Capitolul 7

Data mining

Ce este Data mining?

- Iniţial “data mining” (căutarea în date, extragerea de cunostinte din date) a fost un termen din statistică însemnând suprautilizarea datelor pentru a deduce inferenţe invalide.
- Actualmente: descoperirea unor informatii sumare utile despre date (cunostinte, sabloane, etc)

Exemple de succes

- Arbori de decizie construiţi din istoria împrumuturilor bancare pentru a produce algoritmi de decizie in vederea acordării împrumuturilor.
- Şabloane privind comportamentul călătorilor folosite pentru a gestiona vânzarea locurilor cu reducere la cursele aeriene, a camerelor de hotel, etc.

Exemple de succes

- "Scutece si bere": observaţia că cei care cumpără scutece cumpără bere mai mult decât media a permis supermagazinelor să plaseze berea şi scutecele aproape unele de altele, ştiind că mulţi cumpărători vor circula între ele. Plasarea cipsurilor de cartofi între cele două a crescut vânzările la cele trei articole.

Exemple de succes

- Skycat si Sloan Sky Survey: gruparea obiectelor de pe cer după nivelul lor de radiaţie în 7 benzi a permis astronomilor să delimiteze galaxii, stele apropiate şi alte feluri de obiecte cereşti (clustering intr-un spatiu 7-dimensional)

Exemple de succes

- Compararea genotipului persoanelor îndeplinind sau nu o anumită condiţie a permis descoperirea unei mulţimi de gene care împreună determină multe cazuri de diabet. Acest mod de căutare în date va deveni mult mai important în momentul construirii genomului uman.

Comunitati implicate

- Statistica
- Inteligenţa artificială (IA) unde este denumită “machine learning”.
- Cercetatorii în “clustering algorithms” (algoritmi de grupare)
- Cercetatorii in “visualization” (vizualizarea datelor)

Comunitati implicate

- Baze de date. Vom continua bineînţeles cu această abordare, concentrându-ne asupra provocărilor care apar atunci când cantitatea de date este mare iar calculele sunt complexe.
- Într-un anumit sens data mining poate fi văzută ca mulţimea algoritmilor pentru execuţia unor cereri foarte complexe asupra unor date care nu sunt în memoria centrala a calculatorului.

Domeniu heterogen

- Exista multe clase de probleme in domeniul data mining.
- Vom studia la curs cateva clase de probleme: articole si multimi frecvente, articole corelate, clustering
- Exista insa multe altele, pentru fiecare clasa de probleme existand algoritmi specifici

Nu orice e Data Mining

- Exemplu faimos: David Rhine, un "parapsiholog" de la Duke a testat în anii 1950 studenţii pentru "percepţie extrasenzorială" (PE) cerându-le să ghicească pentru 10 cărţi culoarea - roşu sau negru.
- A descoperit că 1/1000 din ei au ghicit toate cele 10 cărţi.

Nu orice e Data Mining

- în loc să realizeze că este ceea ce trebuia aşteptat din ghicirea aleatoare i-a declarat pe cei care au ghicit toate cartile ca având percepţie extrasenzorială.
- Când i-a retestat a descoperit că nu sunt mai buni decât media. Concluzia sa: a spune oamenilor că au percepţie extrasenzorială duce la pierderea acesteia!

De ce?

- Teorema lui Bonferroni ne avertizează ca atunci când exista prea multe concluzii posibile unele dintre acestea vor fi adevărate *din motive pur statistice*.
- In cazul PE, $1/2^{10}$ (=~ 1/1000) este probabilitatea ca cineva sa ghiceasca toate cele 10 carti (culoarea lor).
- Deci in acest caz era vorba de o concluzie care putea fi trasa si fara a testa efectiv vreun student, fiind vorba de un adevar statistic

Etapele procesului

- *Colectarea datelor*, e.g. din depozitele de date (data warehouse), parcurgerea webului.
- De remarcat ca algoritmii de data mining sunt ganditi pentru masive mari de date care nu incap in memoria centrala a calculatorului
- Multi algoritmi optimizeaza numarul de treceri prin date (care sunt pe disc)

Etapele procesului

- *Curatarea datelor*: eliminarea erorilor evidente din date, e.g. temperatura pacientului=125
- Aceste erori se pot elimina automat prin fixarea unor reguli formale pe care valorile trebuie sa le respecte.
- Nu se face manual, volumul datelor este in astfel de cazuri foarte mare

Etapele procesului

- *Extragerea proprietatilor*: obtinerea doar a atributelor de interes ale datelor,
- Exemple:
  - "data colectarii datelor" nu este probabil de interes in gruparea obiectelor ceresti in cazul Skycat.
  - Numele casierei, numarul casei nu sunt de interes in cazul cautarii multimilor de produse care se gasesc frecvent impreuna in cosurile de cumparaturi

Etapele procesului

- *Extragerea sablonelor si descoperirea*: Aceasta este etapa considerata adesea ca fiind "data mining" si aici ne vom concentra eforturile.
- Pentru fiecare clasa de probleme discutate vom prezenta algoritmi specifici

Etapele procesului

- Vizualizarea datelor: rezultatele obtinute sunt de multe ori voluminoase.
- Pentru interpretarea lor e necesara etapa de vizualizare care are ca scop prezentarea acestora intr-o forma vizuala, grafica, facilitand intelegerea rezultatelor de catre operatorul uman.

Etapele procesului

- Evaluarea rezultatelor: nu orice fapt descoperit este si util ori adevarat, asa cum s-a vazut si din exemplul cu PE.
- Este necesara o evaluare de catre specialisti si beneficiari inainte de a urma concluziile programelor de Data mining.

Prima clasa de probleme

Reguli de asociere si multimi frecvente de articole (frequent itemsets)

Problema

- Problema cosului de produse presupune ca avem un mare numar de articole, e.g. “paine”, “lapte”.
- Cumparatorii pun in cosul lor de produse anumite submultimi de articole iar noi vom afla ce articole sunt cumparate impreuna chiar daca nu si de cine.
- Vanzatorii utilizeaza aceste informatii pentru asezarea articolelor in rafturi si controlez modul in care un cumparator tipic traverseaza magazinul.

Alte utilizari

- Cos = documente; articole = cuvinte.
- Cuvintele aparand frecvent impreuna in documente pot reprezenta fraze su concepte legate intre ele.
- Poate fi utilizat pentru colectarea de informatii (intelligence).

Alte utilizari

- Cos = propozitii, articole = documente.
- Doua documente continand celeasi propozitii pot reprezenta un plagiat sau “mirror sites” pe web.

Obiective DM

- Gasirea de:
  - Reguli de asociere
  - Cauzalitate
  - Multimi frecvente de articole

Reguli de asociere

- *Regulile de asociere* sunt propozitii de forma $\{X_1, X_2, ..., X_n\} \Rightarrow Y$ insemnand ca daca gasim toate $X_1, X_2, ..., X_n$ in cos atunci sunt mari sanse sa-l gasim si pe Y.
- Probabilitatea de a-l gasi pe Y pentru a accepta aceasta regula este numita *increderea* acelei reguli.
- In mod normal vom cauta doar reguli care au o incredere peste un anumit prag.

Reguli de asociere

- Putem de asemenea cere ca increderea sa fie semnificativ mai mare decat in cazul in care articolele ar fi plasate aleator in cos.
- De exemplu putem gasi o regula ca $\{lapte, unt\} \Rightarrow paine$ doar pentru ca foarte multa lume cumpara paine.
- Totusi exemplul bere/scutece arata ca regula $\{scutece\} \Rightarrow \{bere\}$ este verificata cu o incredere semnificativ mai mare decat a submultimii de cosuri continand bere.

Cauzalitate

- *Cauzalitate.* Ideal, a, vrea sa stim daca intr-o regula de asociere prezenta elementelor X_1, X_2, ..., X_n efectiv "cauzeaza" (determina) cumpararea lui Y.
- "Cauzalitatea" este insa un concept echivoc.
- Exemplu:

Cauzalitate

- Daca scadem pretul scutecelor si crestem pretul berii putem ademenii cumparatorii de scutece care au inclinatia de a cumpara bere din magazin, acoperind astfel pierderile din vanzarea scutecelor.
- Aceasta strategie este valabila deoarece “scutece determina bere”.

Cauzalitate

- Actiunea inversa, micsorarea pretului la bere si marirea pretului scutecelor nu va determina cumparatorii de bere sa cumpere scutece in numar mare si vom pierde bani deoarece “bere determina scutece” nu este adevarata.

Multimi frecvente

- *Multimi frecvente de articole (frequent itemsets)*: in multe situatii (dar nu in toate) ne intereseaza doar regulile de asociere si cauzalitatea in ceea ce priveste multimi de articole care apar frecvent in cosuri.
- De exemplu, nu putem conduce o buna strategie de marketing care implica produse pe care oricum nu le cumpara nimeni.

Multimi frecvente

- Astfel, cautarile in date pornesc de la premiza ca ne intereseaza doar multimile de articole cu un larg suport;
- Larg suport = ele apar impreuna in multe cosuri de produse.
- Gasim apoi reguli de asociere sau cauzalitatile implicand doar articolele cu larg suport (i.e. $\{X_1, X_2, \ldots, X_n, Y\}$ trebuie sa apara in cel putin un anumit procent din cosuri, numit *prag de suport*)

Cadru de cautare

- Utilizam termenul *multimi frecvente de articole (frequent itemsets)* pentru "o multime de articole S care apare in cel putin a "s"-a parte din cosuri", unde s este o constanta aleasa, de obicei 0.01 sau 1%.
- Vom presupune ca avem o cantitate de date care nu incape in memoria centrala a calculatorului.

Cadru de cautare

- Stocare (pentru exemplele urmatoare:
  - Fie sunt stocate intr-o baza de date relationale (BDR), de exemplu o relatie (tabela) *Cosuri(IdCos, articol)*
  - Fie ca un fisier de inregistrari de forma *(ICos, articol1, articol2, …, articol-n)*. Cand evaluam timpul de rulare al algoritmilor:

Cadru de cautare

- Cand evaluam timpul de rulare al algoritmilor parametrul optimizat este numarul de treceri prin date.
- Deoarece costul principal este dat adesea de timpul necesar citirii datelor de pe disc, numarul necesar de citiri pentru fiecare data este adesea cea mai buna masura a timpului de rulare al algoritmului.

Principiul a-priori

- Exista un principiu cheie, numit **monotonicitate* (monotonicity) sau *principiul a-priori* care ne ajuta sa gasim multimile frecvente de articole:
- Daca o multime de articole S este frecventa (i.e., apare cel putin in a "s"-a parte a cosurilor), atunci orice submultime a lui S este de asemenea frecventa.

Abordari

- Pentru a gasi multimile frecvente de articole avem doua abordari:

1. Nivel cu nivel (aplicand a-priori)
2. Toate multimile de articole - de orice dimensiune - in cateva treceri (2-3 treceri)

Nivel cu nivel

- Procedam nivel cu nivel, gasind intai articolele frecvente (multimi de dimensiune 1), apoi perechile frecvente, tripletele frecvente, etc.
- In cele ce urmeaza ne vom concentra pe gasirea perechilor frecvente deoarece:
  - Adeseori perechile sunt suficiente.

Nivel cu nivel

- In multe multimi de date partea cea mai grea este gasirea perechilor; continuarea pe nivelele superioare necesita mai putin timp decat gasirea perechilor frecvente

- Algoritmii de acest tip utilizeaza o trecere per nivel.

Toate multimile

- Gasim toate *multimile frecvente de articole maximale* (i.e., multimile S a.i. nici o multime care include strict pe S nu este frecventa) intr-o singura trecere sau in cateva treceri.
- Tehnicile de acest tip includ fie esantionarea datelor fie citirea lor in transe
- De obicei sunt suficiente 2 treceri prin date (de ex. O trecere pentru esantionare si inca una pentru verificarea finala)

Algoritmul a-priori

Acest algoritm procedeaza nivel cu nivel.

1. Dandu-se pragul de suport *s*, in prima trecere gasim articolele care apar in cel putin a "*s*"-a parte a cosurilor. Aceasta multime este notata L_1, multimea articolelor frecvente.

Algoritmul a-priori

2. Perechile de articole din L_1 devin multimea C_2 a *perechilor candidate* pentru a doua trecere. Speram ca dimensiunea lui C_2 nu este atat de mare pentru ca altfel nu este suficient spatiu in memoria centrala pentru un contor numeric intreg al aparitiei fiecarei perechi. Perechile din C_2 al caror contor ajunge sau depaseste pe s formeaza multimea L_2 a perechilor frecvente.

Algoritmul a-priori

3. Tripletele candidate C_3 sunt multimile $\{A, B, C\}$ pentru care $\{A, B\}$, $\{A, C\}$ si $\{B, C\}$ sunt in L_2. In a treia trecere sunt numarate aparitiile tripletelor din C_3; cele al caror contor este cel putin s formeaza multimea L_3 a tripletelor frecvente.

Algoritmul a-priori

4. Se poate merge oricat de departe se doreste (sau pana multimile devin vide). L_i contime multimile frecvente de articole de dimensiune i; C_{i+1} este multimea candidatelor de dimensiune $i+1$ a.i. fiecare submultime a lor de dimensiune i este inclusa in L_i.

Efect a-priori

- Sa consideram cererea SQL din slide-ul urmator
  - pe o relatie (tabela) *Cosuri(IdCos, articol)*
  - avand 10^8 tupluri
  - care contin date despre 10^7 cosuri
  - de cate 10 articole fiecare.
  - Presupunem existenta a 100.000 articole diferite (tipic pentru o retea ca Wal-Mart de exemplu).

Efect a-priori

```
SELECT b1.articol, b2.articol,
  COUNT(*)
FROM Cosuri b1, Cosuri b2
WHERE b1.IdCos = b2.IdCos AND
  b1.articol < b2.articol
GROUP BY b1.articol, b2.articol
HAVING count(*) >= s;
```

Efect a-priori

- *s* este pragul de suport (nu in procente ci in valoare absoluta) iar al doilea termen al clauzei WHERE elimina perechile formate din acelasi produs si aparitia de doua ori a aceleiasi perechi.
- In joinul *Cosuri* ⋈ *Cosuri* fiecare cos contribuie cu $C_2^{10} = 45$ de perechi astfel incat joinul are $4,5 \times 10^8$ tupluri (multe!).

Efect a-priori

- A-priori "impinge clauza HAVING in jos pe arborele expresiei", determinandu-ne in primul rand sa inlocuim *Cosuri* cu rezultatul cererii:

```
SELECT *
FROM Cosuri
GROUP BY articol
HAVING COUNT(*) >= s;
```

Efect a-priori

- Daca s = 0,01 atunci cel mult 1000 de grupuri de articole pot trece de clauza HAVING.
- Motivul: sunt 10^8 linii continand articole in relatia Cosuri iar fiecare articol are nevoie de 0,01 x 10^7 = 10^5 dintre acestea pentru a aparea in 1% din cosuri.
- Rezulta o diminuare a tabelei Cosuri si implicit a volumului de date pentru calculul joinului cu ea insasi.

Dar ...

- Desi 99% dintre articole sunt eliminate de algoritmul a-priori nu trebuie sa concluzionam ca relatia *Cosuri* care rezulta are doar 10^6 tupluri.
- In fapt, *toate* tuplele pot fi pentru prodse cu larg suport.
- Totusi, in situatiile reale, micsorarea relatiei *Cosuri* este substantiala si dimensiunea joinului scade cu patratul acestei micsorari.

Imbunatatiri pentru a-priori

De doua tipuri:

1. Micsorarea dimensiunii multimilor candidat C_i pentru i ≥ 2.

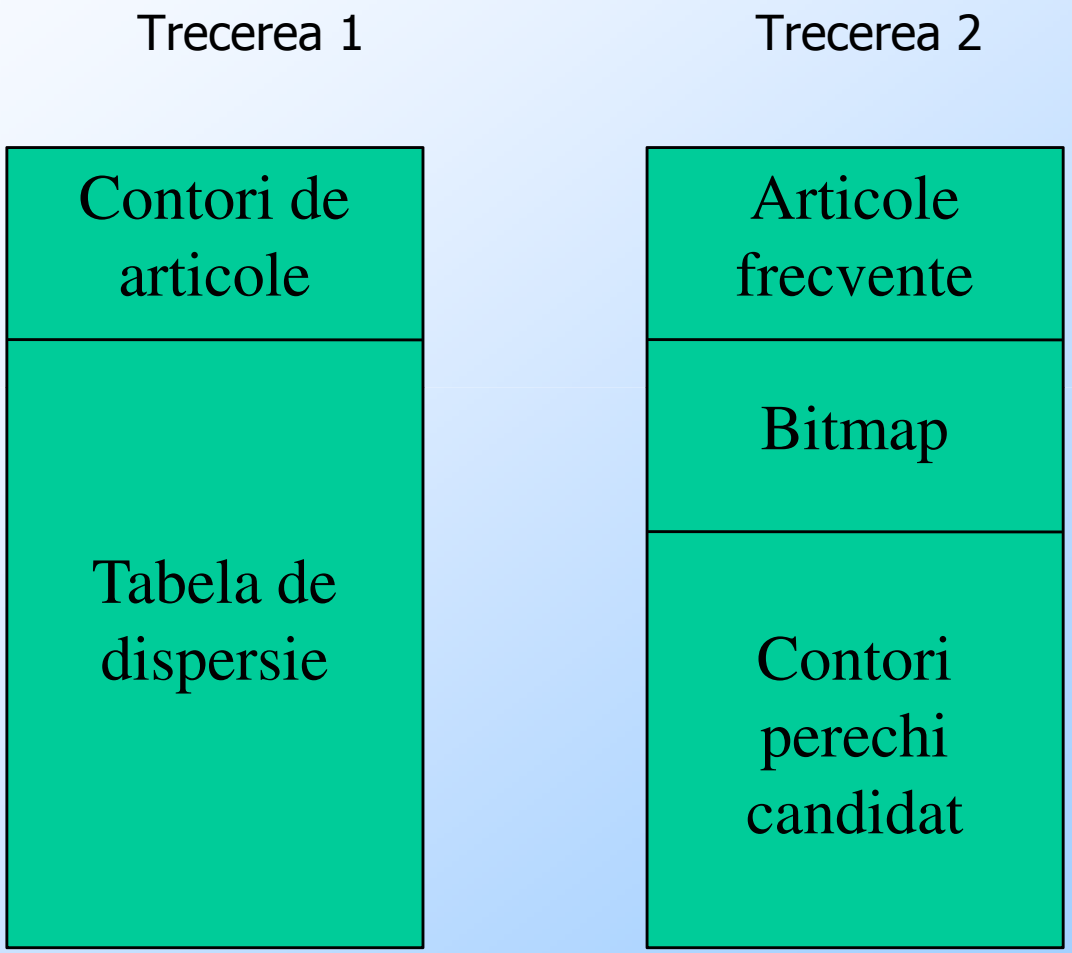
- Aceasta optiune este importanta chiar si pentru gasirea perechilor frecvente deoarece numarul de elemente trebuie sa fie suficient de mic pentru ca un contor de aparitii pentru fiecare sa fie tinut in memoria centrala.

2. Contopirea incercarilor de gasire a L_1, L_2, L_3, ... in doar una sau doua treceri in loc de o trecere per nivel.

PCY

- Park, Chen si Yu au propus, utilizand o tabela de dispersie, sa determine la prima trecere (cand este calculat L_1) ca multe perechi nu sunt frecvent posibile.
- Profita de faptul ca memori centrala este uzual *mult* mai mare decat numarul de articole.
- In timpul celor doua faze pentru gasirea lui L_2 memoria centrala este ocupata ca in figura urmatoare.
- Presupunem ca datele sunt stocate intr-un fisier cu inregistrari constand dintr-un identificator IdCos si o lista cu articolele sale.

PCY


Trecerea 1
Trecerea 2
Contori de articole
Tabela de dispersie
Articole frecvente
Bitmap
Contori perechi candidat

Trecerea 1

- Se numara aparitiile fiecarui articol
- Pentru fiecare cos constand din articolele $\{i_1, ..., i_k\}$, se aplica functia de dispersie fiecarei perechi asociind-o unei intrari a tabelei de dispersie si se incrementeaza contorul acesteia cu 1.
- La sfarsitul trecerii, se determina L_1, articolele cu contorul cel putin *s*.

Trecerea 1

- De asemenea la sfarsitul trecerii se determina acele intrari din tabela de dispersie care au contororul cel putin egal cu *s*.
- Punct cheie: o pereche (i, j) nu poate fi frecventa decat daca este dispersata intr-o intrare frecventa astfel incat perechile care sunt dispersate in alte intrari NU POT fi candidate in C_2.

Trecerea 1

- Se inlocuieste tabela de dispersie cu un bitmap avand un bit per intrare: 1 daca intrarea a fost frecventa, 0 altfel.
- Acest bitmap va fi folosit la trecerea 2 prin date.

Trecerea 2

- Memoria centrala contine o lista cu toate articolele frecvente, i.e. L_1.
- Tot memoria centrala contine un bitmap reprezentand rezultatele dispersiei din prima trecere.
- Punct cheie: intrarile utilizeaza 16 sau 32 de biti pentru un contor dar sunt comptimate la un singur bit.

Trecerea 2

- Astfel, chiar daca tabela de dispersie ocupa aproape intreaga memorie centrala la ptima trecere, bitmapul sau nu ocupa mai mult de 1/16 din memoria centrala la trevecea 2.
- In final, memoria centrala contine de asemenea o tabela cu toate perechile candidat si contorii asociati lor.

Trecerea 2

- O pereche *(i, j)* poate fi candidata in C_2 doar daca *toate* conditiile urmatoare sunt adevarate:
  1. *i* este in L_1.
  2. *j* este in L_1.
  3. *(i, j)* este dispersata intr-o intrare frecventa (se utilizeaza bitmapul)
- Ultima conditie diferentiaza PCY de a-priori clasic si reduce necesarul de memorie in trecerea 2.

Trecerea 2

- In timpul trecerii 2 luam in considerare fiecare cos si fiecare pereche de articole din el, efectuand testul de mai sus.
- Daca o pereche indeplineste tote cele trei conditii, se incrementeaza contorul acesteia din memorie sau se creaza unul daca acesta nu exista inca.
- In final perechile numarate care au un contor de cel putin s formeaza L2.

Q & A

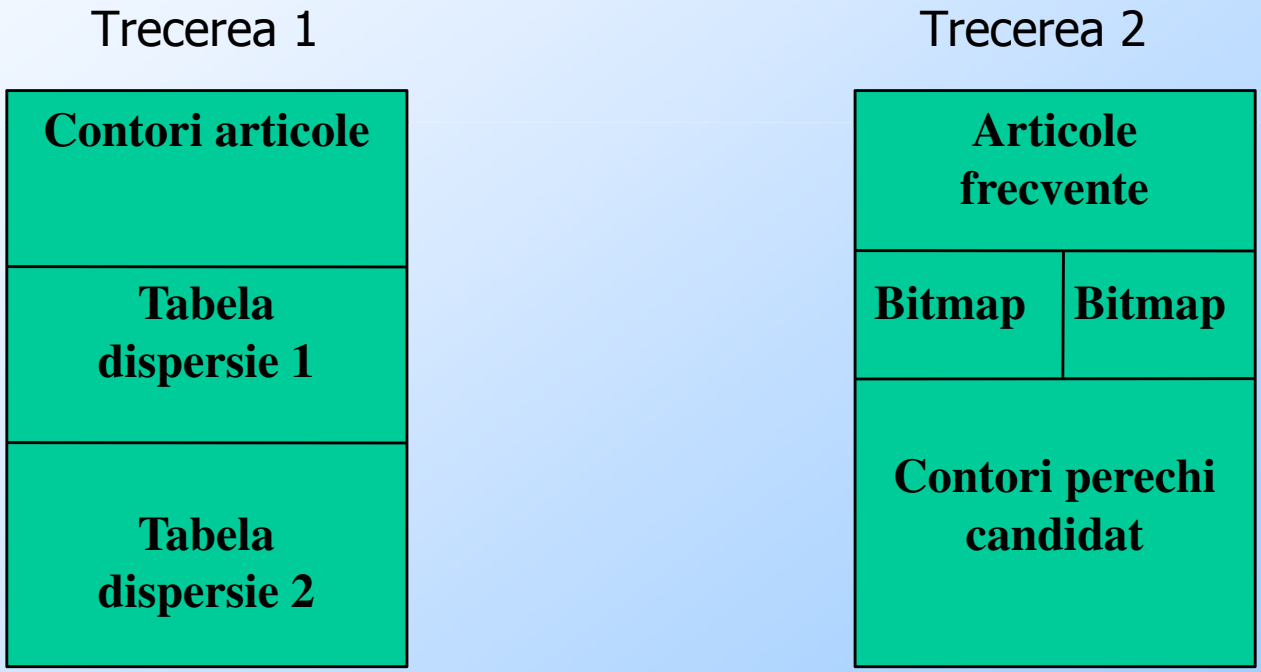
- Q: Cand este mai performant PCY decat a-priori?
- A: Cand sunt prea multe perechi de articole din L_1 pentru a incapea intr-o tabela de perechi candidate si de contori asociati in memoria centrala iar numarul de intrari frecvente din algoritmul PCY este suficient de mic pentru a reduce dimensiunea lui C_2 suficient pentru a incapea in memoria centrala (chiar si fara 1/16 din ea consumata de bitmap).

Q & A

- Q: Cand o mare parte a intrarilor nu vor fi frecvente in PCY?
- A: Cand sunt putine perechi frecvente iar cea mai mare parte a perechilor sunt atat de putin frecvente incat chiar daca sumam contorii tuturor perechilor care sunt dispersate intr-o intrare data nu sunt mari sanse sa se obtina o valoare egala sau mai mare ca *s*.

Tabele de dispersie multiple

- Varianta a PCY
- Se imparte memoria intre doua sau mai multe tabele de dispersie, ca in figura urmatoare:



Tabele de dispersie multiple

- La trecerea 2 se retine in memorie cate un bitmap pentru fiecare dintre acestea; de notat ca spatiul necesar pentru aceste bitmap este exact acelasi cu cel necesar in bitmapul unic din PCY deoarece numarul total de intrari reprezentate este acelasi. Pentru a fi candidata la C_2 o pereche:
- Consta din articole din L_1, si
- Este dispersata intr-o intrare frecventa in *fiecare* tabela de dispersie.

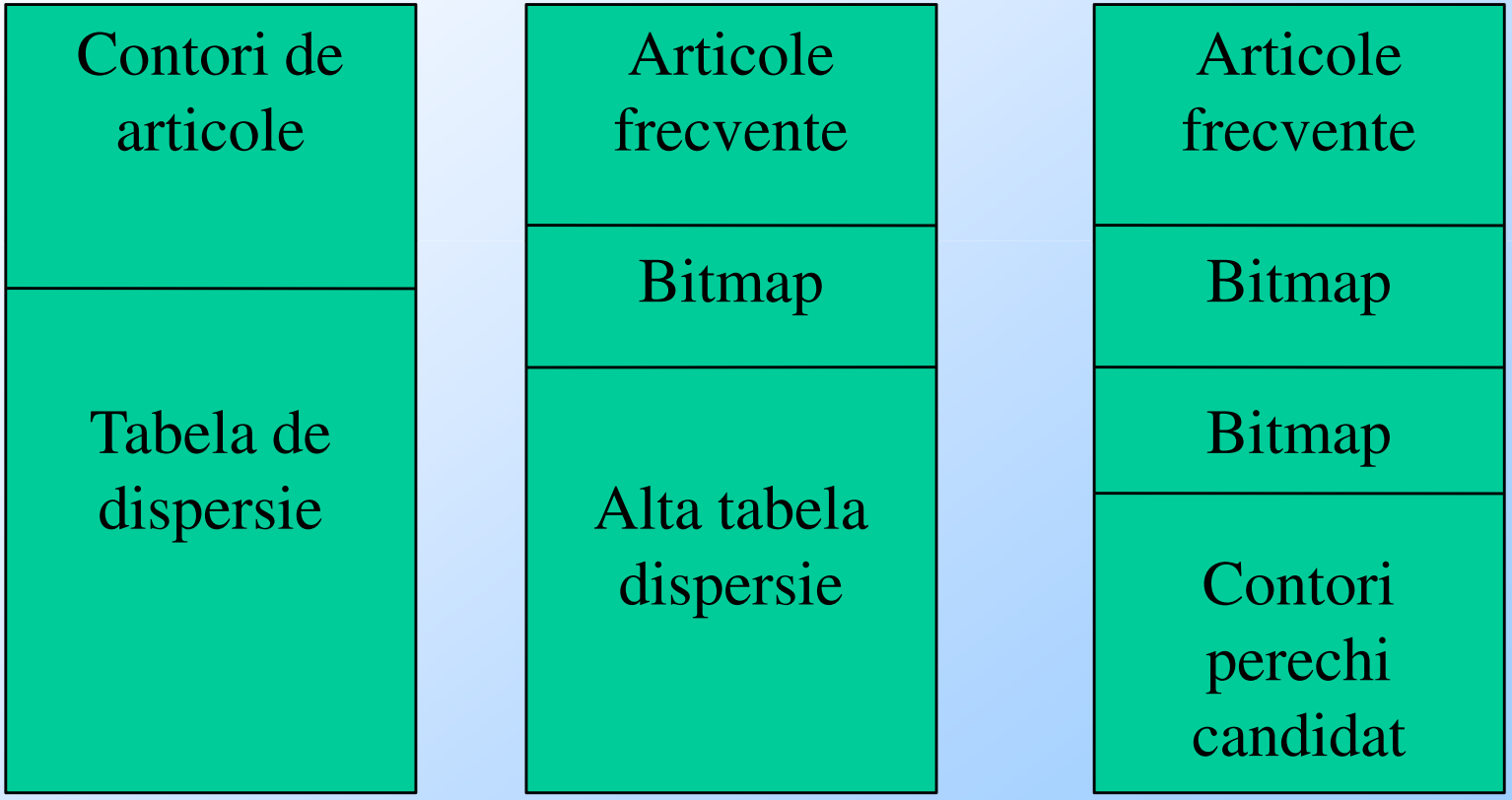
Tabele de dispersie iterate

- Tabele de dispersie iterate *Multistage:*
- In locul verificarii candidatelor in trecerea 2 se creaza o alta tabela de dispersie (alta functie de dispersie) in trecerea 2 dar sunt dispersate doar acele perechi care indeplinesc conditiile pentru PCY; i.e., ambele articole sunt din L_1 si sunt dispersate intr-un intrare frecventa in prima trecere.

Tabele de dispersie iterate

- Intr-a treia trecere, pastram cate un bitmap pentru fiecare tabel ade dispersie si tratam o pereche ca o candidata la C_2 doar daca:
  - Ambele articole sunt in L_1.
  - Perechea a fost dispersata intr-o intrare frecventa in prima trecere.
  - Perechea a fost dispersata de asemenea intr-o intrare frecventa la trecerea 2.

Tabele de dispersie iterate



Q & A

- Q: Cand sunt utile tabelele de dispersie multiple?
- A: Cand cele mai multe dintre intrari de la prima trecere a PCY au contori cu mult sub pragul *s*. Atunci putem dubla contorii intrarilor si totusi cele mai multe vor fi sub prag.

Q & A

- Q: Cand sunt utile tabelele de dispersie Multistage?
- A: Cand numarul intrarilor frecvente din prima trecere este mare (e.g. 50%) - dar nu toate. Atunci, a doua dispersie cu unele dintre perechile ignorate poate reduce semnificativ numarul de intrari.

Toate multimile in 2 treceri

- Metodele de mai sus sunt cele mai bune cand dorim perechile frecvente, cazul cel mai comun.
- Daca dorim toate multimile frecvente maximale de articole, inclusiv multimi mari, sunt necesari prea multi pasi.
- Exista mai multe abordari pentru obtinerea tuturor multimilor frecvente de articole in doua treceri sau mai putin.

Abordarea simpla

- *Abordarea simpla*: Se ia un esantion de date de dimensiunea memoriei centrale.
- Se ruleaza un algoritm nivel cu nivel in memoria centrala (deci nu sunt costuri de I/O) si
- Se spera ca esantionul ne va conduce la adevaratele multimi frecvente.

Abordarea simpla

- De notat ca pragul *s* trebuie scalat prin micsorare; e.g. daca esantionul este 1% din date, se utilizeaza s/100 ca prag de support (in valoare absoluta).
- Se poate face o trecere completa prin date pentru a verifica daca multimile frecvente de articole ale esantionului sunt cu adevarat frecvente,

Abordarea simpla

- Vor fi pierdute insa multimile de articole care sunt frecvente in ansamblul datelor dar nu in esantion.
- Pentru a minimiza falsele negative se poate scadea putin pragul pentru esantion gasindu-se mai multe candidate pentru trecerea prin ansamblul datelor.
- Risc: prea multe candidate pentru a incapea in memoria centrala.

SON95

- *SON95* (Savasere, Omniecinski and Navathe in VLDB 1995; referit de Toivonen).
- Se citesc submultimi (transe) ale datelor in memoria centrala aplicandu-se abordarea simpla pentru descoperirea multimilor candidat.
- Fiecare cos este parte a uneia dintre aceste submultimi.

SON95

- In a doua trecere o multime este candidata daca a fost identificata ca si candidata in una sau mai multe submultimi ale datelor.
- Punct cheie: O multime nu poate fi frecventa in ansamblul datelor daca nu este frecventa in cel putin o submultime a acestora (oare de ce?).

Toivonen

- Se ia un esantion care incape in memoria centrala.
- Se ruleaza abordarea simpla pe aceste date dar cu un prag micsorat astfel incat sa fie improbabila pierderea vreunei adevarate multimi frecvente de articole (e.g. daca esantionul este de 1% din date se foloseste s/125 ca prag de suport).

Toivonen

- Se adauga candidatelor din esantion *marginea negativa*: acele multimi de articole *S* astfel incat *S nu* este identificata ca frecventa in esantion dar *orice* submultime stricta maximala a lui *S* este identificata astfel.
- De exemplu, daca *ABCD* nu este frecventa in esantion dar *ABC, ABD, ACD* si *BCD* sunt frecvente in esantion, atunci *ABCD* este in marginea negativa.

Toivonen

- Se face o trecere prin date, numarand toate multimile frecvente de articole si marginea negativa.
- Daca nici o multime din marginea negative nu este frecventa in ansamblul datelor, atunci multimile frecvente de articole sunt exact acele candidate care sunt deasupra pragului.

Bibliografie

- J.D.Ullman - CS345 --- Lecture Notes, primele doua capitole (Overview of Data Mining, Association-Rules, A-Priori Algorithm)

  http://infolab.stanford.edu/~ullman/cs345-notes.html

Sfârşitul capitolului 7