dovacích pravidel už známe, a to vytvořením pravidel z rozhodovacího stromu. Jiné možnosti vytváření pravidel si ukážeme v tomto oddílu.

5.3.1 Pokrývání množin

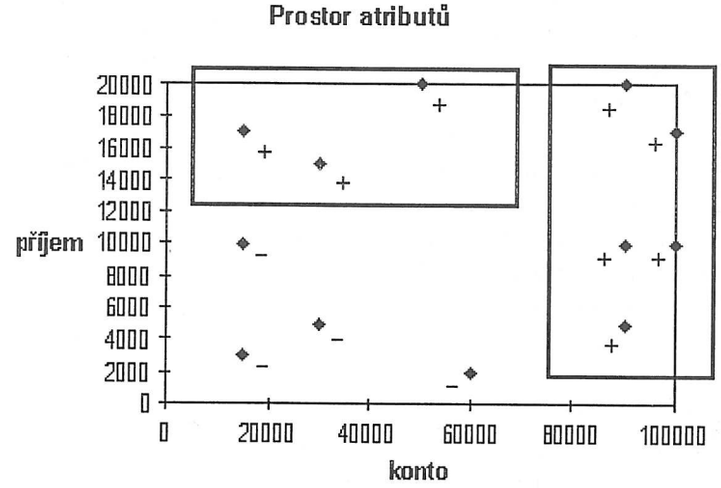
Podobně jako je jméno Rosse Quinlana spojeno s rozhodovacími stromy, je jméno Ryzsarda Michalskiho dáváno do souvislosti s rozhodovacími pravidly, přesněji s algoritmem *pokrývání množin* (set covering algorithm), známým jako *AQ* (Michalski, 1969)[74]. Zatímco algoritmus TDIDT pro tvorbu rozhodovacích stromů bývá nazýván *algoritmus rozděl a panuj* (divide and conquer), algoritmus pokrývání množin bývá nazýván *algoritmus odděl a panuj* (separate and conquer). Při pokrývání množin jde totiž o to nalézt pravidla (konzistentní hypotézy), která pokrývají[75] nějaké příklady hledaného konceptu, a tyto příklady oddělit od jiných příkladů téhož konceptu i od příkladů třídy jiné. V prostoru atributů se tedy postupně vybírají oblasti, které obsahují příklady pouze jedné třídy. Nalezené oblasti mají, podobně jako v případě rozhodovacích stromů,

[74] Postupně vznikla celá řada těchto algoritmů rozlišovaných číslem, tedy např. *AQ15*. Michalski je rovněž autorem myšlenky paprskového prohledávání za využití konceptu hvězda (star), které používá např. CN2 a CN4.

[75] Pravidlo pokrývá příklad, pokud příklad vyhovuje předpokladu pravidla.

podobu mnohorozměrných hranolů rovnoběžných s osami. Rozhodovací pravidla tedy mají stejnou vyjadřovací sílu jako rozhodovací stromy (obr. 62).

Základní podobu algoritmu pokrývání množin pro dvě třídy ukazuje obr. 63. Uvedený algoritmus bude opět fungovat pouze pro kategoriální data (příklady popisujeme jako kombinaci hodnot atributů), která nejsou zatížena šumem (hledáme pravidla pokrývající příklady pouze jedné třídy).



Obr. 62 Vyjadřovací síla rozhodovacích pravidel.

Algoritmus pokrývání množin

1. najdi pravidlo, které pokrývá nějaké pozitivní příklady a žádný negativní,
2. odstraň pokryté příklady z trénovací množiny $\mathbf{D}_{TR}$,
3. pokud v $\mathbf{D}_{TR}$ zbývají nějaké nepokryté pozitivní příklady, vrať se k bodu 1, jinak skonči.

Obr. 63 Algoritmus pokrývání množin.

Rozšíření algoritmu pro více tříd se obvykle[76] provede tak, že pro každou třídu $C(v_t)$ se data rozdělí na příklady této třídy

$$\{C(+)\} = \{\mathbf{o}_i : y_i = v_t\} = \mathbf{D}^+_{TR}$$

a protipříklady této třídy

[76] Jinou možností je vytvářet pravidla ke všem třídám současně. Tak postupuje algoritmus CN2, popř. CN4, v modifikaci pro tvorbu rozhodovacích seznamů a algoritmus *ESOD*, které jsou popsány v dalších částech této kapitoly.

$$\{C(-)\} = \{o_i : y_i \neq v_t\} = \mathbf{D}_{\overline{TR}}.$$

Rozšíření algoritmu pro práci s daty zatíženými šumem spočívá v tom, že v kroku 1 nepožadujeme, aby pravidlo pokrývalo příklady pouze jedné třídy[77]. Numerické atributy je třeba opět diskretizovat („uvnitř" algoritmu pro tvorbu pravidel nebo ve fázi přípravy dat).

Klíčovým bodem algoritmu pokrývání množin je krok 1 – nalezení jednoho pravidla. Zatímco algoritmus TDIDT postupoval v prostoru hypotéz „shora dolů", při pokrývání množin najdeme obě varianty postupu: „zdola nahoru"[78] (tedy metodu generalizace – odstraňování kategorie z kombinace) i „shora dolů"[79] (metodou specializace – přidávání kategorie do kombinace).

Uveďme nejprve postup zdola nahoru, který odpovídá algoritmům AQ. Počáteční hypotéza pokrývající jeden pozitivní příklad, se postupně zobecňuje tak, aby pokrývala více pozitivních příkladů a žádný negativní příklad[80]. Krok 1 z algoritmu na obr. 63 bude tedy mít podobu:

1. vezmi jeden pozitivní příklad jako jádro (seed),
2. najdi jeho generalizaci, která pokrývá nějaké pozitivní příklady a žádný negativní.

Vezměme si k ruce opět naše známá data (tab. 37). Budeme-li při volbě příkladů v kroku 1 postupovat sekvenčně[81], bude hledání pravidel vypadat takto: V prvním kroku vybereme příklad k1 popsaný kombinací

příjem(*vysoký*) ∧ *konto*(*vysoké*) ∧ *pohlaví*(*žena*) ∧ *nezaměstnaný*(*ne*).

Tuto kombinaci postupně zobecňujeme v sekvenci (v závorce jsou uvedeny pokryté příklady):

příjem(*vysoký*) ∧ *konto*(*vysoké*) ∧ *pohlaví*(*žena*) ∧ *nezaměstnaný*(*ne*) (k1),
příjem(*vysoký*) ∧ *konto*(*vysoké*) ∧ *nezaměstnaný*(*ne*) (k1,k2),
konto(*vysoké*) (k1,k2,k4,k5).

Výsledná kombinace *konto*(*vysoké*) pokrývá pozitivní příklady k1, k2, k4 a k5 třídy *úvěr*(*ano*). Tyto příklady odstraníme z trénovací množiny a pokračujeme příkladem k7 pokrytým kombinací

příjem(*vysoký*) ∧ *konto*(*nízké*) ∧ *pohlaví*(*muž*) ∧ *nezaměstnaný*(*ne*).

[77] Připomeňme, že v případě rozhodovacích stromů byl tento problém řešen analogickým způsobem, a to tak, že jsme nepožadovali, aby v listovém uzlu stromu byly příklady pouze jedné třídy.
[78] Tento postup bývá rovněž nazýván *postup řízený daty* (data driven).
[79] Tento postup bývá rovněž nazýván *postup generování a testování* (generate and test). Generování zde znamená prohledávání prostoru možných pravidel.
[80] Tento postup jsme již viděli u algoritmu FindS, který je popsán v kapitole o strojovém učení.
[81] Při výběru jak jádra, tak příkladů pokrytých generalizací jádra bude hrát roli pořadí příkladů v trénovacích datech. Volba příkladu tedy může ovlivnit výslednou podobu pravidel.

Tabulka 37 Data pro tvorbu rozhodovacích pravidel

klient	*příjem*	*konto*	*pohlaví*	*nezaměstnaný*	*úvěr*
k1	*vysoký*	*vysoké*	*žena*	*ne*	*ano*
k2	*vysoký*	*vysoké*	*muž*	*ne*	*ano*
k3	*nízký*	*nízké*	*muž*	*ne*	*ne*
k4	*nízký*	*vysoké*	*žena*	*ano*	*ano*
k5	*nízký*	*vysoké*	*muž*	*ano*	*ano*
k6	*nízký*	*nízké*	*žena*	*ano*	*ne*
k7	*vysoký*	*nízké*	*muž*	*ne*	*ano*
k8	*vysoký*	*nízké*	*žena*	*ano*	*ano*
k9	*nízký*	*střední*	*muž*	*ano*	*ne*
k10	*vysoký*	*střední*	*žena*	*ne*	*ano*
k11	*nízký*	*střední*	*žena*	*ano*	*ne*
k12	*nízký*	*střední*	*muž*	*ne*	*ano*

Tuto kombinaci postupně zobecníme[82] na kombinaci

příjem(*vysoký*),

která pokrývá příklady k7, k8 a k10. Zbývá příklad k12 popsaný kombinací

příjem(*nízký*) ∧ *konto*(*střední*) ∧ *pohlaví*(*muž*) ∧ *nezaměstnaný*(*ne*).

Jde o poslední pozitivní příklad v trénovacích datech. Při zobecňování tohoto příkladu nám pomohou negativní příklady; nalezená kombinace nesmí pokrývat žádný z nich. Tento požadavek splňuje kombinace

konto(*střední*) ∧ *nezaměstnaný*(*ne*).

Pravidla nalezená algoritmem tedy budou[83]:

IF *konto*(*vysoké*) THEN *úvěr*(*ano*),
IF *příjem*(*vysoký*) THEN *úvěr*(*ano*),
IF *konto*(*střední*) ∧ *nezaměstnaný*(*ne*) THEN *úvěr*(*ano*).

Použití rozhodovacích pravidel pro klasifikaci nových příkladů je velice prosté. Postupně procházíme soubor pravidel až nalezneme pravidlo, které lze použít. Závěr pravidla pak určí třídu, do které máme uvažovaný příklad zařadit.

V následujícím odstavci ukážeme postup vytváření pravidel metodou specializace – tedy přidáváním kategorií do kombinace tvořící předpoklad pravidla. Pro generování předpokladů pravidel můžeme použít různé metody prohledávání prostoru kombinací. Nejpoužívanějšími způsoby pro nalezení jednoho pravidla (v souvislosti s algoritmem pokrývání množin) je *gradientní*

[82] Budeme-li procházet klienty postupně, pak zobecňujeme v sekvenci *příjem*(*vysoký*) ∧ *konto*(*nízké*) ∧ *pohlaví*(*muž*) ∧ *nezaměstnaný*(*ne*) (k7) → *příjem*(*vysoký*) ∧ *konto*(*nízké*) (k7, k8) → *příjem*(*vysoký*) (k7, k8, k10).
[83] Můžeme pozorovat značnou shodu s pravidly vytvořenými z rozhodovacího stromu, jak jsme uvedli v předchozím textu.

prohledávání a *paprskové prohledávání*. V prvním případě jde o prohledávání do hloubky bez navracení: v každém kroku se vybere ta „nejlepší" kategorie, která specializuje předpoklad pravidla (kritéria kvality pravidel rovněž uvidíme v následujícím odstavci). Ve druhém případě se paralelně sleduje předem daný počet nejvhodnějších „kandidátů" toho být předpokladem pravidla: v každém kroku se pak přidá nejlepší kategorie ke každému potenciálnímu předpokladu a výsledná množina (starých a nových kandidátů) se opět redukuje na požadovaný počet těch nejlepších.

5.3.2 Rozhodovací seznam

Soubor pravidel IF-THEN, tak jak jsme ho poznali v předcházejícím odstavci, bývá někdy nazýván „neuspořádaný" soubor pravidel. Opakem je „uspořádaný" soubor pravidel, neboli *rozhodovací seznam* (decision list). V tomto druhém případě jde o strukturu typu

IF Ant_1 THEN $Class_i$,
ELSE IF Ant_2 THEN $Class_j$
ELSE IF Ant_3 THEN $Class_k$ ...,

kde v závěrech THEN se mohou objevovat různé třídy. Uspořádání zde spočívá v tom, že v každé podmínce ELSE IF se implicitně skrývá negace všech podmínek předcházejících pravidel. Nelze tedy již pravidla chápat jako navzájem nezávislá.

Příkladem systému, který vytváří jak rozhodovací pravidla (čti neuspořádaný soubor pravidel), tak rozhodovací seznamy (čti uspořádaný soubor pravidel), je CN2 (Clark, Nibblet, 1989), resp. jeho rozšíření CN4 (Bruha, Kočková, 1994)[84]. Opět jde o algoritmus pokrývání množin (pokryté příklady se odstraňují z trénovacích dat), postupuje se ale metodou specializace, tedy „shora dolů".

Jádrem algoritmu (který odpovídá kroku 1 v obecném schématu algoritmu pokrývání množin) je funkce *Search*(*Ant*,$\mathbf{D}_{TR}$) hledající jedno pravidlo. Podoba této funkce je uvedena na obr. 64. Algoritmus hledá pravidlo, které pokrývá velký počet objektů třídy *Class* a malý počet objektů ostatních tříd. Tvorba pravidel končí, když už se nepodaří nalézt vyhovující pravidlo. Specializace pravidla se provádí přidáním kategorie ke kombinaci tvořící předpoklad pravidla[85]. Potenciální předpoklady se uchovávají v množině *Star*. Velikost této

[84] Tato verze nabízí řadu zlepšení ve srovnání s původním algoritmem: ošetření chybějících hodnot, cenu vyhodnocování atributu, práci s numerickými atributy.

[85] V původním algoritmu se používá termín *selektor* (selector) pro kategorii a termín *komplex* (complex) pro kombinaci.

Funkce Search(*Ant*,$\mathbf{D}_{TR}$)

1. nechť *Star* je množina obsahující prázdnou kombinaci [ ]
2. nechť *Ant* je prázdná kombinace
3. nechť *Sel* je množina všech kategorií $A(v)$ vyskytujících se v $\mathbf{D}_{TR}$
4. dokud *Star* je prázdné nebo dokud nebyly testovány všechny kategorie $A(v)$ v *Sel*
   4.1. nechť *NewStar* je prázdné
   4.2. pro každou kombinaci *Comb* ∈ *Star*
      4.2.1. proveď specializaci přidáním kategorie $A(v)$ ze *Sel*
      4.2.2. vyhodnoť kvalitu kombinace $CombA = Comb \wedge A(v)$ pomocí funkce $F(CombA)$
      4.2.3. zařaď kombinaci *CombA* do *NewStar*
   4.3. pro každou kombinaci *Comb* ∈ *NewStar*
      4.3.1. pokud *Comb* je (signifikantně) lepší než *Ant*, přiřaď *Ant* := *Comb*
   4.4. pokud počet kombinací v *NewStar* překročí zadaný práh, vyhoď nejhorší kombinaci
   4.5. přiřaď *Star* := *NewStar*

Obr. 64 Funkce „najdi jedno pravidlo" v algoritmu CN4.

množiny určuje šířku paprsku pro paprskové prohledávání. Nejlepší pravidlo (krok 4.2.2 algoritmu) se hledá na základě negativní entropie

$$F(Ant) = \sum_{t=1}^{T} \frac{n_t(Ant)}{n(Ant)} \log_2 \frac{n_t(Ant)}{n(Ant)} ,$$

na základě Laplaceova odhadu očekávané spolehlivosti pravidla[86]

$$F(Ant) = \frac{n_t(Ant)+1}{n(Ant)+T}$$

nebo na základě m-odhadu (m-prob)

$$F(Ant) = \frac{n_t(Ant)+m\,f_t}{n(Ant)+m} ,$$

kde T je počet tříd, $n_t(Ant)$ udává počet příkladů třídy t pokrytých pravidlem, $n(Ant)$ značí počet všech příkladů pokrytých pravidlem, $f_t = n_t/n$ představuje relativní četnost třídy t a m je parametr. Ve všech těchto případech vyšší hodnota znamená lepší pravidlo.

Hlavní cyklus algoritmu se liší podle toho, jestli vytváříme uspořádaná nebo neuspořádaná pravidla. V případě neuspořádaných pravidel systém hledá pravidla pro jednotlivé třídy odděleně. Pro každou třídu projde celou trénovací množinu $\mathbf{D}_{TR}$ s tím, že pozitivní příklady tvoří vždy příklady jedné třídy

[86] Oproti běžně používané spolehlivosti (platnosti) pravidla bere Laplaceova korekce do úvahy počet tříd T.

a negativní příklady tvoří příklady ostatních tříd (podobu hlavního cyklu algoritmu ukazuje obr. 65). Tak se může stát, že se k jednomu příkladu naleznou pravidla, která by jej řadila k různým třídám. Tento spor lze řešit hlasováním aplikovatelných pravidel.

V případě uspořádaných pravidel (rozhodovacího seznamu) se hledají pravidla ke všem třídám najednou. $\mathbf{D}_{TR}$ tedy odpovídá trénovací množině v nezměněné podobě. Třída predikovaná každým pravidlem odpovídá třídě, do které patří většina pokrytých příkladů. Při klasifikaci ke sporům nemůže dojít, protože pravidla mají přidělena pořadí použití.

Algoritmus CN4 – rozhodovací pravidla

1. nechť *ListOfRules* je prázdný seznam
2. pro každou třídu $C(v_t)$, $t = 1, ..., T$
   2.1. dokud množina pozitivních příkladů této třídy $\mathbf{D}_{TR_t}$ není prázdná
      2.1.1. pomocí funkce Search(*Ant*, $\mathbf{D}_{TR_t}$) nalezni nejlepší kombinaci *Ant*
      2.1.2. přiřaď $\mathbf{D}_{TR_t} := \mathbf{D}_{TR_t} - \mathbf{D}_{TR_t}(Ant)$, kde $\mathbf{D}_{TR_t}(Ant)$ jsou příklady pokryté kombinací *Ant*
      2.1.3. do *ListOfRules* přidej pravidlo IF *Ant* THEN $C(v_t)$

Obr. 65 Hlavní cyklus algoritmu CN4 pro neuspořádaná pravidla.

Algoritmus CN4 – rozhodovací seznam

1. nechť *ListOfRules* je prázdný seznam
2. dokud trénovací množina $\mathbf{D}_{TR}$ není prázdná
   2.1. pomocí funkce Search(*Ant*, $\mathbf{D}_{TR}$) nalezni nejlepší kombinaci *Ant*
   2.2. přiřaď $\mathbf{D}_{TR} := \mathbf{D}_{TR} - \mathbf{D}_{TR}(Ant)$, kde $\mathbf{D}_{TR}(Ant)$ jsou příklady pokryté kombinací *Ant*
   2.3. do *ListOfRules* přidej pravidlo IF *Ant* THEN *Class*, kde *Class* je majoritní třída příkladů v $\mathbf{D}_{TR}(Ant)$

Obr. 66 Hlavní cyklus algoritmu CN4 pro uspořádaná pravidla.

```
if příjem=vysoký then class is ano;
   Kr=[ 5 0]; signif=5.850; quality=0.925; cost=1
if konto=vysoké then class is ano;
   Kr=[ 4 0]; signif=4.680; quality=0.900; cost=1
if příjem=nízký && konto=nízké then class is ne;
   Kr=[ 0 2]; signif=6.340; quality=0.900; cost=2
if konto=střední && nezaměstnaný=ano then class is ne;
   Kr=[ 0 2]; signif=6.340; quality=0.900; cost=2
if konto=střední && nezaměstnaný=ne then class is ano;
   Kr=[ 2 0]; signif=2.340; quality=0.850; cost=2
if true then class is ano;
      Kr=[ 8 4]; signif=0.000; quality=0.733; cost=0
```

Obr. 67 CN4, neuspořádaná pravidla.


```
if příjem=vysoký then class is ano;
   Kr=[ 5 0]; signif=5.850; quality=0.925; cost=1
else if konto=vysoké then class is ano;
   Kr=[ 2 0]; signif=2.340; quality=0.850; cost=1
else if nezaměstnaný=ano then class is ne;
   Kr=[ 0 3]; signif=9.510; quality=0.950; cost=1
else if konto=střední then class is ano;
   Kr=[ 1 0]; signif=1.170; quality=0.825; cost=0
else if true then class is ne;
   Kr=[ 0 1]; signif=3.170; quality=0.850; cost=0
```

Obr.68 CN4, uspořádaná pravidla.

Pro data z tab. 37 vytvoří CN4 soubor neuspořádaných pravidel uvedený na obr. 67 a soubor uspořádaných pravidel znázorněný na obr. 68 (jde o autentické výstupy z programu). Na těchto výstupech si můžeme všimnout, že pro naše data odpovídají neuspořádaná pravidla pravidlům vytvořeným „simulací" algoritmu AQ (viz předcházející oddíl) a uspořádaná pravidla odpovídají pravidlům vytvořeným z rozhodovacího stromu (viz oddíl 5.1). Numerickými charakteristikami každého pravidla jsou:

- *Kr* – pro jednotlivé třídy počet příkladů pokrytých pravidlem, tedy četnosti $n_t(Ant)$,
- *signif* – kritérium pro výběr předpokladu pravidla

$$signif = \sum_{t=1}^{T} n_t(Ant) \log_2 \left(\frac{n_t(Ant)}{n(Ant)} \frac{n}{n_t} \right),$$

kde T je počet tříd, n_t vyjadřuje počet příkladů třídy t a n udává počet všech příkladů (v trénovacích datech),

- *quality* – hodnocení založené na počtu příkladů (majoritní) třídy pokrytých a nepokrytých daným pravidlem

$$quality = 0{,}8 \frac{n_{t_m}(Ant)}{n(Ant)} + 0{,}2 \frac{n_{t_m}(Ant)}{n_{t_m}},$$

kde t_m je majoritní třída,

- *cost* – cena vyhodnocení pravidla, standardně počet dotazů v předpokladu.

Poslední pravidlo (to s předpokladem `true`) je implicitní pravidlo, které odpovídá majoritní třídě v trénovacích datech. Toto pravidlo se při klasifikaci uplatní, pokud selhala všechna ostatní.