Algoritem Apriori in povezovalna pravila

↓↓↓
Damjan Strnad

→ **Morda ste se kdaj vprašali, na kakšen način trgovci uporabijo znanje o nakupovalnih navadah strank za izboljšanje kakovosti svoje ponudbe, posledično pa tudi za dvig prometa. Ko na blagajni v trgovini plačate za nakup, se seznam artiklov v vaši košarici zabeleži kot *transakcija* (transaction). Z zbiranjem transakcij si trgovec ustvari obsežno bazo nakupovalnih navad svojih kupcev, s pomočjo t. i. kartic zvestobe pa jih lahko vodi celo za vsako gospodinjstvo posebej. Poznavanje nakupovalnih navad lahko trgovec s pridom uporabi za oblikovanje akcijskih ponudb, ki bodo bolj verjetno pritegnile kupce. Prav tako lahko artikle, ki jih stranke pogosto kupujejo v paketu, trgovec na policah postavi blizu skupaj, kar bo strankam olajšalo nakup in hkrati pospešilo prodajo. V tem prispevku bomo opisali algoritem oz. metodo, s pomočjo katere lahko trgovec iz zbirke transakcij izlušči tovrstne koristne informacije. Opisani postopki so uporabni tudi pri ugotavljanju navad uporabnikov v drugih situacijah, ki omogočajo beleženje neke vrste transakcij (npr. hranjenje brskalnih navad uporabnikov svetovnega spleta za prikazovanje reklam).**

Označimo s $\mathcal{T}$ seznam vseh zabeleženih transakcij. Posamezna transakcija $T_i \in \mathcal{T}, i = 1, \ldots, m$, je množica *postavk* (items), t. j. $T_i = \{x_1, x_2, \ldots, x_{n_i}\}$ za $x_i \in X$, kjer je $X = \{x_i; i = 1 \ldots n\}$ množica vseh možnih postavk. V primeru nakupovalne košarice je npr. postavka posamezen tip kupljenega artikla (dva ali več enakih artiklov štejemo kot eno postavko). Prvi korak, ki ga lahko izvedemo pri obdelavi $\mathcal{T}$, je iskanje *pogostih množic postavk* (frequent itemsets), t. j. stvari, ki jih pogosto kupujemo skupaj. Pogostost neke množice postavk S je določena kot delež transakcij v $\mathcal{T}$, pri katerih S nastopa kot podmnožica. Tak delež imenujemo *relativna podpora* (relative support) množice postavk in ga bomo označevali z r_S. Formalno bi to lahko zapisali kot

$$r_S = \frac{|\mathcal{T}_S|}{|\mathcal{T}|},$$

kjer je $\mathcal{T}_S = \{T_i \in \mathcal{T} | S \subset T_i\}$. Vrednost v števcu je velikost množice $\mathcal{T}_S$ in se imenuje *podpora* (support) za množico postavk S.

Množica postavk je pogosta, če je njena podpora enaka ali večja od določene vnaprej postavljene meje $r_{\min} \in (0, 1)$. Pojasnimo to s preprostim zgledom, v katerem množico vseh postavk X sestavlja pet artiklov, ki jih bomo označevali s črkami od a do e (bralec si lahko predstavlja, da a pomeni *ananas*, b *banane* itd.). Podana naj bo naslednja zbirka $\mathcal{T}$ desetih transakcij:

Transakcija	a	b	c	d	e
T_1	x	x			x
T_2	x	x	x		
T_3		x	x		x
T_4		x			
T_5	x	x		x	x
T_6	x		x		
T_7			x	x	
T_8		x			x
T_9	x		x		x
T_{10}	x	x			x

V tabeli oznaka 'x' v določenem stolpcu in vrstici pomeni, da je pripadajoči artikel prisoten v ustrezni transakciji. Tako lahko npr. transakcijo T_1 zapišemo kot $T_1 = \{a, b, e\}$.

Iskanje pogostih množic postavk z grobim pristopom bi pomenilo pregledovanje vseh neprazniih podmnožic množice X. Za množico postavk $S = \{a, b, c\}$ bi tako ugotovili, da je njena relativna podpora enaka $r_{\{a,b,c\}} = 0.1$, saj se artikli a, b in c skupaj nahajajo samo v eni (T_2) izmed desetih transakcij. V nadaljevanju bomo zaradi preglednosti množice postavk zapisovali v strnjeni obliki, npr. $abd \equiv \{a, b, d\}$.

Število neprazniih množic postavk pri n postavkah je enako $2^n - 1$ (t.j. moč potenčne množice $\mathcal{P}(X)$ brez $\emptyset$), zato je časovna zahtevnost grobega pristopa eksponentna. V zgornjem primeru s petimi postavkami je grobi pristop še izvedljiv, v realnih situacijah z več 100 ali 1000 artikli pa bi bilo pregledovanje celotne potenčne množice praktično neobvladljivo. Algoritem *Apriori* je preprosta izboljšava grobega pristopa, ki potrebno število pregledanih množic postavk zmanjša z upoštevanjem naslednjega dvodelnega pravila:

Pravilo Apriori a) Vse podmnožice pogoste množice postavk so pogoste in b) nobena nadmnožica nepogoste množice postavk ni pogosta.

Označimo z $\mathcal{F}$ seznam vseh pogostih množic postavk, ki jih algoritem Apriori poišče. Za vsako množico postavk bomo vodili tudi njeno relativno podporo, zato bodo elementi seznama $\mathcal{F}$ pari oblike $\langle S, r_S \rangle$. Delovanje algoritma povzema naslednja psevdokoda:

Algoritem 1 Apriori$(\mathcal{T}, X, r_{\min})$

```
1:  F = ∅        # inicializiraj prazen seznam pogostih množic postavk
2:  D_0 = {∅}    # inicializiraj delovni seznam množic postavk
3:  j = 0        # števec iteracij
4:  ponavljaj
5:      j = j + 1
6:      D_j = {Q ∪ {x} | Q ∈ D_{j-1} ∧ x ∈ X\Q}
7:      odstrani iz D_j vse množice S,
            za katere velja ∃y ∈ S : S\{y} ∉ D_{j-1}
8:      izračunaj relativno podporo
            množicam postavk v D_j
9:      odstrani iz D_j vse množice S,
            za katere velja r_S < r_min
10:     F = F ∪ D_j
11: dokler D_j ≠ ∅
12: vrni F
```

Algoritem za svoje delovanje uporablja delovni seznam $\mathcal{D}$, v katerem hrani pogoste množice postavk, najdene v zadnjem koraku. Za lažje sledenje algoritmu bomo z $\mathcal{D}_i$ označevali stanje delovnega seznama po izvedbi i-te iteracije zanke. V prvi iteraciji algoritem Apriori poišče pogoste množice posameznih postavk in jih v kasnejših iteracijah razširja. Pri vsakem nadaljevanju iskanja algoritem upošteva prvi del prej zapisanega pravila v koraku 7, kjer obdrži samo tiste razširjene množice postavk, ki ne vsebujejo nobene nepogoste podmnožice. Drugi del pravila algoritem upošteva v koraku 9, kjer iz nadaljnje obravnave izloči vse nepogoste množice postavk, saj tudi nobena njihova nadmnožica ne more biti pogosta.

Delovanje algoritma demonstrirajmo na prejšnjem primeru za $r_{\min} = 0.3$. Po začetni inicializaciji algoritem med izvajanjem prvega cikla zanke tvori naslednje množice postavk z njihovimi relativnimi podporami:

S	a	b	c	d	e
r_S	0.6	0.7	0.5	0.2	0.6

Množico postavk $\{d\}$ algoritem v koraku 9 odstrani, ker nima zadostne podpore, tako da ob zaključku prve zanke velja

- $\mathcal{D}_1 = \{a, b, c, e\}$ in
  $\mathcal{F} = \{\langle a, 0.6\rangle, \langle b, 0.7\rangle, \langle c, 0.5\rangle, \langle e, 0.6\rangle\}$.

V naslednji iteraciji algoritma v koraku 6 vsako od množic postavk iz $\mathcal{D}_1$ razširimo na vse možne načine z eno dodatno postavko. Iz tako dobljenega delovnega seznama $\mathcal{D}_2$ sproti izločimo podvojene množice postavk (npr. $\{a\} \cup \{b\}$ in $\{b\} \cup \{a\}$), v koraku 7 pa še tiste, katerih podmnožice niso v $\mathcal{D}_1$ (vse kombinacije s postavko d). Po izračunu relativne podpore dobimo

S	ab	ac	ae	bc	be	ce
r_S	0.4	0.3	0.4	0.2	0.5	0.2

Zaradi prenizke podpore tokrat izločimo množici postavk bc in ce, tako da je stanje seznamov po zaključku druge iteracije algoritma naslednje:

- $\mathcal{D}_2 = \{ab, ac, ae, be\}$
  $\mathcal{F} = \{\langle a, 0.6\rangle, \langle b, 0.7\rangle, \langle c, 0.5\rangle, \langle e, 0.6\rangle, \langle ab, 0.4\rangle, \langle ac, 0.3\rangle, \langle ae, 0.4\rangle, \langle be, 0.5\rangle\}$.

V tretji iteraciji algoritma Apriori računamo podporo samo še za množico postavk abe, saj je edina, katere vse podmnožice so v $\mathcal{D}_2$. Z relativno podporo $r_{abe} = 0.3$ množica postavk abe izpolnjuje pogoj za obstoj v seznamu $\mathcal{D}_3$. V četrti iteraciji algoritma pogoja v koraku 7 ne izpolni več nobena razširitev množice postavk abe in algoritem se zaključi. Končni seznam pogostih množic postavk je tako

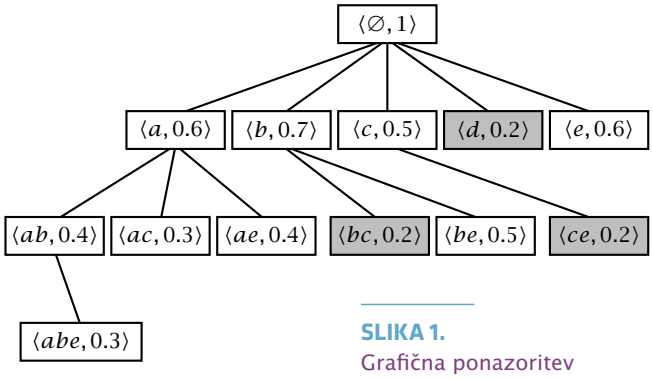
- $\mathcal{F} = \{\langle a, 0.6\rangle, \langle b, 0.7\rangle, \langle c, 0.5\rangle, \langle e, 0.6\rangle, \langle ab, 0.4\rangle, \langle ac, 0.3\rangle, \langle ae, 0.4\rangle, \langle be, 0.5\rangle, \langle abe, 0.3\rangle\}$.

Delovanje algoritma lahko ponazorimo tudi grafično z drevesom, kakršno je za naš primer prikazano na sliki 1. Algoritem Apriori začne v korenu in drevo razvija po nivojih. Vsa vozlišča, obarvana s sivo, predstavljajo nepogoste množice postavk, ki se odstranijo v koraku 9. Izmed vozlišč sinov vsakega vozlišča starša so zaradi preglednosti prikazana samo tista, ki jih ne odstranimo v koraku 7.

Računska zahtevnost algoritma Apriori je v najslabšem primeru še vedno eksponentna, vendar v realnih primerih število pogostih množic postavk z nivoji drevesa hitro upada, zato je praktična časovna zahtevnost običajno precej manjša.

Pridobitev seznama pogostih množic postavk je ponavadi samo prvi korak do praktične uporabe teh informacij. Iz pogoste množice postavk $Z_i \in \mathcal{F}$ lahko izpeljemo eno ali več *povezovalnih pravil* (association rules) oblike $P_{i,j} : X_{i,j} \to Y_{i,j}$, kjer $X_{i,j} \subset Z_i$ in $Y_{i,j} = Z_i \backslash X_{i,j}$. Veljavnost posameznega pravila $P_{i,j}$ izražamo z *zaupanjem* (confidence) $c_{i,j}$, ki se izračuna po naslednji enačbi:

- $c_{i,j} = c_{X_{i,j} \to Y_{i,j}} = \dfrac{r_{Z_i}}{r_{X_{i,j}}}$.



SLIKA 1.
Grafična ponazoritev delovanja algoritma Apriori

Pogosto določimo prag zaupanja $c_{\min} \in (0, 1)$ in obdržimo samo pravila s $c_{i,j} \geq c_{\min}$. Takim pravilom rečemo, da so *zanesljiva* (strong). S povezovalnim pravilom P in pripadajočim zaupanjem c izražamo verjetnost, da se bo v seznamu, ki že vsebuje množico postavk X, pojavila tudi množica postavk Y, npr. da bo kupec ob artiklih X kupil še artikle Y. S pomočjo takšnih pravil lahko trgovec oblikuje politiko akcijskih cen (npr. zniža ceno samo artiklom X, ker pričakuje, da bo kupec potem kupil še neznižane artikle Y) ali postavitev artiklov na policah (npr. postavi artikle Y blizu artiklov X, da vzpodbudi večji nakup).

Izčrpno iskanje vseh zanesljivih povezovalnih pravil lahko optimiziramo s preprosto izboljšavo. Ko izpeljujemo povezovalna pravila iz pogoste množice postavk Z, pričnemo z maksimalno podmnožico X. Če po izračunu zaupanja ugotovimo, da $c_{X \to Y} < c_{\min}$, potem lahko iz nadaljnje obravnave izločimo vse podmnožice od X. Za vsak $W \subset X$ namreč velja $r_W \geq r_X$ in posledično $c_{W \to Z \backslash W} \leq c_{X \to Z \backslash X}$.

Poiščimo zanesljiva povezovalna pravila za naš primer, če je $c_{\min} = 0.75$:

- $P_1 : a \to b$ z zaupanjem $c_1 = \frac{0.4}{0.6} = 0.67$
- $P_2 : b \to a$ z zaupanjem $c_2 = \frac{0.4}{0.7} = 0.57$
- $P_3 : a \to c$ z zaupanjem $c_3 = \frac{0.3}{0.6} = 0.5$
- $P_4 : c \to a$ z zaupanjem $c_4 = \frac{0.3}{0.5} = 0.6$
- $P_5 : a \to e$ z zaupanjem $c_5 = \frac{0.4}{0.6} = 0.67$
- $P_6 : e \to a$ z zaupanjem $c_6 = \frac{0.4}{0.6} = 0.67$
- $P_7 : b \to e$ z zaupanjem $c_7 = \frac{0.5}{0.7} = 0.71$
- $P_8 : e \to b$ z zaupanjem $c_8 = \frac{0.5}{0.6} = 0.83$
- $P_9 : ab \to e$ z zaupanjem $c_9 = \frac{0.3}{0.4} = 0.75$
- $P_{10} : ae \to b$ z zaupanjem $c_{10} = \frac{0.3}{0.4} = 0.75$
- $P_{11} : be \to a$ z zaupanjem $c_{11} = \frac{0.3}{0.5} = 0.6$
- $P_{12} : a \to be$ z zaupanjem $c_{12} = \frac{0.3}{0.6} = 0.5$

Od vseh izpeljanih pravil bi obdržali samo pravila P_8, P_9 in P_{10}. Zaupanja za pravili $b \to ae$ in $e \to ab$ nismo računali, potem ko smo ugotovili, da $c_{11} < c_{\min}$.

Literatura

[1] M. J. Zaki in W. Meira Jr., *Data Mining and Analysis: Fundamental Concepts and Algorithms*, Cambridge University Press, 2014.