Automatické generovanie pravidiel pre BZ ZS použitím algoritmu AQ11

Michal Beca, Kristína Pavlíková,Milan Radovanovich,
Vladyslav Vodopianov

Obsah

Úvod

Indukčné strojové učenie je klasickým problémom umelej inteligencie. Za posledných 20 rokov bolo navrhnutých veľa algoritmov. Medzi nimi sú dve najobľúbenejšie AQ11 a ID3, ktoré navrhli Michelski a Quinlan. Doteraz z nich bolo odvodených veľa algoritmov a vytvorili dve rodiny indukčného strojového učenia.
Skúmaním týchto algoritmov sa zameriavame na interpretáciu veľmi rozsiahlej databázy, data mining. Podľa cieľa, ktorý rozumie databáze, existujú dve kategórie, pochopenie databázy počítačom a pochopenie databázy človekom.

1. Popis algoritmu AQ11

AQ11 je typický algoritmus indukčného strojového učenia s logickou špecifikáciou. Navrhol to Michalski a jeho kolegovia v roku 1980. Princíp AQ11 je rozďeľuj a panuj.

Vstupy: ***E1****...množina pozitívnych trénovacích príkladov*
E2*...množina negatívnych trénovacích príkladov*

Výstupy: ***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* $\boldsymbol{e_i \vee E1}$
  - ***for*** *každý príklad* $\boldsymbol{e_j \vee E2}$
    - *generuj*

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

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

- na $G(e_i/E2)$ aplikuj absorbčný zákon
- z E1 vymaž všetky príklady pokryté $G(e_i/E2)$
  - if E1={}

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

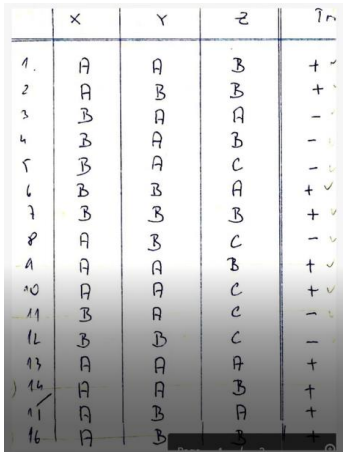
  - then

end

Tento algoritmus môže byť použitý na multi-triednu klasifikáciu do tried T1, T2, ... ,TN: potom E1 tvoria príklady triedy Ti a E2 tvoria príklad ostatných vyskúšaných. Algoritmus nevyžaduje vzájomnú nezávislosť trénovacích príkladov, lebo nepoužíva pravdepodobnosť

2. Postup vypracovania

Trénovacia množina:



	X	Y	Z	TR
0	A	A	B	+
1	A	B	B	+
2	B	A	A	-
3	B	A	B	-
4	B	A	C	-
5	B	B	A	+
6	B	B	B	+
7	A	B	C	-
8	A	A	B	+
9	A	A	C	+
10	B	A	C	-
11	B	B	C	-
12	A	A	A	+
13	A	A	B	+
14	A	B	A	+
15	A	B	B	+

V python kóde sú riadky označované od nuly. V pripravenej trénovacej množine si vyberieme hocijaký príklad z plusovej triedy a budeme ho porovnávať zo všetkými príkladmi z mínusovej triedy. Príklady zapíšeme do obálky.

Napr.

G(1, 3) = X#B Z#A
G(1, 4) = X#B
G(1, 3) = X#B Z#C
G(1, 8) = Y#B Z#C
G(1,12) = X#B Y#B Z#C

V kóde postupujeme po poradí a rozdelíme si riadky podľa tried:

```
Počet výsledkov jednotlivých tried:
 {'+': 10, '-': 6}
Indexy riadkov jednotlivých tried:
 {'+': [0, 1, 5, 6, 8, 9, 12, 13, 14, 15], '-': [2, 3, 4, 7, 10, 11]}
```

Následne porovnávame riadok prvej triedy s ostatnými z druhej triedy a prejdeme všetky dostupné riadky:

```
OBÁLKY G(+)/(-):
G(0/2): ['X#B', 'Z#A']
G(0/3): ['X#B']
G(0/4): ['X#B', 'Z#C']
G(0/7): ['Y#B', 'Z#C']
G(0/10): ['X#B', 'Z#C']
G(0/11): ['X#B', 'Y#B', 'Z#C']
G(0/(-)): [['X#B'], ['X#B', 'Z#A'], ['X#B', 'Z#C'], ['Y#B', 'Z#C'], ['X#B', 'Y#B', 'Z#C']]
Zakon absorbcie: G(0/(-)): [['X#B'], ['Y#B', 'Z#C']]

G(1/2): ['X#B', 'Y#A', 'Z#A']
G(1/3): ['X#B', 'Y#A']
G(1/4): ['X#B', 'Y#A', 'Z#C']
G(1/7): ['Z#C']
G(1/10): ['X#B', 'Y#A', 'Z#C']
G(1/11): ['X#B', 'Z#C']
G(1/(-)): [['Z#C'], ['X#B', 'Y#A'], ['X#B', 'Z#C'], ['X#B', 'Y#A', 'Z#A'], ['X#B', 'Y#A', 'Z#C']]
Zakon absorbcie: G(1/(-)): [['Z#C'], ['X#B', 'Y#A']]
```

Všetky logické výrazy, ktoré sme urobili, dáme do jednej veľkej obálky, a aplikujeme zákon absorbcie, podobne ako v kóde, pričom v kóde je konjunkcia v rámci zátvorky a disjunkcia medzi zátvorkami:

G((1)/(-)) = ((X#B) Λ (Z#A)) V (X#B) V ((X#B) Λ (Z#C)) V ((Y#B) Λ (Z#C)) V (X#B Λ Y#B Λ Z#C)
Zákon absorbcie = (X#B) Λ (Y#B) V (Z#C) = (X#B) Λ (Y#B) V ((X#B) Λ (Z#C))

Takto postupujeme pri každom príklade z plusovej triedy. Napokon Vytvoríme obálku G((+)/(-)), ktorá bude pokrývať každý príklad z plusovej triedy a nebude pokrývať žiaden z mínusovej triedy. Na túto obálku taktiež aplikujeme zákon absorbcie.

```
Vysledok pred Absrobciou:
G((+)/(-)): [[['X#B'], ['Y#B', 'Z#C']], [['Z#C'], ['X#B', 'Y#A']], [['Y#A'], ['Z#C']], [['X#B'], ['Y#B']]]
Absorbcia:
G((+)/(-)): [['X#B', 'Z#C'], ['Z#C', 'Y#A'], ['X#B', 'Y#B']]
```

V pripravenej trénovacej množine si vyberieme hocijaký príklad z mínusovej triedy a budeme ho porovnávať zo všetkými príkladmi z plusovej triedy. Príklady opäť zapíšeme do obálky. Napr.

G(3, 1) = X#A Z#B
G(3, 2) = X#A Y#B Z#B
G(3, 6) = Y#B
G(3, 7) = Y#B Z#B
G(3,10) = X#A Z#C
G(3,13) = X#A
G(3,15) = X#B Y#B

Zákon absorbcie:
G((3)/(+)) = X#A Λ Y#B

Netreba zabudnúť, že v python kóde sú riadky posunuté o jeden menej.

```
OBÁLKY G(-)/(+):
G(2/0): ['X#A', 'Z#B']
G(2/1): ['X#A', 'Y#B', 'Z#B']
G(2/5): ['Y#B']
G(2/6): ['Y#B', 'Z#B']
G(2/8): ['X#A', 'Z#B']
G(2/9): ['X#A', 'Z#C']
G(2/12): ['X#A']
G(2/13): ['X#A', 'Z#B']
G(2/14): ['X#A', 'Y#B']
G(2/15): ['X#A', 'Y#B', 'Z#B']
G(2/(+)): [['Y#B'], ['X#A'], ['X#A', 'Z#B'], ['Y#B', 'Z#B'], ['X#A', 'Z#C'], ['X#A', 'Y#B'], ['X#A', 'Y#B', 'Z#B']]
Zakon absorbcie: G((+)/2): [['Y#B'], ['X#A']]
```

Takto postupujeme pri každom príklade z mínusovej triedy. Napokon vytvoríme obálku G((-)/(+)), ktorá bude pokrývať každý príklad z mínusovej triedy a nebude pokrývať žiaden z plusovej triedy. Na túto obálku taktiež aplikujeme zákon absorbcie.

```
Vysledok pred Absrobciou:
G((-)/(+)): [[['Y#B'], ['X#A']], [['X#A'], ['Y#B']], [['X#A'], ['Y#B', 'Z#A'], ['Y#B', 'Z#B']], [['Z#B'], ['Y#A'],
['Z#A']], [['Z#A'], ['Z#B'], ['X#A', 'Y#A']]]
Absorbcia:
G((-)/(+)): [['Y#B', 'X#A'], ['X#A', 'Z#A'], ['Y#B', 'Z#A', 'Y#B'], ['Y#A', 'Z#A'], ['Z#A', 'Z#B'], ['Z#B', 'X#A'],
 ['Z#B', 'Y#A']]
```

Takto máme vytvorené pravidlá pre identifikáciu tried nových riadkov.

3. Záver

Algoritmus AQ11 je jedna z metód strojového učenia na získavania znalostí a tvorby pravidiel, vidíme že postup tvorby obálok je pomerne komplikovaný a preto je jeho automatizácia veľmi užitočná a ušetrí množstvo času.

4. Použitá literatúra

Na prípravu dokumentácie boli použité cvičenia a prednášky z predmetu Znalostné systémy prof. Kristíny Machovej