Učenje pravila

Matej Mihelčić

Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu

matmih@math.hr

26. ožujka, 2026.

Učenje pravila - uvod

- Klasifikacijski model baziran na pravilima se sastoji od skupa *if-then* pravila.
- Svako pravilo ima uvjet (konjunkcija vrijednosti atributa) i konzekventu (labelu ciljne klase).
- Vjerojatnosna pravila uz predviđenu vrijednost ciljne labele sadrže i listu vjerojatnosti ili broj pokrivenih primjera entiteta za svaku vrijednost ciljne labele.
- Skupovi pravila su često **jednostavniji** i **razumljiviji** od stabala odlučivanja.
- Stabla odlučivanja se mogu pretvoriti u skup pravila, tako da stvorimo po jedno pravilo za svaki list stabla.
- Takva pretvorba nužno stvara **nepreklapajuća** pravila (ukoliko promatramo skupove bez nedostajućih vrijednosti).
- Dopuštanje preklapanja između pravila, koja ne opisuju nužno sve entitete iz skupa podataka, može omogućiti stvaranje znatno manjih skupova pravila.

Učenje pravila - uvod

- Kod stvaranja skupova pravila gdje je dopušteno preklapanje, moramo:
  - uvesti mehanizme koji će omogućiti **stvaranje predviđanja u slučaju kada imamo nekoliko jednako vrijednih alternativa**.
  - uvesti **standardno pravilo** koje će klasificirati sve primjere koji nisu pokriveni pravilima iz našeg skupa pravila.
- Ukoliko više pravila opisuje neki entitet, predviđanje najčešće radi pravilo koje ima veću točnost na skupu za treniranje (sadrži veći postotak primjera čija vrijednost ciljne klase odgovara predviđanju pravila).
- Standardno pravilo uobičajeno stavljamo na kraj skupa pravila.
- Za razliku od stabala odlučivanja koja koriste podijeli pa vladaj (eng. *divide and conquer*) postupak izgradnje, pravila se rade koristeći postupak odvoji pa vladaj (eng. *separate and conquer*). Umjesto rekurzivnog dijeljenja skupa entiteta na disjunktne podskupove, prvo se iterativno poboljšava jedno pravilo, nakon čega se svi entiteti opisani tim pravilom izbacuju iz skupa za učenje. Na modificiranom skupu se uči drugo pravilo itd.

Učenje pravila - uvod

- Podrezivanje pravila je također koristan postupak, koji se uobičajno radi nakon što je pravilo naučeno.
- Pokazano je (Rivest 1987) da je skup pravila s maksimalno k uvjeta (atributa) u svakom pravilu strogo jače ekspresivan od stabla dubine k.
- Uvjet stabala odlučivanja da se pravila ne smiju preklapati (dijeljenje na disjunktne podskupove) često dovodi do toga da se identična (ili jako slična) podstabla moraju graditi na više mjesta u velikom stablu odlučivanja.
- Pretpostavimo da imamo 4 atributa, A, B, C, D, gdje svaki atribut poprima tri različite vrijednosti.
- Definiramo koncept:

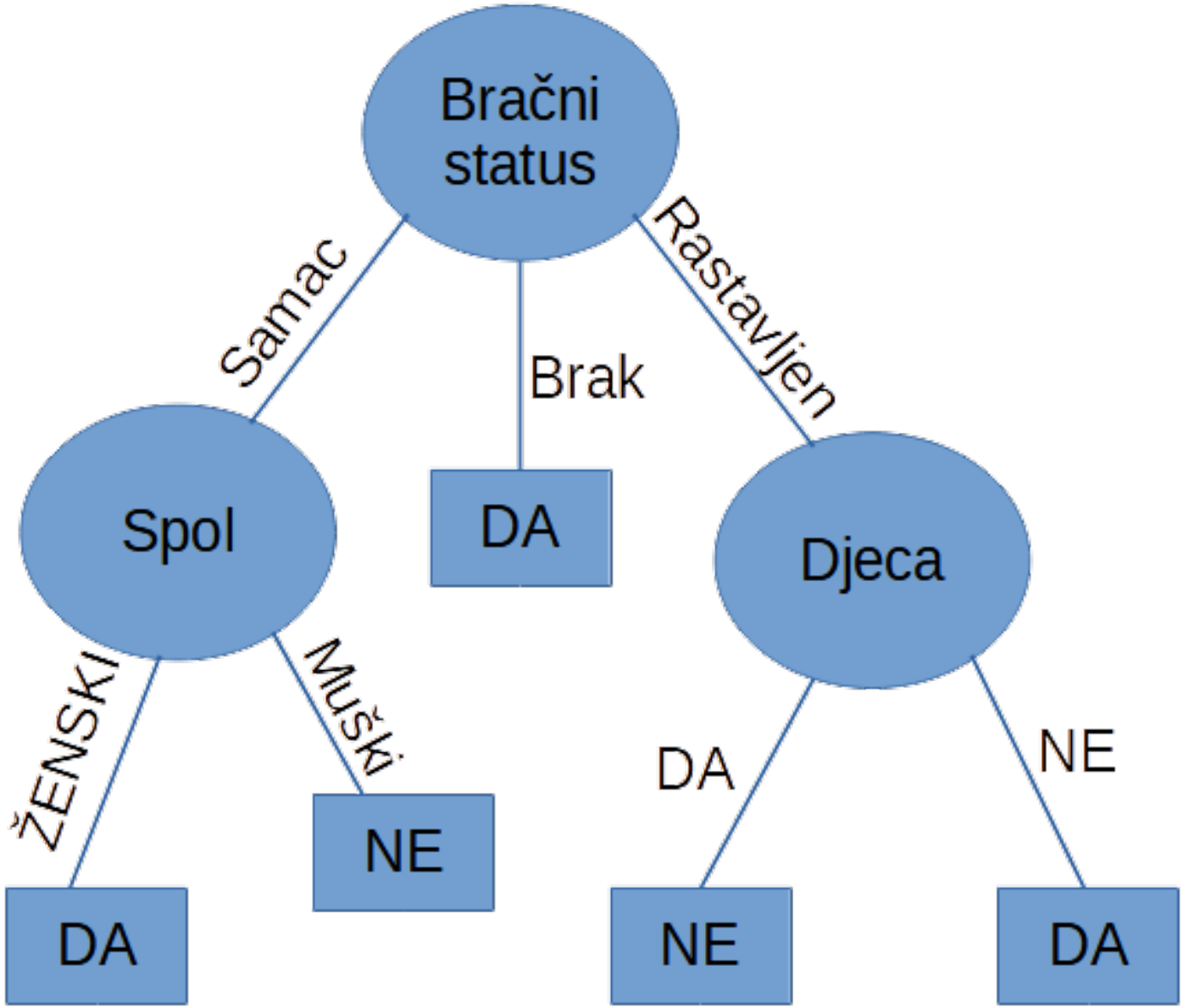
```
IF A = 3 AND B = 3 THEN Class = x
IF C = 3 AND D = 3 THEN Class = x
```

- Minimalno stablo koje može opisati ovaj koncept se sastoji od 10 unutarnjih čvorova i 21 lista.

Učenje pravila - primjer

Broj	Obrazovanje	Bračni status	Spol	Djeca	Slaganje
1	Primarno	Samac	Muški	NE	NE
2	Primarno	Samac	Muški	DA	NE
3	Primarno	Brak	Muški	NE	DA
4	Sveučilišno	Rastavljen	Ženski	NE	DA
5	Sveučilišno	Brak	Ženski	DA	DA
6	Sekundarno	Samac	Muški	NE	NE
7	Sveučilišno	Samac	Ženski	NE	DA
8	Sekundarno	Rastavljen	Ženski	NE	DA
9	Sekundarno	Samac	Ženski	DA	DA
10	Sekundarno	Brak	Muški	DA	DA
11	Primarno	Brak	Ženski	NE	DA
12	Sekundarno	Rastavljen	Muški	DA	NE
13	Sveučilišno	Rastavljen	Ženski	DA	NE
14	Sekundarno	Rastavljen	Muški	NE	DA

Učenje pravila - primjer



Učenje pravila - primjer

Skup pravila dobiven iz stabla odlučivanja:

```
IF BračniStatus = Samac AND Spol = ŽENSKI
THEN Slaganje = DA                (DA 2/9, NE 0/5)

IF BračniStatus = Samac AND Spol = MUŠKI
THEN Slaganje = NE                (DA 0/9, NE 3/5)

IF BračniStatus = Brak
THEN Slaganje = DA                (DA 4/9, NE 0/5)

IF BračniStatus = Rastavljen AND Djeca = DA
THEN Slaganje = NE                (DA 0/9, NE 2/5)

IF BračniStatus = Rastavljen AND Djeca = NE
THEN Slaganje = DA                (DA 3/9, NE 0/5)
```

Učenje pravila - primjer

Skup pravila dobiven postupcima učenja pravila može biti:

```
IF BračniStatus = Brak
THEN Slaganje = DA              (DA 4/9, NE 0/5)

IF Spol = ŽENSKI
THEN Slaganje = DA              (DA 6/9, NE 1/5)

IF Spol = MUŠKI
THEN Slaganje = NE              (DA 3/9, NE 4/5)

STANDARDNO: SLAGANJE = DA
```

Proces učenja pravila

Tipičan proces učenja pravila se sastoji od tri bitne faze:

- Konstrukcija značajki - značajke se grade iz atributa originalnog skupa podataka i čine gradivne jedinice pravila.
- Konstrukcija pravila - stvaramo pravila koristeći značajke, gdje svako pravilo opisuje podskup entiteta iz skupa za treniranje. Tipično biramo vrijednost ciljne varijable koju želimo opisati, te računamo konjunkcije značajki koje najbolje odvajaju entitete s tom vrijednosti ciljne varijable od ostalih entiteta iz skupa za učenje.
- Konstrukcija hipoteza - hipoteza se sastoji od skupa pravila. Često hipoteze dobivamo sekvencijalnim učenjem pravila dok ne opišemo sve ili većinu entiteta iz skupa za treniranje.

Generalni algoritam za traženje jednog pravila

Algoritam PronadiNajboljePravilo

function PronadiNajboljePravilo($\mathcal{E}$)
Ulaz: $\mathcal{E} = P \cup N$: skup pozitivnih i negativnih entiteta za klasu c, opisanih značajkama $\mathcal{F}$
Algoritam:
$r^* :=$ InicijalizirajPravilo($\mathcal{E}$)
$h^* :=$ EvaluirajPravilo(r^*)
$\mathcal{R} := \{r^*\}$
while $\mathcal{R} \neq \emptyset$ **do**
    $\mathcal{R}_{cand} :=$ IzaberiKandidate($\mathcal{R}, \mathcal{E}$)
    $\mathcal{R} := \mathcal{R} \setminus \mathcal{R}_{cand}$
    for all $r \in \mathcal{R}_{cand}$ **do**
        $\rho(r) =$ ProfiniPravilo($r, \mathcal{E}$)
        for all $r' \in \rho(r)$ **do**
            if KriterijZaustavljanja($r', \mathcal{E}$) **then**
                sljedeci r'
            end if
            $h' :=$ EvaluirajPravilo($r', \mathcal{E}$)
            $\mathcal{R} :=$ UbaciSortirano($r', \mathcal{R}$)
            if $h' > h^*$ **then**
                $h^* := h', r^* := r'$
            end if
        end for
    end for
    $\mathcal{R} :=$ FiltrirajPravila($\mathcal{R}, \mathcal{E}$)
end while
Izlaz: r^*: najbolje pravilo za dani skup entiteta

Generalni algoritam za traženje jednog pravila

- Algoritam `PronadiNajboljePravilo` pretražuje prostor hipoteza i vraća pravilo koje optimizira zadanu mjeru kvalitete definiranu u `EvaluirajPravilo`.
- U principu se vrijednost mjera evaluacije povećava s brojem opisanih pozitivnih primjera, a smanjuje s brojem negativno opisanih primjera.
- Algoritam `PronadiNajboljePravilo` održava sortiranu listu pravila kandidata $\mathcal{R}$, koja se inicijalizira funkcijom `InicijalizirajPravilo`. Nova pravila se umeću na odgovarajuće pozicije korištenjem `UbaciSortirano`. U svakom koraku metoda `IzaberiKandidate` izabire podskup pravila kandidata koja se poboljšavaju unutar metode `ProfiniPravilo`.
- `ProfiniPravilo` implementira neki postupak poboljšavanja pravila - stvaranja novih kandidata koji poboljšavaju mjeru kvalitete. Specijalizacija, generalizacija, slučajna mutacija itd. su primjeri postupaka koji poboljšavaju kvalitetu pravila.

Generalni algoritam za traženje jednog pravila

- Poboljšane varijante pravila se dodaju u skup $\mathcal{R}$ dok nije zadovoljne kriterij zaustavljanja `KriterijZaustavljanja`.
- U svakom koraku pamtimo trenutno najbolje generirano pravilo r^*.
- Najbolje pravilo pronađeno kroz sve iteracije vraćamo korisniku.

- `InicijalizirajPravilo` i `ProfiniPravilo` čine strategiju pretrage.
- `IzaberiKandidate` i `FiltrirajPravila` čine algoritam pretraživanja.
- `EvaluirajPravilo` čini huristiku pretraživanja.

Postupci pretraživanja prostora

- Heuristički:
  - Pretraživanje penjanjem (eng. Hill-Climbing) - kontinuirano poboljšavamo pravilo i stajemo kada poboljšanja više nisu moguća. Pokušavamo doći do optimuma provodeći poboljšanja koja u svakom koraku biraju trenutno najbolje poboljšanje.
  - Pretraživanje snopom zraka (eng. Beam search) - uz najboljeg kandidata, u memoriji čuvamo i fiksan broj alternativa (snop). Poboljšava rezultat pretraživanja penjanjem zato što poboljšava veći broj pravila (pretražuje veći dio prostora hipoteza).
- Sveobuhvatno:
  - Pretraživanje najboljeg-prvog (eng. best-first search) - izabire najboljeg kandidata, međutim umjesto da računa fiksan broj poboljšanja, računa **sva** poboljšanja kandidata u listu pravila.
  - Uređeno pretraživanje - posjećuje svako pravilo samo jednom. Kombinira se illi s pretraživanjem najboljeg prvog ili s pretraživanjem u širinu. Definira se neki uređaj između značajki, generiraju se sva pravila koja sadrže jednu značajku, zatim se pravila prošire tako da sadrže dvije značajke itd.

Postupci pretraživanja prostora

- Sveobuhvatno:
  - Pretraživanje po razinama - koristi pretraživanje u širinu za generiranje pravila s jednom, dvije značajke itd. Međutim, u svakomo koraku odbacuje pravila ukoliko vrijedi svojstvo anti-monotonosti. Navedeno svojstvo kaže da nadskup $\mathcal{F}' \supset \mathcal{F}$ značajki ne može imati bolju kvalitetu od $\mathcal{F}$. Jedan kriterij kvalitete koji zadovoljava to svojstvo je broj opisanih entiteta. Na ovaj način otkrivamo skupove frekventnih značajki.
- Stohastičko pretraživanje - dozvoljavamo slučajnost u metodi `ProfiniPravilo`. Omogućava provođenje većih skokova.

Smjerovi pretraživanja

Prostor pretraživanja se može strukturirati kao **rešetka** (eng. lattice) generalnosti. U njoj je pravilo r_A generalnije od pravila r_B ako i samo ako r_A pokriva (opisuje) sve entitete koje pokriva i r_B.

Definiramo operator **poboljšavanja** ρ kao operator $\rho : \mathcal{L} \mapsto 2^{\mathcal{L}}$, za jezik $\mathcal{L}$. Pravila $r' \in \rho(r)$ se naziva poboljšanje pravila r.

Minimalni operator poboljšanja je operator poboljšanja takav da za svaki element $r' \in \rho(r)$, r' je susjed od r u rešetki generalnosti.

Postoje tri glavna smjera pretraživanja:

- Pretraživanje od vrha prema dnu (eng. *top-down search*) - pravila se poboljšavaju korištenjem operacije **specijalizacije** (dodavanjem novih značajki).
- Pretraživanje od dna prema vrhu (eng. *bottom-up search*) - pravila se stvaraju generalizacijom najspecifičnijeg pravila (eliminacijom značajki).
- Dvostrano pretraživanje - kombinira prethodna dva načina pretraživanja.

Smjerovi pretraživanja

- Kod pristupa pretraživanja od vrha prema dnu koristimo **operator specijalizacije**. Operator specijalizacije σ mapira pravilo r u skup pravila $\sigma(r)$ takav da su sva pravila iz $\sigma(r)$ manje generalna od r. $\forall r' \in \sigma(r) : r' \subset r$. Operator specijalizacije je **minimalan** ako je $\sigma(r)$ skup maksimalno generalnih specijalizacije pravila r, tj. $\sigma(r) = \{r' | \ (r' \subset r) \ \wedge \ (\nexists \bar{r} : r' \subset \bar{r} \subset r)\}$.
- Najčešće korišteni operator specijalizacije dodaje jednu novu značajku (atribut) pravilu. $r = C \leftarrow f_1 \ \wedge \ f_2 \ \wedge \ \ldots \ \wedge \ f_n$ specijaliziramo u $r = C \leftarrow f_1 \ \wedge \ f_2 \ \wedge \ \ldots \ \wedge \ f_n \ \wedge f_{n+1}$.

Smjerovi pretraživanja

- Kod pristupa pretraživanja od dna prema gore koristimo **operator generalizacije**. Operator generalizacije γ mapira pravilo r u skup pravila $\gamma(r)$ tako da su sva pravila iz $\gamma(r)$ generalnija od r, odnosno $\forall r' \in \gamma(r) : r' \supset r$. Operator generalizacije je **minimalan** ako je $\gamma(r)$ skup najmanje generalnih generalizacija pravila r.
  $\gamma(r) = \{r' | \ (r' \supset r) \ \wedge \ (\nexists \bar{r} : r' \supset \bar{r} \supset r)\}$.
- Najjednostavniji operator generalizacije dobijemo brisanjem značajke (atributa) iz pravila. Tim postupkom možemo dobiti najviše n generalizacija (poboljšanja) pravila $r = C \leftarrow f_1 \ \wedge \ f_2 \ \ldots \ \wedge \ f_n$. Npr. $r' = C \leftarrow f_1 \ \wedge \ f_3 \ \wedge \ldots \ \wedge \ f_n$ je jedna generalizacija pravila r.
- Najpoznatiji načini pretraživanja od dna prema gore su **inverzna rezolucija** i **najmanje generalna generalizacija**.

Učenje skupova pravila

- Cilj učenja skupova pravila je stvoriti skup pravila koja pokrivaju **sve pozitivne primjere**, a niti jedan negativan primjer neke klase.
- Kod ciljnih klasa koje sadrže više kategorijskih vrijednosti, učimo podskupove pravila koji odvajaju primjere s zadanom kategorijom ciljne klase od preostalih primjera.
- Dodatno uvodimo uvjet da pravila (i njihov broj) moraju biti što manji mogući da se osigura dobro generaliziranje i izbjegnu trivialna rješenja (pravila koja opisuju svaki entitet individualno).
- Jedna strategija je učenje pravila koja pokrivaju (opisuju) podskupove entiteta iz skupa za treniranje. Iterativno se primjenjuje algoritam za pronalazak skupa pravila koji zajedno pokrivaju sve primjere.
- Glavni problem je pronaći dovoljno raznolika pravila koja pokrivaju sve primjere iz skupa za treniranje.
- Jednostavna strategija - igoriramo entitete koji su već pokriveni prethodno otkrivenim pravilima.

Algoritam pokrivanja (odvoji pa vladaj)

function PRONADIPRAVILA($\mathcal{E}$)
Ulaz: $\mathcal{E} = \mathcal{P} \cup \mathcal{N}$: skup pozitivnih i negativnih entiteta za klasu c, opisane skupom značajki $\mathcal{F}$
$R := \emptyset$
$\mathcal{E}^{cur} := \mathcal{E}$
while $\mathcal{P}^{cur} \neq \emptyset$ **do**
  $r :=$ PRONADINAJBOLJEPRAVILO($\mathcal{E}^{cur}$)
  if KRITERIJZAUSTAVLJANJATEORIJE($R, \mathcal{E}^{cur}$) **then**
    break while
  end if
  $R := R \cup \{r\}$
  $\mathcal{E}^{cur} :=$ AZURIRAJPRIMJERE($R, \mathcal{E}^{cur}$)
end while
$R :=$ POSTPROCESIRANJEPRAVILA($R, \mathcal{E}$)
Izlaz: R naučeni skup pravila

- Pokrivanje kreće od prazne teorije.
- Ako postoji pozitivnih entiteta u skupu za treiranje, pozivamo `PronadiNajboljePravilo` koje pronalazi pravilo koje pokriva (opisuje) podskup entiteta.

Algoritam pokrivanja (odvoji pa vladaj)

- `AzurirajPrimjere` prilagođava skup za treniranje novoj situaciji brisanjem svih opisanih entiteta ili samo pozitivnih opisanih entiteta.
- Pravila iterativno učimo dok nije zadovoljen `KriterijZaustavljanjaTeorije`.
- Na kraju slijedi `PostProcesiranjePravila` gdje možemo posebno optimizirati pravila ili podrezivati pravila.

Vidjeli smo da `AzurirajPrimjere` možemo implementirati na dva načina:

- eliminacijom svih pokrivenih entiteta. Najčešće se koristi kod **uređenih** lista pravila. Kod takvih lista, pravilo koje prvo pokrije (opiše) entitet daje predviđanje. Ostala pravila ne sudjeluju u predviđanju entiteta za koje već imamo predviđanja.
- eliminacijom samo pozitivnih pokrivenih entiteta. Najčešće se koristi kod **neuređenih** lista pravila. Ne postoji fiksan poredak između pravila. Moramo implementirati strategiju donošenja konsenzusa (najtočnije pravilo koje pokriva primjer, većinsko glasanje itd.).

Težinsko pokrivanje

- Odgovor na probleme koji nastaju izbacivanjem pokrivenih entiteta. Najveći problem je što se iz izbačenih entiteta više ne može učiti. Također, iterativnim izbacivanjem entiteta iz skupa za treniranje često završimo sa skupom koji ima jako asimetričnu distribuciju i izolirane, raspršene entitete.
- Težinsko pokrivanje ne briše pokrivene entitete već im pridodaje težinu koja ovisi o broju pravila koja pokrivaju entitet.
- Težine se koriste pri računanju heuristike učenja i utječu na izbor atributa.

Algoritam težinskog pokrivanja

function TezinskoPokrivanje($\mathcal{E}$)
Ulaz:
$\mathcal{E} = \mathcal{P} \cup \mathcal{N}$: skup pozitivnih i negativnih primjera klase c, opisani značajkama $\mathcal{F}$
Algoritam:
$\mathcal{R} \leftarrow \emptyset$

for all $e \in \mathcal{E}$ **do**
    $w(e) :=$ InicijalizirajTezinePrimjera($\mathcal{E}$)
end for

for $i = 1 \ldots I$ **do**
    $r :=$ PronadiNajboljePravilo($\mathcal{E}$)
    $\mathcal{R} := \mathcal{R} \cup r$
    for all $e \in \mathcal{E}$ **do**
        $w(e) :=$ AzurirajTezinePrimjera($\mathcal{R}, \mathcal{E}$)
    end for
end for

Izlaz:
$\mathcal{R}$ naučeni skup pravila

Algoritam težinskog pokrivanja

- Inicijalno postavljamo težine svih entiteta na 1.0.
- Težine pokrivenih primjera se reduciraju na vrijednost između 0 i 1, što daje do znanja algoritmu da se ne treba previše truditi pokriti taj entitet.
- Postoji više načina za reduciranje težina primjera. Dva poznatija pristupa su:
  - **Multiplikativne težine** - za zadani $0 < \gamma < 1$, težina entiteta koji je pokriven od strane C pravila se određuje kao $w(e) = \gamma^C$. $\gamma = 1$ bi rezultirao pronalaženjem istog pravila u svakoj iteraciji, dok bi $\gamma = 0$ rezultirao običnim algoritmom pokrivanja.
  - Kod **aditivnog ažuriranja težina**, težine entiteta koji su pokriveni s C pravila se računaju kao $w(e) = \frac{1}{C+1}$.
- Heuristike u `PronadiNajboljePravilo` se modificiraju tako da $E = P + N$ zamijenimo s $E = \sum_{e \in \varepsilon} w(e)$. Broj pokrivenih primjera $\hat{P}$ definiramo kao $\hat{P} = \sum_{e \in \hat{P}} w(e)$.

Ripper

procedure IREP(*Poz*, *Neg*)
SkupPravila $\leftarrow \emptyset$
while *Poz* $\neq \emptyset$ **do**
  podijeli (*Poz*, *Neg*) u (*PozGradi*, *NegGradi*) i (*PozPodr*, *NegPodr*)
  Pravilo $\leftarrow$ IzgradiPravilo(*PozGradi*, *NegGradi*)
  Pravilo $\leftarrow$ PodreziPravilo(*Rule*, *PozdPodr*, *NegPodr*)
  if stopa pogreske *Pravilo* na (*PozPodr*, *NegPodr*) prelazi 50% **then**
    return *SkupPravila*
  else
    dodaj *Pravilo* u *SkupPravila*
    eliminiraj entitete pokrivene od *Pravila* iz (*Poz*, *Neg*)
  end if
end while
return *SkupPravila*

- Pravila se grade počevši od praznih konjunkcija, iterativno dodajući atribute oblika $A_n = v$, $A_c \leq \theta$, $A_c \geq \theta$.

Ripper

- Koristimo information gain kao kriterij dodavanja atributa, te povećavamo pravilo dok ono opisuje samo pozitivne primjere.
- Nakon konstrukcije, pravilo se odmah podrezuje. Biramo maksimalni niz atributa koji maksimiziraju:
  $v(r, PozPodr, NegPodr) = \frac{p + (N - n)}{P + N}$, gdje P i N označavaju broj entieta u $PozPodr$, $NegPodr$, a p i n broj entiteta iz $PozPodr$, $NegPodr$ pokrivenih pravilom r.
- U slučaju prisutnosti nedostajućih vrijednosti, algoritam definira da pravilo ne opisuje entitet ukoliko je njegova vrijednost nedostajuća za bilo koji atribut sadržan u pravilu.

Ripper

function RIPPER(E)
$SkupPravila \leftarrow \emptyset$
Sortiraj klase u uzlaznom poretku $C_1, \ldots, C_k$, tako da postoji najmanji broj entiteta koji sadrže vrijednost ciljne klase C_1, a najveći broj entiteta koji sadrže vrijednost ciljne klase C_k
for $c = C_1, \ldots, C_{k-1}$ **do**
  $Poz \leftarrow E_{C_c}$, $Neg \leftarrow E_{\cup_{i \neq c} C_i}$
  $SkupPravila \leftarrow SkupPravila \cup \text{IREP}(Poz, Neg)$
end for
$SkupPravila \leftarrow SkupPravila \cup \{DEFAULT \leftarrow (C_k \leftarrow *)\}$
return $SkupPravila$

- Iterativno pozivamo IREP da stvara skupove pravila koji odvajaju entitete s vrijednosti ciljne labele C_i od entiteta s vrijednostima različitim od C_i, za sve $i \neq k$.
- Standardno pravilo glasi, za svaki nepokriveni primjer predvidi vrijednost klase C_k (one čiju vrijednost sadrži najveći broj entiteta iz skupa za treniranje).

CN2

E - skup entiteta
A - skup atributa

function CN2(E, A, M)
$LP \leftarrow []$
repeat
  $Najbolji\_K \leftarrow$ PronadiNajboljePravilo(E, A, M)
  if $Najbolji\_K \neq \emptyset$ **then**
    Neka su E' entiteti pokriveni od $Najbolji\_K$
    Briši E' iz E
    Neka je C najčešća vrijednost klase entiteta iz E'
    Dodaj pravilo "If $Najbolji\_K$ then C"
      na kraj LP
  end if
until $Najbolji\_K = \emptyset$ **or** $E = \emptyset$
return LP

- $Najbolji\_K$ - najbolja elementarna konjunkcija (konjunkcija atributa) najveće točnosti u danoj iteraciji.

CN2

Procedure PronadiNajboljePravilo(E, A, M)
Neka je $B = \{\{\emptyset\}\}$
Neka je $Najbolji\_k = \emptyset$
while $B \neq \emptyset$ **do**
  Specijaliziraj sve konjunkcije u B:
  Neka je B' skup $\{x \wedge y \mid x \in B, y \in A\}$.
  Briši sve konjunkcije iz B' koje su u B (nisu specijalizirane) ili su prazne (npr. $big = y \wedge big = n$).
  for svaku konjunkciju $K_i \in B'$ **do**
    if K_i je statistički značajan i bolji od $Najbolji\_k$ prema korisničkom kriteriju kada se testira na E **then**
      $Najbolji\_K = K_i$.
    end if
  end for
  repeat
    Izbriši najgoru konjunkciju iz B'.
  until $|B'| \leq M$
  $B \leftarrow B'$.
end while
return $Najbolji\_K$.

CN2

- Algoritam CN2 specijalizira konjunkcije dodavanjem atributa.
- Održava zraku od maksimalno B konjunkcija koje iterativno poboljšava.
- Algoritam koristi entropiju da odredi najbolju konjunkciju.
- Značajnost elementarne konjunkcije se račun testom $\sum_{i=1}^{n} f_i log(f_i/e_i)$, gdje f_i označava frekvenciju pojavljivanja i-te vrijednosti ciljne labele među entitetima koje pokriva elementarna konjunkcija, a e_i je očekivana frekvencija i-te vrijednosti ciljne labele u slučajno odabranom podskupu entiteta jednake veličine. Pretpostavlja se da entitet ima i-tu vrijednost ciljne labele s vjerojatnošću jednakoj vjerojatnosti pojavljivanja te vrijednosti u skupu za treniranje. Značajnost računamo iz χ^2 distribucije s $n-1$ stupnjeva slobode.
- CN2 za entitete koji imaju nedostajuću vrijednost za neki atribut iz elementarne konjunkcije uzima najčešću vrijednost iz skupa za treniranje ili srednju vrijednost podintervala koji se najčešće javlja.

CN2

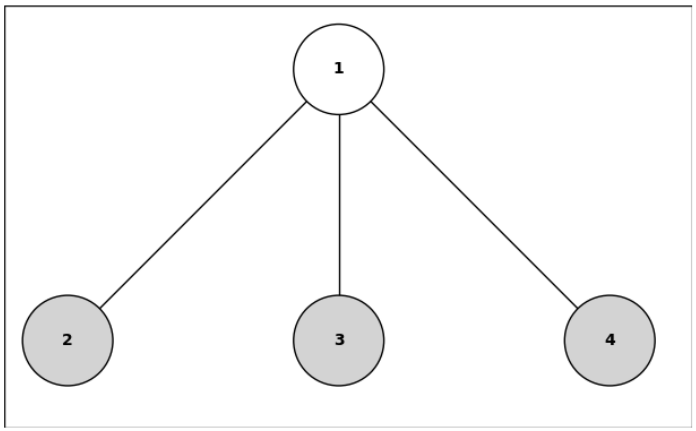
- CN2 stvara uređenu listu pravila.
- Prvo pravilo koje pokrije primjer, predviđa vrijednost ciljne varijable.
- Interpretacija uređene liste je malo složenija od interpretacije neuređene liste.
- Interpretacija mora uzeti u obzir pravilo koje pokriva primjer i sva prethodna pravila koja ga ne pokrivaju.
- Ne treba implementirati konsenzus između pravila.

PART

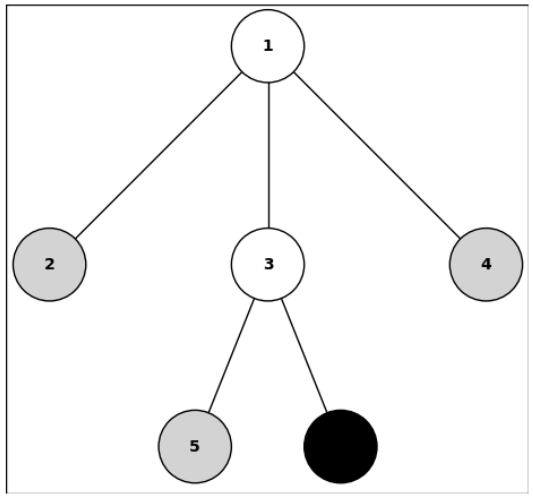
- Koristi standardnu odvoji pa vladaj strategiju. Iterativno bira najbolje pravilo, briše entitete pokrivene tim pravilom dok nije zadovoljen kriterij zaustavljanja (nemogućnost pronalaska pravila odgovarajuće kvalitete ili su pokriveni svi entiteti).
- Standardno pravilo opisuje sve preostale primjere predviđajući klasu koju sadrži najveći broj entiteta sadržanih u podskupu.
- Najbolje pravilo traži konstrukcijom parcijalnih stabala odlučivanja `C4.5` tipa.
- Prilikom izgradnje parcijalnog stabla, nedostajuće vrijednosti tretira kao i `C4.5` algoritam.
- Listove određuje identično kao i `C4.5` algoritam.
- Standardno stvara uređenu listu pravila.

KRAJ$\leftarrow$ FALSE
Procedure PronadiNajboljePravilo(E, A)
if KRAJ and postojiRoditelj() **then**
    return
end if
if KRAJ and !postojiRoditelj() **then**
    $L_k \leftarrow$ pronadiNajgeneralnijiListStabla(korjen)
    return pravilo(L_k)
end if
korjen $\leftarrow$ *kreirajKorjen*(E, A) kao kod `C4.5`
sortiraj djecu $D_1, \ldots, D_k$ prema entropiji tako da D_1 ima najmanju entropiju.
for $D_1, \ldots D_k$ **do**
    korjen $\leftarrow$ PronadiNajboljePravilo(N_i, A) gdje N_i označava entitete čvora D_i
    if D_i je list, ista definicija kao kod `C4.5` **then**
        oznaciList(D_i)
        return
    end if
    if Sva djeca od D_i su listovi **then**
        $D_i \leftarrow$ podrezi(D_i) na isti način kao kod `C4.5`
    end if
    if podrezan(D_i) **then**
        return
    else
        KRAJ = TRUE
        return
    end if
end for

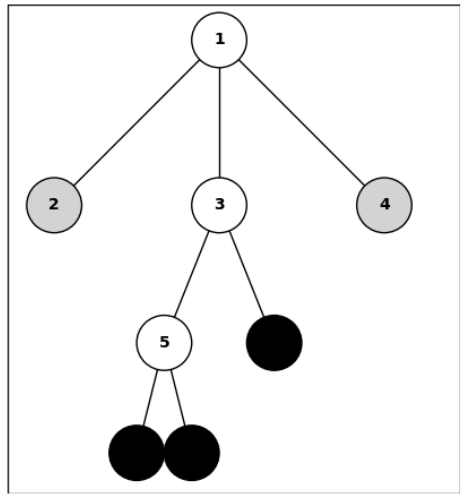
Faza 1



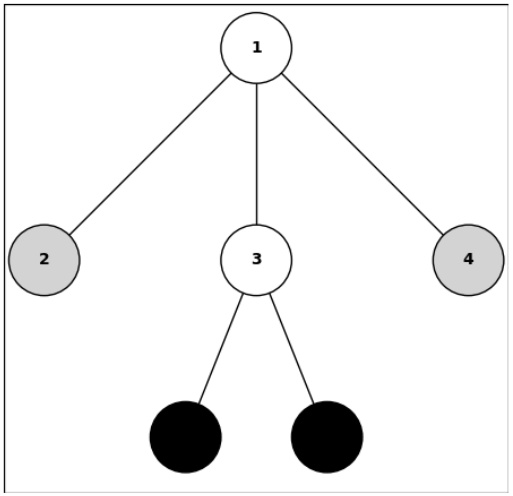
Faza 2



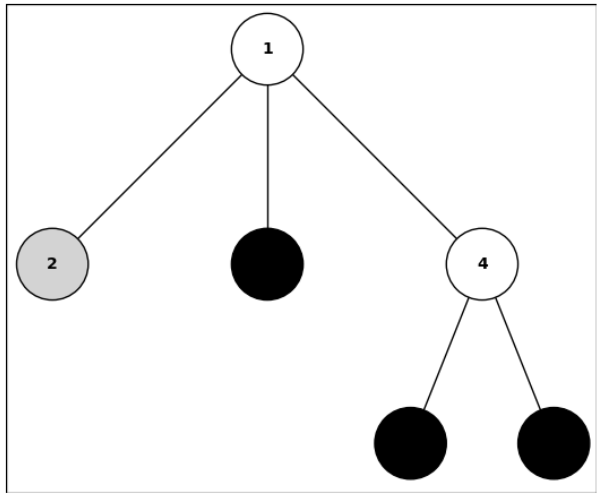
Faza 3



Faza 4



Faza 5



Dodatne mjere evaluacije

- $\hat{P}$ - broj pokrivenih pozitivnih primjera, $\hat{N}$ - broj pokrivenih negativnih primjera.
- P - broj pozitivnih primjera u skupu na kojem vršimo evaluaciju, N - broj negativnih primjera u evaluacijskom skupu podataka.
- `NepokriveniNegativni`(r, D_{test})$= N - \hat{N}$.
- `PokriveniPozitivni`(r,D_{test})$= \hat{P}$.
- `RazlikaPokrivenosti`(r,D_{test})$=$ `PokriveniPozitivni(r)` + `NepokriveniNegativni(r)`
- `Toc(r,` D_{test})$= \frac{\hat{P}+(N-\hat{N})}{P+N}$.
- `Pokrivenost(r,` D_{test}) $= \frac{\hat{P}+\hat{N}}{P+N}$.
- `RazOmj`(r,D_{test})$= \frac{\hat{P}}{P} - \frac{\hat{N}}{N}$.
- `TezToc(r,` D_{test})$= \frac{\hat{P}+\hat{N}}{P+N} \cdot (\frac{\hat{P}}{\hat{P}+\hat{N}} - \frac{P}{P+N})$.

Dodatne mjere evaluacije

- $\hat{P}$ - broj pokrivenih pozitivnih primjera, $\hat{N}$ - broj pokrivenih negativnih primjera.
- Laplace-ova procjena uz pretpostavku uniformne apriori vjerojatnosti pozitivne i negativne klase: $\texttt{Laplace}(\texttt{r}, D_{test}) = \frac{\hat{P}+1}{\hat{P}+\hat{N}+2}$.
- m procjena, generalizira Laplace-ovu procjenu tako da apriori vjerojatnost računa kao $\pi = \frac{P}{P+N}$ iz skupa podataka za treniranje. $\texttt{MProcjena}(\texttt{r}, D_{test}) = \frac{\hat{P}+m\cdot\pi}{\hat{P}+\hat{N}+m}$
- $\texttt{JMjera}(\texttt{r}, D_{test}) = \texttt{Pokrivenost}(\texttt{r}, D_{test})\cdot\texttt{KL}(\texttt{r}, D_{test}) = \frac{1}{P+N}(\hat{P}\cdot\log_2\frac{\frac{\hat{P}}{\hat{P}+\hat{N}}}{\frac{P}{P+N}} + \hat{N}\cdot\log_2\frac{\frac{\hat{N}}{\hat{P}+\hat{N}}}{\frac{N}{P+N}})$.
- $\texttt{Lift}(\texttt{r}, D_{test}) = \frac{\frac{\hat{P}}{\hat{P}+\hat{N}}}{\frac{P}{P+N}}$

Dodatne mjere evaluacije

Uz mjere točnost, kod učenja pravila su nam bitne i mjere složenosti pravila.

- `Duljina(r)`$= |r|$ - broj atributa koji grade pravilo.
- Minimalna duljina opisa (eng. minimal description length) $MDO(r) = I(r) + I(\varepsilon|r)$. $I(r)$ označava količinu informacija potrebnu za slanje pravila r, a $I(\varepsilon|r)$, količinu informacija potrebnu za slanje skupa primjera ε uz pomoć r. Navedene informacije nisu jednostavno izračunljive stoga se koriste razne računalne aproksimacije.

Često se koriste i mjere koje kombiniraju više različitih mjera koristeći npr. težinsku sumu.

Učenje pravila - primjeri

- WEKA - `RIPPER`, odnosno Java implementacija `JRIP`, te `PART`.
- Python (`Orange` paket) - `CN2`.