslucháčom 4. ročníka Fakulty elektrotechniky a informatiky v odbore Umelá inteligencia pre rovnomenný predmet Strojové učenie. Tento predmet nadväzuje na predmet Znalostné systémy. Rozširuje hlavne tú časť znalostných systémov, ktorá sa venuje získavaniu znalostí pre znalostné systémy automatickým spôsobom – induktívnym učením. Na predmet strojové učenie nadväzuje predmet Objavovanie znalostí, kde sa využívajú niektoré algoritmy strojového učenia.

Daný učebný text obsahuje tri hlavné okruhy problémov: **problém učenia logickej reprezentácie s učiteľom** (patria sem kapitoly: Generovanie logických konjunkcií, Generovanie produkčných pravidiel, Rozhodovacie stromy, Generovanie rozhodovacích zoznamov), **problém učenia s prvkami kvantitatívneho usudzovania s učiteľom** (Indukcia prahových pojmov, Indukcia etalónov, Pravdepodobnostný popis) a **problém učenia bez učiteľa** (Učenie odmenou a trestom, Zhlukovanie). Štruktúra kapitol prvých dvoch okruhov problémov je jednotná. Kapitoly sa spravidla začínajú podkapitolami venovanými reprezentácii pojmu a použitiu danej reprezentácie. Napokon sa pozornosť venuje indukcii danej reprezentácie pojmu, kde sú uvedené konkrétne algoritmy používané na indukciu.

Vyššie uvedené témy predstavujú určité kombinácie „logického kalkulu“ a „odvodzovacieho usporiadania“, ktoré sa ustálili v praxi. Je potrebné si uvedomiť, že ide o pragmaticky prístup a do úvahy prichádzajú aj iné možnosti. Ďalej je potrebné podotknúť, že členenie logických prístupov má dve hlavné dimenzie. Prvou je tvar logických formúl, ktorý je povolený. To znamená, akého typu môžu byť literály (atribút - hodnota, interná disjunkcia, množiny hodnôt) a aká môže byť celá formula (striktná konjunkcia, booleovský výraz, predikátové formule)?. Druhou dimenziou členenia je to, ako sú formule usporiadané pre rozhodovanie (skladanie príspevkov na rovnakej úrovni, „voting“, orientovaný acyklický graf, lineárny zoznam). Od toho sa odvíja postup používaný pri indukcii.

Moja vďaka patrí Doc. Ing. Marianovi Machovi CSc., Ing. Vojtechovi Svátekovi PhD. a Ing. Janovi Paraličovi PhD. za starostlivé prečítanie rukopisu a hlavne za podnetné návrhy , ktoré v konečnej verzii textu boli zohľadnené.

V Košiciach 16.9.2002 autor

OBSAH

1 ÚVOD DO STROJOVÉHO UČENIA

V mnohých technických disciplínach sa vedci inšpirujú človekom. V strojovom učení sa inšpirujeme kognitívnymi procesmi, ktoré prebiehajú u človeka. Na ilustráciu veľmi jednoduchý príklad. Predstavme si situáciu, v ktorej dieťa čupí pri poľnom kvietku a volá „aká pekná trávička“. Rodič mu povie, že je to kvietok, lebo je farebný. Na kvet si sadne motýľ. Dieťa povie „priletel kvietok farebný“. Rodič to uvedie na pravú mieru „to je farebný motýľ, kvietok nemá krídla a nelieta“. Dieťa zbadá sadnúť na konár slávika. „Aha motýľ má krídla a lieta“. Rodič mu vysvetlí, že je to vták, ktorý na rozdiel od motýľa znáša vajíčka. Dieťaťu na nos sadne mucha. Dieťa, už opatrnejšie sonduje „znáša to vajíčka?“ Rodič povie, že áno. Teda to musí byť vták. Od rodiča sa dozvie, že vtáci na rozdiel od hmyzu, znášajú vajíčka so škrupinou. Výsledkom tejto výmeny otázok a odpovedí je, že si dieťa v hlavičke sformuje svoju predstavu o pojme vták. Vták je farebný, má krídla, lieta a znáša vajíčka so škrupinou. Pojem bude reprezentovaný množinou vlastností – atribútov, ktoré sú charakteristické pre daný pojem. V podobnej situácii je študent predmetu strojové učenie. Musí sa naučiť odlíšiť jednotlivé reprezentácie pojmu a rôzne učiace sa algoritmy ako aj ich princípy.

Zjednodušene povedané, ľudia sa učia pojmy tak, že hľadajú pre tieto pojmy charakteristické vlastnosti. Aj v strojovom učení ide prevažne o to, sformulovať popis pojmu pomocou jeho charakteristických vlastností, resp. atribútov.

Strojové učenie je predmetom skúmania kognitívnej psychológie, tak ako aj umelej inteligencie. Schopnosť učiť sa je považovaná za jednou z kľúčových vlastností inteligencie. Podobne aj strojové učenie ako vedná disciplína sa vyhranilo na poli umelej inteligencie. (Carbonell-Michalski-Mitchell, 1983), (Dietterich at al., 1982) a (Langley-Carbonell, 1984) uskutočnili vo svojich prácach podrobný prehľad oblasti strojového učenia z rozličných perspektív. Strojové učenie sa zameriava na výpočtové procesy, ktoré tvoria základ učenia nielen u ľudí, ale aj u strojov.

1.1 Ciele strojového učenia

Prúdy strojového učenia sa zjednocovali sústredením sa na učenie. Rôzni výskumníci sa však sústreďovali na problém učenia z rozličných dôvodov.

Jedna skupina výskumníkov si kládla za cieľ modelovanie mechanizmov, ktoré tvoria základ ľudského učenia. V tomto rámci vyvíjali výskumníci učiace algoritmy, ktoré sú konzistentné s ľudským spôsobom poznávania a architektúrou, ktorú človek používa na ukladanie znalostí a vzťahov medzi nimi. Tieto algoritmy sú navrhované s cieľom, aby vysvetlili špecifiká pozorovaných návykov učenia. Niektoré metódy vysvetľujú tieto návyky na kvalitatívnej úrovni, zatiaľ čo iné pracujú v kvantitatívnej rovine, napr. narábajú s rozsahom chyby a reakčným časom ľudských subjektov. Keďže tieto metódy vytvárajú predikcie návykov učenia, sú potenciálne použiteľné v návrhu inštrukčných materiálov pre vzdelávanie ľudských subjektov.

Ďalšia skupina pristupovala k štúdiu strojového učenia empiricky. Empirický prístup predstavoval snahu objaviť všeobecné princípy, ktoré sa vzťahujú na charakteristiky učiacich

algoritmov a všeobecné princípy domén, v rámci ktorých tieto algoritmy operovali. Robili sa experimenty, v ktorých sa menili tak algoritmy ako aj domény a pozorovali sa dôsledky tejto manipulácie na učenie. Experimentálne štúdie zo strojového učenia vyprodukovali veľké množstvo empirických zovšeobecnení o alternatívnych metódach, ktoré poukazovali:

- na použiteľnosť jednotlivých tried algoritmov pre riešenie rôznych typov úloh
- na slabosti skúmaných algoritmov
- na možnosti ich vylepšenia
- na zdroje obtiažnosti úloh.

Iná skupina sa sústreďovala na všeobecné princípy. So strojovým učením nakladala ako s matematickou oblasťou. Ukladala si za cieľ formulovať a dokazovať teorémy o spracovateľnosti celých tried problémov učenia a algoritmov navrhnutých na riešenie týchto problémov.

Posledná skupina pristupovala k strojovému učeniu aplikačne. Zameriavala sa na:

- formuláciu problému v termínoch strojového učenia,
- návrh reprezentácie trénovacích príkladov, ako aj naučených znalostí,
- zhromaždenie trénovacích príkladov,
- generovanie bázy znalostí použitím strojového učenia.

Čo spájalo všetky tieto prístupy, bol záujem o rozvoj, porozumenie a ohodnotenie učiacich algoritmov. Strojové učenie je teda veda o algoritmoch.

1.2 Rámec strojového učenia

Čitateľ pravdepodobne na tomto mieste očakáva jasnú definíciu učenia. Pokusy určiť fixné hranice tak širokého pojmu neuspeli, podobne ako by to bolo pri pokuse definovať pojmy "život" alebo "láska".

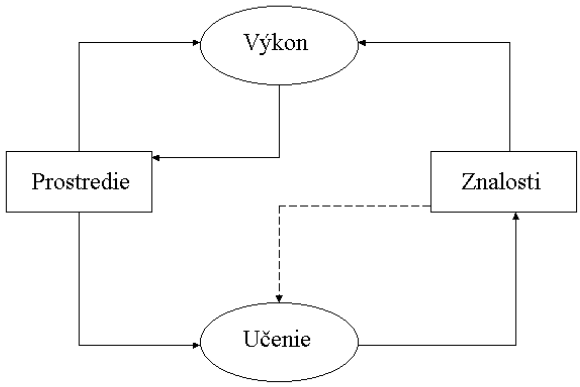
> **Učenie môže predstavovať zdokonalenie výkonu v nejakom prostredí prostredníctvom získavania znalostí zo skúseností v tomto prostredí.**

Výpočtový program by mal byť tiež schopný naučiť sa, resp. prebrať skúsenosti. Učiaci sa subjekt existuje vždy v nejakom prostredí, v rámci ktorého sa mieni učiť. Tento subjekt je spojený s nejakou bázou znalostí, v ktorej môže zhromažďovať získané znalosti, a z ktorej môže tieto znalosti čerpať. Má snahu vylepšovať chovanie niektorého výkonového elementu, pričom toto zlepšenie výkonu je dosahované nepriamo cez bázu znalostí, ktorá je používaná na riadenie interakcií s prostredím. Z toho vyplýva nasledovná definícia strojového učenia.

> *Počítačový program je považovaný za schopný učiť sa zo skúsenosti (experience) vo vzťahu k niektorej triede úloh (task) a miere výkonnosti (performance), ak sa jeho výkon zvýšil pri úlohách z danej triedy úloh, meraný mierou výkonnosti vďaka skúsenosti.*

Vzájomné interakcie medzi učením, prostredím, výkonom a znalosťami ilustruje Obr. 1. Prostredie logicky reprezentuje skúsenosť. Čiarkovaná čiara predstavuje interakciu, ktorá sa nemusí pri učení vyskytovať. Avšak učenie s touto interakciou je optimálne. Ide o prípad, keď

naučené znalosti ovplyvňujú spätne proces učenia. Učenie samozrejme vplýva na znalosti, pretože ich vytvára. Naučené znalosti môžu ovplyvniť výkon. Podobne vplývajú na výkon skúsenosti z prostredia. Žiaduce je aby zvyšovali výkon. Prostredie samozrejme vplýva na proces učenia.



Obr. 1: Interakcie medzi učením, výkonom, znalosťami a prostredím.

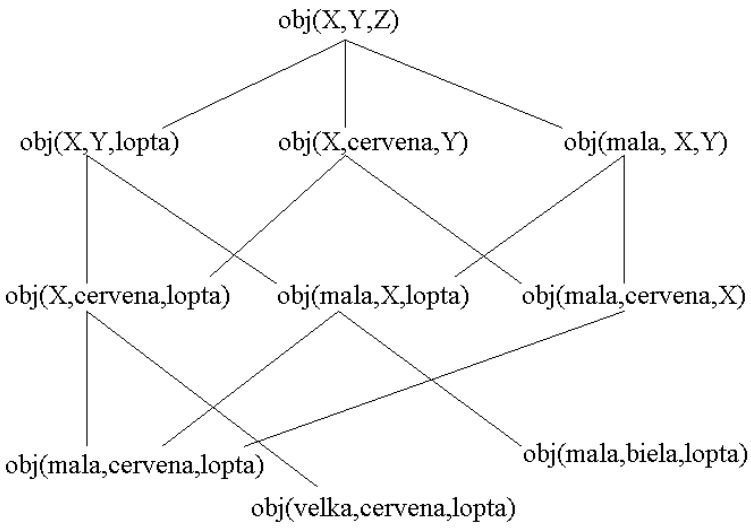
1.3 Základné pojmy

Nasledujú základné pojmy týkajúce sa učiaceho problému a učiaceho algoritmu. Učiaci problém môže byť v zásade klasifikačného a sekvenčného typu.

1.3.1 Učiaci problém

1.3.1.1 Priestor pojmov

Strojové učenie sa uskutočňuje nad priestorom pojmov. Priestor pojmov sa niekedy nazýva aj priestorom riešení (pojem predstavuje riešenie úlohy: naučiť sa pojem).



Obr. 2: Fragment usporiadaného priestoru pojmov.

Tento priestor pojmov spravidla usporiadaný napr. podľa všeobecnosti. Učenie spočíva v prehľadávaní tohto usporiadaného priestoru pojmov.

Predpokladajme, že máme skupinu objektov opísaných pomocou troch vlastností t.j. atribútov: veľkosti, farby a tvaru. Tieto objekty sú reprezentované pomocou predikátu obj(Veľkosť, Farba, Tvar). Priestor pojmov pre dané objekty usporiadaný podľa všeobecnosti je uvedený na Obr. 2.

Pojmy, ktoré sa nachádzajú v danom priestore pojmov na najnižšej úrovni všeobecnosti predstavujú jednotlivé pozorovania, na základe ktorých prebehne proces učenia. Týmto pozorovaniam budeme hovoriť typické príklady. Sú charakteristické tým, že každý atribút príkladu je daný konkrétnou hodnotou. Typické príklady sú vstupmi učiacich algoritmov. Priestor typických príkladov sa niekedy nazýva aj **problémovým priestorom**.

Všetky ostatné pojmy usporiadaného priestoru (mimo najnižšej úrovne všeobecnosti) predstavujú možné riešenie. Teda môžu predstavovať pojem, ktorý je výsledkom učenia. Tieto pojmy sa zvyknú označovať termínom **priestor riešení**. Pri extenzionálnej forme (definovanej v nasledovnom) predstavuje problémový priestor zároveň priestor riešení. Na najvyššej úrovni všeobecnosti sa nachádza takzvaná nulová hypotéza: obj(X,Y,Z), ktorá pokrýva všetky pojmy priestoru a zároveň nehovorí nič konkrétne. Pridávaním ďalších typických príkladov sa môže priestor pojmov zväčšovať nielen horizontálne ale aj vertikálne.

V súvislosti s usporiadaním priestoru príkladov podľa všeobecnosti je potrebné definovať **pojem pokrytia**. Definícia pojmu pokrytia je nasledovná:

Nech p(x) a q(x) sú definície pojmov, ktoré klasifikujú príklad x rovnakým spôsobom, napr. ako pozitívny príklad nejakej triedy. Potom p(x) pokrýva q(x) ak q(x) je logickým dôsledkom p(x).

Môžeme povedať, že ak pojem p je všeobecnejší ako pojem q potom p pokrýva q. Napríklad obj(X, červená, Y) pokrýva obj(lopta, červená, Y). Niektoré reprezentácie pojmu pracujú s tzv. parciálnym pokrytím. **Parciálne pokrytie** je taký prístup ku klasifikácii, ktorý zatrieďuje trénovacie príklady do tried aj v prípade, keď daný príklad má iba niektoré atribúty z atribútov tvoriacich popis triedy, pričom **stupeň pokrytia** určuje v koľkých atribútoch sa musí trénovací príklad zhodovať s popisom triedy.

1.3.1.2 Reprezentácia

Učiace algoritmy majú na vstupe spravidla množinu typických príkladov a na výstupe definíciu učeného pojmu. Typický príklad je reprezentovaný množinou **n** atribútov, ktoré predstavujú vlastnosti daného príkladu. Posledný **n**-tý atribút môže reprezentovať triedu, do ktorej je typický príklad zatriedený, keď takáto informácia o triede je súčasťou daného pozorovania (kontrolované učenie). V takom prípade má typický príklad **dátovú časť** a **rozhodovaciu časť**. Dátová časť predstavuje súbor **n-1** hodnôt atribútov. Rozhodovacia časť predstavuje informáciu o triede, do ktorej je príklad zaradený.

Podľa toho, aké hodnoty atribúty nadobúdajú, rozlišujeme rôzne typy atribútov:

- **binárne atribúty**, v ktorých každá hodnota môže špecifikovať buď prítomnosť alebo absenciu vlastnosti. Všeobecnejšie povedané, nadobúdajú hodnoty z dvojprvkovej množiny, napr. {ÁNO/NIE}.
- **nominálne** atribúty pripúšťajú viac ako dve hodnoty atribútu. Nadobúdajú hodnoty z konečnej neusporiadanej množiny, napr.:{zem, oheň, voda, vzduch}. Je možná transformácia nominálnej reprezentácie na binárny formalizmus, keď sa všetky hodnoty nominálneho atribútu združia do dvoch skupín, ktoré budú vystupovať ako dve hodnoty binárneho atribútu.
- **numerické** atribúty môžu nadobúdať reálne, celočíselné usporiadané hodnoty atribútov. Ak máme daných **k** atribútov, trénovací príklad môže reprezentovať bod v **k** dimenzionálnom priestore. Ak sú všetky numerické, je možné znázorniť trénovací príklad ako bod v **k** dimenzionálnom priestore, pričom jednotlivé atribúty definujú súradné osi. Niekedy sa takýto priestor nazýva aj priestorom príkladov alebo problémovým priestorom.
- **ordinárne** – nadobúdajú hodnoty z konečnej usporiadanej množiny napr. {slabá, mierna, silná}. Často uvádzaným príkladom je usporiadanie farieb podľa vlnových dĺžok.
- **hierarchické** – jednotlivé hodnoty nie sú nezávisle, sú súčasťou nejakej hierarchickej štruktúry. Jednotlivé hodnoty nemusia byť disjunktné, napr. {ovál, kruh, elipsa, mnohouholník, trojuholník, štvoruholník, štvorec, obdĺžnik, päťuholník, ...}.

Množina typických príkladov je spravidla udávaná vo forme tabuľky, kde riadky predstavujú jednotlivé pozorovania resp. typické príklady a stĺpce jednotlivé atribúty s ich hodnotami. Posledný stĺpec obsahuje triedu. Príklad takejto tabuľky sa nachádza v kapitole Rozhodovacie stromy na Obr. 15. Táto tabuľka obsahuje jeden binárny atribút **A2**-patologický prietok, nadobúdajúci hodnoty {áno, nie} a dva ordinárne atribúty s možnými hodnotami:
A1-prírastok hmotnosti ženy = {malý, v_norme, veľký}
A3-typ placenty = {nezrelý, čiast_vyzretý, vyzretý}, pričom ide o klasifikáciu do dvoch tried **T**={+(patologický_plod),-(zdravý_plod)}.

Cieľom induktívneho učenia je z takejto tabuľky indukovať popis hľadaného pojmu, ktorý bude rozhodovacou procedúrou, ktorá pre dátovú časť príkladu nájde príslušnú rozhodovaciu časť, teda triedu.

Ak sú známe všetky typické príklady, potom je rozhodovacia procedúra perfektná. To je ale v praxi veľmi zriedkavé. Ak nie sú známe všetky typické príklady, zvykne sa množina typických príkladov deliť na množinu **trénovacích príkladov** a množinu **testovacích príkladov.** Tieto množiny sa zvyknú skrátene označovať pojmami **trénovacia množina** a **testovacia množina**. Trénovacia množina slúži na naučenie pojmu. Testovacia množina sa používa na overenie stupňa presnosti klasifikácie. Pojem trénovací príklad je v strojovom učení bežnejší ako typický príklad.

Na výstupe učiaceho algoritmu je naučený pojem. Hľadáme spravidla taký popis pojmu, ktorý je konzistentný s údajmi. **Konzistentný** je taký popis pojmu, ktorý jednoznačne diskriminuje pozitívne príklady pojmu od negatívnych. Na druhej strane **kompletný** je taký popis pojmu, ktorý pokrýva všetky pozitívne príklady a to aj v prípade, že spolu s nimi pokryje aj niektoré negatívne príklady (Kubát, 1993).

Pojem môže byť reprezentovaný extenzionálne alebo intenzionálne. **Extenzionálna** definícia pojmu predstavuje naivný prístup k reprezentácii pojmu, ktorý jednoducho pokladá všetky pozorované trénovacie príklady za pojmy. Teda extenzionálna definícia pojmu je reprezentovaná vymenovaním tých príkladov, ktoré ho reprezentujú. Takáto definícia iba sumarizuje predchádzajúce skúsenosti. Nie je schopná vykonávať predikciu zatriedenia neznámych trénovacích príkladov, ktoré raz budú reprezentanti pojmu. **Intenzionálna** definícia je kompaktnejšia a všeobecnejšia, pretože sa získava generalizáciou množiny trénovacích príkladov. Je schopná predikovať zatriedenie nových trénovacích príkladov.

Majme **interpreter**, ktorého úlohou je rozhodnúť, ktorý z pojmov pokrýva daný príklad. Interpreter je potrebný na to, aby interpretoval výsledok učenia reprezentovaný intenzionálne. Nájdená intenzionálna definícia pojmu môže byť použitá na klasifikáciu nových príkladov a to tak, že sa definícia pojmu porovná s definíciou príkladu. Ak príklad vyhovuje všetkým podmienkam v definícii pojmu, je označený za pozitívny príklad daného pojmu. Inak je označený za negatívny príklad pojmu.

Extenzionálna reprezentácia sa dá priamo použiť. Extenzionálna definícia pojmu (t.j. typické príklady v pamäti) sa jednoducho porovná s definíciou nového príkladu. Intenzionálna definícia sa nedá takto priamo použiť. Inými slovami interpreter môže byť použitý na transformáciu intenzionálnej definície na extenzionálnu definíciu jednoducho tým, že je aplikovaný na všetky možné príklady. Teda:

Intenzionálna definícia + Interpreter = Extenzionálna definícia

Naučené pojmy môžu byť rôzne interpretované. Každá interpretácia má vlastnú reprezentáciu pojmu. Najčastejšie sa vyskytujú nasledovné interpretácie pojmov:

- Logická interpretácia. Pojmy sú reprezentované logickými kombináciami hodnôt atribútov. Proces pokrývania, obstarávaný interpretom, je úplný. Musia platiť všetky podmienky, alebo žiadna.
- Prahová interpretácia. Interpreter obstaráva čiastočné pokrývanie, v ktorom iba niektoré aspekty intenzionálneho popisu musia byť pokryté. To vyžaduje špecifikáciu nejakej prahovej hodnoty, ktorá určuje akceptovateľný stupeň zhody.
- Súťažná interpretácia. Interpreter zabezpečuje čiastočné pokrývanie. Namiesto určenia prahovej hodnoty, vypočíta stupeň pokrývania pre jednotlivé alternatívne súťažiace pojmy a vyberá najlepšieho kandidáta.

1.3.1.3 Definícia klasifikačnej úlohy

Nech **M** je množina všetkých trénovacích príkladov, **C** je množina cieľových pojmov a **H** je množina hypotéz, ktoré aproximujú cieľové pojmy z **C**. Ak je nejaký cieľový pojem **c** z množiny **C** aproximovaný svojou hypotézou h z **H**, potom **D** je distribúcia pravdepodobnosti, že h bude chybne klasifikovať náhodne vybratý príklad (nemusí patriť do trénovacej množiny **M**). Túto chybu klasifikácie je možné definovať:

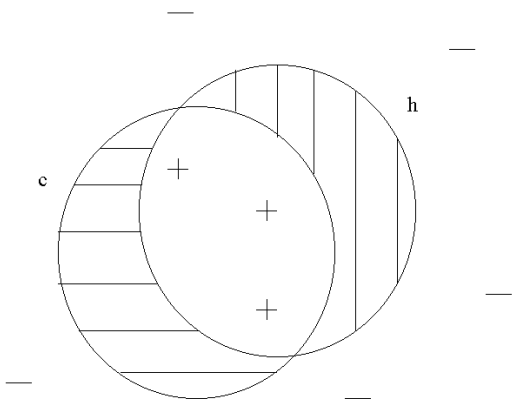
$$chyba_D(h) = P_{x \in D}[c(x) \neq h(x)] .$$

Obr. 3 ilustruje definíciu chyby v grafickej forme. Chyba hypotézy **h** je pravdepodobnosť, že ľubovoľný trénovací príklad nebude patriť do oblasti, kde sa **c** a **h** prekrývajú. Sú možné dve variácie tohto prípadu:

Príklad **x** je pozitívnym príkladom pojmu **c**, ale nie je súčasťou trénovacej množiny ***M***, preto nemôže byť pokrytý hypotézou pojmu **h**, ktorá bola naučená z množiny **M**. Teda príklad je pokrytý **c** a nie je pokrytý **h** (vertikálne šrafovaná oblasť v Obr. 3).

Príklad **x** nie je pozitívnym príkladom pojmu **c**, ale hypotézou **h** daného pojmu je chybne klasifikovaný ako pozitívny príklad. Príklad nie je pokrytý **c** ale je pokrytý **h** (horizontálne šrafovaná oblasť v Obr. 3).

Chyba hypotézy pojmu **h** s ohľadom na **c** nie je priamo pozorovateľná učiacim sa subjektom (učiacim algoritmom). Ten môže pozorovať iba prejavy **h** nad trénovacími príkladmi a na tomto základe vybrať konečnú výstupnú hypotézu pojmu. V tejto súvislosti budeme používať pojem trénovacia chyba, ktorý bude vzťahovaný k tej časti trénovacích príkladov, ktoré boli chybne klasifikované **h** (horizontálne šrafovaná oblasť v Obr. 3).



Obr. 3: Definícia chyby hypotézy pojmu v grafickej podobe.

1.3.1.4 Komplexnosť učenia

Posledný činiteľ, ktorý ovplyvňuje učenie sú aspekty okolia. Najbežnejší aspekt okolia je komplexnosť cieľových znalostí, ktoré majú byť získané. Táto komplexnosť môže byť závislá od:

- komplexnosti popisu pojmu. Napr. pojem, ktorý obsahuje veľa atribútov alebo podmienok, je ťažší na naučenie ako ten, ktorý obsahuje menej atribútov, resp. podmienok. Tento aspekt zohľadňuje generovanie rozhodovacieho stromu pomocou minimálnej dĺžky popisu viď (Quinlan-Rivest, 1989).
- štruktúry problémového priestoru. Napr., ak pojem obsahuje veľa irelevantných atribútov, učiaci systém môže mať ťažkosti s ich odlíšením od relevantných atribútov.
- výskytu šumu. V kontrolovanom učení existujú dva druhy šumu. **Klasifikačný šum**, t.j. šum pri určovaní klasifikačnej triedy. Predstavuje chybu spätnej väzby. Druhý typ je tzv. **Atribútový šum**, ktorý zahŕňa chybu v popise atribútov trénovacieho príkladu. Väčšinou ide o to, že pri konkrétnych trénovacích príkladoch chýbajú niektoré hodnoty atribútov. Chýbajúce hodnoty atribútov môžu byť nahradené (napr. najčastejšie sa vyskytujúcou hodnotou v rámci daného atribútu), alebo môžu byť vylúčené trénovacie príklady s touto chybou z trénovacej množiny. Väčšie množstvo šumu má tendenciu robiť učenie zložitejším. V zašumenej doméne, dokonca aj jedna neoznačená trieda alebo vlastnosť môže seriózne prekaziť niektorým algoritmom nájdenie konzistentného popisu. Iné algoritmy, ktoré nevyžadujú striktné oddelenie negatívnych a pozitívnych príkladov môžu uspieť. Čo sa objaví ako šum v jednom učiacom systéme, v inom učiacom systéme s prídavnými atribútmi sa môže prejavovať ako nezašumené.

1.3.2 Učiaci algoritmus

1.3.2.1 Delenie strojového učenia

Strojové učenie v zásade delíme na induktívne a deduktívne. **Induktívne učenie** z informácií na najnižšej úrovni všeobecnosti (pozorovaných prípadov alebo trénovacích príkladov) indukuje, resp. generuje všeobecnejšiu znalosť. **Deduktívne učenie** naopak zo znalosti na vysokej úrovni všeobecnosti dedukuje znalosť menej všeobecnú a zložitú, ktorá je lepšie prispôsobená na riešenie konkrétneho druhu problémov.

Ďalšia dimenzia, ktorá vplýva na učenie je stupeň kontroly. Podľa stupňa kontroly rozoznávame kontrolované a nekontrolované učenie. Oba typy, tak kontrolované ako nekontrolované učenie sa používajú na riešenie klasifikačných úloh aj sekvenčných úloh. Pri klasifikácii ide o rozhodnutie o triede. Príklad, ktorý je definovaný hodnotami atribútov, resp. vlastností klasifikujeme, teda zaradíme do triedy. Pri sekvenčných úlohách je potrebné určiť postupnosť krokov, ktorá povedie k želanému riešeniu. **Kontrolované učenie** disponuje spätnou väzbou o úspešnosti učenia. To znamená, že pri klasifikácii sa predpokladá, že pre každý trénovací príklad je vopred určená trieda daného príkladu a cieľom je indukovať popis, resp definíciu pojmu, ktorý presne predikuje túto triedu pre každý nový príklad (bez triedy). Indukovaný popis môže mať formu: logickej konjunkcie, prahového pojmu, etalónu, rozhodovacieho stromu alebo rozhodovacieho zoznamu. Pri sekvenčných úlohách tútor, alebo expert navrhne korektný krok v každom stave v prehľadávacom, alebo vysvetľovacom procese. Systémy, ktoré operujú s takouto spätnou väzbou sa niekedy nazývajú "učiaci sa učni" (learning apprentices).

Na druhej strane pri **nekontrolovanom učení** neexistuje spätná väzba o vhodnosti výkonu. Teda klasifikačné úlohy nemajú vopred špecifikovanú triedu. Preto pri nekontrolovanom učení sa príklady iba zhlukujú do zhlukov podľa nejakého kritéria, najčastejšie podobnosti (zhlukovanie). Pri sekvenčných úlohách učiaci sa subjekt musí odlíšiť želaný alebo vhodný krok od neželaného. Napríklad keď hrá šachy a nemá učiteľa, ktorý by hneď spätne okomentoval každý jeho krok z hľadiska vhodnosti. Vzniká podúloha a to určenie kreditu, ktorý by pomohol učiacemu sa subjektu rozlíšiť kroky zodpovedné za úspech alebo neúspech (učenie odmenou a trestom).

Na základe spôsobu, akým sa získavajú trénovacie príklady, delíme učiace úlohy na úlohy typu online a offline. Úlohy typu **online** získavajú trénovacie príklady jednotlivo a postupne (zodpovedajú inkrementálnemu učeniu). Úlohy, ktoré rieši človek majú väčšinou online povahu, pretože ľudia existujú v časovom svete.

Úlohy typu **offline** disponujú všetkými trénovacími príkladmi súčasne (zodpovedajú neinkrementálnemu učeniu). Aj u človeka sa môže výnimočne vyskytnúť takýto prípad, keď je konfrontovaný s masou zhromaždených znalostí, napr. keď študent prechádza do novej oblasti. Kompromisom medzi úlohami typu online a offline je učiaca úloha, ktorá získa množinu trénovacích príkladov raz za čas.

Algoritmy, ktoré sa používajú na riešenie učiacich úloh (typu online aj offline) delíme obdobným spôsobom na inkrementálne a neinkrementálne. **Inkrementálne učenie** je učenie

pomocou algoritmu, ktorý spracováva jeden trénovací príklad za druhým. Po každom príklade poskytuje algoritmus riešenie. Inkrementálne učenie zodpovedá online učiacim úlohám. **Neinkrementálne učenie** je učenie pomocou algoritmu, ktorý spracováva celú množinu trénovacích príkladov odrazu. Zodpovedá offline učiacim úlohám.

Avšak je možné aplikovať neinkrementálne učenie aj na online úlohy, a to tak, že sa predchádzajúce trénovacie príklady zdržia v pamäti a každý nový príklad sa pridá k tejto množine. Algoritmus sa potom spustí nad celou trénovacou množinou, pričom zabudne čo sa naučil na predchádzajúcej (menšej) množine príkladov. Podobne inkrementálne metódy je možné použiť aj na riešenie offline úloh prostredníctvom iteratívneho behu cez tú istú trénovaciu množinu. Pritom sa v každej iterácii pridáva jeden príklad z tejto množiny. V poslednej iterácii sa spracuje celá množina.

Hovoríme, že algoritmus je inkrementálny v stupni **K**, ak spracováva online údaje nasledujúcim spôsobom: po narazení na nový trénovací príklad spracuje vždy **K** predchádzajúcich príkladov. Taktiež hovoríme, že algoritmus je neinkrementálny, ak spracuje všetky skoršie trénovacie príklady.

1.3.2.2 Preferencie systému

Učiaci systém musí nejakým spôsobom limitovať svoje prehľadávanie priestoru možných znalostných štruktúr. Tým sa vytvárajú tzv. **preferencie systému** (bias). Jedna dôležitá forma preferencie známa ako **reprezentačné preferencie** (represent bias) obmedzuje priestor možných štruktúr popisu pojmu. Napríklad, keď pracujeme s numerickými atribútmi a výsledný popis pojmu očakávame v tvare logických konjunkcií, výsledný popis pojmu obsahuje podmienky v tvare: **A1 < x1**, alebo **A2 > x2**, kde **A1** a **A2** sú numerické atribúty a **x1, x2** sú reálne čísla. Takéto podmienky vytvárajú prirodzene v priestore príkladov obdĺžnikové oblasti odpovedajúce popisom pojmu. Teda vytvárajú reprezentačné preferencie hyperobdĺžnikov.

Flexibilnejší prístup zahŕňa **prehľadávacie preferencie** (search bias), ktorý berie do úvahy možné popisy pojmov, ale skúša niektoré skôr a iné neskôr v rámci procesu prehľadávania. Napríklad, algoritmus pre indukciu logických konjunkcií môže preferovať jednoduchšie popisy pred komplexnejšími a tak vytvárať **induktívne preferencie** (inductive bias). Tieto preferencie spočívajú v tom, že sa uvažujú najprv popisy pojmov s menším počtom testov. Pojem induktívnych preferencií zaviedol (Mitchell, 1980), zatiaľ čo (Rendell, 1986) stanovil rozdiel medzi reprezentačným a prehľadávacím biasom. V niektorých prípadoch tieto preferencie sú vyjadrené v explicitnej ohodnocovacej metrike, zatiaľ čo v iných prípadoch sú vyjadrené v štruktúre samotného algoritmu prehľadávania. Ďalším zdrojom preferencií sú znalosti okolia prístupné učiacemu systému.

1.4 Úlohy na precvičenie:

1. Ako delíme učiace úlohy z hľadiska spôsobu prijímania vstupných údajov a ako z hľadiska toho, načo sa výsledná reprezentácia používa?
2. Ako delíme učiace algoritmy?
3. Aké druhy preferencií (bias) systému poznáte a ako by ste ich charakterizovali?
4. Zotrieďte priestor pojmov – grafov podľa všeobecnosti. Grafy môžu obsahovať 1 až 5 uzlov. Grafy sú spojité. Všeobecnejšie sú grafy, ktoré obsahujú menej hrán. Špecifikácia o jednu podmienku znamená pridanie jednej hrany.

1.5 Literatúra:

Carbonell, J.G., Michalski, R.S., Mitchell, T.M.: An overview of machine learning. In R.S.Michalski, J.G.Carbonell, T.M.Mitchell (Eds.), Machine learning: An artificial intelligence approach. San Francisco, CA: Morgan Kaufmann, 1983.

Dietterich, T.G., London, B., Clarkson, K., Dromey, G.: Learning and inductive inference. In P.R. Cohen, E.A. Feigenbaum (Eds.), The handbook of artificial intelligence, Vol. 3. San Francisco, CA: Morgan Kaufmann, 1982.

Kubát, M.: Strojové učení. In: Mařík, V., Štepánková, O., Lažanský, J.: Umělá inteligence (1). Academia Praha, 1993, 264 s.

Langley, P., Carbonell, J.G.: Approaches to machine learning, Journal of the American Society for Information Science, 35, 1984, pp. 306-316.

Kubát, M.: Strojové učení. In. Mařík, V., Štepánková, O., Lažanský, J.: Umělá inteligence. ACADEMIA PRAHA, Praha, 1993, ISBN 80-200.0502-3.

Mitchell, T.M.: The need for biases in learning generalizations (Technical Report No. CBM-TR-117). New Brunswick, NJ: Rutgers University, Department of Computer Science, 1980. Reprinted in J.W. Shavlik, T.G. Dietterich (Eds.), Readings in machine learning. San Francisco, CA: Morgan Kaufmann, 1990.

Quinlan, J.R., Rivest, R.L.: Inferring Decision Trees Using the Minimum Description Length Principle. Information and Computation, Vol. 80, No. 3, 1989, pp. 227-248.

Rendell, L.A.: A general framework for induction and a study of selective induction. Machine Learning, 1,1986, pp. 177-226.

Časť I

Učenie logickej reprezentácie s učiteľom

2 GENEROVANIE LOGICKÝCH KONJUNKCIÍ

2.1 Prehľadávanie priestoru kandidátov pojmov

Najjednoduchší spôsob ako reprezentovať pojem je v tvare konjunkcie podmienok predstavujúcich vlastnosti typické pre daný pojem. Výskumy na poli indukcie logických konjunkcií zaujímali centrálnu pozíciu v strojovom učení. Základné myšlienky tohto prístupu boli prezentované v (Bruner at al., 1957), kde je taktiež reprezentovaná štúdia problematiky učenia pojmov u človeka. Prvé pokusy o automatizáciu učenia pojmov v tvare logických konjunkcií boli prezentované v (Hunt-Hovland, 1963). (Mitchell, 1977) zavádza termín priestoru pojmov. V rámci tejto kapitoly bude pojem reprezentovaný konjunkciou atribútov. Je potrebné si uvedomiť, že v priestore pojmov sa nenachádzajú všetky možné pojmy, ale iba tie, ktoré majú tvar logickej konjunkcie podmienok. Uplatňujú sa tu reprezentačné preferencie. Najšpecifickejší pojem je popísaný konjunkciou konkrétnych hodnôt atribútov a predstavuje trénovací príklad. Najvšeobecnejší pojem je popísaný konjunkciou premenných predstavujúcich jednotlivé atribúty, ktoré nie sú dané konkrétnymi hodnotami.

Prehľadávanie priestoru pojmov (Version Space Search) predstavuje implementáciu induktívneho učenia nad týmto priestorom. Tomuto prehľadávaniu predchádza usporiadanie priestoru pojmov podľa všeobecnosti. Preto hovoríme o **riadenom prehľadávaní**, pričom na riadenie prehľadávania je používané práve spomínané usporiadanie podľa všeobecnosti.

Usporiadanie pojmov v priestore sa uskutočňuje pomocou **operátorov zovšeobecnenia**, resp. operátorov **špecifikácie**. Často citovaná dizertačná práca (Winston, 1970) uvádza prehľadávanie použitím operátorov zovšeobecnenia a špecifikácie. Zovšeobecnenie a špecifikácia sú najbežnejšie typy operácií na pohyb v priestore pojmov. Primárne operácie zovšeobecnenia používané v strojovom učení sú:

1. nahradenie konštanty premennou

$$farba(lopta, cervena) \Rightarrow farba(lopta, X)$$

2. vypustenie podmienky z konjunktívneho výrazu

$$t\,\text{var}(X, lopta)\,\&\,velkost(X, mala)\,\&\,farba(X, biela) \Rightarrow t\,\text{var}(X, lopta)\,\&\,farba(X, biela)$$

3. pridanie disjunkcie do výrazu

$$t\,\text{var}(X, lopta)\,\&\,farba(X, biela) \Rightarrow t\,\text{var}(X, lopta)\,\&\,farba(X, biela \vee cervena)$$

4. nahradenie vlastnosti jej rodičom v hierarchii tried

$$farba(X, cervena) \Rightarrow farba(X, primarna\_farba)$$

Predpokladá sa, že je definovaná množina primárnych farieb, do ktorej nepatria všetky farby a do ktorej okrem iných farieb patrí červená. Atribút „farba“ je hierarchickým atribútom.

Majme množinu trénovacích príkladov daných predikátom obj(Velkost, Farba, Tvar), pričom jednotlivé atribúty môžu nadobúdať nasledovné hodnoty:

Veľkosť = {veľká, malá}
Farba = {červená, biela, modrá}
Tvar = {lopta, tehla, kocka}.

Operácie zovšeobecnenia postupne nahrádzajú konštanty premennými a tým usporadúvajú priestor pojmov podľa všeobecnosti tak, ako je to uvedené na Obr. 2.

Induktívne učenie môžeme chápať ako prehľadávanie tohto priestoru pojmov, ktoré majú rôzne stupne všeobecnosti a sú v tomto priestore podľa všeobecnosti usporiadané. V (Simon-Lea, 1973) sa nachádzajú prvé zmienky o učení ako prehľadávaní priestoru pojmov. Ďalšie z prvých systémov založených na učení pojmov sú prezentované v (Popplestone, 1969), (Michalski, 1973), (Buchanan, 1974), (Vere, 1975) a (Hayes-Roth, 1974).

Priestor pojmov obsahuje všetky trénovacie príklady a to na najnižšej úrovni všeobecnosti. Na najvyššej úrovni všeobecnosti sa nachádza takzvaná nulová hypotéza obj(X,Y,Z), ktorá pokrýva všetky pojmy priestoru a zároveň nehovorí nič konkrétne.

2.2 Reprezentácia a použitie logických konjunkcií

Algoritmy tejto kapitoly sú navrhované tak, aby hľadali a našli konjunktívny popis pojmu, ktorý je konzistentný s trénovacími príkladmi. Tento pojem je reprezentovaný logickou konjunkciou hodnôt atribútov.

Nájdená definícia pojmu môže byť použitá na klasifikáciu nových príkladov a to tak, že sa definícia pojmu porovná s definíciou príkladu. Ak príklad vyhovuje všetkým podmienkam v definícii pojmu, je označený za pozitívny príklad daného pojmu. Inak je označený za negatívny príklad pojmu. Túto klasifikáciu uskutočňuje logický interpreter implicitnej reprezentácie. V tomto prípade má implicitná reprezentácia tvar logickej konjunkcie. Ide o striktnú klasifikáciu – všetky podmienky musia byť splnené.

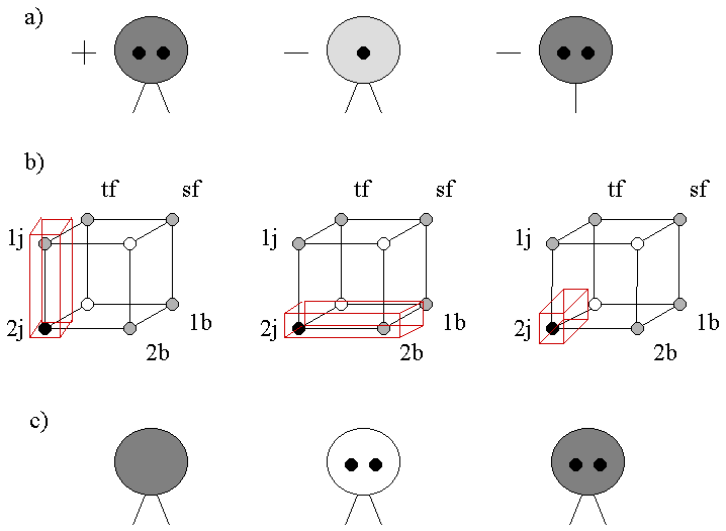
Tento prístup buduje rozhodovacie hranice v priestore trénovacích príkladov. Tieto hranice ohraničujú oblasti, ktoré znázorňujú definície hľadaných pojmov a pokrývajú pozitívne príklady daného pojmu. Naviac platí, že pre každý pár pozitívnych príkladov existuje cesta – spojnica medzi nimi, ktorá obsahuje iba pozitívne príklady. Taká oblasť sa nazýva **konvexná**. Tieto oblasti majú tvar hyper-obdĺžnikov s povrchmi kolmými na osi, reprezentujúce relevantné atribúty a s povrchmi paralelnými s osami irelevantných atribútov. Preto sa o metódach indukujúcich logické konjunkcie hovorí, že používajú reprezentačné preferencie pravouhlého obdĺžnika.

Uvažujme nominálnu doménu, ktorá zahŕňa tri atribúty (s hodnotami), opisujúce bunku:

Počet_bičíkov = {jeden, dva}
Farba_tela_bunky = {tmavá, svetlá}
Počet_jadier = {jeden, dva}.

Obr. 4 ilustruje indukciu pojmu chorá ľudská bunka. V časti a) sú znázornené trénovacie príklady: jeden pozitívny a dva negatívne. V časti b) sú znázornené tri osovo paralelné oblasti odpovedajúce trom logickým konjunkciám. Časť c) predstavuje tri logické konjunkcie

konzistentné s trénovacími príkladmi, teda tri možné riešenia – popisy hľadaného pojmu „chorá bunka“.



Obr. 4: Príklad nominálnej domény – bunky opísané tromi atribútmi (tf...tmavá farba tela bunky, sf...svetlá farba tela bunky, 1j...jedno jadro, 2j...dve jadrá, 1b...jeden bičík, 2b...dva bičíky).

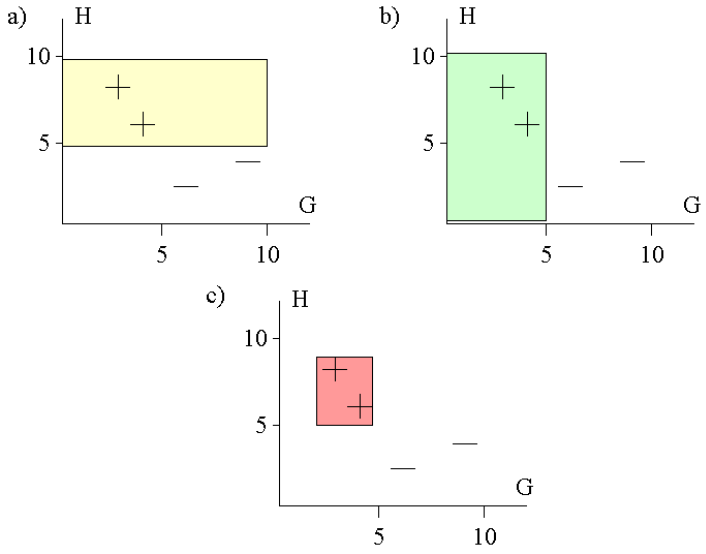
Pretože každý atribút v danej doméne má dve možné hodnoty, je možné priestor trénovacích príkladov znázorniť kockou (8 možných kombinácií hodnôt atribútov sa zhoduje s počtom rohov kocky). Čierny bod predstavuje pozitívny príklad pojmu, biele body sú negatívne príklady pojmu a šedé body znázorňujú neznáme trénovacie príklady, ktoré zatiaľ nepatria do trénovacej množiny. Uvedený Obr. 4 ukazuje ako je možné vytvárať predikcie nad neznámymi príkladmi (šedé body).

Teraz uvažujme numerickú doménu s dvoma numerickými atribútmi **H** a **G**. Logické konjunkcie budú v tejto doméne obsahovať podmienky v tvare:

$H > x_1$, $H < x_2$, $G > x_3$, $G < x_4$, kde x_i je celé číslo.

Logická konjunkcia v tejto doméne bude obsahovať minimálne jednu a maximálne štyri podmienky, teda pre každý atribút 0, 1 alebo 2 podmienky.

Obr. 5 obsahuje dva pozitívne a dva negatívne trénovacie príklady. Podobne ako v nominálnej doméne majú rozhodovacie hranice formu obdĺžnikov, ktoré majú jednu stranu paralelnú s niektorou súradnou osou v priestore príkladov. Takáto súradná os reprezentuje „irelevantný atribút“ t.j. atribút, ktorý z hľadiska klasifikácie do tried môže byť v danom príklade považovaný za irelevantný.



Obr. 5: Logické konjunkcie konzistentné s trénovacími údajmi v numerickej doméne.

Dve najvšeobecnejšie možné logické konjunkcie (prvá charakterizovaná podmienkou **H> 5**, druhá **G<5**), sa nachádzajú v častiach a) a b) obrázku

Obr. 5. Oblasť znázornená v časti a) spomínaného obrázku má jednu hranicu danú podmienkou **H>5.** Ostatné hranice tejto oblasti sú dané hranicami domény, keďže platí **0<H<10** a **0<G<10**. To isté platí pre oblasť v časti b), kde jediná hranica, ktorá nie je daná obmedzeniami domény je daná podmienkou **G<5**. Najšpecifickejšia logická konjunkcia je zobrazená v c) časti obrázku. Je definovaná štyrmi podmienkami: **H>5&H<9&G>2&G<5**), ktoré sú premietnuté do štyroch hraníc jej odpovedajúcej oblasti.

2.3 Úloha indukcie logických konjunkcií

Úlohu indukcie logických konjunkcií môžeme formulovať nasledovne.

Dané: množina pozitívnych trénovacích príkladov pre triedu T
množina negatívnych trénovacích príkladov pre triedu T

Hľadané:logická konjunkcia, ktorá v najväčšom možnom rozsahu korektne klasifikuje nové testovacie príklady.

Táto úloha predpokladá aktuálny popis pojmu, alebo aspoň aproximáciu tohto pojmu použitím konjunkcie jednoduchých podmienok. Presnosť aproximácie závisí od množstva

trénovacích príkladov. Cieľom nie je nevyhnutne nájsť konjunkciu, ktorá perfektne klasifikuje trénovacie príklady, ale indukovať popis pojmu, ktorý prijateľne klasifikuje nové príklady.

Tieto metódy majú ťažkosti v doménach obsahujúcich šum, keďže uprednostňujú pravouhlé tvary (reprezentačné preferencie) a podmienky v konjunkcii musia striktne platiť alebo neplatiť.

Doteraz sa hovorilo iba o indukcii popisu jedného pojmu. Vyššie uvedenú schému je možné použiť aj na multitriednu indukciu. Multitriedna indukcia indukuje popisy pre množinu **N** pojmov, ktoré majú slúžiť na klasifikáciu do **N** tried. Vždy jeden pojem pre jednu triedu. Pre každú triedu $\mathbf{T_i}$ sa považujú všetky trénovacie príklady ostatných **N-1** tried za negatívne príklady $\mathbf{T_i}$. Z toho vyplýva, že základnú indukciu je potrebné zopakovať **N**-krát.

2.3.1 Čiastočné usporiadanie tried a pojmov

Ako už bolo uvedené, priestor pojmov môže byť usporiadaný podľa všeobecnosti. Na najnižšej úrovni všeobecnosti sú v tomto priestore trénovacie príklady a na najvyššej úrovni všeobecnosti je nulová hypotéza. Medzitým sa nachádzajú pojmy na rôznej úrovni všeobecnosti. Niektoré z nich sú hľadané pojmy, ktoré slúžia na klasifikáciu do príslušných tried. Ak si množinu týchto tried nazveme priestorom tried, potom tento priestor tried bude podpriestorom priestoru pojmov.

Aj priestor tried môže byť čiastočne usporiadaný podľa všeobecnosti. Ak trieda **A** zahŕňa všetky príklady triedy **B** a taktiež ďalšie príklady, potom **A** je striktne všeobecnejšia ako **B**. Tento čiastočne usporiadaný priestor je v jednom extréme ohraničený najvšeobecnejšou triedou , ktorá obsahuje všetky príklady a v druhom extréme je ohraničený množinou najšpecifickejších tried, ktoré obsahujú jeden príklad. Takáto najšpecifickejšia trieda zodpovedá extenzionálnej reprezentácii pojmu.

Toto parciálne usporiadanie hrá dôležitú úlohu vo väčšine metód, indukujúcich logický popis pojmu, pretože existuje priamy vzťah medzi relatívnou všeobecnosťou dvoch tried a ich popismi. Všeobecnejší popis pojmu obsahuje menej podmienok (je kratší), ako špecifickejší pojem. To neplatí pri numerických atribútoch, kde sa miera všeobecnosti môže meniť aj zmenou hraničných hodnôt v podmienkach, nielen vypustením (vložením) podmienky. Vypúšťanie nominálnych podmienok z popisu pojmu zovšeobecňuje pojem. Na druhej strane, pridávanie nominálnych podmienok vytvára špecifickejší pojem.

2.4 VSS algoritmy (Version Space Search)

V ďalšom budú prezentované tri algoritmy na prehľadávanie priestoru pojmov. Tieto algoritmy pracujú tak, že redukujú veľkosť priestoru pojmov v závislosti od toho, ako prichádzajú nové príklady. Prvý algoritmus redukuje priestor pojmov, resp. možných hypotéz, v smere od špecifického k všeobecnému. Druhý algoritmus pracuje v smere od všeobecného

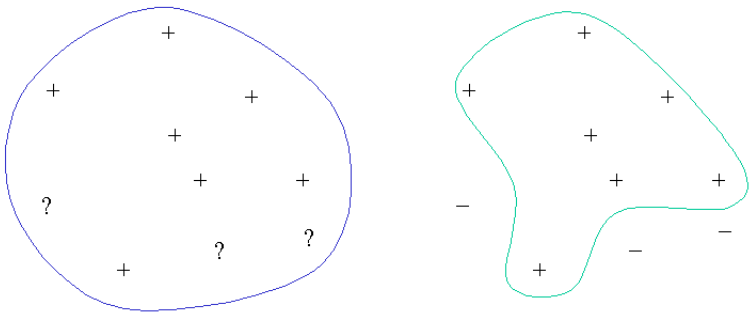
ku špecifickému. Tretí algoritmus nazývaný "eliminácia kandidátov" kombinuje tieto dva prístupy do obojsmerného prehľadávania. Podrobnejší popis týchto troch algoritmov je možné nájsť v (Mitchell, 1982) a (Mitchell, 1997).

Tieto algoritmy sú riadené údajmi t.j. trénovacími príkladmi. Postupné zovšeobecňovanie hľadaného popisu pojmu sa zakladá na pravidelnostiach nachádzajúcich sa medzi trénovacími príkladmi v trénovacej množine. Ide o prípad, keď sa v trénovacej množine s určitou pravidelnosťou vyskytujú tie isté skupiny údajov. Napríklad vlastnosť vysoký sa väčšinou vyskytuje spolu s vlastnosťou chudý, resp. blonďák. Taktiež pri používaní trénovacích príkladov na klasifikáciu, uskutočňujú uvedené algoritmy rôzne variácie riadeného učenia.

V súvislosti s danou tematikou je potrebné objasniť pojem **prílišného zovšeobecnenia**. Prehľadávanie priestoru pojmov používa nielen pozitívne, ale aj negatívne príklady. Aj keď je možné zovšeobecňovať iba z pozitívnych príkladov, negatívne príklady sú dôležité pri prevencii možného prílišného zovšeobecnenia. Učený pojem musí byť nielen dostatočne všeobecný aby pokryl všetky pozitívne príklady, ale musí byť aj dostatočne špecifický, aby vylúčil všetky negatívne príklady. Prílišne zovšeobecnený pojem by mohol obsahovať aj niektoré negatívne trénovacie príklady. Napríklad obj(X,Y,Z) na Obr. 2 je príliš všeobecný pojem. Zahŕňa úplne všetko, čo sa dá definovať troma atribútmi: veľkosťou, farbou a tvarom. Existujú dva spôsoby, ako sa vyhnúť prílišnému zovšeobecneniu:

- Zovšeobecňovať iba toľko, koľko je nutné na pokrytie všetkých pozitívnych príkladov.
- Používať negatívne príklady na elimináciu príliš všeobecných pojmov.

Algoritmy tejto skupiny používajú obidve uvádzané techniky. Obr. 6 ilustruje, ako negatívne príklady predchádzajú prílišnému zovšeobecneniu pojmu. Negatívne príklady tlačia proces učenia do špecifikácie pojmu prostredníctvom vylúčenia negatívnych príkladov.



Pojem indukovaný z pozitívnych príkladov.

Pojem indukovaný z pozitívnych a negatívnych príkladov.

Obr. 6: Úloha negatívnych príkladov v predchádzaní prílišnému zovšeobecneniu.

2.4.1 Prehľadávanie od špecifického k všeobecnému

Tento algoritmus uskutočňuje zovšeobecňovanie. Pracuje inkrementálne. Buduje množinu **S** kandidátov pojmov, ktorých definíciu (popis) hľadáme. Názov **S** je od "specific", pretože obsahuje najšpecifickejšie pojmy, konzistentné so vstupnými príkladmi, ktoré sa postupne zovšeobecňujú použitím operátorov zovšeobecnenia. Aby sme sa vyhli prílišnému zovšeobecneniu, definície pojmov sú zovšeobecnené najšpecifickejšie.

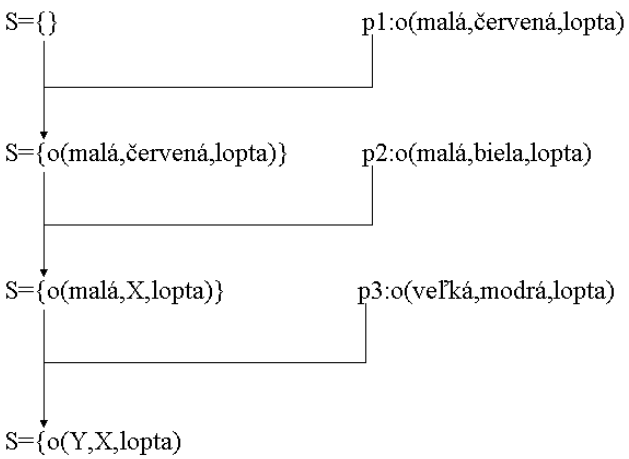
> **Pojem P je najšpecifickejšie zovšeobecnený, ak pokrýva všetky pozitívne príklady, žiadny negatívny príklad, a pre každý ďalší pojem P', tiež pokrývajúci pozitívne príklady platí: P ≤ P'.**

Pričom symbol ≤ v tomto prípade znamená špecifickejší alebo rovnako špecifický. Symbol ≥ znamená všeobecnejší alebo rovnako všeobecný.

Vstupy: ***M****...množina trénovacích príkladov*
N*.....množina negatívnych trénovacích príkladov dosiaľ nájdených*
Výstup: ***S****.....množina kandidátov hľadaných pojmov*

```
begin
N={}
S=prvý pozitívny trénovací príklad
for     každý pozitívny príklad p
        begin
        for     každé s z S
                if      s nepokrýva p
                then    nahraď s jeho najšpecifickejším zovšeobecnením, ktoré
                        pokrýva p
                        vymaž z S hypotézy všeobecnejšie ako iné hypotézy v S
                        vymaž z S hypotézy pokrývajúce negatívne príklady v N
end
for     každý negatívny príklad n
        begin
        vymaž členov S pokrývajúcich n
        pridaj n do N pre kontrolu prílišného zovšeobecnenia budúcich hypotéz
        end
end.
```

Obr. 7 ilustruje použitie algoritmu prehľadávania od špecifického k všeobecnému na učenie pojmu "lopta" z troch trénovacích príkladov (o predstavuje objekt - náhrada obj).



Obr. 7: Učenie pojmu „lopta“ prehľadávaním priestoru pojmov od špecifického k všeobecnému.

2.4.2 Prehľadávanie od všeobecného k špecifickému

Tento algoritmus buduje množinu **G** najvšeobecnejších pojmov, ktoré sú konzistentné s trénovacími príkladmi. Pojmy tejto množiny sa postupne špecifikujú, pričom je snaha vytvárať najvšeobecnejšie špecifikácie. Pojem P je najvšeobecnejšou špecifikáciou, ak nepokrýva žiaden z negatívnych trénovacích príkladov, a pre každý ďalší pojem P', ktorý taktiež nepokrýva žiadny negatívny trénovací príklad, platí: $P \geq P'$.

V tomto algoritme vedú negatívne trénovacie príklady ku špecifikácii pojmov. Pozitívne trénovacie príklady používa algoritmus na elimináciu prílišnej špecifikácie.

Obr. 8 je príkladom aplikácie predmetného algoritmu nad priestorom pojmov z Obr. 2. V tomto prípade algoritmus používa znalosti okolia (background knowledge) v tvare definície hodnôt atribútov: Veľkosť = {veľká, malá}

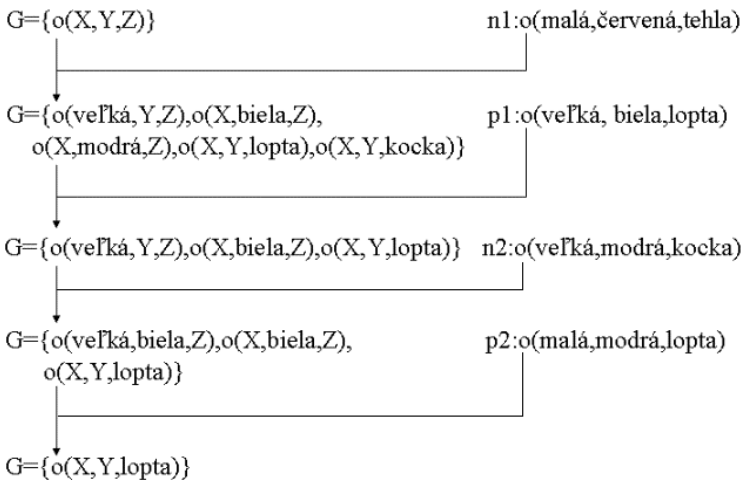
Farba = {červená, biela, modrá}

Tvar = {lopta, tehla, kocka}

Znalosti okolia sú potrebné pri procese špecifikácie pojmov, konkrétne pri náhrade premenných konštantami. Keď napríklad príde negatívny príklad, predstavujúci „malú červenú tehlu“, algoritmus vie, že množine **G** môžu byť iba pojmy, popisujúce objekty, ktoré sú veľké, biele alebo modré a je to lopta alebo kocka.

```
Vstupy:        M...množina trénovacích príkladov
               P.....množina pozitívnych trénovacích príkladov dosiaľ nájdených
Výstup:        G.....množina kandidátov hľadaných pojmov

begin
P={}
G=najvšeobecnejší pojem priestoru
for     každý negatívny príklad n
        begin
        for     každé g z G
                if      g pokrýva n
                then    nahraď g jeho najvšeobecnejšou špecifikáciou, ktorá nepokrýva n
                        vymaž z G hypotézy špecifickejšie ako iné hypotézy v G
                        vymaž z G hypotézy nepokrývajúce niektoré pozitívne príklady v P
        end
for     každý pozitívny príklad p
        begin
        vymaž z G  hypotézy, nepokrývajúce p
        pridaj p do P pre kontrolu prílišnej špecifikácie budúcich hypotéz
        end
end
```



Obr. 8: Učenie pojmu „lopta“ prehľadávaním priestoru pojmov od všeobecného k špecifickému.

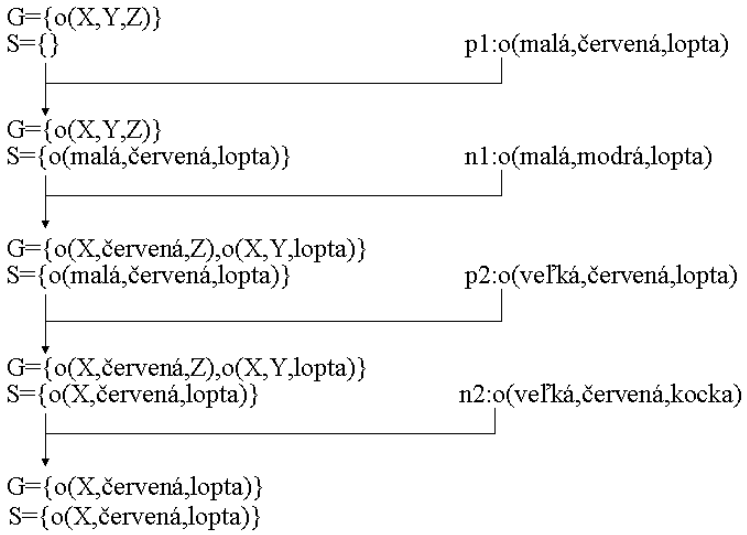
2.4.3 Algoritmus eliminácie kandidátov pojmov

Algoritmus eliminácie kandidátov pojmov bol predstavený v prácach (Mitchell, 1977), (Mitchell, 1982) a (Mitchell, 1997). Kombinuje dva prístupy: prehľadávanie od špecifického k všeobecnému a prehľadávanie od všeobecného k špecifickému do obojsmerného prehľadávania priestoru pojmov. Algoritmus buduje dve množiny kandidátov pojmov:

G...množinu maximálne všeobecných kandidátov pojmov a

S...množinu maximálne špecifických kandidátov pojmov.

Algoritmus špecifikuje obsah **G** a zovšeobecňuje obsah **S** až kým obe neskonvergujú k cieľovému pojmu. Nasledovný obrázok Obr. 9 ilustruje chovanie algoritmu eliminácie kandidátov pojmov prehľadávaním priestoru pojmov z Obr. 2.



Obr. 9: Algoritmus eliminácie kandidáta pri učení pojmu „červená lopta.

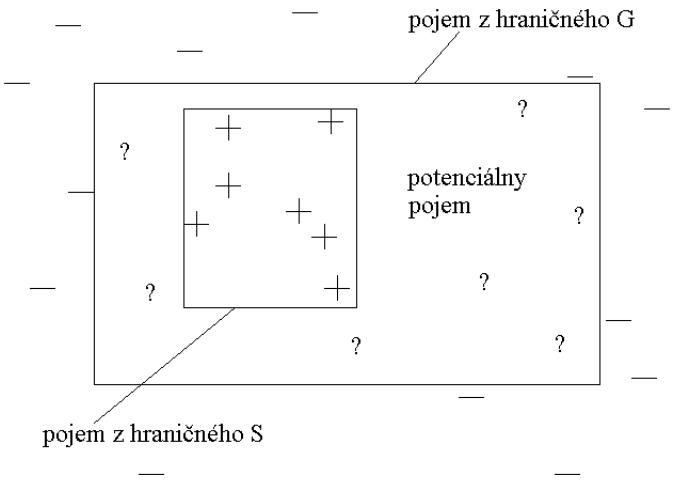
Kombinovanie dvoch smerov prehľadávania v jednom algoritme má tú výhodu, že množiny **G** a **S** sumarizujú každá zvlášť informácie o negatívnych a pozitívnych príkladoch, čím eliminujú potrebu uchovávať tieto príklady. Preto nie je potrebné používať množiny **P** a **N** z predchádzajúcich algoritmov.

G je množina maximálne všeobecných pojmov, ktoré nepokrývajú žiadne negatívne trénovacie príklady. Preto každý prvok množiny **S**, ktorý je všeobecnejší, ako niektorý prvok **G**, musí pokrývať nejaký negatívny trénovací príklad a musí byť vymazaný. Podobne **S** je množina

maximálne špecifických pojmov, ktoré pokrývajú všetky pozitívne príklady. Preto každý nový prvok **G**, ktorý je špecifickejší ako niektorý prvok **S**, nepokrýva niektoré pozitívne príklady a musí byť tiež eliminovaný. Obr. 10 ilustruje prácu algoritmu eliminácie kandidátov, ktorého popis nasleduje:

Vstupy: ***M****...množina trénovacích príkladov*
Výstupy: ***S****.....množina maximálne špecifických kandidátov hľadaných pojmov*
G*.....množina maximálne všeobecných kandidátov hľadaných pojmov*

begin
S*=prvý pozitívny trénovací príklad*
G*=najvšeobecnejší pojem v priestore*
for *každý nový pozitívny trénovací príklad* ***p***
begin
vymaž z ***G*** *hypotézy, ktoré nepokrývajú* ***p***
for *každé* ***s*** *z* ***S***
if ***s*** *nepokrýva* ***p***
then *nahraď* ***s*** *jeho najšpecifickejším zovšeobecnením,*
ktoré pokrýva ***p***
. *vymaž z* ***S*** *hypotézy všeobecnejšie ako iné hypotézy v* ***S***
vymaž z ***S*** *hypotézy všeobecnejšie ako nejaká hypotéza v* ***G***
end
for *každý nový negatívny príklad* ***n***
begin
vymaž z ***S*** *hypotézy, ktoré pokrývajú* ***n***
for *každé* ***g*** *z* ***G***
if ***g*** *pokrýva* ***n***
then *nahraď* ***g*** *jeho najvšeobecnejšou špecifikáciou,*
ktorá nepokrýva ***n***
vymaž z ***G*** *hypotézy špecifickejšie ako iné hypotézy v* ***G***
vymaž z ***G*** *hypotézy špecifickejšie ako niektorá hypotéza v* ***S***
end
if ***G=S***
then *algoritmus našiel jednoduchý pojem, ktorý je konzistentný*
so všetkými údajmi v trénovacej množine
end
if ***G*** *a* ***S*** *sú prázdne*
then *neexistuje pojem, ktorý by pokrýval všetky pozitívne príklady*
a žiaden negatívny
end



Obr. 10: Hranice pokrývania množín G a S v algoritme eliminácie kandidátov.

Algoritmus "sťahuje" hraničné **G** natoľko, nakoľko je to nutné na vylúčenie všetkých negatívnych trénovacích príkladov. Zároveň "rozťahuje" hraničné **S** do tej miery, aby zahrnul všetky pozitívne trénovacie príklady.

Ak **G** a **S** skonvergujú do toho istého pojmu, daný pojem je riešením a algoritmus môže ukončiť svoju prácu. Ak **G** a **S** ostanú prázdne, potom neexistuje pojem, ktorý by bol konzistentný s ténovacími príkladmi. Neexistuje také riešenie. To sa môže stať, ak sú trénovacie údaje nekonzistentné, teda na vstupe existuje šum. Alebo sa cieľový pojem nedá vyjadriť v reprezentačnom jazyku (v striktnej forme logickej konjunkcie). Ak **G** a **S** skonvergujú k dvom rôznym pojmom, potom obidva pojmy sú riešeniami a takisto za riešenie môžeme považovať každý potenciálny pojem, ktorý sa v priestore pojmov nachádza medzi pojmami s **G** a **S**. To je znázornené na Obr. 10. Algoritmus produkuje všetky možné riešenia naraz.

Zaujímavým aspektom tohto algoritmu je jeho inkrementálna povaha, čo znamená, že akceptuje trénovacie príklady postupne jeden za druhým. Formuje použiteľné, aj keď možno neúplné zovšeobecnenie po každom príklade.

Ešte predtým, ako tento algoritmus skonverguje do jedného pojmu, množiny **G** a **S** vytvoria obmedzenia tohto pojmu. Tieto obmedzenia je možné definovať nasledovne:

Ak c je cieľový pojem, potom pre všetky g z G a s z S platí $s \leq c \leq g$. Každý pojem, ktorý je všeobecnejší (väčšia oblasť) ako nejaký pojem v **G**, bude pokrývať negatívne príklady. Každý pojem ktorý je špecifickejší (menšia oblasť) ako nejaký pojem v **S**, nebude pokrývať niektoré pozitívne príklady.

Algoritmus eliminácie kandidátov bol aplikovaný na také problémy, ako pravidelnosti učenia v chemickej spektroskopii hmoty (Mitchell, 1979), alebo učenie riadiacich pravidiel pre

heuristické prehľadávanie (Mitchell, 1983). (Haussler, 1988) dokazuje, že veľkosť všeobecných hraníc konjunktívnej oblasti môže rásť exponenciálne v závislosti od počtu trénovacích príkladov, dokonca aj v prípade, že priestor hypotéz pozostáva iba z jednoduchých konjunkcií vlastností. V určitých prípadoch je možné zvýšiť komplexnosť jednoduchou zmenou reprezentácie množiny **G** (Smith-Rosenbloom, 1990). Učenie môže byť aj polynomiálne závislé od počtu trénovacích príkladov (Hirsh, 1991), a to v niektorých prípadoch, keď množina **G** nie je celá uchovávaná. V (Subramanian-Feigenbaum, 1986) je diskutovaná metóda, ktorá môže generovať efektívne otázky v určitých prípadoch. Najväčším praktickým obmedzením algoritmu eliminácie kandidátov je požiadavka nezašumených trénovacích údajov. (Mitchell, 1979) popisuje rozšírenie tohto algoritmu, ktoré dokáže spracovať ohraničený a vopred definovaný počet nesprávne klasifikovaných príkladov. (Hirsh, 1990) a (Hirsh, 1994) definuje elegantné rozšírenie pre spracovanie ohraničeného šumu v reálnych atribútoch.

2.5 Neinkrementálna indukcia logických konjunkcií

Ak máme priestor popisov pojmov usporiadaný podľa všeobecnosti, ponúkajú sa nám pri neikrementálnej, podobne ako pri VS algoritmoch, dva jednoduché spôsoby generovania:

- od všeobecného popisu pojmu ku špecifickému. Algoritmy tejto povahy majú vo svojom názve skratku GS (General to Specific).
- od špecifického popisu pojmu ku všeobecnému. Tieto algoritmy majú v názve SG (Specific to General).

Budeme sa zameriavať na dve skupiny neinkrementálnych algoritmov:

- algoritmy indukcie logických konjunkcií úplným prehľadávaním
- algoritmy heuristickej indukcie logických konjunkcií.

2.5.1 Indukcia logických konjunkcií úplným prehľadávaním

Logickú indukciu neinkrementálnej povahy úplným prehľadávaním začneme prehľadávaním od všeobecného ku špecifickému. Reprezentantom tohto prístupu je algoritmus **EGS (Exhaustive General-to-Specific)**. Tento algoritmus je definovaný v (Langley, 1996). Na vstupe potrebuje tento algoritmus množinu pozitívnych a množinu negatívnych trénovacích príkladov. Na výstupe poskytuje množinu popisov pojmov, ktoré pokrývajú všetky pozitívne a žiaden negatívny trénovací príklad. Je to rekurzívny algoritmus s prehľadávaním do šírky v priestore pojmov.

Na každej úrovni prehľadávania, resp. úrovni rekurzie **EGS** zvažuje všetky špecifikácie **S** všeobecných popisov **H**, t.j. pracovných popisov, ktoré je potrebné preskúmať. Tieto špecifikácie sa tvoria pridaním jednej podmienky alebo pri numerických atribútoch zmenou hraničných hodnôt v podmienkach. Pre nominálny atribút, ktorý ešte nebol použitý v danom popise, to predstavuje generovanie jednej alternatívnej podmienky pre každú hodnotu atribútu.

Pre každý takto špecifikovaný popis **H** uskutočňuje algoritmus test, aby sa uistil, že **H** stále pokrýva všetky pozitívne trénovacie príklady. V opačnom prípade vypustí tento popis ako príliš špecifický z ďalšieho uvažovania, t.j. vymaže ho. Je potrebné si uvedomiť, že tým vymaže aj všetkých jeho prípadných potomkov.

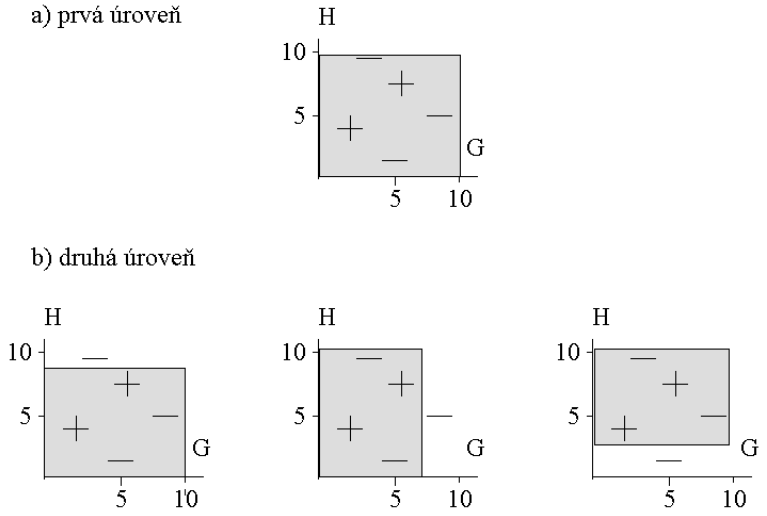
Vstupy: ***PSET****...množina pozitívnych trénovacích príkladov*
NSET*...množina negatívnych trénovacích príkladov*
CSET*...množina konzistentných popisov hľadaného pojmu*
HSET*...množina príliš všeobecných popisov hľadaného pojmu*
Výstup: Množina najvšeobecnejších logických konjunkcií, konzistentných s príkladmi.
Volanie na najvyššej úrovni: ***egs(PSET,NSET,{},{{}})***

Procedúra ***egs(PSET,NSET,CSET,HSET)***
for *každý pojem* ***H*** *v* ***HSET***
if ***H*** *nepokrýva všetkých členov* ***PSET***
then *vymaž* ***H*** *z* ***HSET***
if ***H*** *nepokrýva žiadny člen z* ***NSET***
then *vymaž* ***H*** *z* ***HSET*** *&* *pridaj* ***H*** *do* ***CSET***
if ***HSET****={}*
then *vráť* ***CSET***
else *nech* ***NEWSET****={}*
for *každý pojem* ***H*** *v* ***HSET***
nech ***SPECS*** *sú všetky jedno-podmienkové špecifikácie* ***H***
for *každý pojem* ***S*** *v* ***SPECS***
if ***CSET*** *neobsahuje pojem všeobecnejší ako* ***S***
(kontrola prílišnej špecifikácie)
then *pridaj* ***S*** *do* ***NEWSET***
egs(PSET,NSET,CSET,NEWSET)

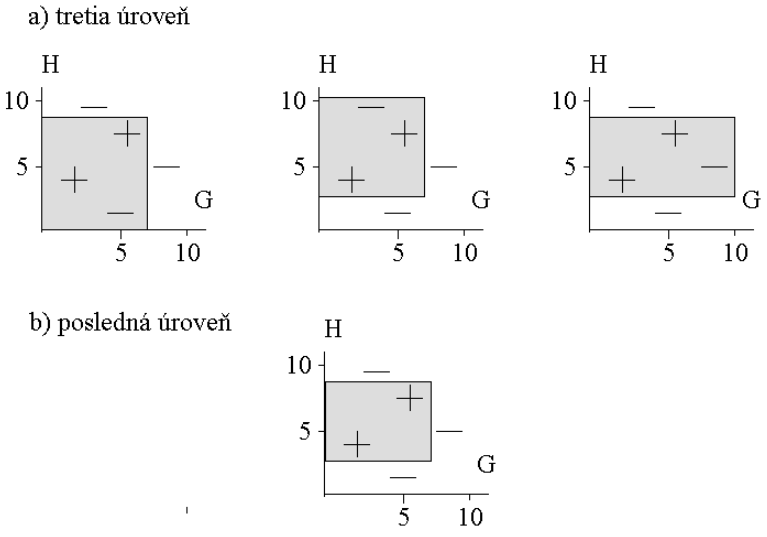
Pre všetky ostávajúce popisy algoritmus kontroluje, či **H** stále pokrýva nejaký negatívny trénovací príklad. Ak nie, **H** je pridané do množiny konzistentných popisov, t.j. do množiny prípadných riešení. Algoritmus končí, keď množina príliš všeobecných popisov bude prázdna. Vtedy vráti množinu konzistentných popisov pojmov z predchádzajúcej úrovne rekurzie. Každý popis z tejto množiny garantovane pokrýva všetky pozitívne a žiaden negatívny trénovací príklad.

Logické konjunkcie generované algoritmom EGS zahŕňajú iba relevantné atribúty, ktoré sú nevyhnutné na diskrimináciu negatívnych príkladov.

Ďalšia ilustrácia Obr. 11 znázorňuje popisy pojmov odpovedajúce prvým dvom úrovniam rekurzie algoritmu EGS nad numerickou doménou. Obr. 12 predstavuje posledné dve úrovne rekuzie tohto algoritmu nad danou doménou.



Obr. 11: Prvé dve úrovne rekurzie EGS nad numerickou doménou.



Obr. 12: Posledné dve úrovne rekurzie EGS nad numerickou doménou.

Nevýhodou algoritmu EGS je narastanie jeho výpočtovej zložitosti, ktorá rastie exponenciálne v dôsledku úplného prehľadávania do šírky.

Alternatívou k predchádzajúcemu algoritmu je prehľadávanie od špecifického k všeobecnému **ESG (Exhaustive Specific-to-General)**. Podobne ide o logickú indukciu neinkrementálnej povahy úplným prehľadávaním Táto metóda hľadá najšpecifickejšiu logickú konjunkciu konzistentnú s trénovacími príkladmi. Algoritmus ESG má niekoľko výpočtových výhod v porovnaní s EGS algoritmom. ESG jednoducho počíta najšpecifickejší popis bez akejkoľvek potreby prehľadávania. Nepracuje s negatívnymi príkladmi, keďže tieto nemôžu ovplyvniť výsledok. Pre nominálne atribúty, táto metóda hľadá jednoducho hodnoty atribútov zdieľané všetkými pozitívnymi trénovacími príkladmi. Pre numerické atribúty nájde ich minimálnu a maximálnu hodnotu vyhovujúcu všetkým pozitívnym trénovacím príkladom.

Obmedzením oboch týchto algoritmov je ich neschopnosť zvládať situácie, v ktorých žiadna logická konjunkcia nie je konzistentná s trénovacími údajmi. Takisto majú tieto algoritmy problémy pracovať so zašumenými doménami. V zašumenej doméne, dokonca aj jedna neoznačená trieda alebo vlastnosť môže seriózne prekaziť nájdenie konzistentného popisu v tvare logickej konjunkcie.

Keby sme mali porovnať kvalitu výsledkov algoritmov EGS a ESG, ESG dáva presnejšie výsledky, pretože sú spravidla špecifickejšie ako výsledky EGS nad tou istou trénovacou množinou. To znamená, že je menšia pravdepodobnosť, že popis pojmu generovaný algoritmom ESG zahŕňa nejaké zatiaľ neevidované negatívne príklady.

2.5.2 Heuristická indukcia logických konjunkcií

Heuristické metódy uskutočňujú selektívnejšie prehľadávanie ako algoritmy využívajúce systematické prehľadávanie, ako sú algoritmy VS, EGS, ESG, atď. Heuristické algoritmy nájdu v mnohých prípadoch uspokojujúce riešenie, hoci prehľadávajú iba časť celého priestoru pojmov. Heuristická indukcia je riadená hodnotiacou funkciou a takzvaným lúčovým prehľadávaním (beam search).

Prvý algoritmus tejto skupiny **HGS (Heuristic General-to-Specific)** akceptuje na vstupe množiny negatívnych a pozitívnych trénovacích príkladov (Michalski, 1980). Na výstupe algoritmu sú dve množiny:

- CLOSED-SET, ktorá uchováva kandidátov popisov hľadaného pojmu, ktoré nemôžu byť vylepšené ďalšou špecifikáciou
- HSET, ktorá uchováva popisy, ktoré môžu byť ďalej vylepšené.

Na každej úrovni prehľadávania HGS uvažuje všetky špecifikácie popisov pojmov v **HSET**, ktoré sa získajú pridaním jednej podmienky alebo pri numerických atribútoch zmenou hraničných hodnôt v podmienkach. Pre každý špecifikovaný popis **S** používa algoritmus hodnotiacu funkciu na meranie stupňa zhody s trénovacou množinou t.j. stupeň pokrytia pozitívnych a nepokrytia negatívnych trénovacích príkladov. Ak je skóre **S** väčšie ako skóre jeho rodiča **H**, pridáva hypotézu **S** ku množine nových pojmov **NEW-SET**. Ak žiadna zo špecifikácií nemá lepšie skóre ako jej rodič, potom je H pridané do množiny **CLOSED-SET**, keďže nemôže

byť v ďalšom vylepšené. Ak má rodič nejakých lepšie skórovaných potomkov, tak zaniká – zabúda sa.

Pred zahájením pokračovania na ďalšej (špecifickejšej) úrovni, algoritmus musí redukovať hypotetickú množinu popisov pojmov do zvládnuteľnej veľkosti. Preto vyberá **Beam-Size** (veľkosť lúča, záber lúča) počet členov s najvyšším skóre zo zjednotenia **CLOSED-SET** a **OPEN-SET**. Potom tieto vybrané popisy pojmov opätovne rozdelí do množín **CLOSED-SET** a **OPEN-SET**. Napokon volá rekurzívne sám seba s redukovanou množinou **OPEN-SET** hrajúcou úlohu **HSET**.

Vstupy: ***PSET****...množina pozitívnych trénovacích príkladov*
NSET*...množina negatívnych trénovacích príkladov*
CLOSED-SET*...množina lokálne optimálnych popisov hľadaného pojmu*
*H****SET****...množina popisov hľadaného pojmu*
Výstup: popis pojmu v tvare logickej konjunkcie
Parameter: ***Beam-Size*** *počet popisov pojmov na novej iteračnej úrovni*
Volanie na najvyššej úrovni: ***hgs(PSET,NSET,{},{{}})***

Procedúra: ***hgs(PSET,NSET,CLOSED-SET,HSET)***
nech ***OPEN-SET****={}*
for *každý pojem* ***H*** *v* ***HSET***
nech ***SPECS*** *sú všetky jedno-podmienkové špecifikácie* ***H****,*
nech ***NEWSET****={}*
for *každý špecifikovaný pojem* ***S*** *v* ***SPECS***
if ***Score (S,PSET,NSET)>Score (H,PSET,NSET)***
then *pridaj* ***S*** *do* ***NEWSET***
if ***NEW-SET****={}*
then *pridaj* ***H*** *do* ***CLOSED-SET***
else ***for*** *každý pojem* ***S*** *v* ***NEW-SET***
pridaj ***S*** *do* ***OPEN-SET***
for *každý pojem* ***C*** *v* ***CLOSED-SET***
if ***S*** *je aspoň tak špecifický ako* ***C***
then ***if Score (C,PSET,NSET)>Score (S,PSET,NSET)***
then *vymaž* ***S*** *z* ***OPEN-SET***
else *vymaž* ***C*** *z* ***CLOSED-SET***
if ***OPEN-SET****={}*
then *vráť člena s najvyšším skóre v* ***CLOSED-SET***
else
nech ***BEST-SET*** *je* ***Beam-Size*** *počet najvyššie skórovaných*
členov zjednotenia ***OPEN-SET*** *a* ***CLOSED-SET***
nech ***CLOSED-SET*** *je množina členov* ***CLOSED-SET*** *v* ***BEST-SET***
nech ***OPEN-SET*** *je množina členov* ***OPEN-SET*** *v* ***BEST-SET***
hgs (PSET,NSET,CLOSED-SET,OPEN-SET)*.*

Algoritmus vymazáva nezvolených kandidátov pojmov z pamäti, dokonca aj keď môžu viesť k použiteľným popisom pojmov. HGS takto vytvára kompromis medzi optimálnosťou a neobmedzeným prehľadávaním.

Popis pojmu generovaný HGS negarantuje pokrytie všetkých pozitívnych trénovacích príkladov a žiadneho negatívneho trénovacieho príkladu. Generované popisy pojmov nie sú garantovane minimálne, pretože môžu existovať všeobecnejšie popisy hľadaného pojmu v niektorej odmietnutej vetve. Je to cena zaplatená za efektívnosť heuristického prehľadávania. Na druhej strane je HGS úspešnejší v zašumených doménach.

Teda vidíme, že výsledkom HGS nemusí byť striktne logická konjunkcia, ktorá je konzistentná s trénovacími príkladmi. Výsledná logická konjunkcia má iba pokrývať čo najviac pozitívnych a čo najmenej negatívnych trénovacích príkladov. To koľko pozitívnych trénovacích príkladov pojmu **H** nebude pokrytých a koľko negatívnych bude, závisí od hodnotiacej funkcie **Score(H, PSET, NSET)**.

Podobne ako pri úplnom (exhaustive) prehľadávaní, má algoritmus HGS alternatívu **HSG (Heuristic Specific-to-General)** , ktorá heuristicky prehľadáva priestor pojmov opačným smerom od špecifického ku všeobecnému. V mnohých prípadoch dáva lepšie výsledky, ako algoritmus HGS. Dôvodom je, že dokáže nájsť špecifickejšie riešenie. To znamená, že je menšia pravdepodobnosť, že popis pojmu generovaný algoritmom HSG zahŕňa nejaké zatiaľ neevidované negatívne príklady.

2.5.2.1 Heuristické ohodnocovanie popisov pojmov

Chovanie HGS veľmi závisí od výberu ohodnocovacej funkcie **Score(H, P, N)**, ktorá má tri argumenty:
H hodnotený popis pojmu
P množina pozitívnych trénovacích príkladov
N množina negatívnych trénovacích príkladov.

Vo všeobecnosti funkčné skóre narastá tak s $\mathbf{P_c}$ počtom pokrytých pozitívnych trénovacích príkladov, ako aj s $\mathbf{N_{nc}}$ počtom nepokrytých negatívnych trénovacích príkladov. Dobrá metrika berie do úvahy aj celkové množstvo pozitívnych a negatívnych príkladov. Príkladom často používanej miery, ktorá uspokojuje obidve uvádzané požiadavky je:

$$Score(H, P, N) = \frac{P_c + N_{nc}}{P + N}.$$

Hodnoty tejto ohodnocovacej funkcie sa pohybujú v intervale od 0 do 1. Nulová hodnota znamená, že žiaden pozitívny trénovací príklad nie je pokrytý pojmom **H** a zároveň všetky negatívne trénovacie príklady sú pokryté. Jednotková hodnota predstavuje ideálny prípad, keď sú pokryté všetky pozitívne a žiaden negatívny trénovací príklad. Táto funkcia meria celkovú mieru konzistencie uvažovaného pojmu s trénovacími príkladmi.

Zložitejšie prístupy používajú štatistické alebo informačné miery. Avšak aj celkom jednoduchá miera:

$$Score(H, P, N) = P_c + N_{nc}$$

dokáže vhodne riadiť smerovanie algoritmu HGS. Predpokladajme, že meráme skóre popisu pojmu podľa prvého vzťahu a položme **Beam-Size** rovné nekonečnu. V tom prípade algoritmus HGS simuluje činnosť algoritmu EGS.

2.6 Úlohy na precvičenie:

1. Doplňte chýbajúci operátor zovšeobecnenia:
   a/ nahradenie konštanty premennou
   b/ .....
   c/ pridanie disjunkcie do výrazu
   d/ nahradenie atribútu jeho rodičom
2. V čom spočíva riziko prílišnej špecifikácie a prílišného zovšeobecnenia?
3. Ktorá základná charakteristika odlišuje od seba tri algoritmy VSS?
4. V čom sa líši heuristické od úplného prehľadávania priestoru pojmov?
5. Uveďte najčastejšie používanú heuristickú skórovaciu funkciu.
6. Aký bude rozdiel v chovaní algoritmu HGS, ak sa ako ohodnocovacia funkcia použije jedna z troch funkcií: $\frac{P_c + N_n}{P + N}, \frac{P_c}{P} + \frac{N_n}{N}, P_c + N_n$.
7. Simulujte prácu „algoritmu eliminácie kandidáta“ v nominálnej doméne ľudí danej nasledovnou množinou trénovacích príkladov:

	VÝŠKA	VLASY	OČI	TRIEDA
1	nízky	blond	hnedé	-
2	vysoký	tmavé	hnedé	-
3	vysoký	blond	modré	+
4	vysoký	tmavé	modré	-
5	nízky	tmavé	modré	-
6	vysoký	ryšavé	modré	+
7	vysoký	blond	hnedé	-
8	nízky	blond	modré	+

Simulujte prácu algoritmu HGS v nominálnej doméne ľudí z predchádzajúcej úlohy. Porovnajte výsledky.

2.7 Literatúra

Bruner, J.S., Goodnow, J.J., Austin, G.A.: A study of thinking. New York: John Wiley, 1957.

Buchanan, B.G.: Scientific theory formation by computer. In J.C. Simon (Ed.), Computer Oriented Learning Processes. Leyden: Noordhoff, 1974.

Haussler, D.: Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. Artificial Intelligence, 36, 1988, pp. 177-221.

Hayes-Roth, F.: Schematic classification problems and their solution. Patern Recognition, 6, 1974, pp. 105-113.

Hirsh, H.: Incremental version space merging: A general framework for concept learning. Boston, 1990.

Hirsh, H.: Theoretical underpinnings of version spaces. Proceedings of the 12th IJCAI, Sydney, 1991, pp. 665-670.

Hirsh, H.: Generalizing version spaces. Machine Learning, 17(1), 1994, pp. 5-46.

Hunt, E.G., Hovland, D.I.: Programming a model of human concept formation. In E.Feigenbaum & J.Feldman (Eds.), Computers and thought (pp. 310-325), New York: McGraw Hill, 1963.

Langley, P.: Elements of Machine Learning. Morgan Kaufmann Publishers, Inc. San Francisco, California, 1996, 419 ps.

Michalski, R.S.: AQVAL/1: Computer implementation of a variable valued logic system VL1 and examples of its application to pattern recognition. Proceedings of the 1st International Joint Conference on Pattern Recognition, 1973, pp. 3-17.

Michalski, R.S.: Pattern recognition as rule-guided inductive inference. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2, 1980, pp. 349-361.

Mitchell, T.M.: Version spaces: A candidate elimination approach to rule learning. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge, MA: Morgan Kaufmann, 1977, pp. 305-310.

Mitchell, T.M.: Version spaces: An approach to concept learning, (Ph.D. dissertation). Electrical Engineering Dept., Stanford University, Stanford, CA, 1979.

Mitchell, T.M.: Generalization as search. Artificial Intelligence, 18(2), 1982, 203-226.

Mitchell, T.M., Utgoff, P.E., Banerji, R.: Learning by experimentation: Acquiring and modifying problem-solving heuristics. In Michalski, Carbonell, Mitchell (Eds.), Machine Learning Vol.1, Tioga Press, 1983, pp. 163-190.

Mitchell, T.M.: Machine Learning. The McGraw-Hill Companies, Inc. New York, 1997, 414 ps.

Popplestone, R.J.: An experiment in automatic induction. In Meltzer & Mitchie (Eds.), Machine Intelligence 5, Edinburgh University Press, 1969, pp. 204-215.

Simon, H.A., Lea, G.: Problem solving and rule induction: A unified view. In Gregg (Ed.), Knowledge and Cognition, New Jersey: Lawrence Erlbaum Associates, 1973.

Smith, B.D., Rosenbloom, P.: Incremental non-backtracking focusing: A polynomially bounded generalization algorithm for version spaces. Proceedings of the 1990 National Conference on Artificial Intelligence, Boston, 1990, pp. 848-853.

Subramanian, D., Feigenbaum, J.: Factorization in experiment generation. Proceedings of the 1986 National Conference on Artificial Intelligence, Morgan Kaufmann, 1986, pp. 518-522.

Vere, S.A.: Induction of concepts in the predicate calculus. Fourth International Joint Conference on AI, Tbilisi, USSR, 1975, pp. 281-287.
Winston, P.H.: Learning structural descriptions from examples, (Ph.D. dissertation). [MIT Technical Report AI-TR-231], 1970.

3 G ENEROVANIE PRODUKČNÝCH PRAVIDIEL

3.1 Reprezentácia a použitie produkčných pravidiel

Logické konjunkcie majú množstvo obmedzení. Najvážnejšie z nich sú reprezentačné preferencie, ktoré znamenajú, že všetky pozitívne príklady musíme vtesnať do jednej oblasti v tvare hyperobdĺžnika. Pritom nesmieme zahrnúť ani jeden negatívny príklad pojmu. Pri mnohých reálnych množinách trénovacích príkladov je toto obmedzenie neprekonateľným problémom. Produkčné pravidlá sú ďalšou možnou reprezentáciou znalostí naučených pomocou techník strojového učenia z typických resp. trénovacích príkladov (Arbab-Mitchie, 1985).Táto reprezentácia umožňuje na rozdiel od logických konjunkcií aj takzvanú "disjunkciu konjunkcií". To znamená, že všetky pozitívne príklady nemusíme zahrnúť do jednej obdĺžnikovej oblasti, ale môžeme ich rozdeliť do viacerých obdĺžnikových oblastí. Tým stúpa pravdepodobnosť, že sa nám podarí vyhnúť negatívnym príkladom a zahrnúť všetky pozitívne príklady pojmu. Typickou oblasťou použitia produkčných pravidiel sú znalostné systémy.

Množina produkčných pravidiel realizujúcich klasifikáciu do triedy, predstavuje logický výraz. Niekedy hovoríme o klasifikačných pravidlách. Daný logický výraz popisuje klasifikačnú triedu resp. pojem tým, že definuje nutné a postačujúce podmienky pre príslušnosť trénovacieho príkladu do danej triedy. Uvedený výraz má tvar disjunkcie:

podmienky_pravidla1 ∨ podmienky_pravidla2...

PdmienkyPravidlaX sú v tomto prípade reprezentované konjunkciou podmienok. Preto vyššie uvedený výraz môže predstavovať disjunktívnu normálnu formu (DNF), teda disjunkciu konjunkcií podmienok.

Doteraz bol uvažovaný popis klasifikovaného pojmu buď vo veľmi striktnej konjunktívnej forme, alebo v niektorej z voľnejších foriem, ako sú prahové pojmy, etalóny alebo pravdepodobnostná reprezentácia. Teraz sa pokúsime o indukciu popisu pojmu vo forme disjunktívnej normálnej formy. Pri logických konjunkciách bolo potrebné popísať klasifikovaný pojem jedným konjuktívnym výrazom, ktorý by bol reprezentovaný jednou obdĺžnikovou oblasťou, pokrývajúcou všetky pozitívne príklady pojmu alebo aspoň čo najväčší počet pozitívnych príkladov (algoritmus HGS). DNF pripúšťa viac konjunktívnych výrazov spojených logickou funkciou disjunkcie. Preto môže byť reprezentovaná viacerými oblasťami, sústredenými v miestach najväčšieho výskytu pozitívnych príkladov pojmu.

Indukovanú DNF môžeme použiť na klasifikáciu do dvoch tried, pričom úlohu indukcie môžeme redukovať na konštrukciu DNF výrazu pre jednu triedu a druhú triedu klasifikovať implicitne (výraz pre prvú triedu neuspeje). Je možná aj klasifikácia do viacerých tried. V takom prípade je potrebné indukovať produkčné pravidlá pre každú triedu zvlášť.

Vyššie vedené podmienky DNF definujú, aké hodnoty môžu jednotlivé atribúty nadobúdať. Tieto podmienky predstavujú ľavú stranu produkčného pravidla. Pravá strana

produkčného pravidla predstavuje záver, ktorý platí keď sú splnené podmienky ľavej strany daného pravidla. Týmto záverom môže byť napríklad klasifikácia do triedy $\mathbf{T_i}$.

if podmienky_pravidla1 $\vee$ podmienky_pravidla2... then T_i

Spravidla sa uprednostňuje zjednodušená forma zápisu tohto produkčného pravidla, keď sa ono transformuje do sady **N** alternatívnych konjunktívnych pravidiel:

if podmienky_pravidla1... then T_i
if podmienky_pravidla2... then T_i
...
if podmienky_pravidlaN... then T_i

Disjunkcia jednotlivých produkčných pravidiel predstavuje naučené znalosti získané indukciou zo vstupných údajov, napríklad množiny typických príkladov. Teda na vstupe algoritmov indukujúcich DNF máme množinu typických príkladov a na výstupe očakávame rozhodovaciu procedúru (produkčné pravidlo).

Publikácia (Svátek, 1994) sa zaoberá problémom, ktorý vzniká generovaním znalostí teda produkčných pravidiel zo zašumených alebo nie celkom presných údajov. Také znalosti môžu byť nekompletné a čiastočne nesprávne. Tento problém rieši zavádzaním vážených pravidiel.

3.2 Indukcia produkčných pravidiel

Produkčné pravidlá vieme indukovať dvoma spôsobmi:

- priamo, hovoríme o priamom generovaní produkčných pravidiel
- nepriamo, akúkoľvek inú reprezentáciu, uvedenú v tomto texte, je možné pretransformovať do sady produkčných pravidiel

Vo všeobecnosti platí, že akákoľvek reprezentácia znalostí používaná v strojovom učení sa dá transformovať na ktorúkoľvek inú reprezentáciu znalostí. Najčastejšie ide o klasifikáciu príkladov do tried pomocou produkčných pravidiel. V oblasti strojového učenia sa preto často používa pojem klasifikačné pravidlo namiesto produkčného pravidla.

3.2.1 Metóda rozdeľuj a panuj

Riešením úlohy nájdenia popisu pojmu v tvare DNF môže byť napríklad algoritmus **NSC (Nonincremental Separate-and-Conquer)** (Langley, 1996). Je to neikrementálny algoritmus, ktorého stratégiu by bolo možné charakterizovať dvoma slovami: separuj a panuj. Tento algoritmus je definovaný nasledovne:

Vstupy: ***PSET****...množina pozitívnych trénovacích príkladov*
NSET*...množina negatívnych trénovacích príkladov*
Výstup: ***DNF****.....disjunkcia konjunktívnych popisov jednotlivých oblastí.*
Volanie na najvyššej úrovni: ***NSC(PSET,NSET,{})***

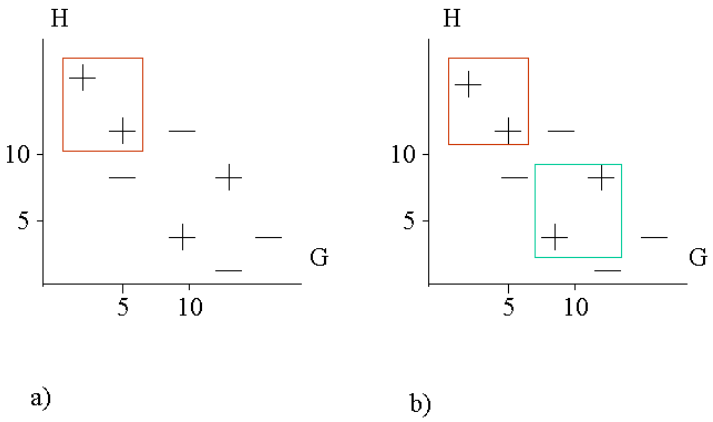
Procedúra: ***NSC(PSET,NSET,DNF)***
if ***PSET****={}*
then *vráť* ***DNF***
else
volaj podprogram, ktorý nájde popis ***D*** *jednotlivej oblasti pokrývajúcej niektoré príklady*
(nie všetky) z ***PSET*** *a nepokrývajúcej žiadne príklady z* ***NSET***
pridaj popis ***D*** *do* ***DNF***
z ***PSET*** *vymaž príklady pokryté* ***D***
vráť NSC(PSET,NSET,DNF)

NSC vyvoláva v prvom kroku ako podprogram induktívny algoritmus na konštrukciu popisu, ktorý pokrýva niektoré pozitívne príklady (nie nutne všetky), ale žiadny negatívny trénova cí príklad. Úlohu takéhoto podprogramu môže splniť niektorý z algoritmov na konštrukciu konjunktívneho popisu (HGS, HSG, ...), alebo iná z prahových metód (algoritmus sférickej prahovej jednotky).

Algoritmus NSC je vhodný pre takú trénovaciu množinu, v ktorej nie je možné vytvoriť súvislú oblasť (pri dvoch atribútoch – obdĺžnik) okolo všetkých pozitívnych príkladov tak, aby nezahŕňal aj niektoré negatívne príklady. Tento algoritmus rieši problém tak, že ho rozdelí na podproblémy. Vytvorí súvislú oblasť nad časťou množiny pozitívnych príkladov. Nad ostatnými pozitívnymi príkladmi sa pokúsi vytvoriť novú súvislú oblasť.

NSC pridáva nájdený popis **D** do **DNF** výrazu, vypúšťa príklady pokryté týmto výrazom a volá sám seba rekurzívne s redukovanou trénovacou množinou a rozšíreným **DNF** výrazom. Algoritmus končí, keď v trénovacej množine neostanú žiadne pozitívne príklady. Záverom vráti aktuálnu **DNF**.

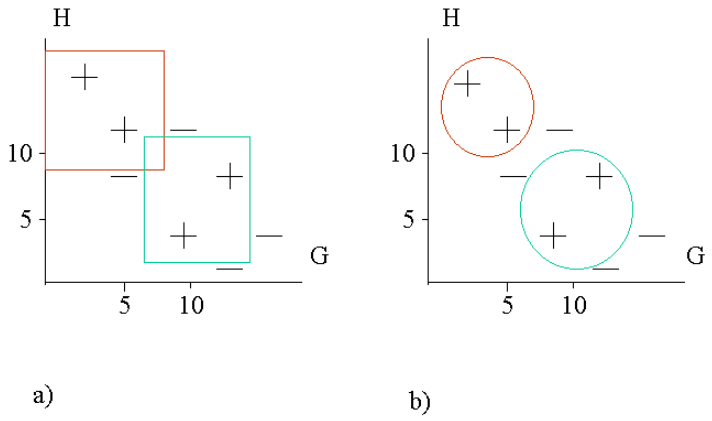
Chovanie algoritmu NSC ilustruje na numerickej doméne Obr. 13, kde v časti a) je výsledok po jednom volaní podprogramu HSG a v časti b) je výsledok po druhom volaní toho istého podprogramu.



Obr. 13: Chovanie algoritmu NSC na numerickej doméne.

Je jasné, že v numerickej doméne na Obr. 13 nie je možné nakresliť jeden obdĺžnik okolo všetkých pozitívnych príkladov. Teda jednoduchý konjunktívny popis neprichádza do úvahy. Je potrebné generovať DNF, napríklad pomocou algoritmu NSC, ktorý volá podprogram HSG. Tento podprogram vyberie ako východzí jeden z pozitívnych príkladov. Lúčové prehľadávanie postupne buduje všeobecnejšiu verziu v tvare obdĺžnikovej deliacej hranice v a) časti Obr. 13. Ďalšie zovšeobecňovanie (rozširovanie hraníc) nie je žiadúce, pretože by zahrnulo aj negatívne príklady. NSC preto vymaže dva pokryté pozitívne príklady z trénovacej množiny a nad takto aktualizovanou trénovacou množinou opätovne vyvolá podprogram, čoho výsledkom je b) časť Obr. 13.

Nie akýkoľvek algoritmus na indukciu popisov pojmov sa hodí ako podprogram na volanie v rámci NSC. Tento fakt dokumentuje Obr. 14. V časti a) tohto obrázku bol ako podprogram volaný HGS, ktorý sa ukázal ako menej vhodný, pretože ním generované oblasti sú maximálne všeobecné t.j. tak veľké ako to dovolia negatívne príklady. V budúcnosti (po rozšírení množiny príkladov) môže dôjsť k tomu, že do oblasti padne nejaký negatívny trénovací príklad a teda dôjde k nejednoznačnej klasifikácii. V časti b) tohto obrázku bol ako podprogram volaný modifikovaný algoritmus pre sférickú prahovú jednotku, ktorý sa ukázal ako vhodnejší, pretože ním generované oblasti sú menej všeobecné.



Obr. 14: Deliace hranice produkované NSC pri volaní rôznych podprogramov.

3.2.2 Algoritmy AQ

AQ systém autorov (Michalski- Chilausky, 1980) bol jedným z prvých pokusov o implementáciu neinkrementálneho prístupu k tvorbe pravidiel metódou „rozdeľuj a panuj“ (separate-and-conquer). Najznámejším reprezentantom tejto skupiny je algoritmus AQ11. Existujú rôzne variácie a vylepšenia tohto algoritmu. Z ďalších reprezentantov sa často cituje algoritmus AQ15 (Michalski-Mozetic, 1986).

Podmienka v produkčnom pravidle, t.j. forma špecifikácie hodnoty atribútu sa nazýva selektor a má tvar:

$\mathbf{A_i}$ # $\mathbf{R_i}$ kde: $\mathbf{R_i}$ je disjunkcia hodnôt atribútu $\mathbf{A_i}$
označuje jeden z operátorov ”=” alebo ”≠”

Na ilustráciu sú uvedené dve ukážky selektorov:

Tvorivosť = slabá OR žiadna
Inteligencia ≠ priemerná.

Metódy generujúce popis klasifikačnej triedy v tvare disjunkcie typicky využívajú princíp pokrývania. Pri uvedenom zápise selektorov príklad je pokrytý nejakým výrazom práve vtedy, ak pre každý selektor výrazu hodnota atribútu príslušného selektora príkladu patrí do množiny hodnôt atribútu daného selektora výrazu. Inak povedané, každý selektor výrazu v sebe zahŕňa príslušný selektor príkladu. Napríklad:

Tvorivosť = výnimočná & Inteligencia = nadpriemerná & Pracovitosť = lenivý
je pokrytý výrazom

Tvorivosť = výnimočná & Inteligencia ≠ priemerná
nie je pokrytý výrazom

Tvorivosť ≠ priemerná & Pracovitosť = usilovný.

3.2.2.1 Algoritmus AQ11

Algoritmus AQ11 uvedený v (Mach, 1997) využíva princíp pokrývania. Bol navrhnutý Michalskim už v roku 1969. Pracuje s pojmom **obálka**. Napríklad obálka **G(e1/e2)** predstavuje logický výraz, ktorý pokrýva príklad „**e1**“ a zároveň nepokrýva príklad „**e2**“. Podobne obálka **G(e/E)** predstavuje výraz, ktorý pokrýva príklad „**e**“ a zároveň nepokrýva ani jeden príklad z množiny **E** a obálka **G(E1/E2)** pokrýva všetky príklady z množiny **E1** a zároveň ani jeden príklad z množiny **E2**. V súvislosti s algoritmom AQ11 je zaužívaný pojem kontrapríklad. Kontrapríklad je negatívny príklad, ktorý sa porovnáva s pozitívnym príkladom za účelom vylúčenia niektorých hodnôt atribútov z hľadaného popisu triedy

Tento algoritmus sa snaží o perfektnú klasifikáciu všetkých príkladov z trénovacej množiny. Inými slovami, hľadá popisy klasifikačných tried takým spôsobom, aby popis konkrétnej triedy pokrýval všetky príklady danej triedy a nepokrýval ani jeden príklad inej triedy.

AQ11 rozdeľuje problém na podproblémy, keďže je aplikáciou princípu „rozdeľuj a panuj“. Tieto podproblémy predstavujú jednotlivé kroky algoritmu:

Nájdenie popisu jedného (napríklad prvého) pozitívneho príkladu voči jednotlivému kontrapríkladu. Tento pozitívny príklad sa zvykne označovať ako jadro popisu hľadanej triedy. Príkladom takého popisu môže byť obálka **G(e1,e2)**.

Nájdenie popisu jadra voči kontrapríkladom ako množine, napríklad obálky **G(e/E)**. Kontrapríklady plnia úlohu ohraničení výsledného popisu, žiadny z nich nesmie byť pokrytý nájdeným popisom. Tento popis okrem jadra môže zahŕňať aj iné príklady popisovanej triedy. (Krok 2) zodpovedá volaniu podprogramu v algoritme NSC.)

Nájdenie popisu množiny všetkých príkladov patriacich do danej triedy voči množine všetkých kontrapríkladov, **G(E1/E2)**. Tento popis už predstavuje hľadaný popis triedy. Týmto popisom sú pokryté všetky príklady popisovanej triedy a nie je pokrytý ani jeden príklad kontratriedy.

E1 a **E2** sú navzájom disjunktné množiny typických príkladov. Na základe týchto množín indukujeme produkčné pravidlá, ktoré korektne klasifikujú všetky trénovacie príklady do správnych tried. Z vyššie uvedeného vyplýva, že **e1 (e2)** je prvkom množiny **E1 (E2)**.

Nech všetky príklady z množiny **E1** patria do triedy **T1** a všetky príklady z množiny **E2** patria do triedy **T2**. Potom algoritmus pre generovanie produkčných pravidiel na klasifikáciu do triedy **T1** bude pozostávať z nasledovných krokov:

1.krok:

Z množiny **E1** sa náhodne vyberie jeden príklad $\mathbf{e_i}$. Vygenerujú sa postupne obálky $\mathbf{G(e_i/e_j}$), kde $\mathbf{e_j}$ je kontrapríklad z množiny **E2**.

Elementárnu obálku $\mathbf{G(e_i/e_j)}$ vytvoríme nasledovným spôsobom. Najprv nájdeme všetky atribúty, ktoré majú rozličné hodnoty v príkladoch $\mathbf{e_i}$ a $\mathbf{e_j}$. Pre tieto atribúty sa generuje selektor v tvare $\mathbf{A_j \neq v_j}$, kde $\mathbf{v_j}$ je hodnota, ktorú nadobúda $\mathbf{A_j}$ v kontrapríklade $\mathbf{e_j}$. Celkovú obálku $\mathbf{G(e_i/e_j)}$ dostaneme ako logický súčet všetkých selektorov

$$G(e_i / e_j) = \bigcup_{j=1}^{n} (A_j \neq v_j)$$

kde **n** je počet tých atribútov, ktoré majú rôzne hodnoty v príkladoch $\mathbf{e_i}$ a $\mathbf{e_j}$.

Inak povedané, generuje sa maximálna oblasť pokrývajúca príklad a nepokrývajúca kontrapríklad. Počet selektorov je maximálne počet všetkých atribútov (príklady sa líšia hodnotou každého atribútu) a minimálne jeden (aby boli príklady rôzne musia sa líšiť aspoň hodnotou jedného atribútu).

2.krok:

Vytvorí sa celková obálka $\mathbf{G(e_i/E2)}$ ako logický súčin všetkých elementárnych obálok $G(e_i/e_j)$

$$G(e_i / E2) = \bigcap_{e_j \in E2} G(e_i / e_j)$$

Pri zjednotení použijeme absorbčný zákon, vďaka čomu dostaneme neredundantný logický výraz.

3.krok:

Z množiny **E1** sa vylúčia všetky príklady, ktoré sú pokryté obálkou vygenerovanou v kroku 2. Ak **E1** neostane potom prázdne, opäť sa pokračuje krokom 1. Ak je množina **E1** už prázdna, pokračuje sa krokom 4.

4.krok:

Zo všetkých obálok z kroku 2 sa vytvorí disjunkciou celková obálka **G(E1/E2)**.

$$G(E1 / E2) = \bigcup_{i=1}^{m} G(e_i / E2)$$

Táto obálka predstavuje klasifikačné pravidlá pre zaradenie typických príkladov do triedy **T1**, pričom **m** je počet rôznych obálok, vygenerovaných v kroku 2.

Vstupy: ***E1****...množina pozitívnych trénovacích príkladov*
E2*...množina negatívnych trénovacích príkladov*
Výstup: ***G(E1/E2)****...popis triedy v tvare DNF*
Volanie na najvyššej úrovni: ***AQ11(E1, E2, {})***

Procedúra: ***AQ11(E1, E2, G(E1/E2))***
for *každý príklad* $\boldsymbol{e_i}$ *v* ***E1***
for *každý príklad* $\boldsymbol{e_j}$ *v* ***E2***
generuj $G(e_i / e_j) = \bigcup_{j=1}^{n} (A_j \neq v_j)$
generuj $G(e_i / E2) = \bigcap_{e_j \in E2} G(e_i / e_j)$
na $\boldsymbol{G(e_i/E2)}$ *aplikuj absorbčný zákon*
z ***E1*** *vymaž všetky príklady pokryté* $\boldsymbol{G(e_i/E2)}$
if ***E1****={}*
then $G(E1 / E2) = \bigcup_{i=1}^{m} G(e_i / E2)$
end

Ten istý postup sa môže použiť na generovanie celkovej obálky druhej triedy. Stačí vymeniť množiny **E1** a **E2**. Algoritmus AQ11 je možné použiť aj na generovanie produkčných pravidiel pre klasifikáciu príkladov do viacerých tried. Jedinou podmienkou použitia algoritmu v takom prípade je rozdelenie všetkých trénovacích príkladov do dvoch množín **E1** (množina pozitívnych príkladov) a **E2** (množina negatívnych kontrapríkladov). Majme dané klasifikačné triedy **T1, T2, ... ,Tn**. Potom pri generovaní pravidla pre klasifikáciu do triedy **Ti** budú množinu **E1** tvoriť všetky typické príklady klaskifikované do triedy **Ti** a množinu **E2** všetky ostatné trénovacie príklady.

Môže sa stať, že sa obálky rôznych tried budú prekrývať v mieste, pre ktoré nebol k dispozícii žiaden trénovací príklad. Ak v budúcnosti príde príklad, prislúchajúci miestu prekrytia, bude podľa generovaných produkčných pravidiel klasifikovaný do viacerých tried. Túto nevýhodu je možné kompenzovať nasledovným postupom: obálka triedy **T2** sa negeneruje voči príkladom z **E1**, ale voči obálke týchto príkladov. To zabezpečí disjunktnosť obálok pre jednotlivé triedy. Tento postup má však jednu nevýhodu. Obálky neskoršie generovaných tried budú obopínať svoje príklady stále tesnejšie.

Pri generovaní obálky nie sú všetky trénovacie príklady rovnako dôležité. Najvýznamnejšie sú okrajové príklady. Z dôvodu zvýšenia rýchlosti je možné vstupnú množinu príkladov predspracovať s cieľom výberu reprezentatívnych okrajových príkladov.

Oproti algoritmom generujúcim rozhodovací strom má niektoré výhody, ako napríklad:

- nevyžaduje vzájomnú nezávislosť jednotlivých typických príkladov, keďže nepoužíva pravdepodobnosti
- nevykoľají ho redundantnosť príkladov
- je už v princípe inkrementálnej povahy.

3.3 Úlohy na precvičenie:

1. Ako sa dá graficky znázorniť popis pojmu reprezentovaný DNF t.j. disjunktívnou normálnou formou?
2. Charakterizujte algoritmus NSC.
3. Ktorá z nasledovných obálok môže byť výsledkom algoritmu AQ11?
   a/ $G(e_2,E_2)$
   b/ $G(E_1,E_2)$
   c/ $G(e_7,e_2)$.
4. Aké sú podľa Vás aplikačné možnosti generovaných produkčných pravidiel?
5. Majme danú nominálnu doménu ľudí z úlohy na precvičenie číslo 7. v kapitole „Generovanie logických konjunkcií“. Pomocou týchto trénovacích príkladov nájdite NDF pre triedu “-“. Použite algoritmus NSC a v ňom vnorený podprogram HGS. Použite ohodnocovaciu funkciu $S_{core}=P_c-6N_c$ a BS=2.
6. Majme danú numerickú doménu s dvoma atribútmi X a Y s nasledovnými pozitívnymi: $e_2=[4,3]$, $e_3=[1,2]$ a negatívnymi trénovacími príkladmi: $e_1=[2,4]$, $e_4=[5,2]$ a $e_5=[3,1]$. Generujte produkčné pravidlá pre triedu “-“ pomocou algoritmu AQ11.
7. Navrhnite produkčné pravidlá na klasifikáciu skúšky z predmetu strojové učenie.

3.4 Literatúra:

Arbab, B., Mitchie, D.: Generating rules from examples. In: Proc. of the 9th Int. Joint Conference on Artificial Intelligence. Morgan Kaufmann. Los Angeles, 1985, pp. 631-636.

Langley, P.: Elements of Machine Learning. Morgan Kaufmann Publishers, Inc. San Francisco, California, 1996, 419 pp.

Mach, M.: Získavanie znalostí pre znalostné systémy. Vydavateľstvo ELFA s.r.o., Košice, 1997, 104 pp.

Michalski, R.S., Chilausky, R.L.: Learning by being told and learning from examples: An experimental comparison of two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. International Journal of Policy Analysis and Information Systems, 4, 1980.

Michalski, R.S., Mozetic, I, Hong, J., Lavrac, N.: The multipurpose incremental learning system AQ15 and its testing application to three medical domains. Proc. of the Fifth National Conference on Artificial Intelligence, Philadelphia, PA: Morgan Kaufmann, 1986, pp. 1041-1045.

Svátek, V.: Automatic knowledge base construction using expert knowledge and data. Kybernetika a umelá inteligencia, Herľany, 1994.

4 ROZHODOVACIE STROMY

4.1 Reprezentácia a použitie rozhodovacích stromov

Rozhodovací strom je reprezentácia pojmu, ktorá je veľmi blízka reprezentácii pojmu produkčným pravidlom. Napokon je veľmi ľahké transformovať rozhodovací strom na produkčné pravidlá. Rozhodovací strom má výhodu v tom, že veľmi názorne ilustruje proces učenia. Preto je táto reprezentácia vhodná na riešenie úloh, ktoré vyžadujú spoluprácu s odborníkmi iných vedných odborov a teda laikmi v strojovom učení. Používajú sa v znalostných systémoch na automatické generovanie báz znalostí, v objavovaní znalostí a ďalších oblastiach.

Rozhodovací strom predstavuje reprezentáciu rozhodovacej procedúry (Quinlan,1990) pre klasifikáciu príkladov do príslušných tried. Je to grafová štruktúra vo forme stromu obsahujúca koreňový, medziľahlé a listové uzly. Uzly reprezentujú triedu alebo testovací atribút. Hrany rozhodovacieho stromu reprezentujú hodnoty testovacieho atribútu. Z každého uzla vychádza toľko hrán, koľko hodnôt má testovací atribút v danom uzle.

Na začiatku generovania rozhodovacieho stromu sa nachádzajú všetky trénovacie príklady, (celá trénovacia množina) v koreňovom uzle. Príklad trénovacej množiny z medicínskej oblasti je uvedený na Obr. 15.

Pre každý nelistový uzol (koreňový a medziľahlé uzly) sa vyberie najvhodnejší atribút. Hovoríme, že každý nelistový uzol reprezentuje špecifický test. V rámci medziľahlých uzlov sa trénovacia množina člení na podmnožiny resp. strom sa vetví na podstromy pre každý výsledok špecifického testu, t.j. pre každú hodnotu zvoleného atribútu. Vo všeobecnosti test môže byť aj niečo zložitejšie (napríklad LTU). Inak povedané každý nelistový uzol obsahuje testovaciu podmienku, ktorá rozdeľuje priestor príkladov podľa možného výsledku testu.

Každý listový uzol rozhodovacieho stromu reprezentuje klasifikačnú triedu.

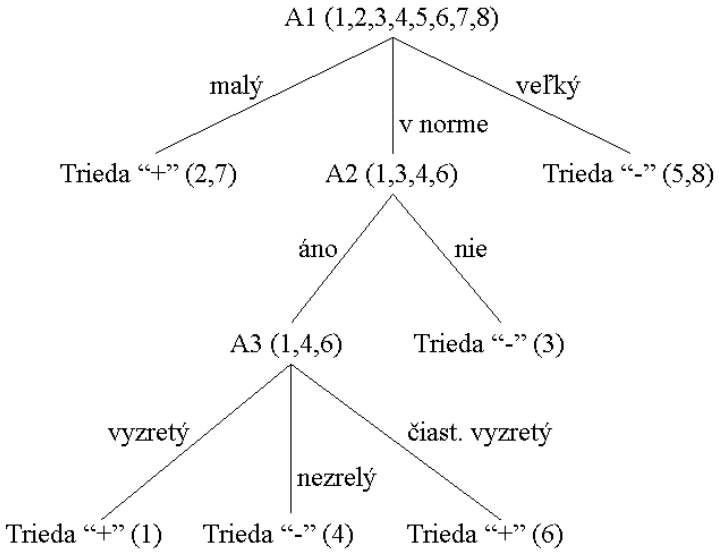
Z vyššie uvedeného vyplýva, že pre definíciu rozhodovacieho stromu je potrebné určiť:

- pre listový uzol názov triedy, do ktorej budú klasifikované všetky typické príklady, asociované s daným listovým uzlom
- pre nelistový uzol testovací atribút, podľa ktorého sa rozhodovanie vetví na jednoduchšie podstromy pre každú hodnotu zvoleného testovacieho atribútu.

Z trénovacej množiny znázornenej na Obr. 15 je možné vygenerovať (v tomto prípade algoritmom ID3) rozhodovací strom na Obr. 16, pričom A1 – je prírastok hmotnosti ženy, A2 – je zistený patologický prietok u dieťaťa a A3 – je typ placenty. Čísla v uzloch rozhodovacieho stromu sú čísla trénovacích príkladov.

Por.č.	A1	A2	A2	T
1.	v norme	áno	vyzretý	+
2.	malý	áno	vyzretý	+
3.	v norme	nie	vyzretý	-
4.	v norme	áno	nezrelý	-
5.	veľký	nie	vyzretý	-
6.	v norme	áno	čiast. vyzretý	+
7.	malý	áno	čiast. vyzretý	+
8.	veľký	nie	čiast. vyzretý	-

Obr. 15: Množina trénovacích príkladov z oblasti prenatálnej medicíny.



Obr. 16: Rozhodovací strom vygenerovaný algoritmom ID3.

Každý trénovací príklad je asociovaný s jedným listovým uzlom. Preto je možné rozhodovací strom prepísať do sady produkčných pravidiel. Generovaním produkčných pravidiel z rozhodovacieho stromu sa zaoberá aj (Quinlan, 1987). Klasifikačné, resp. produkčné pravidlá je možné získať nasledovne:

Každá cesta v rozhodovacom strome od jeho koreňa po niektorý listový uzol predstavuje jedno produkčné pravidlo

Ľavá strana, t.j. podmienková časť pravidla bude obsahovať všetky testovacie podmienky v nelistových uzloch na danej ceste v tvare: $\mathbf{A_i=a_i}$, pričom každý testovací atribút $\mathbf{A_i}$ bude nadobúdať hodnotu $\mathbf{a_i}$ prislúchajúcu vybratej vetve.

Pravá strana pravidla bude obsahovať názov triedy zodpovedajúcej listovému uzlu, v ktorom cesta končí. Do tejto triedy bude zaradený každý trénovací príklad, spĺňajúci podmienky ľavej strany pravidla.

Napríklad rozhodovací strom na Obr. 16 je možné prepísať do množiny pravidiel pre triedu **(+)**, teda pre patologický plod:

AK	A1=malý	POTOM trieda (+)
AK	A1=v norme & A2=áno & A3=vyzretý	POTOM trieda (+)
AK	A1=v norme & A2=áno & A3=čiast vyzretý	POTOM trieda (+)

Podobne je možné generovať z rozhodovacieho stromu pravidlá pre triedu**(-)**, keď nejde o patologický plod.

Klasifikácia nového prílkadu sa potom uskutočňuje nasledovne:

Ak nový príklad spĺňa všetky testy na ceste v strome od koreňa po listový uzol (podmienky podmienkovej časti pravidla), potom je daný príklad zaradený do triedy v listovom uzle (v závere pravidla).

4.2 Indukcia rozhodovacích stromov

Algoritmy tejto skupiny sa zvyknú označovať aj ako algoritmy induktívneho učenia (Russell-Norvig, 1995), (Michalski-Stepp, 1983), (Quinlan, 1986). Zameriavajú sa na generovanie rozhodovacieho stromu. Indukovanie rozhodovacieho stromu je taktiež ukážkou prístupu „rozdeľuj a panuj“, pri ktorom sa úlohy delia na podúlohy (indukovanie podstromov).

Algoritmy pre generovanie rozhodovacích stromov sú typicky založené na princípe budovania stromov zhora nadol. Pri tomto postupe je pôvodný priestor príkladov (trénovacia množina) delený na jednotlivé podpriestory, charakterizované hodnotami testovacích atribútov. Toto delenie sa uskutočňuje rekurzívne, až kým nie je splnená ukončovacia podmienka. Väčšina algoritmov na generovanie rozhodovacieho stromu končí indukciu, keď sa v každom listovom uzle nachádzajú iba príklady jednej triedy. To sú algoritmy snažiace sa o perfektnú klasifikáciu. Môže byť definované aj iné kritérium, napríklad: keď podpriestor obsahuje aspoň. 80% príkladov tej istej triedy, stáva sa listovým uzlom a daná vetva sa ďalej nerozvíja. V prípade, že už boli

vyčerpané všetky atribúty a v niektorom podpriestore sa stále nepodarilo splniť ukončovacie kritérium, je množina atribútov nedostatočná, alebo pôvodná množina príkladov protirečivá.

Všeobecný postup generovania rozhodovacieho stromu zhora nadol môže byť nasledovný:

1) Ak pre každý podpriestor je splnené ukončovacie kritérium, generovanie sa ukončí.
2) Inak:
3) Zvolí sa podpriestor obsahujúci príklady klasifikované do viacerých tried.
4) Pre zvolený podpriestor sa vyberie jeden testovací atribút, ešte nepoužitý pre daný podpriestor príkladov.
5) Zvolený podpriestor príkladov sa rozdelí na ďalšie podpriestory podľa hodnôt zvoleného testovacieho atribútu.

Krok 1) zodpovedá prípadu, keď generovanie rozhodovacieho stromu je už ukončené. Krok 2) pokračuje v generovaní, keď expanduje jeden listový uzol o jednu úroveň.

Ešte je potrebné zodpovedať otázku, ako voliť testovacie atribúty v kroku 2b). Pri vhodnej voľbe bude rozhodovací strom minimálny. Na druhej strane, nevhodná voľba testovacích atribútov môže značne zväčšiť rozmery generovaného stromu. Konkrétne algoritmy môžu používať rôzne spôsoby výberu testovacieho atribútu. Preto sa odpoveď na nastolenú otázku nachádza v popisoch konkrétnych algoritmov.

Rozhodovacie stromy sa dnes používajú v mnohých systémoch. Jedným z prvých bol systém EPAM (Elementary Perceiver and Memorizer) publikovaný v (Feigenbaum, 1961). Tento systém bol považovaný za kognitívno-simulačný model učenia pojmov človekom. Ďalší model CLS (Hunt at al., 1966), používa pri generovaní rozhodovacieho stromu heuristický prístup (Look ahead Method). V Huntovom učení pojmov (Hunt's Concept Learning) má pôvod aj známy algoritmus C4.5, o ktorom pojednáva jedna z nasledujúcich podkapitol.

4.2.1 Algoritmus ID3

Najznámejším algoritmom generujúcim rozhodovací strom metódou zhora nadol je algoritmus **ID3 (Iterative Dichotomizer 3)**. Voľne by sa dal názov algoritmu preložiť ako Iteratívny dvojtriedny klasifikátor. Bol vyvinutý Rossom Quinlainom v 1979 v doméne znalostí šachistu o hraní šachových koncoviek.

Ukončovacie kritérium tohto algoritmu je, že každý podpriestor obsahuje iba príklady jednej triedy. Ak je množina atribútov dostatočná, možno uvedeným postupom zostrojiť rozhodovacie stromy, korektne klasifikujúce trénovacie príklady. Hovoríme, že rozhodovací strom generovaný algoritmom je:

- **perfektný** keď správne klasifikuje všetky trénovacie príklady
- **rozmerovo optimálny resp. minimálny** keď je rozhodovacia procedúra čo najjednoduchšia, s čo najmenším počtom testov hodnôt atribútov.

Algoritmus ID3 pre výber testovacieho atribútu využíva Shannonovu teóriu informácie (Shannon-Weaver, 1949), ktorá na meranie množstva informácie používa entrópiu. Claude

Shannon objavil informačnú teóriu, ktorá spôsobila revolúciu v oblasti komunikácie a taktiež prispela k úspechu učenia pomocou rozhodovacích stromov.

Ak správy $\mathbf{x_1, x_2, ... , x_n}$ sú možné s pravdepodobnosťami $\mathbf{p(x_1), p(x_2), ... , p(x_N)}$ a tieto vytvárajú úplný súbor pravdepodobností, teda:

$$\sum_{j=1}^{N} p(x_j) = 1$$

potom entrópiu, resp. neurčitosť daného súboru správ možno vyjadriť nasledovne:

$$H = -\sum_{j=1}^{N} p(x_j) \log_2 p(x_j)$$

Tento vzťah je známy ako Shannonova entrópia. Čím je entrópia súboru správ vyššia, tým menej určitý je obsah budúcej správy a tým väčšie množstvo informácie získame, keď správu prijmeme.

Rozhodovací strom môže byť považovaný za zdroj informácie, ktorá pre nejaký konkrétny príklad formuje správu o klasifikácii daného príkladu do nejakej triedy. Keď niektorý uzol stromu obsahuje iba príklady tej istej triedy, entrópia v tomto uzle je nulová, pretože klasifikačné rozhodnutie pre príklady prislúchajúce tomuto uzlu je pevne dané. (Teda pravdepodobnosť jednej triedy je 1 a ostatných nulová.)

Z toho vyplýva, že entrópia v listových uzloch je nulová, zatiaľ čo v koreňovom uzle a medziľahlých uzloch má nenulovú hodnotu. Pre vygenerovanie minimálneho rozhodovacieho stromu by entrópia mala čo najrýchlejšie klesnúť na nulovú hodnotu. Algoritmus ID3 používa heuristiku, ktorá usiluje o čo najvyšší pokles entrópie lokálne v každom kroku.

Nech uzol **S** obsahuje $\mathbf{n_1}$ príkladov klasifikovaných do triedy **T1** a $\mathbf{n_2}$ príkladov zaradených do triedy **T2**. Potom pravdepodobnosť, že nejaký príklad prislúchajúci uzlu **S** rozhodovacieho stromu bude klasifikovaný do triedy **T1** alebo **T2** je

$$\frac{n_1}{n_1 + n_2} \quad resp. \quad \frac{n_2}{n_1 + n_2}.$$

Teda entrópia v danom uzle S, presnejšie povedané entrópia súboru trénovacích príkladov prislúchajúcich uzlu S, bude určená nasledovne:

$$H(S) = -\sum_{j=1}^{2} \frac{n_j}{n_1 + n_2} \log_2 \frac{n_j}{n_1 + n_2}.$$

Predpokladajme, že je možné v uzle **S** použiť atribút **A** s hodnotami $\mathbf{a_1}$ a $\mathbf{a_2}$ pre rozdelenie príkladov z uzla **S** do dvoch disjunktných podmnožín **S1** a **S2**. Ak v uzle **S** má $\mathbf{m_1}$ príkladov

hodnotu atribútu **A** rovnú $\mathbf{a_1}$ a $\mathbf{m_2}$ príkladov má hodnotu $\mathbf{a_2}$, potom celková entrópia v uzle **S** s použitím atribútu **A** sa určí ako:

$$H(S,A) = \sum_{j=1}^{2} \frac{m_j}{m_1 + m_2} H(S_j).$$

Informácia získaná použitím atribútu **A** v uzle **S** je označovaná ako informačný zisk a počíta sa nasledovne:

$$I(A) = H(S) - H(S,A).$$

V každej iterácii sa v kroku 2b) algoritmu pre generovanie rozhodovacieho stromu metódou zhora nadol vyšetrujú všetky ešte nepoužité atribúty a ako testovací atribút sa vyberá ten, ktorý maximalizuje informačný zisk resp. minimalizuje entrópiu t.j. neurčitosť. Stratégiou algoritmu ID3 je čo najrýchlejší pokles entrópie na nulu a tak generovanie minimálneho rozhodovacieho stromu.

Algoritmus ID3 je možné jednoducho rozšíriť aj na prípad, keď príklady budú klasifikované do viacerých tried a keď atribúty môžu nadobúdať viac ako dve hodnoty. Ak jednotlivé triedy budú **T1, T2, ..., TN**, potom vzťah pre výpočet entrópie uzla **S** bude mať tvar:

$$H(S) = -\sum_{j=1}^{N} p(T_j) \log_2 p(T_j)$$

kde $\mathbf{p(T_j)}$ je pravdepodobnosť toho, že nejaký príklad, patriaci uzlu **S**, bude klasifikovaný do triedy $\mathbf{T_j}$. Ak atribút **A** môže nadobúdať hodnoty $\mathbf{a_1, a_2, ..., a_m}$, potom vzťah pre výpočet neurčitosti uzla **S** s testovacím atribútom **A** bude mať tvar:

$$H(S,A) = \sum_{j=1}^{m} p(a_j) H(S_j)$$

kde $\mathbf{p(a_j)}$ je pravdepodobnosť toho, že nejaký príklad, patriaci uzlu **S** má hodnotu atribútu $A=a_j$.

Pre použitie algoritmu musí platiť podmienka neprotirečivosti (nekontradikčnosti) trénovacích príkladov. Nutnosť platnosti tejto podmienky vyplýva z toho, že ID3 nie je odolný voči zašumeným údajom na vstupe. Zo spôsobu určovania pravdepodobností vyplýva, že iba dodržanie dvoch dodatočných podmienok, a to podmienky neredundantnosti príkladov (nepovolenie viacnásobného výskytu trénovacieho príkladu v trénovacej množine) a podmienky vzájomnej nezávislosti atribútov, garantuje nájdenie minimálneho rozhodovacieho stromu.

Jednou z modifikácií algoritmu ID3 je program uvedený v (Štepánková-Hetmerová-Kraus,1998), ktorý je využívaný v lekárstve na spracovanie a vyhodnotenie EEG, konkrétnejšie kmeňových sluchových evokovaných potenciálov. V tomto prípade modifikácia spočíva v zavedení diskretizácie hodnôt atribútov, optimálnej hĺbky stromu a v spolupráci s jednotkou zhlukovania.

4.2.2 Algoritmus ID5R

Pomerne známy je aj algoritmus **ID5R (Inductive Dichotomizer 5 Recursive)**, ktorý predstavuje inkrementálnu modifikáciu algoritmu ID3. Využíva sa v prípadoch, keď nie všetky trénovacie príklady sú k dispozícii naraz, ale stávajú sa známymi postupne (odoberanie z reálneho procesu v určitých časových okamihoch). Ide o riešenie učiacej úlohy typu on-line. Niekedy nie je možné čakať, až budú známe všetky trénovacie príklady a teda je potrebné generovať rozhodovací strom na základe tých príkladov, ktoré už sú známe. Po príchode ďalších trénovacích príkladov je možné už vygenerovaný rozhodovací strom aktualizovať.

Pre túto úlohu je samozrejme možné použiť aj niektorý neinkrementálny algoritmus, napr. ID3. Toto riešenie má však niekoľko nevýhod:

- pri príchode každého nového príkladu je potrebné realizovať indukciu opäť od začiatku
- pri novej indukcii sa nevyužívajú informácie získané v predchádzajúcich indukciách
- počet príkladov neustále narastá, čím sa predlžuje doba potrebná pre opätovné vygenerovanie rozhodovacieho stromu.

Preto je vhodné využívať inkrementálnu indukciu, ktorá po príchode nového príkladu inicializuje iba modifikáciu už existujúceho stromu. Takúto inkrementálnu indukciu vykonáva aj algoritmus ID5R. Aj tento algoritmus, podobne ako ID3, využíva maximalizáciu informačného zisku pre výber testovacieho atribútu. Na rozdiel od ID3 si uchováva v každom uzle rozhodovacieho stromu dostatočnú informáciu na to, aby mohol zistiť, či vybraný testovací atribút skutočne maximalizuje informačný zisk, alebo či je potrebné ho nahradiť iným atribútom. Pri nutnosti zmeny testovacieho atribútu mu uchovávaná informácia umožňuje nájsť nový atribút a všetky podstromy reštrukturalizovať podľa nového stavu, zmeneného príchodom nového trénovacieho príkladu. Podrobnejšie o inkrementálnej indukcii rozhodovacích stromov pojednáva (Utgoff-Bradly, 1991).

4.2.3 Algoritmus C4.5

Ďalším zaujímavým algoritmom tejto skupiny je algoritmus **C4.5** (Quinlan, 1993), ktorý navrhol Ross Quinlan. Ide vlastne o vylepšený a doplnený algoritmus ID3. Algoritmus C4.5 rozlišuje dva typy atribútov: nominálne a reálne. Používa pomenovanie diskrétne (nominálne) a spojité (reálne) atribúty. Ak je potrebné používať iné typy atribútov, je potrebné ich vhodne pretransformovať na jeden z týchto atribútov. Je to neinkrementálny algoritmus, budujúci strom zhora nadol. Tento algoritmus je založený na metóde "Rozdeľuj a panuj" panuje nad a rozdeľuje množinu **M** trénovacích príkladov. Predpokladajme, že trénovacie príklady majú byť klasifikované do tried $\mathbf{T_1, T_2, ...T_K}$.

Potom môžu nastať tri prípady:

- Množina **M** obsahuje jeden alebo viac príkladov patriacich do tej istej triedy $\mathbf{T_i}$. Rozhodovací strom pre množinu **M** je listový uzol s triedou $\mathbf{T_i}$.

- Množina **M** neobsahuje žiadne príklady. Rozhodovací strom je opäť listový uzol, ale triedu nie je možné určiť z množiny **M**. Môže byť určená napr. ako majoritná trieda zo všetkých trénovacích príkladov, alebo ako majoritná trieda v rodičovskom uzle.
- Množina **M** obsahuje príklady patriace do viacerých tried. Vyberie sa test vedúci k rozdeleniu množiny **M** do podmnožín, obsahujúcich príklady s rovnakou hodnotou testovacieho atribútu. Testovacia podmienka predstavuje atribút, podľa hodnôt ktorého $\{\mathbf{a_1}, \ldots \mathbf{a_n}\}$ sa **M** rozdelí do podmnožín $\mathbf{M_1}, \mathbf{M_2}, \ldots \mathbf{M_n}$, kde $\mathbf{M_i}$ obsahuje všetky príklady z **M**, vyhovujúce podmienke s hodnotou atribútu $\mathbf{a_i}$. Rozhodovací strom sa v tomto prípade skladá z rozhodovacieho uzla s testovacím atribútom a jednou podmienkou pre každú hodnotu atribútu. Tento postup je opakovaný rekurzívne pre každú vetvu rozhodovacieho uzla.

Algoritmus C4.5 sa od ID3 líši okrem iného aj druhým prípadom, keď množina M neobsahuje žiadne príklady. Tento algoritmus vytvára perfektný rozhodovací strom za predpokladu neprotirečivosti trénovacích príkladov, podobne ako algoritmus ID3. Existencia protirečivých príkladov môže signalizovať nedostatočné množstvo atribútov, a teda nie dosť precízny popis príkladov v údajovej časti. V prípade protirečivých príkladov je možné ukončovacie kritérium, predpokladajúce klasifikáciu všetkých príkladov v listovom uzle do jednej triedy upraviť tak, aby majoritná trieda v listovom uzle tvorila aspoň 85%.

4.2.3.1 Pomerové kritérium zisku

Algoritmus C4.5 používa na výber testovacej podmienky (testovacieho atribútu) kritérium zisku, ktoré je založené na informačnej teórii, podobne ako algoritmus ID3. V takom tvare, ako ho používa ID3, však toto kritérium zisku uprednostňuje výber testovacích atribútov s väčším počtom hodnôt. Tým vytvára určitý druh prehľadávacej preferencie. Ak sa v trénovacej množine nachádza diskrétny atribút s rôznymi hodnotami pre každý príklad, má najväčšiu šancu pre výber do testovacej podmienky. Lenže to znamená delenie aktuálnej trénovacej množiny na podmnožiny s práve jedným príkladom. Takéto vetvenie je neefektívne. Uvedený nedostatok je možné odstrániť normalizovaním informačného zisku, t.j. zavedením pomerového kritéria zisku.

Nech sa uzol **S** použitím atribútu **A** rozvetví na **m** vetiev. Potom pomerová entrópia bude daná:

$$H_p(S, A) = -\sum_{j=1}^{m} p(a_j) \log_2 (p(a_j))$$

kde $\mathbf{p(a_j)}$ je pravdepodobnosť toho, že nejaký príklad patriaci uzlu **S** prešiel vetvou $\mathbf{a_j}$ do poduzla $\mathbf{S_j}$, keďže nadobúda pre atribút **A** hodnotu $\mathbf{a_i}$. Pomerová entrópia má tendencie narastať s rastúcim počtom vetiev. Potom pomerovým informačným ziskom, daným nasledovným vzťahom:

$$I_p(S, A) = \frac{I(S, A)}{H_p(S, A)},$$

sa obmedzí výber testovacej podmienky s mnohými výstupmi, čím sa vytvorí nový druh prehľadávacej preferencie. I(S,A) sa určí podľa vzťahu, ktorý na určenie informačného zisku používa algoritmus ID3. Pomerový informačný zisk vedie ku generovaniu rozmerovo menších rozhodovacích stromov.

4.2.3.2 Spojité atribúty

Testovacia podmienka pre spojité atribúty pozostáva z prahovej hodnoty, ktorá rozdeľuje usporiadanú množinu čísel na dve podmnožiny. Trénovacie príklady sú najprv usporiadané vzostupne podľa hodnôt daného spojitého atribútu. Vznikne usporiadaná množina hodnôt spojitého atribútu: $\{v_1, v_2, \ldots, v_m\}$.

Prahová hodnota ležiaca medzi dvoma hodnotami v_i a v_{i+1}, rozdelí množinu príkladov na množiny: $\{v_1, v_2, \ldots, v_i\}$, $\{v_{i+1}, v_{i+2}, \ldots, v_m\}$.Existuje **m-1** takýchto rozdelení. Pre všetky rozdelenia sa vypočíta ohodnocovacia funkcia a vyberie sa rozdelenie s maximálnym ohodnotením. Ohodnocovacou funkciou môže byť informačný zisk alebo pomerový informačný zisk. Prahová hodnota medzi dvoma hodnotami spojitého atribútu v_i a v_{i+1} sa určí ako ich priemerná hodnota.

4.2.3.3 Neznáme hodnoty atribútov

V reálnych údajoch sa často stáva, že niektoré hodnoty atribútov nie sú známe. Môžu chýbať z rôznych príčin. Z dôvodu zašumenia, nedôležitosti atribútu, nedbanlivosti vkladajúceho, alebo sú skutočne neznáme. Algoritmus generovania rozhodovacieho stromu je potrebné upraviť, aby bol schopný vysporiadať sa s takýmto neúplným vstupom.

Príklady trénovacej množiny s neurčenou triedou sú zbytočné (informačne nezaujímavé), preto sú pri spracovávaní odfiltrované. Odfiltrovať aj príklady s chýbajúcimi hodnotami atribútov by bolo najjednoduchšie, ale nie vždy to je vhodné. Do úvahy pripadá napríklad doplnenie chýbajúcich hodnôt najčastejšie sa vyskytujúcou hodnotou.

Algoritmus C4.5 nepredspracováva údaje pred začatím generovania rozhodovacieho stromu (Quinlan, 1989). Je schopný sa s tým vyrovnať počas samotného generovania.

K tomu je potrebné upraviť kritérium pre výber testovacej podmienky, a taktiež spôsob zaradzovania neúplných príkladov do jednotlivých uzlov. Entrópia v danom uzle **H(S)** sa vypočíta tým istým spôsobom ako predtým. K vypočítaniu **H(S,A)** sú uvažované iba príklady so známymi hodnotami uvažovaného atribútu. Modifikovaný informačný zisk sa vypočíta nasledovne:

$$I(S, A) = p(A_k)(H(S) - H(S, A)) - p(A_{unk})$$

kde: $\mathbf{p(A_k)}$ je pravdepodobnosť výskytu známych hodnôt atribútu a
$\mathbf{p(A_{unk})}$ je pravdepodobnosť výskytu neznámych hodnôt atribútu.

Podobne sa upraví pomerová entrópia:

$$H_p(S,A) = -\sum p(a_j)\log_2 p(a_j) - p(a_{unk})\log_2 p(a_{unk})$$

kde: $\mathbf{p(a_j)}$ je pravdepodobnosť toho, že príklad patriaci uzlu **S** prešiel vetvou $\mathbf{a_j}$ do poduzla $\mathbf{S_j}$. Pomerový informačný zisk ostáva v pôvodnej podobe.

Algoritmus C4.5 je z vyššie uvedených algoritmov generovania rozhodovacích stromov najčastejšie používaný v súčasnosti, napríklad aj v oblasti objavovania znalostí (Paralič-Andrássyová, 1999a), (Paralič-Andrássyová, 1999b).

4.3 Orezávanie rozhodovacích stromov

Keď sa pri generovaní rozhodovacieho stromu kladie dôraz na presnosť klasifikácie, môže sa stať, že rozhodovací strom bude preučený teda „overfitting“. Pri takomto strome sa dá zaručiť presná klasifikácia iba u trénovacích príkladov. Pri klasifikácii testovacích príkladov môže dôjsť k chybám. Rozhodovací strom totiž odráža aj náhodné súvislosti, nesúvisiace s oblasťou, spôsobené nedokonalým výberom údajov. Tieto problémy sa prejavia vznikom veľmi tenkých vetiev, ktoré nezriedka obsahujú iba jeden trénovací príklad. Taký strom je vhodné orezať teda skrátiť niektoré tenké vetvy. Je možné uviesť horný odhad dĺžky vetiev stromu, ktorý je zárukou požadovanej presnosti (Russell-Norwig, 1995). Vhodná dĺžka vetiev je menšia ako podiel logaritmov rozsahu trénovacej množiny a počtu uvažovaných atribútov. Toto číslo sa nazýva **optimálna hĺbka stromu** (Štepánková-Hetmerová-Kraus,1998).

Orezávanie rozhodovacieho stromu je jednou z možností, ako znížiť chybu klasifikácie a zároveň zmenšiť rozhodovací strom. Presnosť klasifikácie na učiacich údajoch sa síce zmenší, ale dá sa predpokladať, že sa zvýši presnosť na neznámych príkladoch.

Pri klasických metódach orezávania sa už vygenerovaný rozhodovací strom postupne zjednodušuje. Existujú metódy, ktoré zjednodušujú rozhodovací strom už počas jeho generovania. Budovanie a následné orezávanie rozhodovacieho stromu je síce pomalšie, ale hodnovernejšie a v neposlednom rade aj jednoduchšie, pretože je založené na náhrade podstromu listovým uzlom.

Techniky orezávania, podrobnejšie popísané aj v (Mach, 1997), sa delia podľa toho, či pracujú aj s testovacou, alebo iba s trénovacou množinou. V prvom prípade sa rozhodovací strom vygeneruje na základe trénovacej množiny a následnému orezaniu poslúži iná, testovacia množina. Za predpokladu dostatočného množstva dôveryhodných údajov vznikne týmto postupom kvalitnejšia rozhodovacia procedúra.

Príkladom takéhoto spôsobu orezávania rozhodovacieho stromu sú nasledovné techniky:

- Orezávanie cena-komplexnosť
- Redukcia chyby

Metóda **redukcie chyby** rozdistribuuje testovacie príklady do listových uzlov originálneho rozhodovacieho stromu. Potom pre každý nelistový uzol **S** stromu **T** sa preskúma ako by sa zmenil počet chybne klasifikovaných príkladov, ak by sa **S** nahradil jeho najlepším listovým uzlom. Ak nový strom klasifikuje s rovnakým, alebo menším počtom chýb a **S** neobsahuje žiadny podstrom s takouto vlastnosťou, potom je **S** nahradený daným listom. Tento proces sa ukončí vtedy, keď by už každá ďalšia náhrada zvýšila počet chybne klasifikovaných príkladov z testovacej množiny. Táto metóda predpokladá dokonalú testovaciu množinu, aby sa nestalo, že bude odstránená tá časť stromu, ktorá nebola zastúpená v testovacej množine.

Reprezentantom druhého typu techník orezávania, ktoré pracujú iba s trénovacou množinou, je **pesimistické orezávanie**. Táto technika odhaduje chybu klasifikácie. Používa pritom heuristickú metódu, ktorá vyžaduje počet všetkých trénovacích príkladov N a počet chybne klasifikovaných príkladov v danom uzle E. Algoritmus C4.5 rozširuje túto heuristiku o voľbu úrovne pesimistického orezávania CF. V ďalšom sa navrhuje prírastok predikovanej chyby klasifikácie U(E,N). Celková predikovaná chyba je daná súčtom prírastku chyby a počtu chybne klasifikovaných príkladov:

$$E' = E + U(E, N).$$

Predikovaná chyba nelistového uzla sa určí súčtom predikovaných chýb jeho poduzlov.
Orezávanie postupuje rozhodovacím stromom zdola nahor. Je možné nahradiť uzol:
listovým uzlom s triedou, ktorá sa najčastejšie vyskytuje v podstrome
alebo jedným z poduzlov v jeho vetvách.

Pre každú možnosť sa vypočíta predikovaná chyba uzla a porovnáva sa s predikovanou chybou uzla pred orezaním. Aplikuje sa tá možnosť, ktorou sa dosiahne menšia predikovaná chyba. Ak sa nahradí uzol poduzlom, poduzol môže byť taktiež orezaný v ďalšej iterácii. Keďže predikovaná chyba uzla sa môže len znížiť, výsledkom je menšia predikovaná chyba celého stromu. Výhodou tejto metódy je, že jej stačí trénovacia množina. Táto výhoda je veľmi cennou pri veľmi často sa vyskytujúcom nedostatku príkladov.

Existujú aj iné spôsoby ako zjednodušiť vygenerovaný rozhodovací strom, a to napríklad pomocou Booleanovských funkcií (Mařík-Štepánková-Lažanský, 1993). Optimalizácia rozhodovacích stromov je možná aj s využitím heuristického prístupu (Martelli-Montaneri, 1978). Zjednodušovaniu rozhodovacích stromov je venovaná aj práca (Quinlan, 1987b).

4.4 Technika malého okna

Pri spracovaní veľkých databáz sa môže ukázať operačná pamäť nedostatočnou. Tento problém môže vyriešiť nepriama metóda spracovania údajov, nazývaná technika malého okna (windowing). Táto metóda náhodne vyberie podmnožinu príkladov z množiny trénovacích príkladov, nazývanú „okno“. Z okna je nasledovne vygenerovaný rozhodovací strom. Vygenerovaným stromom sa klasifikujú príklady, ktoré nie sú zahrnuté v okne. Chybne

klasifikované príklady sa zaradia do okna, z ktorého sa opäť vygeneruje nový rozhodovací strom. To sa opakuje dovtedy, kým posledný vygenerovaný strom nebude správne klasifikovať všetky príklady mimo okna. Často sa stáva, že konečné okno obsahuje iba malú časť z celkovej množiny trénovacích príkladov.

V algoritme C4.5 sa nevyberajú príklady do okna náhodne, ale preferovaním výberu, ktorý vytvorí distribúciu tried v okne zhodnú s distribúciou tried v trénovacej množine. Algoritmus sa môže zastaviť skôr, než rozhodovací strom správne klasifikuje všetky príklady mimo okna. Napríklad, keď sa už neznižuje celková chyba klasifikácie, alebo ak narastie veľkosť okna nad obmedzujúcu hranicu. Včasné ukončenie môže v prípade zašumených údajov zabrániť neúprosnému rastu okna, ktoré smeruje k pohlteniu všetkých trénovacích príkladov.

Okrem možnosti obísť obmedzenie nedostatku pamäti, je použitie techniky malého okna výhodné pri redundancii príkladov. Na druhej strane, generovaním rozhodovacieho stromu v každom cykle sa predĺži celkový čas spracovania. Neočakávaným prínosom je vyššia presnosť ním vytvorených stromov. Je možné vygenerovať viacero rozličných stromov a následne vybrať najlepší vzhľadom na predikovanú chybu. Takto vygenerovaný strom je často presnejší, ako strom generovaný z celej trénovacej množiny. Ide to však na úkor času, ktorý s počtom generovaných stromov narastá.

4.5 Zoskupovanie hodnôt diskrétnych atribútov

Pri generovaní stromu klasickým spôsobom sa rekurzívne generujú podstromy pre každú hodnotu atribútu. Ak má niektorý atribút veľmi veľa hodnôt (rádovo desiatky) výsledný rozhodovací strom sa stane neprehľadným. Preto je potrebné atribúty s veľkým počtom hodnôt **diskretizovať**, teda nahradiť pôvodnú rozsiahlu novou menšou množinou diskrétnych hodnôt atribútov. Počet hodnôt môžu ovplyvniť aj požiadavky na tvar výsledného rozhodovacieho stromu. Môže byť požadovaná rôzna hĺbka a šírka stromu. Ak je požadovaný strom zrozumiteľný pre používateľa, potom je vhodné zaviesť menej ako osem rôznych diskrétnych hodnôt pre jeden atribút. Psychologické testy často definujú tento počet ako maximálny počet alternatív, ktoré je človek schopný alebo ochotný uvažovať.

Diskrétne hodnoty atribútov je možné zoskupovať viacerými spôsobmi (Quinlan, 1988). Jednou z metód je binárne rozdelenie s dvoma výslednými podmnožinami. Ďalším spôsobom je určenie všetkých rozdelení množiny hodnôt daného atribútu a výber najvhodnejšieho, podľa nejakého ohodnocovacieho kritéria. Tento prístup však vedie ku kombinatorickej explózii výpočtového procesu. Ďalšou možnosťou je využitie heuristiky (ako to robí algoritmus C4.5), ktorá je menej výpočtovo náročná, a ktorá pracuje nasledovným spôsobom:

- Najprv sa v rámci inicializácie vytvorí inicializačné rozdelenie, pozostávajúce z N jednoprvkových množín (práve toľko, koľko je hodnôt atribútu A). Pre toto rozdelenie sa vypočíta informačný zisk alebo pomerový informačný zisk.
- Vytvoria sa všetky možné rozdelenia zlúčením dvoch množín (všetky možné dvojice). Rozdelenia sa opäť ohodnotia skórovacou funkciou a vyberie sa najlepšie rozdelenie.

- Zlučovanie množín rekurzívne postupuje dovtedy, kým nie je dosiahnuté hraničné rozdelenie s dvoma množinami.
- Zo všetkých rozdelení sa vyberie výsledné rozdelenie s maximálnym ohodnotením.

Pri výbere testovacieho atribútu v konkrétnom uzle sa nielen vyberá ešte nepoužitý atribút, ale sa aj spúšťa vyššie uvedený algoritmus, ktorý určí najvhodnejšie zoskupenie hodnôt daného atribútu.

Zoskupovanie hodnôt atribútov výrazne zvýši celkový čas generovania rozhodovacieho stromu. Z hľadiska dĺžky výpočtu by bolo výhodnejšie vykonať zoskupovanie hodnôt atribútov už pred začiatkom generovania rozhodovacieho stromu. Tento prístup má však tiež svoju nevýhodu. Nedokáže podchytiť rozličné znalosti v rôznych častiach rozhodovacieho stromu.

4.6 Úlohy na precvičenie:

1. Aké používa algoritmus ID3 ukončovacie kritérium a aké kritérium na výber testovacej podmienky?
2. Ktorým algoritmom je možné spracovať spojité atribúty a akým mechanizmom?
3. Priraďte nasledovné: trieda T, trénovacia množina M a testovací atribút TA, jednotlivým uzlom rozhodovacieho stromu:
   koreňovému .....
   medziľahlému .....
   listovému .....
4. Nárast informačného zisku znamená:
   a/ nárast
   b/ pokles entrópie.
5. Čo je podstatou algoritmu MDPL tj. generovanie rozhodovacieho stromu s minimálnou dĺžkou popisu?
6. Vypočítajte aká bude entrópia listového uzla, ktorému odpovedá rovnaký počet trénovacích príkladov triedy „+“ aj triedy „-“
   H(S(x+, x-))=.....*)* a aká bude entrópia uzla obsahujúceho iba príklady jednej triedy?
   H(S(x+,0-))=H(S(0+,x-))=…….
7. Ako sa líši algoritmus ID5R od algoritmu ID3?
8. Čo spája techniky: orezávania RS a zoskupovania hodnôt diskrétnych atribútov?
9. Aký typ problému vyžaduje použitie techniky malého okna?
10. Majme danú nominálnu doménu ľudí z úlohy na precvičenie číslo 7. v kapitole „Generovanie logických konjunkcií“. Generujte rozhodovací strom z týchto trénovacích príkladov použitím algoritmu ID3 a MDPL. Porovnajte výsledky.

4.7 Literatúra

Feigenbaum, E.A.: The simulation of verbal learning behavior. Proc. of the Western Joint Computers Conference, 19, 1961, pp. 121-131.

Hunt, E.B., Marin, J., Stone, P.T.: Experiments in Induction. Academic Press, New York, 1966.

Mach, M.: Získavanie znalostí pre znalostné systémy. Vydavateľstvo ELFA s.r.o., Košice, 1997, 104 pp.

Martelli, A., Montaneri, U.: Optimizing decision trees through heuristically guided search. CACM, 21(12), 1978, pp. 1025-1039.

Mařík, V., Štepánková, O., Lažanský, J.: Umělá inteligence. ACADEMIA PRAHA, Praha, 1993, ISBN 80-200.0502-3.

Michalski, R.S., Stepp, R.E.: A Theory and Methodology of Inductive Learning. In: Michalski R.S., Carbonnell J.G., Mitchell T.M. (eds.): Machine Learning: An Artificial Intelligence Approach, Morgan Kaufmann, 1983.

Paralič, J., Andrássyová, E.: Knowledge Discovery in Databases. A comparison of different views. In Zborník Radova, Journal of information and organizational sciences, Vol. 23, No. 2, Varaždin, Croatia, 1999a, pp. 95-102.

Paralič, J., Andrássyová, E.: Intelligent Knowledge Discovery. Proc. of the 3rd IEEE International conference on Intelligent Engineering Systems INES'99, Stará Lesná, 1999b, pp. 307-310.

Quinlan, J.R.: Induction of decision trees. Machine Learning 1, 1, 1986, pp. 81-106.

Quinlan, J.R.: Generating production rules from decision trees. Proc. of the Tenth International Joint Conference on Artificial Intelligence, San Mateo, CA: Morgan Kaufmann, 1987a, pp. 304-307.

Quinlan, J.R.: Simplifying decision trees. International Journal of Man-Machine Studies 27, 1987b, pp. 221-234.

Quinlan, J.R.: Decision trees and multi valued attributes. In J.E. Hayes, D. Michie, and J. Richards (eds.), Machine Intelligence 11, Oxford, UK: Oxford University Press, 1988, pp. 305-318.

Quinlan, J.R.: Unknown attribute values in induction. Proc. of the Sixth International Machine Learning Workshop, San Mateo, CA: Morgan Kaufmann, 1989, pp. 164-168.

Quinlan, J.R.: Decision trees and decision-making. IEEE Transactions Systems, Man and Cybernetics 20, 2, 1990, pp. 339-346.

Quinlan, J.R.: C4.5 Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, California, 1993, ISBN 1-55860-238-0.

Russell S.J., Norvig P.: Artificial Intelligence. A Modern Approach. Prentice-Hall International, Inc., USA, 1995, ISBN 0-13-360124-2.k

Shannon, C.E., Weaver, W.: The Mathematical Theory of Communication. University of Illinois Press, Urbana, 1949.

Štepánková, O., Hetmerová, A., Kraus, J.: Některé možnosti použití strojového učení v lékařství. Lékař a Technika, 29, 1998, pp. 56-62.

Utgoff, P.E., Bradly, C.F.: Incremental induction of decision trees. Machine Learning 4, 2, 1991, pp. 161-186.

5 GENEROVANIE ROZHODOVACÍCH ZOZNAMOV

5.1 Reprezentácia a použitie rozhodovacích zoznamov

Výsledný popis pojmu môže byť reprezentovaný aj vo forme rozhodovacieho zoznamu. Rozhodovací zoznam generuje usporiadané pravidlá, v ktorých sa žiadna podmienka nevyšetruje dvakrát, preto sa jednoduchšie a rýchlejšie interpretujú ako produkčné pravidlá. Výsledný popis pojmu v tvare rozhodovacích zoznamov je taktiež kratší ako u produkčných pravidiel. Typická oblasť použitia je objavovanie znalostí.

Doteraz sme pracovali s produkčnými pravidlami resp. klasifikačnými pravidlami, ktoré boli navzájom nezávislé. Zmena poradia pravidiel nemohla ovplyvniť výsledok klasifikácie. Súbor takýchto pravidiel je neusporiadaný. **Neusporiadaný súbor pravidiel** predstavuje navzájom nezávislé pravidlá typu if - then.

Usporiadaný súbor pravidiel má nasledovnú podobu:

```
if      then
        else    if      then
                        else    . . .   if      then
                                                else
```

To znamená, že podmienky, ktoré sa vyšetrovali v if časti predchádzajúceho pravidla sa v nasledujúcom pravidle už nevyšetrujú, pretože neplatia, keďže celé nasledujúce pravidlo sa nachádza v else časti predchádzajúceho pravidla. V tomto prípade pochopiteľne veľmi záleží na poradí jednotlivých pravidiel.

Rozhodovací zoznam zodpovedá usporiadanému súboru pravidiel a je konštruovaný nasledovne:

[[podmienky_pravidla1, T_i],…,[podmienky_pravidlaN, T_i]].

Podmienky_pravidlaX sú predstavované jednou alebo viacerými podmienkami v tvare selektora $\mathbf{A_i}=\mathbf{a_i}$. , kde $\mathbf{A_i}$ je atribút a $\mathbf{a_i}$ jemu prislúchajúca hodnota. Jedna podmienka sa v zozname môže nachádzať iba raz. Klasifikácia podľa takéhoto rozhodovacieho zoznamu prebieha nasledovne: príklad je klasifikovaný do triedy, ktorá je súčasťou tej položky rozhodovacieho zoznamu, ktorej podmienky spĺňa. Pričom sa predpokladá, že klasifikovaný príklad nesplnil podmienky predchádzajúcich položiek zoznamu (prešiel do else časti) a zároveň už nie je podstatné či splní podmienky nasledujúcich položiek.

Pri tvorbe neusporiadaných pravidiel môže byť vytvorené pravidlo, ktoré dáva aj pre málo početnú triedu nadpriemernú úspešnosť predikcie. Z toho vyplýva, že algoritmus môže generovať kontradikčné (protirečivé) pravidlá. V prípade neusporiadaných pravidiel sa preto pracuje s konfidenčným faktorom (resp. faktorom spoľahlivosti), ktorý pomáha riešiť sporné prípady, keď je jeden trénovací príklad zaradený pravidlami do viacerých tried.

Pri usporiadaných pravidlách a teda aj pri rozhodovacích zoznamoch k sporom nemôže dôjsť, pretože je pevne dané poradie použitia jednotlivých pravidiel. Trieda predikovaná každým pravidlom odpovedá triede, do ktorej patrí väčšina pokrytých pravidiel.

Uvažujme nasledovnú trénovaciu množinu:

Príklad	atribut1	atribut2	atribut3	Trieda
e1	x	r	m	+
e2	y	r	n	+
e3	y	s	n	+
e4	x	s	m	-
e5	z	t	n	-
e6	z	r	n	-

Z tejto trénovacej množiny je možné generovať nasledovné súbory pravidiel:

Neusporiadaný:

```
if      atribut1=y                          then    trieda is +
if      atribut1=z                          then    trieda is -
if      atribut1=x  &  atribut2=r   then    trieda is +
if      atribut1=x  &  atribut2=s   then    trieda is –
if      true                                then    trieda is +
```

Usporiadaný:

```
if      atribut1=y      then    trieda is +
else
        if      atribut1=z      then    trieda is -
else
        if      atribut2=r      then    trieda is +
else
        if      true            then    trieda is –
```

Ako vidíme pri neusporiadaných pravidlách je potrebné overiť všetky podmienky predpokladu pravidla. Pri usporiadaných pravidlách sa podmienky reťazia, keďže zodpovedajú rozhodovaciemu zoznamu.

Pre neinkrementálne algoritmy platí:

- ak algoritmus buduje zoznam od konca (z prava do ľava), potom zvyčajne pracuje stále so všetkými príkladmi Príkladom tohto prístupu je algoritmus NEX.
- ak algoritmus buduje zoznam od začiatku (z ľava do prava), potom tie príklady, ktoré správne spracoval, zabúda. Príkladom je algoritmus CN2.

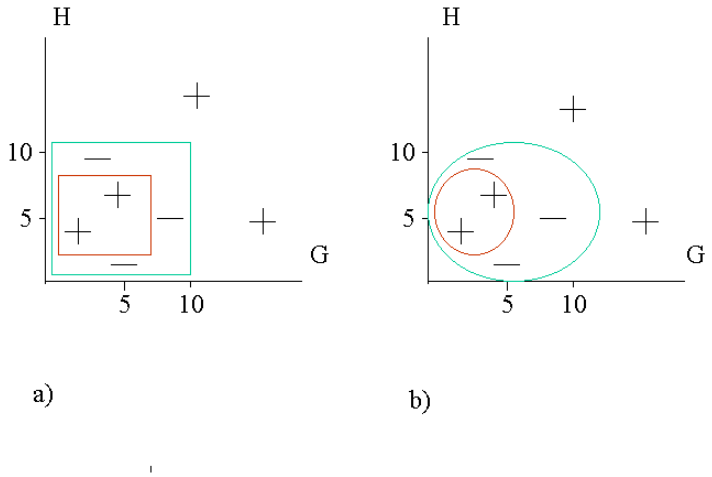
5.2 Indukcia rozhodovacích zoznamov

Podrobnejšie sa budeme venovať dvom algoritmom, navrhnutým na indukciu rozhodovacích zoznamov, a to algoritmu NEX a CN2.

5.2.1 Algoritmus NEX

Algoritmus NEX (**N**oninkremental Induction of Decision List with **Ex**clusions) je neinkrementálny algoritmus indukcie rozhodovacích zoznamov použitím výnimiek. Algoritmus buduje rozhodovací zoznam od konca. Nové položky (pravidlá) ukladá na začiatok zoznamu. Výsledok zodpovedá usporiadanému súboru pravidiel.

Tento algoritmus inicializuje svoj rozhodovací zoznam tým, že formuje implicitné (default) pravidlo najfrekventovanejšej triedy. V každej iterácii metóda aplikuje aktuálny rozhodovací zoznam na všetky trénovacie príklady, aby určila tie, ktoré boli chybne klasifikované. Chybne klasifikované príklady zaradí medzi výnimky. Pre výnimky formuje použitím podprogramu nové pravidlo, ktoré pridá na začiatok rozhodovacieho zoznamu. Algoritmus končí, keď rozhodovací zoznam korektne klasifikuje všetky trénovacie príklady. Chovanie algoritmu NEX v numerickej doméne s dvoma triedami je znázornené na Obr. 17.



Obr. 17: Chovanie algoritmu NEX.

Obr. 17 a) ukazuje výsledky spracovania numerickej domény algoritmom NEX, ktorý ako podprogram na nájdenie podoblasti použil algoritmus pre hľadanie konjunkcií HGS (obdĺžnikové preferencie). Keďže najbežnejšie sa vyskytujúca trieda v rámci danej domény je trieda „+“, je vybratá ako implicitná trieda. Pre všetky výminky tejto implicitnej triedy t.j. pre všetky príklady triedy „-“ algoritmus generuje obdĺžnikovú oblasť obopínajúcu výnimky triedy „-“. Táto oblasť však ešte stále obsahuje dva pozitívne príklady t.j. dve výnimky. Pre tieto dva príklady algoritmus NEX generuje obdĺžnikovú oblasť, ktorá už obsahuje iba príklady jednej triedy „+“ (bez výnimiek). Preto sa činnosť algoritmu môže ukončiť. Výsledkom je zoznam, ktorý obsahuje tri položky – tri pravidlá pre triedu „+“, triedu „-“ a triedu „+“. Obr. 17 b) ilustruje činnosť algoritmu NEX v tej istej doméne, pričom na generovanie podoblasti bola použitá modifikovaná technika pre sférickú prahovú jednotku.

Vstupy: ***ISET****...množina klasifikovaných trénovacích príkladov*
CSET*...množina dvoch alebo viacerých tried*
Výstup: ***DLIST****...rozhodovací zoznam popisov jednoduchých oblastí*

Procedúra: ***NEX(CSET,ISET)***
Nech ***C*** *je najbežnejšia trieda v* ***ISET***
Inicializuj ***DLIST=C***
NEX-AUX(CSET,ISET,DLIST)
vráť rozhodovací zoznam ***DLIST****.*

Procedúra: ***NEX-AUX(CSET,ISET,DLIST)****.*
nech ***MISSED*** *sú tie trénovacie príklady z* ***ISET****, ktoré* ***DLIST*** *nesprávne klasifikuje*
if ***MISSED*** *= {}*
then *vráť* ***DLIST***
else *nech* ***C*** *je najbežnejšia trieda v* ***MISSED***
indukuj najšpecifickejší popis ***D****, ktorý pokrýva príklady triedy* ***C*** *v* ***MISSED***
if ***D*** *je odlišné od prvého termu v* ***DLIST***
then *pridaj „****if D then C****“ na začiatok* ***DLIST***
NEX-AUX(CSET,ISET,DLIST).
else *vyber ľubovoľný atribút* ***A*** *v rozsahu* ***[A1,A2]***
nech ***M1*** *sú príklady z* ***MISSED****, pre ktoré :****A < (A2-A1)/2***
nech ***M2*** *sú príklady z* ***MISSED****, pre ktoré :****A >= (A2-A1)/2***
NEX-AUX (CSET, M1, DLIST)
NEX-AUX (CSET, M2, DLIST)

Dá sa vymyslieť trénovacia množina, pre ktorú nájde algoritmus s úspechom dvakrát tú istú množinu. Napríklad keby boli štyri trénovacie príklady (dva pozitívne a dva negatívne) rozmiestnené v rohoch štvorca v prípade, že doména obsahuje dva atribúty. Pozitívny príklad by sa striedal s negatívnym. V takom prípade najšpecifickejší popis pozitívnych príkladov a najšpecifickejší popis negatívnych príkladov je tá istá oblasť. Hrozí nekonečná rekurzia. V tomto

špeciálnom prípade sa vyberá atribút tak, že sa vypočíta bod uprostred oblasti. Najprv sa rozdelí oblasť na dve podoblasti a následne sa rozdelia aj trénovacie príklady do dvoch podmnožín. Jedna, ktorej trénovacie príklady nadobúdajú hodnotu atribútu väčšiu ako stredový bod a druhá, v ktorej príklady nadobúdajú hodnotu atribútu menšiu ako stredový bod.

V publikáciách (Vere, 1980) a (Helmbold-Sloan-Warmuth,1990) sú popísané systémy využívajúce NEX algoritmus.

Je možná aj inkrementálna verzia NEX algoritmu, takzvaný IEX algoritmus. IEX neuchováva žiadne trénovacie príklady. Keď aktuálny rozhodovací zoznam nesprávne klasifikuje niektorý trénovací príklad, algoritmus zmení triedu termu, ktorý sa dopustil nesprávnej klasifikácie na triedu termu, ktorý sa nachádza v zozname pred ním. Ak nový term má požadovanú triedu, potom IEX zovšeobecní tento term práve natoľko, aby pokryl sporný príklad. Ak nie, algoritmus pridá na začiatok rozhodovacieho zoznamu nový term s korektnou triedou vyhovujúcou spornému príkladu. Implicitné pravidlo používa triedu počiatočného trénovacieho príkladu.

Inkrementálna indukcia rozhodovacích zoznamov je menej bežná (Langley, 1996). Zmienku o nej je možné nájsť v (Iba et al., 1988).

5.2.2 Algoritmus CN2

Algoritmus CN2 (Clark-Nibblet, 1989) patrí medzi systémy typu AQ. Vytvára pravidlá z predložených príkladov a ukladá ich do zoznamu. Môže pracovať s viacerými triedami cieľového atribútu súčasne, aj so zašumenými údajmi.

Algoritmus CN2 pracuje v iteratívnom režime. V každom kroku vygeneruje jedno pravidlo a odstráni pokryté príklady z trénovacej množiny. Hľadá pravidlo, ktoré pokrýva veľký počet trénovacích príkladov najčastejšie sa vyskytujúcej triedy (**C**) a malý počet trénovacích príkladov ostatných tried. Tvorba pravidiel končí, keď sa už nepodarí nájsť vyhovujúce pravidlo. Algoritmus realizuje pri hľadaní jedného pravidla prehľadávanie zhora-nadol (general to specific). Špecifikácia sa uskutočňuje pridaním podmienky do konjunkcie predpokladu, alebo odstránením elementu z disjunkcie hodnôt atribútu. Pod komplexom sa rozumie konjunkcia selektorov t.j. dvojíc atribút-hodnota. Podmienky pre zastavenie procedúry v cykle while podprogramu SEARCH(T,BEST) sú splnené, keď je STAR prázdne, alebo boli generované a testované všetky selektory. Vyhodnotenie komplexu v starom cykle sa uskutočňuje na základe entrópie nasledovným spôsobom:

$$H = -\sum_{r} \frac{K_r}{K} \log_2 \frac{K_r}{K}$$

kde: **K** je počet všetkých trénovacích príkladov pokrytých komplexom

K_r je počet trénovacích príkladov danej triedy pokrytých komplexom.

Spôsob výpočtu signifikancie bude uvedený ďalej v rámci numerických charakteristík pravidla. Algoritmus CN2 je možné definovať nasledovným spôsobom:

Vstupy: ***M****...množina klasifikovaných trénovacích príkladov*
Výstup: ***LISTofRULES****...rozhodovací zoznam popisov jednoduchých oblastí*

CN2(M,LISTofRULES)
nech ***LISTofRULES****={}*
while *trénovacia množina* ***M*** *nie je prázdna* ***do***
begin
    nech ***CMPLX*** *je najlepší komplex* ***BEST***
    nájdený procedúrou ***SEARCH(M,BEST)***
    if ***CMPLX*** *nie je prázdne*
    then *begin*
        nech ***M'*** $\subset$ ***M*** *sú príklady pokryté* ***CMPLX***
        M=M-M'
        *priraď pravidlo "****if CMPLX then class is C****" na koniec* ***LISTofRULES***
        kde ***C*** *je majoritná trieda pre príklady z* ***M'***
end

SEARCH(T,BEST)*:*
nech ***STAR*** *je množina obsahujúca komplex* ***TRUE***
nech komplex ***BEST={}***
nech ***SEL*** *je množina všetkých selektorov, ktoré sa vyskytujú v* ***M***
while *nie sú splnené podmienky pre zastavenie procedúry* ***do***
begin
    nech ***NEWSTAR****={}*
    špecifikuj každý komplex ***p*** *zo* ***STAR*** *pridaním selektoru* ***s*** *zo* ***SEL***
    vyhodnoť nový komplex ***s & p*** *a zaraď ho do* ***NEWSTAR***
    for *každý komplex* ***p*** *z* ***NEWSTAR*** ***do***
        if ***p*** *je signifikantné a lepšie ako* ***BEST***
        then ***BEST=p***
    if *počet komplexov v* ***NEWSTAR*** *presiahne zadanú hodnotu*
    then *vyhoď najhoršie komplexy*
    STAR=NEWSTAR
end

Algoritmus CN2 sa s obľubou využíva aj v oblasti objavovania znalostí, napríklad pri spracovávaní údajov o návštevnosti zámkov (Paralič-Bednár, 2002). Tento algoritmus bol taktiež implementovaný v Turingovom ústave v Prahe. Pod názvom CN4 bol modifikovaný a ponúka radu zlepšení, ako napríklad: ošetrenie chýbajúcich hodnôt, cenu vyhodnocovania atribútu, práca s numerickými atribútmi, atď.

5.2.2.1 Numerické charakteristiky pravidla

Každé pravidlo generované algoritmom CN2 má svoje numerické charakteristiky: K_r, signif, quality a cost.

K_r je predstavované dvojicou čísel v hranatej zátvorke. Prvé číslo sa vzťahuje k triede + a druhé k triede -. Ak je tried viac, hranatá zátvorka obsahuje viac čísel v poradí, ktoré zodpovedá určenému poradiu tried. Pre jednotlivé triedy **K_r** udáva počty príkladov pokrytých pravidlom.

Ďalšia charakteristika **signif** je kritériom pre výber komplexu, t.j. ľavej strany pravidla a počíta sa podľa vzťahu:

$$signif = 2\sum_{r=1}^{R} K_r \ln \frac{K_r}{K} \frac{N}{N_r}$$

kde: **N** je počet trénovacích príkladov
R je počet tried
N_r je počet trénovacích príkladov triedy **r**
K je počet všetkých trénovacích príkladov pokrytých pravidlom
K_r je počet trénovacích príkladov triedy **r** pokrytých pravidlom.

Charakteristika **quality Q** predstavuje hodnotenie založené na počte príkladov triedy pokrytých a nepokrytých daným pravidlom:

$$Q = 0.8\frac{K_r}{K} + 0.2\frac{K_r}{N_r}$$

kde: **K_r/K**....sa nazýva **konzistencia**
K_r/N_r...sa nazýva **úplnosť**

Posledná charakteristika **cost** predstavuje cenu vyhodnotenia pravidla definovanú počtom otázok v predpoklade. Posledné pravidlo je default pravidlo, resp. implicitné pravidlo. Jeho cost = 0, keďže neobsahuje žiadne podmienky (podmienková časť = true).

Ak si napríklad zoberieme tretie pravidlo z neusporiadaného súboru:
if atribut1=x & atribut2=r then class is +

Jeho numerické charakteristiky budú nadobúdať nasledovné hodnoty: K_r = [1,0] (pretože dané pravidlo pokrýva jeden príklad triedy „+" a žiaden príklad triedy "-" vo vyššie uvedenej tabuľke), signif = 2.000, quality = 0.867, cost = 2 (keďže je potrebné overiť dve podmienky).

5.3 Úlohy na precvičenie:

1. Rozhodovací zoznam predstavuje pravidlá:
   a/ neusporiadané
   b/ usporiadané.
2. Charakterizujte algoritmus NEX.
3. Je daná trénovacia množina v tvare tabuľky:

	1	2	3	4	5	6
A	a	b	b	a	c	c
T	+	+	+	-	-	-

   Vypočítajte:
   Signif(A=a)=.....
   Signif(A=b)=.....
4. Majme dané pravidlo: IF A1=a & A2=b THEN trieda +. Určite pre toto pravidlo numerickú charakteristiku ceny:
   Cost=.....
5. Majme danú trénovaciu množinu z tabuľky uvedenej v podkapitole "Reprezentácia a použitie rozhodovacích zoznamov". Generujte z tejto trénovacej množiny rozhodovací zoznam použitím algoritmu CN2. Predpokladajte, že BS=2.
6. Navrhnite podprogramy na vzájomný prevod troch rôznych reprezentácií pojmov: disjunktívnej normálnej formy (DNF), rozhodovacieho stromu (RS) a rozhodovacieho zoznamu (RZ).

5.4 Literatúra

Clark, P., Nibblet, T.: The CN2 Induction Algorithm. Machine Learning, Vol.3, No.4, 1989, pp. 261-284.

Helmbold, D., Sloan, R., Warmuth, M.K.: Learning nested differences of intersection-closed concept classes. Machine Learning, 5, 1990, pp. 165-196.

Iba, W., Wogulis, J., Langley, P.: Trading off simplicity and coverage in incremental concept learning. Proc. of the Fifth International Conference on Machine Learning, Ann Arbor, MI: Morgan Kaufmann, 1988, pp. 73-79.

Langley, P.: Elements of Machine Learning. Morgan Kaufmann Publishers, Inc. San Francisco, California, 1996, 419 pp.

Paralič, J., Bednar, P: Knowledge Discovery in Texts Supporting e-Democracy. In Proc. of the 6th IEEE International Conference on Intelligent Engineering Systems, INES 2002, Opatija, Croatia, May 2002, pp.327-332.

Vere, S.: Multilevel counterfactuals for generalization of relational concepts and productions. Artificial Intelligence, 14, 1980, pp. 139-164.

Časť II

Učenie s prvkami kvantitatívneho usudzovania s učiteľom

6 INDUKCIA PRAHOVÝCH POJMOV

Reprezentačné preferencie popisov prahových pojmov sa odlišuje od popisov logických konjunkcií, produkčných pravidiel, rozhodovacích stromov a zoznamov. Prahové pojmy predstavujú flexibilnejšiu reprezentáciu znalostí. Kľúčovým je parciálne pokrytie, t.j. taký prístup ku klasifikácii, ktorý zatrieďuje trénovacie príklady do tried aj v prípade, keď daný príklad spĺňa iba niektoré podmienky z podmienok tvoriacich popis triedy. Stačí ak stupeň pokrytia prekročí niektoré vopred definované prahy. Prvé štúdie o prahových pojmoch siahajú do päťdesiatych rokov (Rosenblatt0, 1958), (Rosenblatt, 1962). Výborný prehľad relevantných prác z polovice šesťdesiatych rokov je uvedený v (Nilsson,1965). Ďalší vývoj v tejto oblasti je zdokumentovaný v rámci (Duda-Hart, 1973).

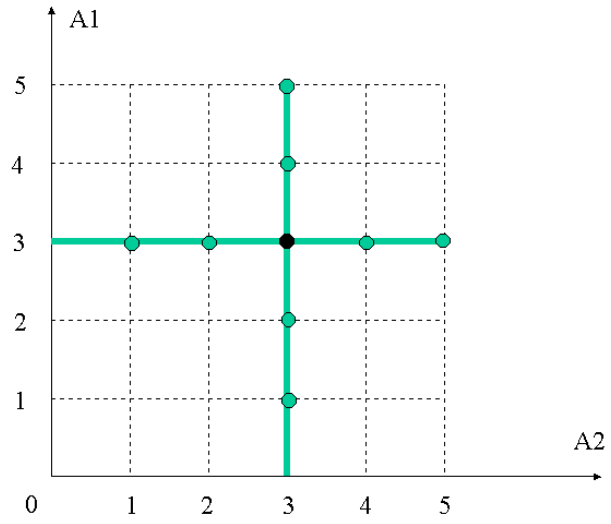
6.1 Reprezentácia a použitie tabuľky kritérií

Najjednoduchšia forma prahového popisu je známa ako **tabuľka kritérií** alebo **m_z_n**. Tabuľka kritérií bola uvedená do oblasti strojového učenia publikáciou (Spackman, 1988). Ale aj iné práce venujú pozornosť tejto problematike, ako napríklad (Baffles-Mooney, 1993) a (Murphy-Pazzani, 1991).

Takýto popis zahŕňa množinu **n** dvojhodnotových atribútov a prah **m**, ktorý reprezentuje prirodzené číslo medzi **1** a **n**. Konjunkcia všetkých **n** definovaných atribútov je často nazývaná ako prototyp resp. etalón, predstavujúci bod v priestore príkladov. Spôsob interpretácie popisu pojmu vo forme tabuľky kritérií je nasledovný: ak je počet atribútov príkladu, ktoré sú pokryté atribútmi etalónu väčší ako prahová hodnota **m**, potom je príklad považovaný za pozitívny príklad pojmu, ktorý je reprezentovaný daným etalónom. Tabuľka kritérií môže reprezentovať konjunkciu a disjunkciu ako špeciálne prípady v závislosti na požadovanej hodnote prahu. Konjunkcia môže byť predstavovaná **n_z_n** pojmom a disjunkciu je možné definovať ako **1_z_n** pojem.

Obr. 18 reprezentuje dvojrozmerný priestor príkladov nad reálnou doménou s atribútmi A1 a A2. Obidva atribúty môžu nadobúdať hodnoty z množiny {1,2,3,4,5}. Tabuľka kritérií 2_z_2 predstavuje čierny bod etalón, ktorý je možné popísať [A1=3,A2=3]. Tento bod je zároveň jediným pozitívnym príkladom pojmu 2_z_2. Nad daným priestorom príkladov je možné definovať ešte pojem 1_z_2. Tento pojem predstavujú dva rezy prechádzajúce etalónom, kolmé na súradné osi. Všetky príklady nachádzajúce sa na týchto rezoch môžu byť klasifikované ako pozitívne príklady pojmu 1_z_2.

Obr. 19 reprezentuje trojrozmerný priestor s atribútmi A1, A2 a A3. V tomto priestore je etalón reprezentovaný pojmom 3_z_3, teda jedným čiernym bodom. Pojem 2_z_3 predstavuje tri rezy. Každý je rovnobežný s jednou osou pre daný rez neuvažovaného atribútu a zároveň kolmý na zvyšné dve osi pre daný rez uvažovaných atribútov. Pojem 1_z_3 je znázornený troma plochami. Každá z nich je rovnobežná s osami dvoch pre danú plochu neuvažovaných atribútov a zároveň kolmá na jednu os pre danú plochu uvažovaného atribútu.

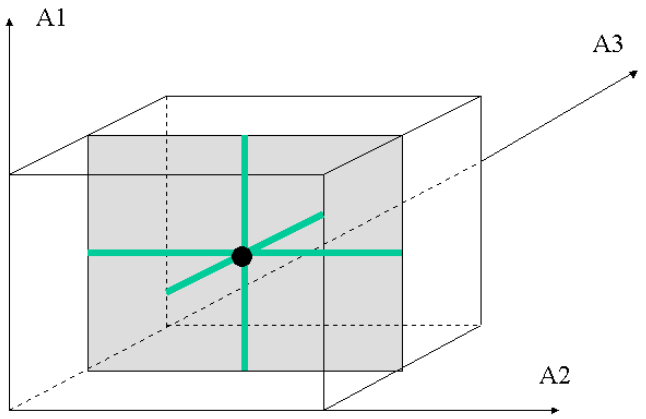


Obr. 18: Tabuľka kritérií v dvojrozmernom priestore trénovacích príkladov.

Tabuľka kritérií má široké použitie v medicíne pri diagnóze chorôb. Napríklad tabuľka kritérií z Obr. 18 môže predstavovať diagnostiku ochorenia horných dýchacích ciest u nemluvňaťa (nevie povedať čo ho bolí). **A1** predstavuje príznak horúčky a **A2** viditeľné ťažkosti s nasledovnými konkrétnymi hodnotami:

A1=1...teplota normálna	A2=1...soplenie
A1=2...teplota 37	A2=2...pokašliavanie
A1=3...teplota 38	A3=3...kašeľ
A1=4...teplota 39	A3=4...silný kašeľ
A1=5...teplota 40	A5=5...dusivý kašeľ.

Typickým reprezentantom, teda etalónom je A1=teplota 38 a zároveň A2=kašeľ. Vtedy väčšinou už vyhľadáme pre dieťa lekársku pomoc.



Obr. 19: Tabuľka kritérií v trojrozmernom priestore trénovacích príkladov.

6.2 Reprezentácia a použitie lineárnej prahovej jednotky

Existujú problémy, ktorým nevyhovuje reprezentácia ani vo forme logických konjunkcií ani v tvare tabuľky kritérií. Prirodzený spôsob ako spracovať túto triedu problémov je priradiť váhu každému atribútu a tak reprezentovať stupeň relevancie daného atribútu. Tento prístup sa nazýva lineárna prahová jednotka a je veľmi podrobne prezentovaný v (Langley, 1996), ako napokon celá problematika prahových pojmov.
Napríklad klasifikačné pravidlo:

$$1 * jedno\_jadro + 1 * jeden\_bičič + 2 * hrubá\_stena \leq 1{,}5 \Rightarrow chorá\_bunka$$

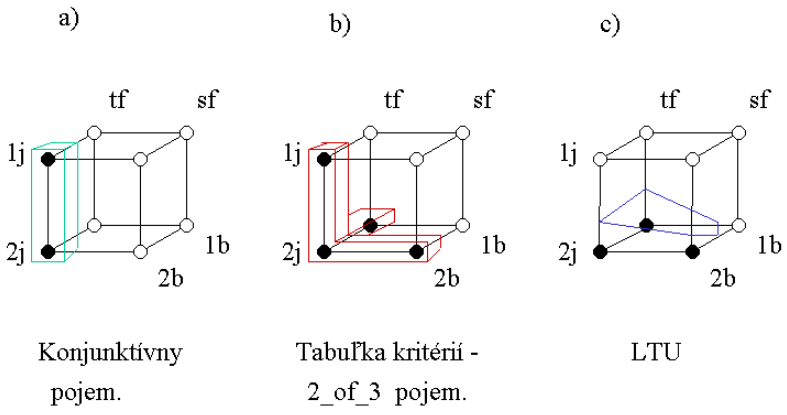
obsahuje vo svojej IF-časti lineárnu prahovú jednotku, ktorá vlastne produkuje intenzionálnu definíciu, ktorú nie je možné vyjadriť vo forme tabuľky kritérií m_z_n.

Klasifikácia nového príkladu potom prebieha nasledovne: ak hodnoty atribútov klasifikovaného príkladu spĺňajú nerovnicu v IF časti klasifikačného pravidla, potom je nový príklad klasifikovaný ako pozitívny príklad daného pojmu.

Kombinácia váh použitých v klasifikačnom pravidle sa nazýva lineárna prahová jednotka alebo LTU (Linear Threshold Unit). Tabuľku kritérií môžeme považovať za špeciálny prípad

LTU, v ktorej váhy nadobúdajú hodnoty 1. LTU môže charakterizovať akúkoľvek extenzionálnu definíciu pojmu. Stačí ak hranica oblasti pozitívnych trénovacích príkladov daného pojmu môže byť ohraničená nejakou hyperrovinou v n-dimenzionálnom priestore. To znamená, že triedy (presnejšie príklady tried) sú lineárne separabilné Váhy atribútov potom určujú orientáciu hyperroviny v priestore a prahová hodnota určuje jej posunutie pozdĺž kolmice na hyperrovinu. O cieľovom pojme, ktorý môže byť reprezentovaný LTU hovoríme, že lineárne separuje pozitívne a negatívne trénovacie príklady. Taká trénovacia množina je lineárne separabilná pomocou LTU.

Na Obr. 20 sú znázornené tri extenzionálne definície dané množinou pozitívnych (čierne body) a negatívnych (biele body) trénovacích príkladov pojmu. Prípad v časti a) môže byť reprezentovaný najstriktnejšou formou, logickou konjunkciou. Takisto môže byť reprezentovaný tabuľkou kritérií a LTU, ktorých špeciálnym prípadom je logická konjunkcia. Trénovacia množina v časti b) nemôže byť reprezentovaná logickou konjunkciou, ale môže byť reprezentovaná tabuľkou kritérií a taktiež všeobecnejšou reprezentáciou LTU. Posledný prípad v časti c) nemôže byť reprezentovaný ani logickou konjunkciou ani tabuľkou kritérií. Môže byť však reprezentovaný LTU.



Obr. 20: Tri rôzne extenzionálne definície pojmu.

Reprezentačné schopnosti LTU sú najjasnejšie preukázateľné pri použití numerických domén viď Obr. 21.

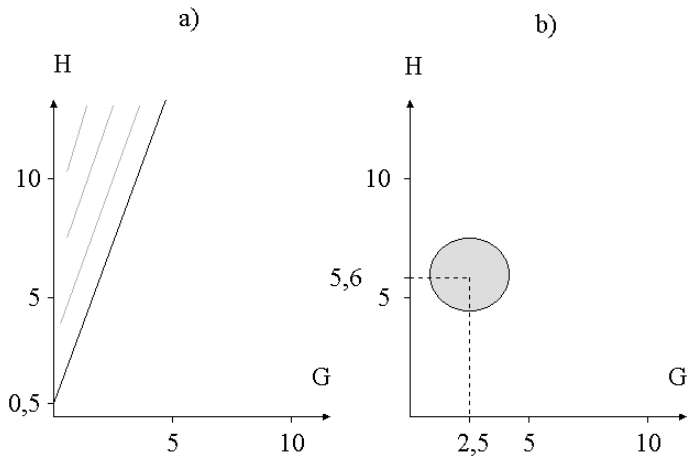
6.3 Reprezentácia a použitie sférických prahových jednotiek

LTU je nevýhodná v tom, že oblasti reprezentujúce pozitívne trénovacie príklady pojmu nie sú ohraničené zo všetkých strán. Napríklad LTU v Obr. 21 v časti a) je ohraničená iba lineárnou rovnicou a hranicami priestoru trénovacích príkladov. Je čiastočne otvorená, čo znamená, že teoreticky do nej spadá väčšie množstvo neznámych trénovacích príkladov, ako pri sférickej prahovej jednotke (Kruschke, 1992), (Moody-Darken, 1991), ktorá je ohraničená zo všetkých strán, ako to ilustruje spomínaný obrázok. Tento obrázok v časti a) znázorňuje LTU v numerickej doméne H/G, ktorá je ohraničená lineárnou rovnicou v podmienkovej časti pravidla:

$$1{,}0 * H - 2{,}2 * G \geq 0{,}5 \Rightarrow \textit{štíhla\_postava}.$$

V časti b) daného obrázku je znázornená sférická prahová jednotka taktiež v numerickej doméne H/G. Táto sférická prahová jednotka pokrýva lokálnu oblasť, ohraničenú sférickou rovnicou v podmienkovej časti pravidla:

$$\sqrt{(H-5{,}6)^2 + (G-2{,}5)^2} \leq 1{,}5 \Rightarrow \textit{normálna\_váha}.$$



Obr. 21: LTU a sférická prahová jednotka v numerickej doméne.

V n-rozmernom priestore príkladov tvorí sférická prahová jednotka hypersféru. Konštanta spojená s každým atribútom špecifikuje polohu centra hypersféry vzhľadom k osi daného atribútu. Prahová hodnota určuje jej polomer. Inou možnosťou je reprezentácia definície pojmu hyperelipsou, s osami rôznej dĺžky. To vyžaduje jeden prídavný parameter pre každý atribút, ktorý umožňuje, že každá os hyperoblasti (odpovedajúca príslušnému atribútu) je paralelná s niektorou osou priestoru príkladov.

6.4 Indukcia prahových pojmov prehľadávaním

Prahové pojmy sa odlišujú od logickej konjunkcie okrem iného aj v povahe operátorov umožňujúcich pohyb v usporiadanom priestore pojmov. Keďže táto oblasť umožňuje parciálne pokrytie, platí:

Pridanie atribútu do podmienky pri konštantnom prahu produkuje skôr všeobecnejšie ako špecifickejšie popisy pojmov, pretože tým vytvára viac možností na prekročenie prahu. Napríklad 1_z_2 pojem je v dvojrozmernom priestore trénovacích príkladov znázornený priamkou. Pridaním jedného atribútu sa priestor trénovacích príkladov zväčší na trojrozmerný a vznikne pojem 1_z_3 znázornený plochou, ktorá je všeobecnejšia ako priamka.

Závislosť na prahových hodnotách zavádza nový typ operátora. Nárast prahovej hodnoty pri zachovaní etalónu vedie ku špecifickejšiemu pojmu. Napríklad 1_z_3 pojem (plocha) sa môže nárastom prahovej hodnoty zmeniť na pojem 2_z_3 (priamka) čím sa stane špecifickejším.

6.4.1 Indukcia tabuľky kritérií

Ak sa sústredíme na domény, ktoré obsahujú iba **r** relevantných atribútov a žiadny irelevantný (teda všetky atribúty budú vystupovať v etalóne), potom môžeme úlohu indukcie tabuľky kritérií zúžiť na dva kroky:

- Prvým krokom je určenie etalónu.
- Druhým krokom je výber vhodného prahu od **1** do **r** vrátane.

Algoritmus revízie prahovej hodnoty TR (Threshold Revision) najprv nájde najfrekventovanejšiu hodnotu pre každý atribút. Množina týchto hodnôt potom tvorí etalón E. Keďže je priestor pojmov usporiadaný podľa všeobecnosti, algoritmus môže štartovať od najšpecifickejšieho pojmu r_z_r, alebo od najvšeobecnejšieho pojmu 1_z_r.

V jednom prístupe, nazvanom **TRG (Threshold Revision General)** jediný operátor zmenšuje prahovú hodnotu a odvádza všeobecnejšiu hypotézu. V druhom prístupe **TRS (Threshold Revision Specific)** jediný operátor zvyšuje prahovú hodnotu a generuje špecifickejšiu hypotézu. Ukončovacou podmienkou oboch algoritmov je neúspech v snahe dosiahnuť lepšie výsledky podľa niektorého kritéria. Hoci celkový priestor pojmov je iba parciálne usporiadaný, podmnožina hypotéz, ktoré sa líšia iba v prahovej hodnote, je úplne usporiadaná.

Obidve tieto metódy sú schopné indukovať tabuľku kritérií iba v prípade, že sú všetky atribúty relevantné. V realistickejších situáciách, keď je relevantných iba **r** z **n** atribútov musíme použiť iný prístup, napríklad heuristickú indukciu tabuľky kritérií.

6.4.2 Algoritmus HCT

HCT je algoritmus pre heuristickú indukciu tabuľky kritérií (**Heuristic Criteria Tables**) (Langley, 1996). Je to neinkrementálny algoritmus postupujúci smerom od špecifického ku všeobecnému. Je riadený ohodnocovacou funkciou Score a používa takzvané Beam Search t.j. prehľadávanie obmedzené lúčovým číslom (Beam Size) t.j. šírkou lúča.

Aby tento algoritmus mohol prehľadávať usporiadaný priestor hypotéz, potrebuje operátory zovšeobecnenia a špecifikácie, ktoré dokážu pracovať s prahovými pojmami. Existujú dva operátory, ktoré vedú od špecifickejšej tabuľky kritérií ku všeobecnejšej, t.j. **operátory generalizácie:**

Prvý operátor generalizácie vymaže atribút a zároveň zníži prahovú hodnotu o jednotku. Majme hypotézu 2_z_{jedno_jadro, jeden_bičík, hrubá_stena}.Operátor z tejto hypotézy generuje tri alternatívy: 1_z_{jedno_jadro, jeden_bičík}
1_z_{jedno_jadro, hrubá_stena}
1_z_{jeden_bičík, hrubá_stena}.

Druhý operátor generalizácie simultánne drží prahovú hodnotu konštantnú a zavádza nový atribút. Tento operátor by pre hypotézu 1_z_{jedno_jadro} generoval dve zovšeobecnenia:
1_z_{jedno_jadro, jeden_bičík}
1_z_{jedno_jadro, hrubá_stena}.

Podobne môže byť organizované prehľadávanie od všeobecného ku špecifickému. V takom prípade hovoríme o **operátoroch špecifikácie:**

Prvý operátor špecifikácie pridáva atribút a zároveň zvyšuje prahovú hodnotu o jednotku.

Druhý operátor špecifikácie zachováva prahovú hodnotu a zároveň vymazáva atribút.

To predpokladá že metóda HCT vypočíta na začiatku zmysluplný etalón, z ktorého operátory preberú atribúty.

Algoritmus HCT je v značnej miere príbuzný algoritmu HGS (HSG). Príbuznosť spočíva hlavne v riadení skórovacou funkciou a v prehľadávaní obmedzenom šírkou lúča (Beam Size). Obidva algoritmy v každej iterácii zo všetkých uvažovaných pojmov vyberajú iba určitý počet (šírka lúča). Výber uskutočňujú za pomoci skórovacej funkcie. S tým súvisí vhodné používanie množín: HSET, SPECS a hlavne OPEN-SET, CLOSED-SET a BEST-SET.

Vstupy: ***PSET****...množina pozitívnych trénovacích príkladov*
NSET*...množina negatívnych trénovacích príkladov*
ATTS*...množina nominálnych atribútov*
Výstup: ***Tabuľka kritérií*** *na klasifikáciu nových príkladov*
Parameter: ***Beam-Size*** *počet popisov pojmov uvažovaných na novej úrovni*

Procedúra: ***hct(PSET,NSET,ATTS)***
nech etalón ***E*** *je množinou najfrekventovanejších hodnôt v* ***PSET***
pre každý z atribútov v ***ATTS***
nech inicializačná prahová hodnota ***T*** *= veľkosť* ***ATTS*** *(počet atribútov)*
nech inicializačná množina hypotéz ***HSET={[T_z_E]}***
htc-aux(PSET,NSET,E,{},HSET)

Procedúra podprogramu:
htc-aux(PSET,NSET,E,CLOSED-SET,HSET)
nech ***OPEN-SET****={}*
for *každý pojem* ***H*** *v* ***HSET***
nech ***SPECS*** *je najšpecifickejšie zovšeobecnenie(****H,E****)*
nech ***NEWSET****={}*
for *každý špecifikovaný pojem* ***S*** *v* ***SPECS***
if ***Score (S,PSET,NSET)>Score (H,PSET,NSET)***
then *pridaj* ***S*** *do* ***NEWSET***
if ***NEW-SET****={}*
then *pridaj* ***H*** *do* ***CLOSED-SET***
else ***for*** *každý pojem* ***S*** *v* ***NEW-SET***
pridaj ***S*** *do* ***OPEN-SET***
if ***OPEN-SET****={}*
then *vráť člena s najvyšším skóre v* ***CLOSED-SET***
else *nech* ***BEST-SET*** *je* ***Beam-Size*** *počet najvyššie skórovaných*
členov zjednotenia ***OPEN-SET*** *a* ***CLOSED-SET***
nech ***CLOSED-SET*** *je množina členov* ***CLOSED-SET*** *v* ***BEST-SET***
nech ***OPEN-SET*** *je množina členov* ***OPEN-SET*** *v* ***BEST-SET***
hct-aux(PSET,NSET,E,CLOSED-SET,OPEN-SET)

V každej etape v procese prehľadávania musí algoritmus HCT ohodnotiť hypotézy podľa nejakého kritéria. Pre malé nezašumené domény je možné jednoducho vylúčiť hypotézy pokrývajúce niektoré negatívne príklady, alebo tie ktoré nepokrývajú pozitívne príklady. Ak je priestor hypotéz väčší, alebo ak sa vyskytujú zašumené údaje, potom je potrebné použiť heuristické prehľadávanie, riadené ohodnocovacou funkciou. Ohodnocovacie funkcie sú uvedené v kapitole Heuristická indukcia logických konjunkcií.

Algoritmus HCT je na prvý pohľad veľmi podobný algoritmu HGS. Obidva algoritmy heuristicky prehľadávajú priestor pojmov, obidva používajú heuristickú ohodnocovaciu (skórovaciu) funkciu a ich prehľadávanie je obmedzené lúčovým číslom (Beam Search). Rozdiel

je v tom, že na výstupe HGS je pojem vo forme prísnej logickej konjunkcie, zatiaľ čo na výstupe HCT je pojem vo voľnejšej forme tabuľky kritérií vykazujúcej prvky kvantitatívneho usudzovania.

6.4.3 Iteratívna váhová perturbácia

Uvažujme schému, ktorá na základe celej trénovacej množiny raz za čas reviduje váhy. Deliace hranice generované LTU majú formu priamky v rovine, roviny v priestore. Vo všeobecnosti v n-rozmernom priestore má LTU formu hyperroviny, ktorú môžeme popísať rovnicou:

$$\sum_{i=1}^{n} w_i x_i = w_0$$

Kde $\mathbf{w_0}$ je prahová hodnota, $\mathbf{w_i}$ je váha atribútu $\mathbf{A_i}$ a $\mathbf{x_i}$ je hodnota atribútu $\mathbf{A_i}$ v príklade. Niektoré trénovacie príklady môžu ležať priamo na deliacej ploche. Pre pozitívne trénovacie príklady môžeme definovať kolmicu, oddeľujúcu bod (trénovací príklad) od deliacej plochy, pričom prenesieme $\mathbf{w_0}$ z pravej na ľavú stranu predchádzajúcej rovnice:

$$\sum_{i=0}^{n} w_i x_{ij} = V_j$$

Kde $\mathbf{x_{ij}}$ je hodnota atribútu $\mathbf{A_i}$ v príklade **j** ($\mathbf{x_{0j}}$=-1). Pre trénovacie príklady (body) na deliacej ploche je $\mathbf{V_j=0}$. Ak $\mathbf{V_j>0}$, príklad **j** leží na jednej strane deliacej plochy a teda bude klasifikovaný ako pozitívny. Negatívne skóre povedie k opačnej polohe príkladu a teda k jeho negatívnej klasifikácii.

Chceme nájsť takú množinu váh, ktorá uskutoční najväčšie množstvo korektných klasifikácií. Potom pre váhu $\mathbf{w_k}$ nejakého atribútu $\mathbf{A_k}$ môžeme definovať:

$$(\sum_{i \neq k} w_i x_{ij}) / x_{kj} = U_{kj} \qquad U_{kj} = V_j / x_{kj} - w_k .$$

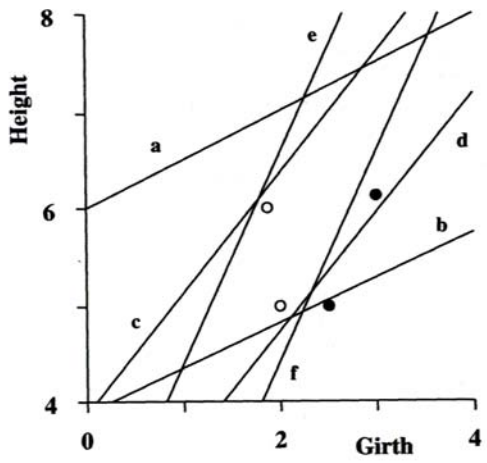
Pre bod, ktorý leží na deliacej ploche teda platí: $\mathbf{U_{kj} = -w_k}$. Čím je nejaký bod (trénovací príklad) vzdialenejší od deliacej plochy, tým menej to platí.

Uvedený prístup bol využitý pri návrhu **IWP (Iterative Weight Perturbation)** algoritmu iteratívnej váhovej perturbácie (Breiman at al., 1984). Príbuzné metódy sú uvedené v (Murthy at al., 1993). Algoritmus IWP opakovane "rozhádže" váhy pre každý atribút a potom ich dolaďuje tak aby zabezpečili čo najlepšiu klasifikáciu. Na určenie najlepšej váhy pre atribút $\mathbf{A_k}$ použije $\mathbf{U_{kj}}$ hodnoty. Ak výsledný váhový vektor korektne klasifikuje všetky trénovacie príklady, potom IWP končí. Inak prebehne cez množinu váh atribútov toľkokrát, kým nevykoná užívateľom špecifikované množstvo cyklov. IWP uchováva najlepší popis generovaný do toho času. Tento najlepší popis je vrátený algoritmom, ak ten nemôže nájsť iný popis, ktorý by perfektne separoval pozitívne príklady od negatívnych.

Vstupy: ***ISET****...množina trénovacích príkladov*
ATTS*...množina atribútov*
Výstup: ***LTU*** *na klasifikáciu nových príkladov*
Parameter: ***Max_Iterations****...maximálny počet iterácií*

Procedúra: ***iwp (ISET,ATTS)****.*
nech ***H*** *je LTU s voliteľnými váhami z intervalu (-1,1)*
nech ***BEST=H***
nech ***COUNT****=Max_Iterations*
repeat kým neplatí ***COUNT****=0:*
for *každý atribút* ***K*** *z* ***ATTS***
for *každý príklad* ***J*** *z* ***ISET***
vypočítaj $\boldsymbol{U_{kj}}$ *použijúc* ***H*** *a* ***J***
ulož $\boldsymbol{U_{kj}}$ *hodnoty v zostupnom usporiadaní (****U'****)*
for *každý susedný pár* ***U'*** *hodnôt*
nech $\boldsymbol{w_k}$***'*** *je zápornou priemernou hodnotou páru*
nech ***H'*** *je* ***H****, v ktorom je* $\boldsymbol{w_k}$ *nahradené* $\boldsymbol{w_k}$***'***
vypočítaj ***Score(H',ISET)***
nech ***H*** *je* ***LTU*** *s najvyšším skóre*
if ***Score(H,ISET)****=1*
then *vráť hypotézu* ***H***
else ***if*** ***Score(H,ISET)<=Score(BEST,ISET)***
then *nech* ***BEST=H***
dekrementuj ***COUNT***
vráť hypotézu ***BEST****.*

Obr. 22 ilustruje deliace hranice generované IWP algoritmom, kde Girth predstavuje súradnú os X a Height súradnú os Y. Bielym krúžkom sú označené pozitívne príklady a čiernym krúžkom negatívne. Algoritmus začína od inicializačnej LTU:
a) 1.0*Y-0.5*X≥6.0. V prvej iterácii sa zmení w_0=6.0 na w_0=3.875, čím sa pôvodná LTU posunie do priamky:
b) 1.0*Y-0.5*X≥3.875. V ďalšej iterácii sa zmení w_1=0.5 na w_1=-1.224, čím sa zmení smernica LTU a výsledkom je priamka:
c) 1.0*Y-1.224*X≥3.875. Priamka d) je získaná posunutím priamky c), spôsobenou zmenou w0=3.875 na w0=2.245. Priamka e) sa získa otočením d), spôsobeným zmenou smernice w1=-1.224 na w1=-2.156. Napokon sa zmení w0=2.245 na w0=0.236 a výsledkom je posunutá priamka:
f) 1.0*Height-2.156*Girth>=2.36, ktorá korektne klasifikuje danú trénovaciu množinu, takže algoritmus IWP končí svoju činnosť.



Obr. 22: Deliace hranice generované algoritmom IWP.

IWP je vlastne algoritmom modifikácie váhového vektora. IWP môže uviaznuť v lokálnom optime. Z toho dôvodu, niektoré verzie algoritmu IWP aplikujú stratégiu náhodnej perturbácie celého váhového vektora. Inou odpoveďou na otázku tohto problému je štart z mnohých odlišných, náhodne generovaných počiatočných váhových vektorov. Týmto spôsobom sa dopracuje algoritmus k viacerým výsledkom a vyberie najlepší.

6.5 Úlohy na precvičenie:

1. Priraďte jednotlivým tabuľkám kritérií 1 z 3, 2 z 3 a 3 z 3 ich reprezentáciu v trojrozmernom priestore: B – bod, U – úsečka, R – rovina.
   a/ 1 z 3 .....
   b/ 2 z 3 .....
   c/ 3 z 3 .....
2. Ktoré sú základné charakteristiky algoritmu HCT?
3. Indukujte tabuľku kritérií použitím algoritmu HCT ak máme danú numerickú doménu s dvoma numerickými atribútmi A1 a A2 obsahujúcu nasledovné pozitívne trénovacie príklady: [1,2][2,1][2,3][3,2] a negatívne trénovacie príklady:[1,1][1,3][3,1][3,3].
4. Majme priestor pozitívnych príkladov pojmu: [0.5,1.0], [0.5,3.0] a negatívnych príkladov pojmu [1.0,0.5], [3.0,0.5], kde [A1,A2] sú atribúty priestoru. Je tento priestor lineárne separabilný LTU v tvare: $A1-A2 \geq 0$?

5. Indukujte LTU použitím algoritmu IWP, ktorá jednoznačne oddelí pozitívne od negatívnych trénovacích príkladov ak máme danú numerickú doménu s dvoma numerickými atribútmi A1 a A2 obsahujúcu nasledovné pozitívne trénovacie príklady: [1.75,6.0][2.0,5.0] a negatívne trénovacie príklady:[2.5,5.0][3.0,6.25].

6.6 Literatúra:

Baffles, P.T., Mooney, R.J.: Symbolic revision of theories with M-of-N rules. Proc. of the Thirteenth International Joint Conference on Artificial Intelligence, Chambéry, France: Morgan Kaufmann, 1993, pp. 1134-1140.

Breiman, L., Friedman, J.H., Olshen, R. A., Stone, C.J.: Classification and regression trees. Belmont, CA: Wadsworth, 1984.

Duda, R.O., Hart, P.E.: Pattern classification and scene analysis. New York: John Wiley, 1973.

Kruschke, J.K.: ALCOVE: An exemplar-based connectionist model of category learning. Psychological Review, 99, 1992, pp. 22-44.

Langley, P.: Elements of Machine Learning. Morgan Kaufmann Publishers, Inc. San Francisco, California, 1996, 419 pp.

Moody, J., Darken, C.: Fast learning in networks of locally-tuned processing units. Neural Computation, 1, 1991, pp. 281-294.

Murphy, P.M., Pazzani, M.J.: ID2-of-3: Constructive induction of M-of-N concepts for discrimination in decision trees. Proc. of the Eighth International Workshop on Machine Learning, Evanston, IL: Morgan Kaufmann, 1991,pp. 173-177.

Murthy, S., Kasif, S., Salzberg, S., Beigel, R.: OC1: Randomized induction of oblique decision trees. Proc. of the Eleventh National Conference on Artificial Intelligence, Washington, DC: AAAI Press, 1993, pp. 322-327.

Nilsson, N.J.: Learning machines. New York: McGraw-Hill, 1965.

Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65, 1958, pp: 386-408.

Rosenblatt, F.: Principles of neurodynamics. New York: Spartan Books, 1962.

Spackman, K.A.: Learning categorical criteria in biomedical domains. Proc. of the Fifth International Conference on Machine Learning, Ann Arbor, MI: Morgan Kaufmann, 1988, pp. 36-46.

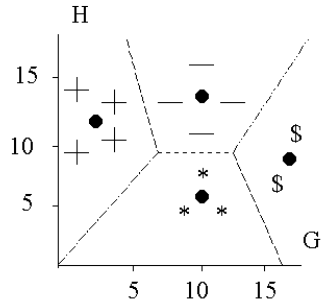
7 INDUKCIA ETALÓNOV

7.1 Reprezentácia a použitie etalónov

Reprezentácia pomocou etalónov, podobne ako pomocou prahových pojmov umožňuje učenie s prvkami kvantitatívneho usudzovania. Je to iný typ reprezentácie, ktorá je flexibilnejšia ako logické reprezentácie. Nemôže sa stať, že sa nenájde popis nejakého pojmu. Pojem je totiž reprezentovaný etalónom, ktorý je typickým resp. "priemerným" reprezentantom pozitívnych príkladov daného pojmu a ten nie problémom určiť.

Metódy učenia založené na prípadoch reprezentujú každý popis pojmu vo forme etalónu, ktorý je jednoduchou sadou párov: atribút-hodnota. Ide vlastne o učenie, kde sa nový prípad rieši resp. klasifikuje pomocou známeho prípadu-etalónu. Vyberá sa etalón, ktorý sa najviac podobá novému príkladu, teda je mu najbližšie. (Kolodner, 1993) obsahuje všeobecný úvod do problematiky učenia založeného na prípadoch.

Ďalší výskum v tejto oblasti je okrem iného prezentovaný v (Haton at al., 1995) a (Watson, 1995). Patrí sem aj Naivný Bayesov klasifikátor, u ktorého je etalón reprezentovaný sadou pravdepodobností, a ktorý je uvedený v kapitole Pravdepodobnostný popis.



Obr. 23: Štvortriedna numerická doména.

Je možné reprezentovať jednu triedu viacerými etalónmi. Teraz sa sústredíme na problémy, v ktorých je jedna trieda reprezentovaná jedným etalónom. Prístup založený na príkladoch sa najjednoduchšie vizualizuje pre numerické domény. Na Obr. 23 je znázornená štvortriedna numerická doména. Etalóny jednotlivých tried ("+", "-", "*", "$") sú na obrázku znázornené čiernym krúžkom a deliaca hranica prerušovanou čiarou. Etalóny neobsahujú žiadne informácie o deliacich hraniciach. Uchovávajú iba špecifické hodnoty. Deliaca hranica medzi dvoma triedami je implicitne definovaná ako kolmica na spojnicu etalónov týchto dvoch tried.

Nasleduje interpretácia implicitného pojmu v tvare etalónu teda pravidla, podľa ktorých prebieha klasifikácia nového príkladu.

Klasifikácia do dvoch tried je uskutočňovaná nasledovne. Ak máme nový príklad **I**, najprv vypočítame jeho vzdialenosť od každého etalónu **E**. Pre numerické domény to môže byť Euklidovská vzdialenosť:

$$V = \sqrt{\sum_{i=1}^{n} (e_i - x_i)^2}$$

kde: **n** je počet atribútov

$\mathbf{e_i}$ je hodnota i-tého atribútu v etalóne **E**

$\mathbf{x_i}$ je hodnota i-tého atribútu v príklade **I**.

Pre nominálne domény je typické použitie variantu Hamingovej vzdialenosti , v ktorej vzdialenosť inverzne závisí na počte atribútov, ktoré sú rovnaké v obidvoch príkladoch. V niektorých prípadoch sa používa podobnosť dvoch príkladov, ktorá je zvyčajne iba inverziou ich vzdialeností.

Po výpočte vzdialenosti nového príkladu od všetkých etalónov sú výsledky použité na klasifikáciu nového príkladu. Najjednoduchšia schéma jednoducho predikuje triedu nového príkladu ako triedu spojenú s najbližším etalónom.

Deliaca hranica definovaná dvoma etalónmi rozdelí priestor príkladov na dve množiny resp. dve oblasti. Všetky príklady, ktoré majú bližšie k etalónu triedy A ležia v oblasti triedy A, zatiaľ čo všetky príklady, ktoré majú bližšie k etalónu triedy B ležia v oblasti triedy B.

Multitriedna klasifikácia. Vo všeobecnosti je možné vykonávať klasifikáciu do viacerých tried. Aj v tomto prípade môžu byť deliace hranice graficky znázornené hyperrovinami (ak je počet atribútov väčší ako tri) a musí existovať jedna hranica pre každý pár etalónov $\mathbf{E_i}$ a $\mathbf{E_j}$, kde **i** nie je totožné s **j**. Prirodzene **i** a **j** môžu nadobúdať hodnoty od **1** po **K**. Každý bod hranice (hyperroviny) je rovnako vzdialený od etalónov susedných oblastí, ktoré daná hyperrovina oddeľuje. **K** tried vedie ku **K(K-1)/2** hraníc. Niektoré neznamenajú prínos pre klasifikáciu, pretože sú redundantné. Napríklad ako hranica medzi dvoma oblasťami, ktoré sa nachádzajú na opačných koncoch priestoru príkladov, a teda sú oddelené viacerými triedami. Na Obr. 23 sa nachádzajú takéto triedy ("+" a "$").

7.1.1 Vzťah reprezentácie etalónmi a LTU

Existuje prevod reprezentácie etalónmi na lineárnu prahovú jednotku. Tento prevod pre dve triedy "**a**" a "**b**" sa môže uskutočniť nasledovne:

$$\sum_{i=1}^{n}(e_{ai}-e_{bi})x_i \geq \frac{1}{2}\sum_{i=1}^{n}(e_{ai}^2-e_{bi}^2)$$

kde $\mathbf{e_{ki}}$ je hodnota **i**-tého atribútu pre etalón triedy **k**. Táto prevodová nerovnica popisuje tú oblasť priestoru trénovacích príkladov, ktorej príklady budú klasifikované do triedy "**a**". Teda trénovacie príklady, ktoré spĺňajú obmedzenie tejto nerovnice sú označené za príklady triedy "**a**". Keby sme v nerovnici vymenili $\mathbf{e_{ai}}$ a $\mathbf{e_{bi}}$, dostali by sme oblasť pre triedu "**b**". Keby sme znamienko nerovnosti zamenili znamienkom rovnosti, dostali by sme deliacu hranicu medzi oblasťami oboch tried, teda dostali by sme priamo lineárnu prahovú jednotku.

Pri multitriednej klasifikácii musíme kontrolovať všetky páry nerovníc, ktoré separujú triedy. Trénovací príklad bude klasifikovaný do tej triedy, pre ktorú spĺňa všetky hraničné podmienky.

7.2 Metóda spriemerňovania príkladov

Otázkou je, ako určiť etalón nejakej triedy. Vybrať nejakého reprezentanta triedy z príkladov, ktoré do danej triedy už patria? Alebo hodnoty atribútov etalónu triedy vypočítať nejakým spôsobom z hodnôt atribútov všetkých trénovacích príkladov danej triedy?

Najjednoduchší prístup sa nazýva metódou spriemerňovania príkladov. Táto metóda určí hodnotu každého atribútu etalónu danej triedy ako priemer všetkých hodnôt daného atribútu od všetkých trénovacích príkladov klasifikovaných v danej triede. Nech $\mathbf{e_i}$ je hodnota **i**-tého atribútu v etalóne. Potom:

$$e_i = \frac{1}{n}\sum_{j=1}^{n}x_{ij}$$

kde $\mathbf{x_{ij}}$ je hodnota **i**-tého atribútu v **j**-tom príklade a **n** je počet príkladov triedy, ktorej etalón hľadáme.

Pri nominálnych atribútoch sa $\mathbf{e_i}$ hodnota **i**-tého atribútu v etalóne určuje ako najčastejšie sa vyskytujúca hodnota v doméne daného atribútu.

7.3 Indukcia etalónov

V princípe je možné etalóny indukovať neinkrementálne a inkrementálne.

7.3.1 Neinkrementálna indukcia etalónov

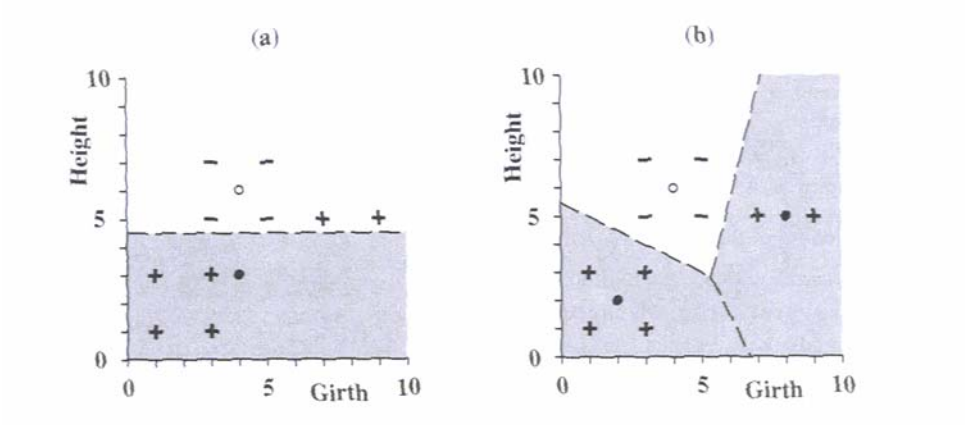
Príkladom neinkrementálnej metódy, ktorá využíva metódu spriemerňovania príkladov, je algoritmus **NCD (Nonincremental Induction of Competitive Disjunctions)** (Langley, 1996) a (Mitchell, 1997). Spriemerňovaním príkladov tvorí inicializačnú množinu etalónov. Tento algoritmus však, na rozdiel od metódy spriemerňovania príkladov, aj zaznamenáva, ktoré výsledky (etalóny) klasifikujú trénovacie príklady korektne a ktoré chybne. Naviac tento algoritmus umožňuje, aby jedna trieda bola reprezentovaná viacerými etalónmi. Čo je vhodné, keď sú príklady jednej triedy rozptýlené ďaleko od seba v niekoľkých skupinách alebo keď medzi tieto skupiny zasahuje skupina príkladov inej triedy. Príklad takejto trénovacej množiny je znázornený na Obr. 24.

Výsledky (etalóny reprezentované súťažiacimi disjunkciami) používa algoritmus NCD na klasifikáciu trénovacích príkladov. Tento algoritmus spriemerňuje trénovacie príklady pri výpočte etalónov. Je riadený chybou klasifikácie.

Vstupy: ***ISET****...množina trénovacích príkladov*
CSET*...množina dvoch alebo viacerých tried*
Výstup: ***DISJUNCTS****...kandidáti disjunkcií*

Procedúra: ***ncd(CSET,ISET)***
for *každú triedu* ***C*** *v* ***CSET***
spriemerni trénovacie príklady triedy ***C*** *do formy etalónu* ***C***
etalón ***C*** *pridaj do* ***DISJUNCTS***
if ***DISJUNCTS*** *korektne klasifikuje všetky trénovacie príklady v* ***ISET***
then *vráť* ***DISJUNCTS***
else *nech* ***OLDSET*** *je* ***ISET***
for *každú triedu* ***C*** *v* ***CSET***
for *každú triedu* ***D*** *v* ***CSET*** *rôznu od* ***C***
nech ***MISSED*** *sú tie trénovacie príklady z* ***D****, ktoré* ***DISJUNCTS***
klasifikuje ako členov ***C***
pridaj triedu ***D'*** *do* ***CSET***
for *každý trénovací príklad* ***I*** *v* ***MISSED***
označ ***I*** *ako člena* ***D'***
if ***ISET=OLDSET***
then *vráť* ***DISJUNCTS***
else ***ncd(CSET,ISET)***.

Keď algoritmus zistí, že niektoré trénovacie príklady boli chybne klasifikované, nahradí ich triedy pseudotriedami, ktoré síce majú odlišné interné mená, ale predikujú tú istú triedu. NCD produkuje novú množinu popisov pojmov, založenú na novom rozdelení príkladov do tried. V ďalšej iterácii znova pátra po chybných klasifikáciách a opakuje proces dovtedy, kým nie je generovaný súbor etalónov, ktorý korektne zaradí všetky príklady do ich tried. V prípade, že po určitom množstve iterácií nedôjde ku korektnej klasifikácii, algoritmus končí. Ďalšou možnou ukončovacou podmienkou je nemožnosť ďalšieho vylepšenia, napríklad preto, lebo sa riešenie dostalo do stavu, keď je toľko etalónov ako príkladov. Obr. 24 ilustruje chovanie metódy NCD v porovnaní s jednoduchým programom, ktorý spriemerňuje trénovacie príklady do formy etalónov pre každú triedu.



Obr. 24: Ilustrácia činnosti po a) výsledok spriemerňovania príkladov a zároveň výsledok prvej iterácie NCD b) výsledok NCD algoritmu.

Metóda spriemerňovania príkladov jednoducho určí etalóny jednotlivých tried spriemerňovaním. Pre triedu "+" čierny krúžok a pre triedu "-" biely krúžok. Deliaca hranica je čiarkovaná čiara, kolmá na spojnicu etalónov, vedená stredom tejto spojnice. Výsledný popis chybne klasifikuje dva príklady. Algoritmus NCD dokáže na rozdiel od prvej metódy spracovať túto situáciu. Chybne klasifikované príklady zaradí do novej pseudotriedy " +' " a spriemerní ich, čím vytvorí etalón triedy " +' ". Následne vymaže sporné príklady zo súhrnov triedy "+", aktualizuje etalón triedy „+" a deliace hranice. Tým vznikne množina popisov na Obr. 24 v časti b) ktorá korektne klasifikuje všetky trénovacie príklady a proces sa ukončí.

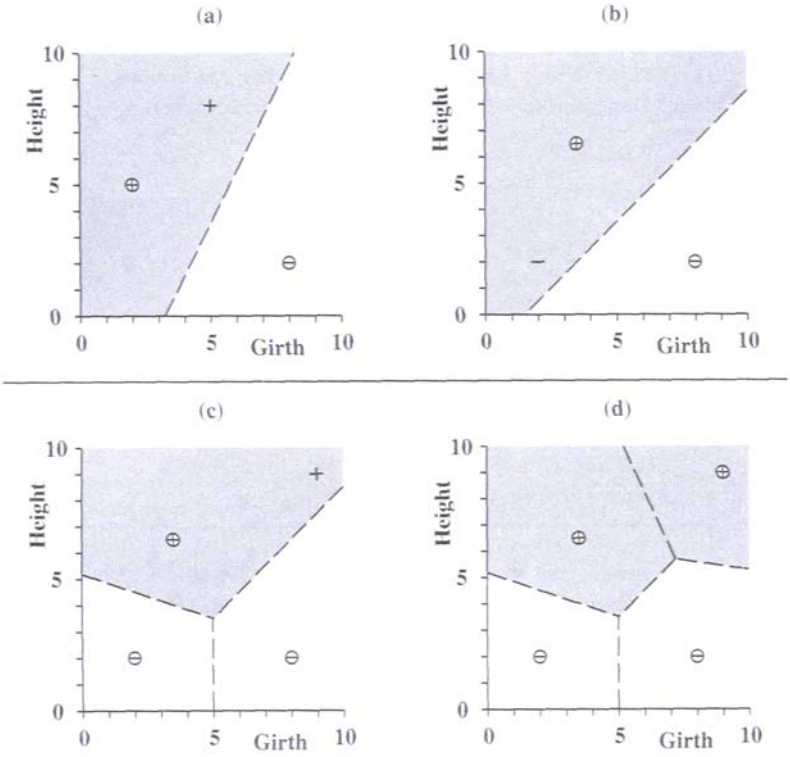
Algoritmus NCD môže byť kombinovaný aj s inými metódami, napríklad s jednoduchým Bayesovským klasifikátorom, ktorému bude venovaná nasledujúca kapitola s názvom "Pravdepodobnostný popis".

Napriek elegancii algoritmu NCD je možné ľahko navrhnúť trénovaciu množinu, ktorá spôsobí tomuto algoritmu ťažkosti. Napríklad trénovaciu množinu, v ktorej ležia etalóny dvoch susedných tried príliš blízko pri sebe. Tieto problémy je možné eliminovať napríklad tak, že

príklady ktoré sú blízko seba sa zoskupujú do zhluku. Pre každý zhluk sa môže spriemernením vytvoriť etalón. Pričom príklady sa spriemerňujú iba ak sú dosť blízko seba. To vyžaduje, aby používateľ špecifikoval parameter algoritmu - maximálnu vzdialenosť jednotlivých trénovacích príkladov, ktoré môžu byť zaradené do toho istého zhluku.

7.3.2 Inkrementálna indukcia etalónov

Myšlienka vytvárania zhlukov príkladov jednej triedy (nie všetkých) je implementovaná aj v algoritme inkrementálnej indukcie etalónov **ICD (Incremental Induction of Competitive Disjunctions)** (Bradshaw, 1987) a (Anderson-Matessa, 1992). Tento algoritmus používa taktiež princíp riadenia chybnou klasifikáciou. Chovanie algoritmu ICD je ilustrované na Obr. 25.



Obr. 25: Chovanie algoritmu ICD.

Keď príde nový trénovací príklad, algoritmus hľadá najlepšie pokrytie aktualizovanej trénovacej množiny disjunktívnymi etalónmi. Jedna trieda môže odpovedať jednému, alebo viacerým etalónom. Ak je toto pokrytie dosť dobré, teda v rámci vzdialenostného limitu, ICD spriemerňuje príklady do etalónu. Ak nie, pridá k disjunkcii etalónov nový etalón, ktorý je založený na hodnotách atribútov a triede nového trénovacieho príkladu.

Po zaregistrovaní jedného príkladu triedy "+" a jedného príkladu triedy "-", má algoritmus jeden etalón pre každú triedu, ako ukazuje Obr. 25 v časti a). Po zadaní ďalšieho trénovacieho príkladu triedy "+", ktorý patrí do oblasti "+" a v rámci vzdialenostného limitu pre etalón danej triedy, učiaci algoritmus aktualizuje etalón tejto triedy (zahrnie do spriemerňovania aj hodnoty nového príkladu) a taktiež reviduje deliace hranice. Ďalší príklad "-" v časti b) je chybne klasifikovaný, čo vedie ku generovaniu nového etalónu pre triedu "-" a následné revidovanie deliacich hraníc. Nový etalón pre triedu "-" je posunutý voči starému etalónu. Napokon posledný príklad "+" v časti c) obrázku je klasifikovaný správne, ale je mimo vzdialenostného limitu (viac ako päť jednotiek). Preto aj v tomto prípade je potrebné vytvoriť nový etalón pre triedu "+" a následne revidovať deliace hranice. Výsledné rozloženie disjunktných oblastí je znázornené na Obr. 25 v časti d).

Vstupy: ***ISET****...množina trénovacích príkladov*
Výstup: ***D****...disjunkcia etalónov (popisov jednotlivých oblastí)*

Procedúra: ***icd(ISET)***
nech ***D={}***
for *každý nový príklad* ***I*** *v* ***ISET***
nech ***C*** *je názov triedy spojenej s* ***I***
if *neexistuje etalón predikujúci triedu* ***C***
then *pridaj nový etalón do* ***D****, založený na* ***I****, predikujúci* ***C***
else *nech* ***T*** *je etalón v* ***D****, ktorý lepšie pokrýva* ***I***
if *etalón* ***T*** *predikuje triedu* ***C*** *a pokrytie medzi* ***I*** *a* ***T*** *spĺňa zdialenostný limit*
then *aktualizuj* ***T*** *zahrnutím* ***I***
else *pridaj nový etalón do* ***D****, založený na* ***I***
vráť ***D****.*

Veľkým nedostatkom algoritmu ICD je jeho citlivosť na dobré určenie vzdialenostného limitu ako parametra algoritmu. Ak používateľ zadá vzdialenostný limit príliš malý, algoritmus bude vytvárať viac etalónov ako je nutné a učenie bude veľmi pomalé. Na druhej strane ak bude tento parameter príliš veľký, potom sa zníži efektívnosť trénovacej postupnosti etalónov. Napríklad, môže sa stať, že dva príklady tej istej triedy sú od seba veľmi ďaleko (môžu byť, pripúšťa to príliš veľký vzdialenostný limit) a medzi nimi sa nachádza oblasť inej, zatiaľ neznámej triedy. Keď sa tieto dva príklady spriemernia, ich etalón padne do oblasti inej (cudzej) triedy. Algoritmus ICD nepozná žiadny spôsob revízie.

7.4 Úlohy na precvičenie:

1. Aký je najčastejšie používaný spôsob určovania hodnôt atribútov etalónu?
2. Akým spôsobom sa klasifikuje nový príklad pri reprezentácii pojmu etalónmi?
3. Charakterizujte algoritmus NCD a uveďte v čom sa zásadne líši *od ICD.*
4. Použitím algoritmu NCD určite pravidlá na klasifikáciu nových príkladov do dvoch tried "+" a "-", ak je daná nasledovná trénovacia množina:

	VÝŠKA	VLASY	OČI	TRIEDA
1	vysoký	blond	hnedé	+
2	vysoký	blond	modré	+
3	nízky	blond	modré	+
4	nízky	blond	hnedé	-
5	vysoký	ryšavé	modré	+
6	vysoký	tmavé	hnedé	-
7	nízky	tmavé	modré	-
8	nízky	tmavé	hnedé	-

7.5 Literatúra:

Anderson, J.R., Matessa, M.: Explorations of an incremental, Bayesian algorithm for categorization. Machine Learning, 9, 1992, 275-308.

Bradshaw, G.: Learning about speech sounds: The NEXUS project. Proc. of the Fourth International Workshop on Machine Learning, Irvine, CA: Morgan Kaufmann, 1987, pp.1-11.

Haton, J.P., Keane, M.., Manago, M.: Advances in case-based reasoning: Second European workshop. Berlin: Springer-Verlag, 1995.

Kolodner, J.L.: Case-Based Reasoning. San Francisco: Morgan Kaufmann, 1993.

Langley, P.: Elements of Machine Learning. Morgan Kaufmann Publishers, Inc. San Francisco, California, 1996, 419 pp.

Mitchell, T.M.: Machine Learning. The McGraw-Hill Companies, Inc. New York, 1997, 414 pp.

Watson,I.: Progress in case-based reasoning: First United Kingdom workshop. Berlin: Springer-Verlag, 1995.

8 PRAVDEPODOBNOSTNÝ POPIS

Reprezentácia pravdepodobnostným popisom je veľmi jednoduchá a flexibilná forma reprezentácie pojmu. Je spojená s Bayesovským klasifikátorom, ktorý sa dá jednoducho použiť v širokom spektre rôznych klasifikačných problémov. Snáď jedinou tienistou stránkou tejto reprezentácie je predpoklad vzájomnej nezávislosti atribútov, ktorý vo väčšine reálnych domén nie je splnený.

8.1 Reprezentácia a klasifikácia

Existuje množstvo úvodných prác k pravdepodobnosti a štatistike, ako napríklad (Casella-Berger, 1990). Ďalšie práce, ako (Maisel, 19971) a (Speigel, 1991) uvádzajú základy pravdepodobnosti a štatistiky vo vzťahu k strojovému učeniu. Reprezentácia pravdepodobnostným popisom používa:

- **$p(c_k)$**...apriórnu pravdepodobnosť **k**-tej triedy **c_k** a
- **$p(v_j/c_k)$**...podmienená pravdepodobnosť výskytu hodnoty **v_j** atribútu **j** v príklade **I** patriacom do triedy **c_k**.

Tieto podmienené pravdepodobnosti špecifikujú rozdelenie pravdepodobnosti pre každý atribút a každý popis triedy. V numerických doménach musí byť reprezentované spojité rozdelenie pravdepodobnosti pre každý atribút. Preto musíme predpokladať nejakú formu modelu, napríklad normálne rozdelenie.

Na klasifikáciu nového príkladu **I** môže byť aplikovaná Bayesova teoréma na určenie pravdepodobnosti, že príklad **I** patrí do triedy **c_k**:

$$p(c_k / I) = \frac{p(c_k)p(I / c_k)}{p(I)}$$

Vieme, že pravdepodobnosť javu môžeme určiť ako sumu podmienených pravdepodobností tohto javu inými javmi, ktoré tvoria úplný súbor. Nakoľko **I** je konjunkciou **v_j** hodnôt, p(I) v predchádzajúcej rovnici môžeme nahradiť nasledovne:

$$p(c_k / \wedge v_j) = \frac{p(c_k)p(\wedge v_j / c_k)}{\sum_i p(c_i)p(\wedge v_j / c_i)}$$

V menovateli výrazu je suma cez všetky triedy, pričom overovaná je trieda **c_k**. Podľa daného vzorca sa vypočíta táto pravdepodobnosť (zaradenia príkladu I do triedy **c_k**) pre každú

triedu resp. pre každý popis. Príklad je klasifikovaný do triedy s najvyššou aposteriórnou pravdepodobnosťou, ktorá je určená na základe apriórnej pravdepodobnosti tejto triedy a pozorovaných dát t.j. trénovacích príkladov. Bayesov systém klasifikácie je podrobnejšie popísaný v (Cheesman at al., 1988).

Ostal nám ešte jeden nevyriešený problém. Ako vypočítať pravdepodobnosť:

$$p(\wedge v_j / c_k).$$

Predpoklad nezávislosti atribútov nám dovolí nasledovnú substitúciu:

$$p(\wedge v_j / c_k) = \prod_j p(v_j / c_k).$$

Potom:

$$p(c_k / \wedge v_j) = \frac{p(c_k)\prod_j p(v_j / c_k)}{\sum_i p(c_i)\prod_j p(v_j / c_i)}.$$

Tento vzťah reprezentuje NAIVNÝ Bayesov klasifikátor, pretože predpokladá nezávislosť atribútov. Naivný Bayesov klasifikátor sa ukázal byť veľmi užitočnou metódou v mnohých praktických aplikáciách. Bayesov klasifikátor môže pracovať vcelku efektívne aj v tom prípade, keď podmienka nezávislosti atribútov nie je celkom splnená. V (Domingos-Pazzani, 1996) uskutočnili analýzu podmienok, pri ktorých naivný Bayesov klasifikátor bude schopný vykonávať "optimálnu klasifikáciu" aj v prípade nesplnenia podmienky nezávislosti atribútov. Optimálnej predikcii pomocou Bayesovho klasifikátora je venovaná aj práca (Opper-Haussler, 1991).

Iným spôsobom sa s neplatnosťou predpokladu nezávislosti atribútov vyrovnávajú Bayesove siete dôvery (Bayes belief networks). Pracujú s nezávislosťou podmnožín atribútov. Týmto sieťam je venované napríklad: (Cooper, 1990), (Cooper-Herskovist, 1992) a (Dagum-Luby, 1993).

Do rámca Bayesovskej klasifikácie spadajú aj metódy, ktoré priamo neaplikujú Bayesovu teorému. Dá sa dokázať, že za určitých podmienok minimalizácia štvorcovej chyby odpovedá MAP klasifikácii, čo je diskutované v (Duda-Hart, 1973).

Bayesova teoréma a informačná teória môžu byť použité na racionalizáciu princípu Minimálnej dĺžky popisu, ktorý uprednostňuje hypotézy s kratším popisom, viď (Rissanen, 1983) a (Rissanen, 1989). V ďalšej publikácii (Quinlan-Rivest, 1989) je opísané použitie tohto princípu na elimináciu preučenia rozhodovacieho stromu.

Do tejto oblasti môžeme zaradiť aj EM algoritmus, ktorý za pomoci pravdepodobnosti je schopný uskutočňovať učenie pri neznámych atribútoch. Tento algoritmus konverguje k lokálnemu maximu hypotézy (pojmu), prostredníctvom estimácie hodnôt stratených atribútov. Použitie metódy M-estimácie na odhad pravdepodobností je diskutované v (Cestnik, 1990).

Experimentálne výsledky porovnávania rôznych Baysovských prístupov s učením rozhodovacích stromov je možné nájsť v (Michie, 1994). V (Chauvin-Rumelhart, 1995) je uskutočnená Bayesovská analýza učenia pomocou neurónových sietí, založenom na algoritme spätného šírenia (Back propagation).

8.2 Indukcia naivného Bayesovho klasifikátora

Z predchádzajúcej podkapitoly vieme, že klasifikovať príklad **I** do triedy $\mathbf{c_k}$ môžeme, ak vieme vypočítať $\mathbf{p(c_k/I)}$ pre každú triedu. To vieme, ak máme k dispozícii hodnoty $\mathbf{p(c_k)}$ a $\mathbf{p(v_j/c_k)}$. Teda indukcia naivného Bayesovho klasifikátora spočíva v algoritme, ktorý by určil na základe frekvencií výskytov hodnôt atribútov a tried v trénovacích príkladoch:

- $\mathbf{p(c_k)}$ pre každú triedu $\mathbf{c_k}$
- $\mathbf{p(v_j/c_k)}$ pre každý možný pár trieda – hodnota atribútu.

Tento algoritmus môže byť tak inkrementálny ako aj neinkrementálny. Obidva prístupy dávajú identické výsledky. Niekedy sa môže stať, že trénovacie údaje produkujú nulovú hodnotu niektorej pravdepodobnosti, keďže sa daná hodnota daného atribútu v trénovacej množine nevyskytuje. Násobenie nám túto nulu prenáša do výsledku, čím ho skresľuje. Tento problém je možné riešiť tak, že sa nula nahradí nejakým veľmi malým číslom. Možno použiť napríklad **1/n**, kde **n** je nejaké veľké číslo, napríklad počet trénovacích príkladov.

Pozitívom Bayesovho klasifikátora je, že každá trieda je charakterizovaná jednoduchým pravdepodobnostným popisom. V dvojhodnotových doménach je to ekvivalentné predpokladu lineárnej separability. Negatívom je predpoklad nezávislosti atribútov.

Vhodnou oblasťou použitia Naivného Bayesovho klasifikátora je oblasť kategorizácie dokumentov (Paralič-Bednár, 2002).

8.3 Úlohy na precvičenie:

1. Uvažujme nový príklad P_i. Pomocou vzorca pre Bayesovu klasifikáciu vypočítame pravdepodobnosť triedy „+“ a „-“: $p(+,P_1)=\frac{\frac{1}{12}}{M}, p(-,P_i)=\frac{\frac{1}{25}}{M}$, kde M je nevyčíslený menovateľ. Je možné P_i klasifikovať na základe týchto informácií? Ak áno, do ktorej triedy?
2) Majme štyri trénovacie príklady triedy „+“. Tri z nich majú hodnotu atribútu Výška rovnú „vysoký“ a jeden „nízky“. Určite:
   p(vysoký, +)=.....?
   p(nízky,-)=.....?
3) Majme danú nominálnu doménu ľudí z úlohy na precvičenie číslo 7. v kapitole „Generovanie logických konjunkcií“. Pomocou týchto trénovacích príkladov natrénujte naivný Bayesovský

klasifikátor. Ďalej overte správnosť klasifikácie na trénovacej množine a klasifikujte všetky nové príklady, ktoré sa v danej doméne môžu vyskytnúť.

8.4 Literatúra:

Casella, G., Berger, R.L.: Statistical inference. Pacific Grove, CA: Wadsworth & Brooks/Cole, 1990.

Cestnik, B.: Estimating probabilities: A crucial task in machine learning. Proceedings of the Ninth European Conference on Artificial Intelligence, London: Pitman, 1990, pp. 147-149.

Chauvin, Y., Rumelhart, D.: Back propagation: Theory, architectures, and applications, (edited collection). Hillsdale NJ: Lawrence Erlbaum Assoc., 1995.

Cheesman, P. at al.: AutoClass: A Bayesian classification system. Proceedings of AAAI 1988, 1988, pp. 607-611.

Cooper, G.: Computational complexity of probabilistic inference using Bayesian belief networks (research note). Artificial Intelligence, 42, 1990, pp. 393-405.

Cooper, G., Herskovits, E.: A Bayesian method for the induction of probabilistic networks from data. Machine Learning, 9, 1992, pp. 309-347.

Dagum, P., Luby, M.: Approximating probabilistic reasoning in Bayesian belief networks is NP-hard. Artificial Intelligence, 60(1), 1993, pp. 141-153.

Domingos, P., Pazzani, M.: Beyond independence: Conditions for the optimality of the simple Bayesian classifier. Proc. of the 13th International Conference on Machine Learning, 1996, pp. 105-112.

Duda, R.O., Hart, P.E.: Pattern classification and scene analysis. New York: John Wiley & Sons, 1973

Maisel, L.: Probability, statistics, and random processes. Simon and Schuster Tech. Outlines. New York: Simon and Schuster, 1971.

Michie, D., Spiegelhalter, D.J., Taylor, C.C.: Machine learning, neural and statistical classification, (edited collection). New York: Ellis Horwood, 1994.

Opper, M., Haussler, D.: Generalization performance of Bayes optimal prediction algorithm for learning a perception. Physics Review Letters, 66, 1991, pp. 2677-2681.

Paralič, J., Bednar, P: Knowledge Discovery in Texts Supporting e-Democracy. In Proc. of the 6th IEEE International Conference on Intelligent Engineering Systems, INES 2002, Opatija, Croatia, May 2002, pp.327-332.

Quinlan, J.R., Rivest, R.: Inferring decision trees using the minimum description length principle. Information and Computation, 80, 1989, pp. 227-248.

Rissanen, J.: A universal prior for integers and estimation by minimum description length. The Annals of Statistics, 11(2), 1983, 416-431.

Rissanen, J.: Stochastic complexity in statistical inquiry. New Jersey: World Scientific Pub, 1989.

Speigel, M.R.: Theory and problems of probability and statistics. Schaum's Outline Series. New York: McGraw Hill, 1991.

Časť III

Učenie bez učiteľa

9 UČENIE ODMENOU A TRESTOM

Doteraz sme sa zaoberali metódami určenými na riešenie úloh klasifikačného typu. Učenie odmenou a trestom bolo motivované snahou o riešenie úloh sekvenčného typu. Sú to úlohy, v ktorých je daný počiatočný stav a konečný stav. Úlohou je nájsť resp. naučiť optimálnu cestu od počiatočného do koncového stavu.

Učenie odmenou a trestom je svojou podstatou nekontrolované učenie, pretože učiaci sa objekt nemá k dispozícii okamžitú spätnú väzbu o vhodnosti svojho rozhodnutia po každom kroku. Spätná väzba nechýba úplne, ale je k dispozícii až na konci celého procesu učenia vo forme výsledku snaženia (vyhral šachovú partiu resp nevyhral). Anglický ekvivalent **reinforcement learning (RL)** sa často používa bez prekladu.

9.1 Získavanie riadiacich znalostí

Aby sme definovali náš problém, musíme určiť čo je dané a čo chceme získať.

Dané sú: čiastočné znalosti problémovej domény
skúsenosti s prehľadávaním problémového priestoru.

Získavame: znalosti, ktoré umožňujú presné rozhodnutie v každom stave.

Tieto znalosti sa používajú resp. aplikujú na sekvenčnú úlohu učenia. Učenie spočíva vo vylepšovaní rozhodovania. Pričom dané rozhodnutia sa môžu vyskytovať tak v mozgu agenta ako aj vo fyzickom svete. Túto úlohu je možné považovať aj za klasifikačnú úlohu. Každý stav sa klasifikuje do nejakej triedy, pričom triedy reprezentujú možné rozhodnutia.

Riadiace znalosti je možné získavať rôznymi spôsobmi. Väčšina prác na túto tému predpokladá, že agent počas prehľadávania generuje vlastné experimenty. Ale niektoré prístupy odvádzajú svoje trénovacie príklady od sledovaného riešenia sekvenčnej úlohy uskutočneného doménovým expertom. Systém môže taktiež získať riadiace znalosti z vyhodnotenia úspešných ciest (vyriešené úlohy, výhry) a z vyhodnotenia neúspešných ciest (slepé uličky, slučky, prehry) ako aj z numerických informácií. Kvalita získaného riešenia sa dá posúdiť na základe:

- efektívnosti hľadaného riešenia
- spoľahlivosť plánov po realizácii v externom svete
- kvality návrhu

Akýkoľvek typ vylepšenia silne závisí na vykonaní presných rozhodnutí v každom stave. Na rozdiel od klasifikačných úloh sekvenčné úlohy vyžadujú viac krokov pred nájdením riešenia. Riešiteľ úlohy sa dozvie, či jeho rozhodnutie bolo správne (nesprávne) resp. presné (nepresné) až dlho po tom čo ho vykonal. To vedie k dvom základným istotám v učení tohto typu:

- priradenie odmeny (kreditu) dobrým rozhodnutiam alebo akciám
- priradenie pokuty nevhodným rozhodnutiam alebo akciám

Priradenie odmeny alebo pokuty je potrebné kvôli absencii okamžitej spätnej väzby.

Jedným z prístupov k získavaniu riadiacich znalostí je **učenie odmenou a trestom**. Tento druh učenia sa sústreďuje na preferenciu znalostí na výber operátora. Dané preferencie sa realizujú pomocou ohodnocovacej funkcie, ktorá prideľuje stavom odmenu resp. trest. **Odmena (reward)** sa prideľuje viac žiadaným stavom. Čím vyššia odmena, tým žiadanejší stav. Na druhej strane je tu t**rest (negative reward)**, ktorý sa prideľuje menej žiadaným alebo nežiadaným stavom. Cieľom učenia je priblížiť sa k prvým a vyhnúť sa druhým. Učenie predstavuje získanie riadiacej stratégie, ktorá vždy povedie k stavu s najvyššou odmenou. Riadiaca stratégia sa získava zo skúseností s odmenami alebo trestami, ktoré sa vyskytli v skúmaných sekvenciách stavov, získaných aplikáciou danej sekvencie operátorov.

9.2 Tabuľkový prístup

9.2.1 Reprezentácia a použitie

Tabuľkový prístup je najjednoduchší z použiteľných prístupov k danému typu učenia. Tento prístup ukladá riadiace znalosti v tabuľke. Nasleduje príklad takejto tabuľky:

STAV	OPERÁTOR	ODMENA
(kocka **a**)(kocka **b**)(kocka **c**)(stôl **t**)(**b** na **a**) (**a** na **c**)(**c** na **t**)(prázdne **b**)(prázdne rameno)	(zlož **b** z **a**)	0.1
(kocka **a**)(kocka **b**)(kocka **c**)(stôl **t**)(**a** na **c**) (**c** na **t**)(prázdne **a**)(drží **b**)	(polož **b** na **t**) (polož **b** na **a**)	0.2 0.0
. . .	. . .	. . .
(kocka **a**)(kocka **b**)(kocka **c**)(stôl **t**)(**b** na **c**) (**c** na **t**)(prázdne **b**)(drží **a**)	(polož **a** na b) (polož **a** na **t**)	0.9 0.0

Každý vstup tabuľky popisuje možný pár **stav – akcia**, spolu s očakávanou **odmenou**. Odmena reprezentuje vhodnosť vykonania danej **akcie** v tom **stave**. Napríklad doména s počtom stavov **p** a počtom akcií **a** bude vytvára tabuľku, ktorej veľkosť je daná súčinom **p** a **a**. Ak sa v danom stave akcia nedá použiť, príslušná bunka tabuľky bude prázdna.

Takúto tabuľku je možné znázorniť ako orientovaný ohodnotený graf, ktorého uzly znázorňujú stavy a hrany znázorňujú akcie. Jednotlivé hrany sú ohodnotené. Hodnotiace skóre (odmena alebo trest) je to čo sa učí. Pri riešení sekvenčných úloh sa používa typicky dopredné zreťazené prehľadávanie. To vyžaduje:

- hľadanie tabuľkových vstupov pre aktuálny stav
- výber akcie s najvyšším skóre
- aplikovanie vybratej akcie na dosiahnutie nového stavu.

Tento trojkrokový cyklus sa opakuje dovtedy, kým sa nedosiahne požadovaný stav, alebo nie je splnená iná ukončovacia podmienka.

Na podporu prieskumu alternatívnych ciest niektoré systémy používajú stochastickú schému, ktorá vyberá akcie pomocou pravdepodobnostnej funkcie ich predikovanej odmeny.

9.2.2 Proces učenia

Algoritmus mení predikovanú odmenu uchovávanú v tabuľke stavov a akcií na základe skúsenosti. Vstupy obvykle štartujú s ľubovoľnými alebo primeranými (odhadnutými na základe apriórnych znalostí o probléme) hodnotami. Skóre pre každý pár stav-akcia **(s,a)** sa mení zakaždým, keď riešiteľ úlohy aplikuje akciu **a** v stave **s**. To vytvorí novú výslednú odmenu, ktorá sa použije k aktualizácii daného vstupu. Napríklad, ak je k dispozícii rada učiteľa a ten vyznačí sekvenciu operátorov, ktorá predstavuje vhodné riešenie, potom sa všetky odmeny na danej ceste zvýšia. Odmeny sú vzťahované k vetvám t.j. spojniciam stavov.

Q-learning je algoritmus učenia odmenou a trestom, ktorého základom je nasledovná aktualizačná schéma. Táto schéma definuje spôsob, akým sa menia ohodnotenia akcií v jednotlivých stavoch:

$$\Delta Q(s,a) \leftarrow \beta[r(s,a) + \gamma U(s') - Q(s,a)]$$

kde:

- $0<\gamma<1$ je redukčný faktor
- $0<\beta<1$ je faktor rýchlosti učenia
- **Q(s,a)** je **interná odmena** resp. tabuľkový vstup pre stav **s** a akciu **a**. Je predmetom procesu učenia.
- **s'** je výsledný stav po aplikácii akcie **a** v stave **s**
- **r(s,a)** je **externá odmena**, vyplývajúca z aplikácie **a** v **s**. Externá odmena môže byť daná učiteľom, alebo môže byť získaná z trénovacích príkladov. Predstavuje apriórne vedomosti o riešenej úlohe. Nie je predmetom učenia. Zadáva sa na začiatku, pred spustením procesu učenia a nemusí byť priradená každému páru **(s,a)**. Najčastejšie sa externá odmena priradí k stavu, v ktorom končí sekvencia operátorov (akcií), z ktorej sa práve učí. Ak to bola úspešná sekvencia, potom je odmena kladná. Inak môže byť aj záporná.
- **U(s')** je maximum z očakávaných odmien **Q(s',a')** pre všetky akcie **a'**, ktoré môžu byť aplikované v stave **s'**.

Teda, ako z vyššie uvedeného vyplýva, pracuje sa z dvoma tabuľkami: **r(s,a)** tabuľkou a **Q(s,a)** tabuľkou. Stratégia Q-learning vypočítava súčet externej odmeny **r(s,a)** a redukovanej maximálnej odmeny, očakávanej v nasledujúcom stave **γU(s')**. Od tohto súčtu odpočíta aktuálny vstup **Q(s,a)**. Výsledok násobí faktorom rýchlosti učenia a získanou hodnotou aktualizuje spracovávaný vstup nasledovne:

$$Q(s,a) \leftarrow Q(s,a) + \Delta Q(s,a)$$

Faktor rýchlosti učenia **β** určuje rozsah, v ktorom učiaci sa môže revidovať tabuľkové vstupy. Redukčný faktor **γ** špecifikuje dôležitosť pomeru aktuálna verzus očakávaná odmena.

Pri dostatočnom počte trénovacích príkladov vyššie uvedená aktualizačná schéma konverguje k nasledovnému výrazu:

$$Q(s,a) = r(s,a) + \gamma U(s').$$

Cieľom učenia je transformovať počiatočnú ohodnocovaciu funkciu (externú odmenu, ktorá môže byť priama, alebo nemonotónna v závislosti od cieľa) na revidovanú ohodnocovaciu funkciu (internú odmenu, ktorá rastie monotónne). Pre každý príklad (sekvenciu operátorov) je potrebné mnohonásobné opakovanie učenia kým sa jednotlivé hodnoty interných odmien vhodne propagujú pozdĺž danej cesty v grafe.

Po dostatočnom množstve experimentov sa môže učiaci algoritmus premiestniť do najžiadanejšieho stavu z akéhokoľvek miesta v problémovom priestore. Učenie je možné urýchliť zmenou faktoru rýchlosti učenia **β** počas učenia. Začína sa s veľkou hodnotou faktoru rýchlosti učenia, čím sa umožní hrubá aproximácia v počiatočnej etape učenia. Postupne sa tento parameter znižuje, čím sa umožní v posledných etapách učenia presnejšie ladenie.

Stratégia Q-learning nadväzuje na metódy dynamického programovania a Markovovského rozhodovacieho procesu (Bellman, 1961), (Blackwell, 1965). Iná obdoba tejto stratégie, pri ktorej nie sú dopredu presne známe funkcie aktualizácie odmien, je uvedená vo (Watkins, 1989).

9.3 Bucket brigade

Táto učiaca stratégia (Holland, 1986) je ďalším bežným algoritmom RL, ktorý predpokladá, že iba niektoré stavy majú priradené externé odmeny. Aj tu má každý pár **(s,a)** priradené číslo – internú odmenu, ktorá odráža odhad žiadanosti akcie **a** v stave **s**. Zakaždým, keď učiaci sa subjekt aplikuje vybranú akciu **a** v stave **s**, algoritmus zníži hodnotu **Q(s,a)** aktuálneho stavu o nejakú časť **f**. Tým istým dychom zvýši odmenu **Q(s,a)** o tú istú časť pre predchádzajúci stav. Prvý stav neodovzdá časť svojej hodnoty nikomu a dostane ju od nasledujúceho stavu. Na druhej strane posledný stav odovzdá časť odmeny predchádzajúcemu a nedostane ju od nikoho.

Pred dosiahnutím stavu ktorý má nejakú externú odmenu **r(s)**, algoritmus zníži skóre pre aktuálny stav o hodnotu **f*r(s,a)**, čím nasmeruje riešiacu cestu k nemu.

9.4 Poznámky k učeniu odmenou a trestom

Tento spôsob učenia má množstvo atraktívnych vlastností, ale aj nevýhod. K výhodám patria:

- nepožaduje znalosti o efektívnosti operátorov
- dokáže zvládnuť neurčité a zašumené domény
- môže spolupracovať s externým svetom.

Medzi nevýhody patria:

- tendencia k nízkej rýchlosti učenia, zvlášť pri dlhých riešiacich cestách
- závislosť na postupnom šírení odmien spätne pozdĺž hľadanej cesty
- riešiteľ úlohy potrebuje prechádzať problémovým priestorom veľa krát, kým odmeny dosiahnu niektoré časti tohto priestoru
- požiadavka veľkého množstva trénovacích ciest na dosiahnutie zmysluplných vstupov pre každý pár (s,a)
- nutnosť uchovávať aspoň jeden vstup pre každý stav spôsobuje exponenciálny nárast veľkosti potrebnej pamäti v prípade narastania dĺžky riešiacej cesty, resp. zložitosti úlohy.

Jednou z odpovedí na tieto problémy je riešenie, v ktorom programátor rozdelí problémový priestor na zmysluplné segmenty a trénuje učiaci systém oddelene nad každým segmentom. Individuálne cesty v jednotlivých segmentoch sú potom prirodzene kratšie, z čoho vyplýva včaššie dosiahnutie odmeny.

Učenie odmenou a trestom je stále aktívnou výskumnou oblasťou. Napríklad (Maclin, Shavlik, 1996) popisujú prístup v ktorom agent RL dokáže akceptovať neúplné rady od učiteľa. (Lin, 1992) sa pokúša o učenie generovaním odhadov postupností akcií. (Dietterich-Flann, 1995) integruje RL s metódami založenými na vysvetľovaní. (Ring, 1994) rozširuje túto oblasť o kontinuálne učenie agentom. Prehľady výskumov na tomto poli je možné nájsť v (Kaelbling et al., 1996) a (Barto et al., 1995).

9.5 Úlohy na precvičenie:

1. Čo je podstatou učenia na základe odmeny a trestu a akým spôsobom sa aktualizujú odmeny?
2. Čo obsahujú stĺpce, riadky a bunky tabuľky v tabuľkovom prístupe k učeniu odmenou a trestom?
3. V čom spočíva hlavný rozdiel medzi Q-learning a vedrovou brigádou (bucket brigade)?
4. Učenie odmenou a trestom je vhodné na riešenie úloh typu:
   a/ klasifikačného
   b/ sekvenčného.

9.6 Literatúra:

Barto, A, Bradtke, S., Singh, S.: Learning to act using real-time dynamic programming. Artificial Intelligence, Special volume: Computational research on interaction and agency, 72(1), 1995, pp. 81-138.

Bellman, R.: Adaptive control processes. Princeton, NJ: Princeton University Press, 1961.

Blackwell, D.: Discounted dynamic programming. Annals of Mathematical Statistics, 36, 1965, pp. 226-235.

Dietterich, T.G, Flann, N.S.: Explanation-based learning and reinforcement learning: A unified view. Proc. of the 12th International Conference on Machine Learning, San Francisco: Morgan Kaufmann, 1995, pp. 176-184.

Holland, J.H.: Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, Carbonell

Mitchell (Eds.), Machine learning: An artificial intelligence approach, Vol 2, San Francisco: Morgan Kaufmann, 1986.

Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of AI Research, 4, 1996, pp. 237-285.

Lin, L.J.: Hierarchical learning of robot skills by reinforcement. Proc. of the International Conference on Neural Networks, 1993.

Maclin, R., Shavlik, J.W.: Creating advice-taking reinforcement learners. Machine Learning, 22, 1996, pp. 251-281.

Ring, M.: Continual learning in reinforcement environments (Ph.D. dissertation). Computer Science Department, University of Texas at Austin, Austin, TX, 1994.

Watkins, C.: Learning from delayed rewards (Ph.D. dissertation). King's College, Cambridge, England, 1989.

10 ZHLUKOVANIE

10.1 Definícia problému zhlukovania

Techniky zhlukovania (clustering) sa aplikujú v prípade, keď jednotlivé trénovacie príklady neobsahujú vopred informáciu o triede a teda trénovacie príklady sú rozdeľované do prirodzených skupín resp. zhlukov. Hovoríme, že ide o techniky nekontrolovaného učenia, keďže neexistuje spätná väzba vo forme zadanej triedy. Na začiatku máme zbierku, resp. kolekciu objektov. Pod objektom môžeme rozumieť napr. trénovací príklad.

Úlohou zhlukovania, ako učenia nekontrolovaného typu, je vytvoriť množinu zhlukov a jednotlivé trénovacie príklady zaradiť do týchto zhlukov. Ide o to združiť trénovacie príklady zmysluplným spôsobom do skupín, teda zhlukov, pričom sa vykryštalizuje aj počet potrebných skupín alebo zhlukov. Avšak niekedy je počet zhlukov daný dopredu. Niektoré algoritmy tento počet vyžadujú. Inak povedané, cieľom zhlukovania je organizácia trénovacích príkladov, ktorá spĺňa nejaký štandard kvality ako napr. maximalizáciu podobnosti trénovacích príkladov v tom istom zhluku. Pre algoritmy založené na meraní podobnosti príkladov musíme definovať pred začatím procesu zhlukovania nejakú mieru podobnosti trénovacích príkladov, ktorá bude v ďalšom predmetom záujmu.

V procese zhlukovania je potrebné určiť zhluky (clusters), definovať ich a zaradiť nový príklad do zhlukov. V rámci zhlukovania je niekedy možné predikovať atribúty zhluku, t.j. určiť tie hodnoty atribútov, ktoré sú typické pre definovaný zhluk. Táto množina hodnôt atribútov môže a nemusí charakterizovať aj konkrétny trénovací príklad z trénovacej množiny. Možnosť predikcie atribútov zhluku závisí od spôsobu popisu zhluku. Z tohto hľadiska rozoznávame nasledovné typy zhlukov:

- Zhluk daný enumeráciou teda vymenovaním trénovacích príkladov patriacich do daného zhluku. Predstavuje extenzionálnu reprezentáciu pojmu. Nový príklad je ťažké zaradiť do hierarchie zhlukov. Obtiažne sa predikujú atribúty zhluku. Jedna z možností predikcie hodnôt atribútov, typických pre zhluk je predikcia pomocou štatistických mier.
- Zhluk reprezentovaný etalónom, pričom to nemusí byť jeden z príkladov, je príkladom intenzionálnej reprezentácie pojmu. Nový príklad sa zaradí do zhluku reprezentovaného najbližším etalónom. Atribúty sú predikované podľa etalónu.
- Zhluk daný distribúciou pravdepodobností nad priestorom možných hodnôt atribútov je intenzionálnou reprezentáciou pojmu. Nový príklad sa zaradí do najpravdepodobnejšieho zhluku. Predikované sú hodnoty atribútov s najväčšou pravdepodobnosťou.
- Zhluk predstavovaný množinou nutných a postačujúcich podmienok (conceptual clustering). Je potrebné nájsť také zhluky, ktorých popisy sú jednoduché. Nový príklad sa zaradí do zhluku, ktorého definíciu spĺňa. Atribúty sú priamo predikované definíciou zhluku. Ide o intenzionálnu reprezentáciu pojmu.

Existuje niekoľko odlišných spôsobov ako reprezentovať výsledky zhlukovania:

- Identifikované zhluky sú disjunktné t.j. trénovací príklad môže byť zaradený iba do jedného zhluku.
- Identifikované zhluky sa prekrývajú, teda príklad môže byť zaradený do niekoľkých zhlukov.
- Identifikované zhluky sú pravdepodobnostné, čo znamená, že trénovací príklad prináleží ku každému zhluku s určitou pravdepodobnosťou.
- Identifikované zhluky sú hierarchické, teda všetky príklady sú najprv zaradené do zhluku na najvyššej úrovni a potom cestou dolu v hierarchii zhlukov až po úroveň individuálnych príkladov.

V teórii zhlukovania sú rozoznávané nasledovné základné prístupy k zhlukovaniu:

Iteratívne zhlukovanie založené na vzdialenosti (iterative distance-based clustering). Ide o klasickú metódu, ktorá rozdeľuje trénovacie príklady do disjunktívnych zhlukov podľa vzdialenosti. Príkladom tohto prístupu ku zhlukovaniu je **aglomeratívna zhlukovacia stratégia**, ktorá môže byť rozšírená na symbolické (binárne, nominálne) domény.

Konceptuálne zhlukovanie (conceptual clustering) využíva algoritmy strojového učenia. Typickým reprezentantom je CLUSTER/2. Je to jednoduchý a priamy algoritmus, ktorý sa používal niekoľko dekád.

Hierarchické inkrementálne zhlukovanie (hierarchical incremental clustering). Tento prístup bol vyvinutý v osemdesiatych rokoch a stelesňuje ho pár systémov: COBWEB (pre nominálne atribúty) a CLASSIT (pre numerické atribúty). Tieto metódy pracujú s hierarchickým zoskupovaním trénovacích príkladov. Pri vytváraní zhlukov merajú ich kvalitu mierou, ktorá sa nazýva užitočnosť zhluku.

Pravdepodobnostné zhlukovanie (probability-based clustering) založené na rozličných rozdeleniach pravdepodobností pre každý zhluk, v dôsledku čoho sú príklady, na rozdiel od predchádzajúcich troch metód, zatrieďované pravdepodobnostne a nie deterministicky.

Okrem týchto základných metód existuje takzvané obsiahle alebo komplexné zhlukovanie (comprehensive clustering), založené na kombinácii základných metód. Typickým predstaviteľom tohto prístupu k zhlukovaniu je systém AutoClass.

V literatúre sa uvádza aj takzvané neurčité zhlukovanie (fuzzy clustering), ktoré pripúšťa príslušnosť jedného trénovacieho príkladu ku viacerým zhlukom. Priradzovanie trénovacieho príkladu k zhluku sa uskutočňuje pomocou funkcie príslušnosti (membership function) (Zadeh, 1965).

10.2 Iteratívne zhlukovanie založené na vzdialenosti

Jedným z najstarších prístupov k problému zhlukovania je **numerické taxonomické zhlukovanie**. Numerické metódy predpokladajú reprezentáciu trénovacích príkladov v tvare množiny numerických atribútov. V takom prípade je možné trénovací príklad, predstavovaný

vektorom N numerických hodnôt atribútov, zobraziť v N dimenzionálnom priestore ako bod. Miera podobnosti môže byť potom daná **euklidovskou vzdialenosťou** medzi bodmi v priestore. Táto mierka je najznámejšia a najpoužívanejšia pri spojitých numerických hodnotách atribútov. Jej špeciálnym prípadom je Minkovského metrika. Podrobnejšia definícia je v (Jain at al., 1999).

10.2.1 Aglomeratívna zhlukovacia stratégia

Takto definovanú mieru podobnosti používa aj **aglomeratívna zhlukovacia stratégia** (Mitchell, 1997), (Rauber-Pampalk-Paralič, 2000). Táto stratégia sa zameriava na budovanie zhlukov od zdola nahor. Spodnú vrstvu v tomto prípade predstavuje jednotlivé trénovacie príklady. Kategórie sú vytvárané nasledovne:

- Vyskúšajú sa všetky páry trénovacích príkladov, vyberú sa páry s najvyšším stupňom podobnosti a vytvoria sa z nich zhluky.
- Definujú sa atribúty každého zhluku. Táto definícia zhluku predstavuje nejakú funkciu (napríklad aritmetický priemer) hodnôt atribútov jednotlivých členov zhluku. Jednotlivé trénovacie príklady sa nahradia zhlukovou definíciou svojho zhluku.
- Tento proces sa opakuje, až kým všetky trénovacie príklady nie sú redukované do jedného zhluku.

Výsledkom tohto algoritmu je binárny strom, ktorého listové uzly znázorňujú trénovacie príklady a vnútorné uzly znázorňujú zhluky rastúcej veľkosti. Uvedený algoritmus môžeme rozšíriť aj na trénovacie príklady reprezentované množinou symbolických atribútov (binárny, nominálny, atď.). Toto rozšírenie prináša problém, ako merať podobnosť symbolických hodnôt. Zmysluplným prístupom je definovanie podobnosti dvoch trénovacích príkladov ako pomer vyskytujúcich sa zhodných hodnôt atribútov ku počtu všetkých atribútov.

Majme dané trénovacie príklady:	objekt1 = {malá, červená, lopta}
	objekt2 = {malá, modrá, lopta}
	objekt3 = {veľká, čierna, lopta}
Podľa uvedenej metriky by:	podobnosť (objekt1, objekt2) = 2/3
	podobnosť (objekt1, objekt3) = 1/3
	podobnosť (objekt2, objekt3) = 1/3

Uvádzané riešenie je veľmi jednoduché. Problémom je ako prispôsobiť miery používané pre spojité numerické hodnoty atribútov na použitie pre diskrétne numerické hodnoty alebo nominálne hodnoty atribútov. (Wilson-Martinez, 1997) navrhli kombináciu modifikovanej Minkowského metriky a vzdialenosti založenej na súčte zhodných hodnôt numerického atribútu. V (Diday-Simon, 1976) a v (Ichino-Yaguchy, 1994) sú prezentované mnohé ďalšie metriky podobnosti medzi vzormi reprezentovanými tak kvantitatívnymi ako aj kvalitatívnymi vlastnosťami.

Zhlukovanie založené na podobnosti nevystihne adekvátne základnú úlohu sémantických znalostí. V definovaných kategóriách často nemajú všetky atribúty rovnakú váhu a podobnostné metriky narábajú so všetkými atribútmi rovnako. V mnohých kontextoch je miera určitosti resp.

vierohodnosti atribútov jednotlivých trénovacích príkladov dôležitejšia ako iné charakteristiky. Ľudské kategórie sú v oveľa väčšej miere závislé na cieli zhlukovania a kvalite apriórnych znalostí domémy ako na vonkajšej podobnosti. Tradičné zhlukovacie algoritmy nedokážu uspokojivým spôsobom vziať do úvahy ciele a znalosti domény a taktiež neuspejú pri tvorbe zmysluplných sémantických vysvetlení generovaných zhlukov.

10.3 Konceptuálne zhlukovanie

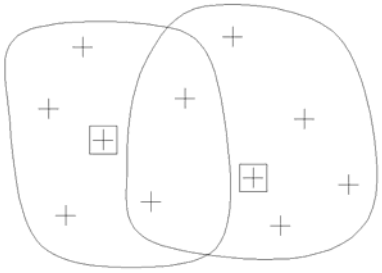
Úspešnejšie sa s týmito problémami vie vysporiadať **konceptuálne zhlukovanie**. Snaží sa ich riešiť použitím algoritmov strojového učenia na generovanie všeobecných definícií popisov pojmov a použitie znalostí okolia pri formovaní zhlukov. Príkladom takéhoto prístupu je algoritmus CLUSTER/2 (Michalski-Stepp, 1983a), (Michalski-Stepp, 1983b).

Tento algoritmus bol navrhnutý pánmi Michalským a Steppom v roku 1983. Najprv je potrebné špecifikovať počet zhlukov, ktoré majú byť generované. Tento počet udáva parameter **K**. Algoritmus formuje **K** zhlukov v okolí **K** jadrových trénovacích príkladov. Parameter **K** môže byť daný alebo modifikovaný používateľom. CLUSTER/2 taktiež hodnotí výsledné zhluky a v prípade potreby vyberá nové jadrá zhlukov. Tento proces opakuje, kým nie sú splnené požadované kritériá kvality. Algoritmus pracuje v nasledujúcich krokoch:

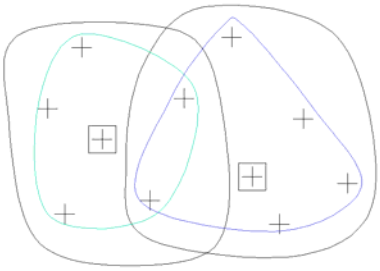
1) Vyber **K** jadier z pozorovaných trénovacích príkladov. Jadrá môžu byť vybrané ľubovoľne alebo podľa nejakej výberovej funkcie.
2) Pre každé jadro použi jadro ako pozitívny príklad a všetky ostatné jadrá ako negatívne príklady. Potom generuj maximálne všeobecnú definíciu, ktorá pokrýva všetky pozitívne a žiadny negatívny príklad. Tento postup môže viesť k viacnásobnej klasifikácii ostatných (nie jadrových) trénovacích príkladov.
3) Klasifikuj všetky príklady v trénovacej množine podľa generovaných popisov zhlukov. Nahraď každý maximálne všeobecný popis (definíciu) maximálne špecifickým popisom, ktorý pokrýva všetky trénovacie príklady zhluku. To znižuje pravdepodobnosť, že sa zhluky budú prekrývať nad zatiaľ neznámymi príkladmi.
4) Zhluky sa stále môžu prekrývať nad danými príkladmi. CLUSTER/2 zahŕňa aj podprogram pre úpravu prekrývajúcich sa definícií.
5) Použitím vzdialenostnej metriky, vyber nové jadrá bližšie k centru každého zhluku. Vzdialenostná metrika môže byť podobná metrike podobnosti, ktorá bola uvádzaná vyššie.
6) Použitím nových jadier opakuj kroky 1 až 5. Zhlukovanie ukonči, keď sú zhluky definované uspokojivo resp. kvalitne. Typickou metrikou kvality je komplexnosť všeobecného popisu zhlukov.
7) Ak zhluky nie sú definované uspokojivo a v priebehu niekoľkých iterácií sa neobjavilo žiadne vylepšenie, vyber nové jadrá radšej bližšie k okraju zhluku ako k jeho centru a opakuj kroky 1 až 5.

Práca algoritmu CLUSTER/2 je prezentovaná nasledovnými obrázkami:

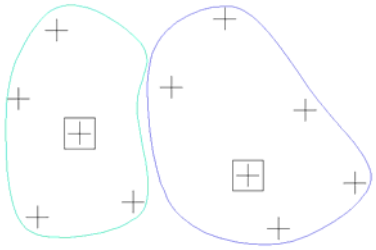
Obr. 26: Prvý krok CLUSTER/2



Obr. 27: Druhý krok CLUSTER/2



Obr. 28: Tretí krok CLUSTER/2



Obr. 29 : Štvrtý krok CLUSTER/2

Obr. 26 predstavuje prvý krok algoritmu t.j. výber jadrových trénovacích príkladov. Ďalší obrázok štvorice znázorňuje formuláciu maximálne všeobecných popisov (definícií), teda druhý krok algoritmu. Obr. 28 je ilustráciou výmeny maximálne všeobecných popisov za maximálne špecifické popisy. Posledný obrázok štvorice je ukážkou riešenia konfliktu záujmov podľa štvrtého kroku algoritmu.

Tento algoritmus sa za vhodných podmienok môže stať slušným požieračom strojového času. Niekedy vyžaduje veľké množstvo iterácií. Sú známe aproximácie tohto algoritmu, ktoré ho značne urýchľujú za cenu nižšej kvality výsledných zhlukov.

10.4 Hierarchické inkrementálne zhlukovanie

Pri iteratívnom zhlukovaní založenom na podobnosti je nutné v každej iterácii znova spracovať celú množinu údajov. V ďalšom sa budeme venovať metódam, ktoré pracujú inkrementálne, príklad za príkladom. Tieto metódy formujú hierarchický strom z celej množiny

údajov. Na začiatku strom obsahuje iba koreňový uzol. Príklady prichádzajú postupne, jeden za druhým a strom je primerane aktualizovaný v každej etape. Táto aktualizácia môže predstavovať iba hľadanie toho pravého miesta na uloženie nového príkladu, alebo môže ísť o radikálnu reštrukturalizáciu časti stromu. Kľúčom k rozhodnutiu ako a kde aktualizovať je užitočnosť kategórie resp. zhluku. Tomuto pojmu sa ešte budeme venovať.

Keď ľudské kategórie definujeme prostredníctvom nutných a postačujúcich podmienok, bude to mať jednu nevýhodu. Nebudeme môcť rozlišovať stupne členstva v kategórii alebo v zhluku. Budeme rozlišovať iba či trénovací príklad je alebo nie je členom zhluku.

Rodinná teória podobnosti je príkladom prístupu, ktorý nedefinuje zhluky nutnými a postačujúcimi podmienkami pre členstvo, ale komplexným systémom podobnosti medzi členmi. Bola navrhnutá Wittgensteinom v roku 1953. Základným stupňom kategórie resp. zhluku je klasifikácia najbežnejšie používaná v opisovaných trénovacích príkladoch. Napr. kategória "stolička" je bežnejšia ako jej zovšeobecnenie "nábytok", alebo jej špecifikácia "kancelárska stolička". "Auto" je bežnejší pojem ako "vozidlo" alebo "sedan".

Na uvedenej rodinnej teórii podobnosti je založený aj algoritmus **COBWEB** (Fisher, 1987a), (Fisher, 1987b). Tento algoritmus bol navrhnutý Fisherom v roku 1987. Možno nepredstavuje ideálny model ľudského poznávania, ale možno ho s úspechom použiť pre základné úrovne kategorizácie a stupne členstva v kategóriách resp. v zhlukoch. Algoritmus COBWEB reprezentuje zhluky ako distribúcie pravdepodobností. Je to inkrementálny algoritmus a teda nevyžaduje, aby boli všetky príklady dostupné pred začatím učenia. Venuje sa nielen generovaniu zhlukov, ale aj problému stanovenia korektného množstva zhlukov a vytvoreniu hierarchie týchto zhlukov. Na určenie počtu zhlukov, hĺbky hierarchie zhlukov a na zaradenie nového príkladu do zhluku používa globálne metriky kvality.

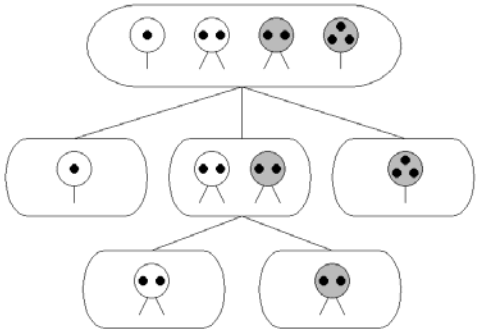
Keď COBWEB dostane nový príklad zvažuje celkovú kvalitu tak zaradenia nového príkladu do existujúceho zhluku, ako aj kvalitu modifikácie hierarchie zhlukov, aby umožňovala zaradenie nového príkladu. Jednoducho povedané uvažuje, či je užitočnejšie nový príklad zaradiť do niektorého z existujúcich zhlukov, vytvoriť preň nový zhluk, alebo modifikovať hierarchiu zhlukov. Pod modifikáciou hierarchie rozumie: spojenie dvoch zhlukov (a následné zaradenie nového príkladu do tohto spojenia) alebo rozdelenie niektorého zhluku (a následné zaradenie nového príkladu do jedného z nových zhlukov.

Kritérium, ktoré algoritmus používa na ohodnotenie kvality zhluku, sa nazýva **užitočnosť zhluku**. Nasledujúca podkapitola je venovaná spôsobom výpočtu tejto ohodnocovacej funkcie.

Algoritmus COBWEB inicializuje vytváranie hierarchie zhlukov vytvorením jedného zhluku, ktorého hodnoty atribútov sú totožné s hodnotami atribútov prvého trénovacieho príkladu. Pre každý ďalší trénovací príklad algoritmus začína s koreňovým zhlukom a premiestňuje sa pozdĺž stromu. Na každej úrovni použije užitočnosť zhluku na ohodnotenie výslednej hierarchii. Jednoduchšie povedané, algoritmus s príchodom nového príkladu, vykoná jeden z nasledujúcich krokov:

- umiestni nový príklad do najlepšieho existujúceho zhluku
- utvorí nový zhluk, obsahujúci iba aktuálny trénovací príklad
- spojí dva existujúce zhluky do jedného a pridá nový príklad k tomuto zhluku
- rozdelí existujúci zhluk a nový príklad umiestni do najlepšieho zhluku (jedného z rozdelených) vo výslednej hierarchii.

Pre ilustráciu uvádzame Obr. 30, ktorý predstavuje hierarchiu zhlukov, ktoré sú výsledkom spracovania množiny štyroch trénovacích príkladov algoritmom COBWEB. Tieto jednobunkové organizmy sú zjednodušene opísané pomocou troch atribútov: počet jadier, počet bičíkov a farba vnútra. Počet jadier môže nadobúdať hodnoty od jedného do troch. Počet bičíkov je jeden alebo dva. Farba vnútra môže byť svetlá alebo tmavá.



Obr. 30: Zhluky generované algoritmom COBWEB nad množinou jednobunkových organizmov.

Uvedený príklad je ilustračný a teda veľmi jednoduchý, keďže obsahuje iba štyri trénovacie príklady. Niekedy môže byť hierarchia zhlukov formovaná z rádovo desiatok až stoviek príkladov. Potom sa stáva značne neprehľadnou. Na obmedzenie veľkosti výslednej hierarchie zhlukov sa môže aplikovať orezávanie. V súvislosti s ním sa definuje orezávací parameter (cutoff). Pri zaraďovaní niektorých príkladov sa ukáže, že sú veľmi podobné iným, a teda nie je dôvod vytvárať pre ne samostatný uzol. Orezávací parameter vytvára niečo ako jednoduchú prahovú hranicu v spolupráci s užitočnosťou zhluku. Keď je nárast užitočnosti zhluku spôsobený pridaním nového uzla veľmi malý (padne pod definovanú hranicu), daný uzol je z hierarchie odrezaný a jemu prislúchajúci príklad je pridelený k podobným príkladom. Toto orezávanie vedie k oveľa vhodnejšej a prehľadnejšej hierarchii zhlukov.

Spojenie zhlukov (merging) a rozdelenie zhluku (splitting) môže byť výpočtovo veľmi náročné. Preto je rozdelenie uvažované iba pre najlepší uzol a spojenie iba pre dva najlepšie uzly vyberané podľa kritéria užitočnosti. Zhluk $\mathbf{c_k}$ má vysokú užitočnosť, ak je charakteristický vysokou pravdepodobnosťou nejakej hodnoty nejakého atribútu $\mathbf{a_i}$. Potom je možné predikovať hodnoty atribútu $\mathbf{a_i}$ s vysokou pravdepodobnosťou.

Nasleduje popis algoritmu COBWEB:

```
COBWEB(Uzol,Priklad)
begin
if      Uzol je listový
then    begin
        generuj dvoch potomkov Uzla, L1 a L2
        nech sa pravdepodobnosti L1 rovnajú pravdepodobnostiam obsiahnutým v Uzle
        nech sa pravdepodobnosti L2 rovnajú pravdepodobnostiam nového Prikladu
        pridaj Príklad k Uzlu aktualizáciou pravdepodobností obsiahnutých v Uzle
        end
else    begin
        pridaj Príklad k Uzlu aktualizáciou pravdepodobnosti Uzla
        pre každého potomka P Uzla vypočítaj užitočnosť zhluku, ktorý vznikne
                umiestnením Prikladu do P
        nech S1 je skóre pre najlepšiu kategorizáciu do zhluku C1
        nech S2 je skóre pre druhú najlepšiu kategorizáciu do zhluku C2
        nech S3 je skóre pre umiestnenie Prikladu do nového zhluku
        nech S4 je skóre pre zlúčenie C1 a C2 do jedného zhluku
        nech S5 je skóre pre rozdelenie C1 (nahradenie dcérskymi zhlukmi)
        end
        if      S1 je najlepšie skóre
        then    COBWEB(C1,Príklad)
        else    if      S3 je najlepšie skóre
                then    nový zhluk bude mať pravdepodobnosti nového príkladu
                else    if      S4 je najlepšie skóre
                        then    begin
                                nech Cm je výsledok spojenia C1 a C2
                                COBWEB(Cm,Príklad)
                                end
                        else    if      S5 je najlepšie skóre
                                then    begin
                                        rozdeľ C1
                                        COBWEB(Uzol,Príklad)
                                        end
end
```

10.4.1 Užitočnosť zhluku

Užitočnosť zhluku predstavuje kritérium kvality, ktoré je používané napr. algoritmom COBWEB. Tento algoritmus definuje zhluky pomocou pravdepodobností. Neurčuje, aké hodnoty musia nadobúdať atribúty trénovacieho príkladu, aby mohol byť zaradený do zhluku. Namiesto toho udáva pravdepodobnosť každej hodnoty atribútu. Napr. $\mathbf{p(a_i=v_{ij}/c_k)}$ je podmienená pravdepodobnosť s ktorou bude mať atribút $\mathbf{a_i}$ hodnotu $\mathbf{v_{ij}}$ ak bude zaradený do zhluku $\mathbf{c_k}$.

Užitočnosť zhluku maximalizuje pravdepodobnosť, že dva príklady toho istého zhluku majú spoločné hodnoty atribútov; a taktiež maximalizuje pravdepodobnosť, že dva príklady z rôznych zhlukov majú rôzne hodnoty atribútov. Užitočnosť zhluku $\mathbf{c_k}$ sa počíta podľa vzťahu:

$$U_1(c_k) = \sum_k \sum_i \sum_j p(a_i = v_{ij} / c_k) p(c_k / a_i = v_{ij}) p(a_i = v_{ij})$$

Kde:

- **predpovedateľnosť** $\mathbf{p(a_i=v_{ij}/c_k)}$ je pravdepodobnosť, že trénovací príklad má hodnotu $\mathbf{v_{ij}}$ atribútu $\mathbf{a_i}$ ak patrí do zhluku $\mathbf{c_k}$. Čím je táto pravdepodobnosť vyššia, tým skôr budú mať dva príklady v tej istej kategórii resp. zhluku tie isté hodnoty atribútov.
- **prediktívnosť** $\mathbf{p(c_k/a_i=v_{ij})}$ je pravdepodobnosť, s ktorou trénovací príklad patrí do kategórie $\mathbf{c_k}$, ak má hodnoty $\mathbf{v_{ij}}$ atribútov $\mathbf{a_i}$. Čím vyššia je táto pravdepodobnosť, tým menej platí, že príklady iných zhlukov budú mať rovnaké hodnoty atribútov.
- **váha** $\mathbf{p(a_i=v_{ij})}$ zabezpečuje, že hodnoty atribútov, ktoré sa častejšie vyskytujú, budú mať silnejší vplyv na ohodnotenie kvality.

Vyššia miera užitočnosti zhluku indikuje vyššiu pravdepodobnosť, že príklady v tom istom zhluku budú zdieľať atribúty a zároveň táto miera indikuje nižšiu pravdepodobnosť, že príklady v rôznych zhlukoch budú mať rovnaké atribúty.

(Witten-Frank, 2000) definujú užitočnosť zhluku nasledovne:

$$U_2(c_k) = \frac{\sum_k p(c_k) \sum_i \sum_j [p(a_i = v_{ij} / c_k)^2 - p(a_i = v_{ij})^2]}{N}$$

kde **N** je počet zhlukov. Táto miera užitočnosti zhluku pomáha vytvárať efektívnejšiu hierarchiu zhlukov, pretože je delená počtom zhlukov. To zabraňuje prílišnému drobeniu zhlukov na každej úrovni pri zaraďovaní nového príkladu. Ak by algoritmus mal tendenciu vytvárať pre takmer každý nový príklad nový zhluk, zväčšovalo by sa číslo **N**, v dôsledku čoho by klesala užitočnosť $\mathbf{U_2}$.

Podstatný je aj rozdiel štvorcov pravdepodobností vo vnútri viacnásobnej sumácie. Musíme si uvedomiť, že to nie je štandardná metrika štvorca rozdielov, ktorá sumuje štvorce rozdielov (čo produkuje symetrický výsledok), ale miera, ktorá sumuje rozdiely štvorcov (čo neprodukuje symetrický výsledok). Teda čitateľ zlomku môže byť veľké číslo (rozdiel štvorcov sa sumuje cez všetky možné hodnoty všetkých atribútov a všetky zhluky). Najlepšia miera užitočnosti sa získa umiestnením každého príkladu do nového zhluku. Potom $\mathbf{p(a_i=v_{ij}/c_k)}$ bude 1, a čitateľ vzorca pre $\mathbf{U_2}$ sa podstatne zjednoduší:

$$n - \sum \sum p(a_i = v_{ij})^2$$

kde **n** je celkové množstvo hodnôt všetkých atribútov. Aj z týchto rozborov vyplýva nutnosť delenia počtom zhlukov.

Tento vzorec (U_2) je možné aplikovať iba na nominálne atribúty. Avšak je možné jeho rozšírenie podľa (Witten-Frank, 2000) aj na numerické atribúty:

$$U_3(c_k) = \frac{1}{N}\sum_k p(c_k)\frac{1}{2\sqrt{\pi}}\sum_i\left(\frac{1}{\sigma_{ik}} - \frac{1}{\sigma_i}\right)$$

Kde σ_i je štandardná odchýlka atribútu $\mathbf{a_i}$. Vzniká tu však problém v prípade, že je štandardná odchýlka nulová, pretože potom môže užitočnosť zhluku nadobúdať nekonečné hodnoty. Riešením takéhoto problému je nahradenie týchto nulových hodnôt nejakými definovanými minimálnymi hodnotami pre každý problémový atribút.

10.4.2 Optimalizácia hierarchického zhlukovania

Zhlukovanie vykonávané algoritmom COBWEB je zhlukovaním hierarchickým. Vytvára nielen zhluky, ale aj hierarchickú štruktúru týchto zhlukov. Rozličné systémy zhlukovania sa od seba odlišujú tým, akú funkciu používajú na ohodnotenie kvality zhlukovania a aké používajú riadiace stratégie na prehľadávanie priestoru zhlukov. Ideálna by bola taká stratégia, ktorá by konštruovala zhluky vysokej kvality a zároveň by nemala vysokú výpočtovú cenu. Vo všeobecnosti nie je možné splniť obidve požiadavky. Tu sa vytvára priestor pre optimalizáciu. To znamená, že použijeme nie veľmi drahú stratégiu vytvárajúcu počiatočné zhluky a pokračujeme niekoľkými riadiacimi stratégiami, ktoré iteratívne optimalizujú vytvorenú hierarchiu zhlukov. Podrobnejšie sa o tejto problematike pojednáva v (Fisher, 1996). V tomto článku sú opísané a zhodnotené tri stratégie iteratívnej optimalizácie. Prvá z nich je inšpirovaná iteratívnou stratégiou výberu jadra algoritmu CLUSTER/2. Druhá je bežnou formou optimalizácie, ktorá iteratívne reklasifikuje jednoduché pozorovania resp. trénovacie príklady. Tretia bola inšpirovaná makro-učiacimi stratégiami (Iba, 1991). Z vývojových dôvodov bola k týmto stratégiám priradená výpočtovo nenáročná procedúra používaná aj vyššie uvedeným algoritmom COBWEB na konštrukciu počiatočnej hierarchie zhlukov. Avšak, namiesto tejto procedúry môže byť použitá aj iná metóda.

Keď je zhlukovanie dokončené je posudzované analytikmi, spravidla podľa kritéria vhodnosti pre špecifickú úlohu. Niektorí autori (Fisher, 1987a), (Fisher, 1987b) a (Anderson-Matessa, 1991) abstrahovali tieto kritériá do zovšeobecnej úlohy príbuznej zakončeniu vývoja zhlukov, kde rozsah chyby nad ukončenými zhlukmi môže byť použitý na externé posúdenie užitočnosti zhlukov. Vo všetkých vyššie uvedených systémoch boli hodnotiace funkcie (ohodnotenie kvality zhlukovania) volené so zreteľom na túto zovšeobecnenú úlohu. Na základe tejto zovšeobecnenej úlohy boli adaptované orezávacie stratégie používané pri kontrolovanom učení na riešenie úlohy zjednodušenia hierarchie zhlukov. Experimentálne bolo dokázané, že hierarchia zhlukov môže byť značne zjednodušená, pričom sa nezvýši rozsah chyby nad ukončenými zhlukmi. Tieto experimenty zo zjednodušovaním zhlukov viedli k návrhu kritéria jednoduchosti a klasifikačnej ceny. Bolo navrhnutých zopár ohodnocovacích funkcií, ktoré vznikli

adaptáciou vybraných mier, používaných pri kontrolovanom učení pri indukcii rozhodovacieho stromu, ktoré je možné veľmi dobre aplikovať na dimenziu jednoduchosti a rozsahu chyby.

10.5 Pravdepodobnostné zhlukovacie metódy

Vyššie uvedené hierarchické zhlukovanie má niektoré zjavné nedostatky. Napríklad nutnosť deliť vypočítanú užitočnosť zhluku počtom zhlukov, nutnosť náhrady nulovej štandardnej odchýlky minimálnou hodnotou, potreba použitia orezávacieho parametra, a pod. Naviac tu vyvstávajú otázky: Akú váhu má výsledok zhlukovania, ktorý závisí na usporiadaní trénovacích príkladov? Je následná reštrukturalizácia (spájanie a rozdeľovanie zhlukov) postačujúca na odstránenie dôsledkov nešťastného usporiadania trénovacích príkladov? Ak opakujeme zhlukovaciu procedúru niekoľko krát, nenegujeme tým inkrementálnu povahu algoritmu?

Pravdepodobnostný prístup k zhlukovaniu môže niektoré z týchto nedostatkov odstrániť. Pri tomto prístupe sa príklady nepriraďujú kategoricky do konkrétnych zhlukov, ale každý príklad má priradenú množinu pravdepodobností, s ktorými prináleží ku každému zhluku. Reprezentantom tohto prístupu je pravdepodobnostný model nazývaný konečná mixáž (finite mixtures), ktorá predstavuje množinu distribúcií pravdepodobností a odpovedá N zhlukom. Táto množina odráža hodnoty atribútu členov jednotlivých zhlukov. Inými slovami, každá distribúcia udáva pravdepodobnosť, že konkrétny príklad bude mať určitú množinu hodnôt atribútov, ak bude členom daného zhluku. Každý zhluk má inú distribúciu pravdepodobnosti. Reálne prináleží každý príklad do jedného a iba jedného zhluku, ale nie je známe do ktorého.

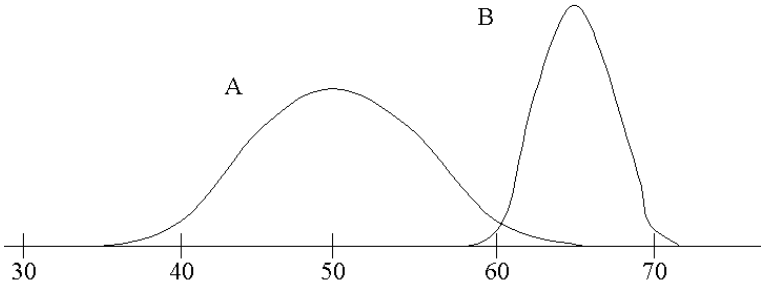
Najjednoduchšia konečná mixáž je v prípade, keď máme iba jeden numerický atribút, ktorý má napríklad normálnu distribúciu pre každý zhluk, ale s rôznymi strednými hodnotami a rôznymi odchýlkami. Samotný problém zhlukovania spočíva v tom, že je potrebné nad danou množinou trénovacích príkladov (v našom príklade je každý príklad iba číslo) definovať počet zhlukov a pre každý zhluk vypočítať strednú hodnotu a odchýlku, ako aj distribúciu pravdepodobností pre každý zhluk. Konečná mixáž kombinuje niekoľko normálnych distribúcií.

Obr. 31ilustruje jednoduchý príklad, kde každý z dvoch zhlukov má normálnu distribúciu pravdepodobnosti. Zhluk **A** má strednú hodnotu $\mu_A=50$ a štandardnú odchýlku $\delta_A=5$, zatiaľ čo zhluk **B** má $\mu_B=65$ a $\delta_B=2$.

A 51	B 62	B 64	A 48	A 39	A 51	A 43	A 47	A 51	B 64	B 62
A 48	B 62	A 52	A 52	A 51	B 64	B 64	B 64	B 64	B 62	B 63
A 52	A 42	A 45	A 51	A 49	A 43	B 63	A 48	A 42	B 65	A 48
B 65	B 64	A 41	A 46	A 48	B 62	B 66	A 48	A 45	A 49	A 43
B 65	B 64	A 45	A 46	A 40	A 46	A 48				

data

model



Obr. 31: Dvojzhlukový mixážny model pre zhluky: A a B.

Príklady sú generované z týchto distribúcií, použitím **A** s pravdepodobnosťou $\mathbf{p_A=0.6}$ a **B** s pravdepodobnosťou $\mathbf{p_B=0.4}$, teda $\mathbf{p_A+p_B=1}$.

Teraz si predstavme, že máme dané údaje t.j. trénovacie príklady bez udania zhlukov, teda samotné čísla. Našou úlohou bude určiť päť parametrov, ktoré charakterizujú model: $\mathbf{\mu_A}$, $\mathbf{\delta_A}$, $\mathbf{\mu_B}$, $\mathbf{\delta_B}$, a $\mathbf{p_A}$ ($\mathbf{p_B}$ je možné vypočítať z $\mathbf{p_A}$). Toto je konečný mixážny problém.

Ak vieme, z ktorej z týchto dvoch distribúcií každý príklad pochádza, nájdenie piatich hľadaných parametrov je jednoduché. Iba odhadneme strednú hodnotu a štandardnú odchýlku osobitne pre príklady zhluku **A** a zhluku **B** použijúc vzťahy:

$$\mu = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

$$\sigma^2 = \frac{(x_1 - \mu)^2 + \ldots + (x_n - \mu)^2}{n-1}$$

Použitie **n-1** namiesto **n** má praktické dôvody (menšia diferencia pri použití **n**).
Kde $\mathbf{x_1,\ldots,x_2}$ sú príklady z distribúcie **A** alebo **B**. Na odhad piateho parametra $\mathbf{p_A}$ stačí zobrať pomer príkladov zhluku **A** ku všetkým príkladom.

Ak poznáme týchto päť parametrov, je ľahké nájsť pravdepodobnosti, s ktorými daný príklad patrí do každého zhluku. Napríklad, pravdepodobnosť, že príklad **x** prináleží ku zhluku **A** je:

$$p(A/x) = \frac{p(x/A)p(A)}{p(x)} = \frac{f(x;\mu_A,\sigma_A)p_A}{p(x)}$$

Kde: $f(x;\mu_A,\sigma_A)$ je normálna distribúcia pravdepodobnosti funkcie **f** pre zhluk **A**, konkrétne:

$$f(x;\mu_A,\sigma_A) = \frac{1}{\sqrt{2\pi\sigma_A}} e^{-\frac{(x-\mu_A)^2}{2\sigma_A{}^2}}.$$

Vypočítame **p(A,x)** a **p(B,x)** a normalizujeme ich. Dalo by sa diskutovať o tom, že $f(x;\mu_A,\sigma_A)$ nie je presne **p(x/A)** pre pravdepodobnosť **x** rovnú nule. Avšak proces normalizácie zaručí korektný konečný výsledok. Všimnime si, že konečný výstup nie je konkrétny zhluk, ale pravdepodobnosť s akou **x** prináleží k **A** a **B**.

Problémom je, že nič z toho nepoznáme. Nepoznáme distribúciu pravdepodobností s ktorými trénovací príklad prináleží k jednotlivým zhlukom, aby sme určili päť parametrov mixážneho modelu. Taktiež nepoznáme päť parametrov, na určenie distribúcií pravdepodobností. Adaptujeme teda na tento problém algoritmus CLUSTER/2. Začneme s počiatočným odhadom piatich parametrov. Tento odhad použijeme na výpočet pravdepodobností zhlukov pre každý príklad. Podobne vypočítané distribúcie pravdepodobností použijeme na opätovné určenie piatich parametrov. To opakujeme v jednotlivých iteráciach, kým nedostaneme uspokojivý výsledok. Samozrejme je možné začínať aj odhadom pravdepodobností zhlukov. Tento algoritmus sa nazýva EM algoritmom (Expectation Maximization algorithm). Prvý krok t.j. výpočet resp. odhad pravdepodobnosti zhlukov (ktoré sú očakávané) je očakávanie – expectation. Druhý, výpočet piatich parametrov je maximalizáciou (maximization) vhodnosti výsledku výpočtu nad danými údajmi.

Hoci EM algoritmus garantuje konvergenciu k maximu, je to lokálne maximum, a nemusí sa nutne rovnať globálnemu maximu. Pre zvýšenie šancí na získanie globálneho maxima by mala byť celá procedúra opakovaná niekoľko krát s rôznymi počiatočnými odhadmi hodnôt parametrov. Výsledkom budú rôzne konečné konfigurácie. Tie je možné porovnať a vybrať najväčšie lokálne maximum.

10.6 Komplexné zhlukovanie

Predstaviteľom obsiahleho alebo komplexného zhlukovania, ktoré je založené na kombinácii vyššie uvedených troch základných prístupov k zhlukovaniu je AutoClass

(Cheeseman-Stutz, 1995). Ide o pravdepodobnostné Bayesovské zhlukovanie, ktoré využíva konečný mixážny model s apriórnymi distribúciami všetkých parametrov. To umožňuje tak použitie numerických ako aj nominálnych atribútov. Taktiež to umožňuje použitie EM algoritmu na odhad parametrov pre distribúcie pravdepodobností, ktoré by čo najlepšie odpovedali daným údajom. Keďže EM algoritmus negarantuje globálne optimum, procedúra je opakovaná pre niekoľko rozličných množín počiatočných hodnôt. Ale to nie je všetko. AutoClass pracuje s rozličným počtom zhlukov. Uvažuje tiež rôzne množstvá kovariancií a rôzne typy distribúcie pravdepodobností pre numerické atribúty. To vedie k vrchnejšej, vyššej globálnejšej úrovni prehľadávania. Napríklad na začiatku budú uvažované počty 2, 3, 5, 7, 10, 15 a 25 zhlukov. Všetky možnosti budú spracované a z výsledných údajov budú náhodne vybraté viaceré hodnoty na overenie. Môžete si predstaviť ako extrémne výpočtovo intenzívny bude algoritmus na najvyššej úrovni. Konkrétna implementácia spravidla štartuje so špecifikovanými časovými hranicami a pokračuje v iteráciach tak dlho, ako to časové obmedzenia umožnia. Viacej času znamená lepší výsledok.

10.7 Diskusia

Uvedené štyri základné zhlukovacie prístupy produkujú celkom odlišný druh výsledkov. Všetky sú schopné pracovať s údajmi v tvare testovacej množiny a začleňovať ich do zhlukov, ktoré boli objavené analýzou trénovacej množiny.

Ak zhlukovacia metóda bola použitá na označenie príkladov trénovacej množiny číslom resp. nejakým iným označením zhluku, táto množina čísel môže byť potom použitá na trénovanie pravidiel alebo rozhodovacích stromov. Výsledné pravidlá alebo stromy môžu formovať výslednú explicitnú definíciu zhlukov. Pravdepodobnostné zhlukovanie môže byť použité na ten istý účel s výnimkou prípadu, keď každý príklad má niekoľko rôznych označení s rôznymi váhami (problém vážených príkladov). Teda príklad rôznymi váhami patrí súčasne do viacerých zhlukov.

Ďalšou možnou aplikáciou zhlukovania je doplnenie chýbajúcej hodnoty atribútu. Napríklad je možné urobiť štatistický odhad neznámej hodnoty atribútu konkrétneho príkladu, ktorý je založený na distribúcii pravdepodobnosti zhluku, do ktorého daný príklad patrí.

Všetky základné zhlukovacie metódy predpokladajú nezávislosť atribútov. AutoClass dovolí používateľovi špecifikovať dodatočne dva alebo viac atribútov ako navzájom závislé, a teda by mali byť modelované prostredníctvom zdieľanej distribúcie pravdepodobnosti.

Algoritmus CLUSTER/2 je klasickou metódou. Umožňuje množstvo definícií zhlukov a variácií (Hartigan, 1975). Inkrementálne zhlukovanie založené na operáciách spájania a rozdeľovania zhlukov bolo uvedené prostredníctvom systému COBWEB pre nominálne atribúty (Fisher, 1987a). K tomuto druhu zhlukovania patrí aj systém CLASSIT pre numerické atribúty (Gennari at al., 1990). Obidva systémy sú založené na meraní užitočnosti zhluku, ktorá bola vyššie definovaná (Gluck-Corter, 1985). Program AutoClass je opísaný v (Cheesman-Stutz, 1995). Sú možné dve implementácie tohto programu. Pôvodná implementácia bola napísaná v jazyku Lisp. Ďalšia implementácia v jazyku C je desať až dvadsať krát rýchlejšia ale čiastočne obmedzená. Napríklad, pre numerické atribúty je implementovaná iba normálna distribúcia pravdepodobností.

Táto metóda spolu s možnosťami použitia v oblasti objavovania znalostí je uvedená v (Rauber-Paralič, 2000).

Záverom by bolo vhodné pozastaviť sa pri vzťahu zhlukovacích algoritmov k Neurónovým sieťam a Genetickým algoritmom. Neurónové siete (ANNs – Artificial Neural Networks) (Hertz at al., 1991) sú motivované biologickými neurónovými sieťami. V posledných troch dekádach boli intenzívne používané tak na klasifikáciu ako na zhlukovanie viď (Sethi-Jain, 1991) a (Jain-Mao, 1994). Neurónové siete typu „víťaz berie všetko" sa často používajú na zhlukovanie vstupných údajov (Jain-Mao, 1996). Podobné trénovacie príklady sú zgrupované pomocou neurónovej siete a reprezentované jednotlivým neurónom.

Genetický prístup, motivovaný prirodzenou evolúciou, používa evolučné operátory a populáciu riešení na získanie globálne optimálneho rozčlenenia údajov. Pri riešení problému zhlukovania genetickým prístupom sú kandidáti riešení kódovaní na spôsob chromozómov. Najčastejšie používané evolučné operátory sú: výber, rekombinácia a mutácia. Každá transformuje jeden alebo viac vstupných chromozómov na výstupné chromozómy. Na začiatku sa vyberie ľubovoľná populácia riešení. Každé riešenie zodpovedá jednému z K zhlukov v algoritme CLUSTER/2. Potom sa aplikujú evolučné operátory iteratívne, kým nie sú splnené ukončovacie podmienky. Najznámejšie evolučné techniky sú:

- genetické algoritmy (GAs) (Holland, 1975) a (Goldberg, 1989)
- evolučné stratégie (ESs) (Schwefel, 1981)
- evolučné programovanie (EP) (Fogel at al., 1965).

Zhlukovacie algoritmy majú rozsiahle aplikačné možnosti. V (Jain at al., 1999) je týmto aplikačným možnostiam zhlukovania venovaný veľký priestor. Na tomto mieste iba vymenujeme hlavné aplikačné oblasti: segmentácia obrazu, rozpoznávanie príkladov, obnova (opätovné nadobúdanie) dokumentov, a dolovanie údajov.

10.8 Úlohy na precvičenie:

1. Vymenujte jednotlivé prístupy k zhlukovaniu.
2. Zhlukovanie je vhodné na riešenie úloh typu:
   a/ klasifikačného
   b/ sekvenčného.

3) Výpočet užitočnosti používa na ohodnotenie kvality zhlukov algoritmus:
   a/ COBWEB
   b/ CLUSTER/2.

4. Majme tri alternatívne hierarchie zhlukov s užitočnosťou U_i kde P_i je počet zhlukov na danej úrovni hierarchie:
   a/ $U_1=2$ $P_1=2$
   b/ $U_2=2.33$ $P_2=2$
   c/ $U_3=2.75$ $P_3=3$.

   Vypočítajte konečnú hodnotu užitočnosti so zohľadnením počtu zhlukov a vyberte najlepšiu alternatívu.

10.9 Literatúra

Anderson, J. R., Matessa, M.: An iterative Bayesian algorithm for categorization. In Fisher, D., Pazzani, M., Langley, P.: Concept formation: Knowledge and Experience in Unsupervised Learning. San Mateo, CA: Morgan Kaufmann, 1991.

Cheeseman, P., Stutz, J.: Bayesian classification (AutoClass): Theory and results. In Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R. editors, Advances in Knowledge Discovery and Data Mining, Menlo Park, CA: AAAI Press, 1995, 153-180.

Diday, E., Simon, J. C.: Clustering analysis. In Digital Pattern Recognition, K. S. Fu, Ed. Springer Verlag, Secaucus, 1976, 47-94.

Fisher, D. H.: Knowledge acquisition via incremental conceptual clustering. Machine Learning, No.2, 1987a, 139-172.

Fisher, D. H.: Knowledge acquisition via incremental conceptual clustering. Ph.D. thesis, University of California, Irvine, CA: Department of Information and Computer Science, 1987b.

Fisher, D.: Iterative Optimization and Simplification of Hierarchical clustering. Journal of Artificial Intelligence Research, No.4, AI Access Foundation and Morgan Kaufman Publishers, 1996, 147-179:

Fogel, L. J.,Owens, A. J., Walsh, M. J.: Artificial Intelligence Through Simulated Evolution. John Wiley and Sons, Inc., New York, NY, 1965.

Gennari, J. H., Langley, P., Fisher, D.: Models of incremental concept formation. Artificial Intelligence 40, 1990, 11-61.

Gluck, M., Corter, J.: Information, uncertainty and the utility of categories. In. Proc. Annual Conference of the Cognitive Science Society, Irvine, CA. Hillsdale, NJ: Lawrence Erlbaum, 1985, pp. 283-287.

Goldberg, D. E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Co., Inc., Redwood City, CA, 1989.

Hartigan, J. A.: CLUSTERing algorithms. John Wiley, New York, 1975.

Hertz, J., Krogh, A., Palmer, R. G.: Introduction to the Theory of Neural Computation. Addison-Wesley Longman Publ. Co., Inc., Reading, MA, 1991.

Holland, J. H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI, 1975.

Iba, G.: A heuristic approach to the discovery of macro operators. Machine Learning, No.3, 1991, 285-317.

Ichino, M., Yaguchi, H.: Generalized Minkowski metrics for mixed feature–type data analysis. IEEE Trans. Syst. Man Cybern. 24, 1994, 698-708.

Jain, A. K., Mao, J.: Neural Networks and Pattern Recognition. In Computational Intelligence: Imitating Life, Zurada-Marks-Robinsom, Eds., 1994, 194-212.

Jain, A. K., Mao, J.: Artificial Neural Networks: A tutorial. IEEE Computer 29, 1996, 31-44.

Jain, A. K., Murty, M. N., Flynn, P. J.: Data clustering: A Review. ACM Computing Surveys, Vol.31, No.3, 1999, 264-323.

Michalski, R. S., Stepp, R.: Automated construction of classification: conceptual clustering versus numerical taxonomy. IEEE Transactions on Pattern Analysis and Machine Intelligence, No.5, 1983a, 219-243.

Michalski, R. S., Stepp, R.: Learning from observation: conceptual clustering. In Michalski, R. S., Carbonell, J. G., Mitchell, T. M.: Machine Learning: An Artificial Intelligence Approach. San Mateo, CA: Morgan Kaufmann, 1983b.

Mitchell, T. M.: Machine Learning. International Editions, Co-published: MITT Press and The McGraw-Hill Companies, Singapore, 1997, 414.

Rauber, A., Pampalk, E., Paralič, J.:Empirical Evaluation of clustering Algorithms. In. Zborník Radova, Journal of information and organizational sciences, Vol. 24. No. 2, Varaždin, Croatia, 2000, pp. 195-209.

Rauber, A., Paralič, J.: Cluster Analysis as a First Step in the Knowledge Discovery Process. Journal of Adv. Comp. Intelligence, Vol.4, No.4, Fuji Technology Press Ltd., Japan, 2000, pp. 158-162.

Schwefel, H. P.: Numerical Optimization of Computer Models. John Wiley and Sons, Inc., New York, NY, 1981.

Sethi, I., Jain, A. K.: Artificial Neural Networks and Pattern Recognition: Old and New Connections. Elsevier Science Inc., New York, NY, 1991.

Wilson, D. R., Martinez, T. R.: Improved heterogeneous distance functions. Artif. Intell. Res. 6, 1997, 1-34.

Witten, I. H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Jawa Implementations. Morgan Kaufmann Publishers, San Francisco, California, USA, 2000, 369 p.

Zadeh, L. A.: Fuzzy sets. Inf. Control 8, 1965, 338-353.