Finding frequent itemsets

Motivacija
Kolica za kupovinu
Aplikacije
Distribuirani File sustavi
GFS
HDFS

Market-basket model

Many-to-many odnos između stvari i kolica
Problem: Kako odrediti stvari koje su često kupljene zajedno?
Rješenje: identifikacija zajedničkih pravila, npr. Otkrivanje čestih skupova stvari

Model

- Velik broj stvari
- Velik broj kolica
- Kolica sadrže mal podskup stvari

Problem: kako se mogu parovi stvari pronaći u najmanje k kolica?

Česti skupovi stvari – pronaći apsolutni broj sigurnih skupova stvari koji se pojavljuju.(često)
Pronalazak sličnih stvari je zapravo pronalazak vrlo sličnih, ali ne nužno istih skupova stvari(često nije važno)

Česti skupovi stvari vs slične stvari

Česti skupovi stvari – pravila trebaju biti česta da bi bila profitabilna – trgovina je fizički entited korišten od svih kupaca
Pronalazak sličnih stvari – web trgovine
Isplati se znati koje stvari se moguće mogu kupiti zajedno, iako je mal broj kupada
Pronalazak korelacije između rijetko kupljenih stvari

Aplikacije

Bilo koje generalno many-to-many mapiranje između 2 tipa entiteta
Market-basket model – pronalazak stvari koje su često kupljene zajedno

Otkrivanje efekta droga
Kolica: pacijenti, stvari: droge i posljedični efekti

Plagijati
Kolica: rečenice, stvari: dokumenti

Biomarkeri
Kolica: informacije o pacijentu, stvari: biomarkeri i bolesti

Pronalaženje zajedništva u grafovima

Kolica: čvorovi, stvari

Pravila asocijacija

Primjena čestih skupova stvari
If-then pravila koja opisuju što bi bilo dobro da kolica sadrže
{i1,i2,i3...in} →implicira da ako kolica sadrže stvari od i1 do in također je vjerojatno da će sadržavati i stvar j

U praksi, želimo pronaći pravila koja se odnose na : **interes i povjerenje**

povjerenje(confidence)
For I={i1,i2,i3,...in}
Conf(I→j)=support(I unija j)/support(I)

Interes
Fr[j] frakcija kolica koja sadrže stvar j
Interest(I→j)=|conf(I→j)-Fr[j]| --POGLEDATI SLAJD 17

„Naivni pristup“

Pročitati sve podatke jednom i izbrojati parove
Za svaka kolica koja sadrže n stvari, izračunati mogući n(n-1)/2 parova sa 2 nested loop-a(ugnježđene petlje)

Pitanja: iako n može biti mal, krajnji broj stvari može biti velik
253 milijuna stvari prodano na amazonu
Ako parovi broje 4 bitni integer, približno 128 gigabajta trebaju kako bi ih se pohranilo

Troškovi traženja čestih skupova stvari

Brojanje podskupova stvari u svakim kolicima
K ugnježđene petlje se koriste za generiranje podskupa veličine k
Uobičajeno, k je mal i kolica su mala(sadrže n stvari)
Primjer : za kolica veličine n=30 i k=2 postoji (30 2) = 435 parova

Usko grlo glavne memorije

Memorija mora biti dovoljno velika za pohranu čestih skupova stvari
Algoritmi su limitirani od strane veličine memorij

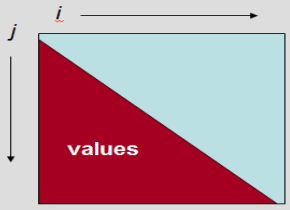
Pohranjivanje u memoriji

Imena stvari bi trebala biti indeksirana kao integeri(hash mape)
2-d matrica →triangularna matrica

Storing counts in memory

– Item names should be indexed as integers (hash map)

– Triangular matrix
- 2-dimensional array *a*[*i*, *j*], *i* < *j*



Storing counts in memory

– Triangular matrix
- More generally one-dimensional array is used
- Pair { *i*, *j* } is stored at *a*[*k*] for 1 ≤ *i* < *j* ≤ *n* where:

$$k = (i-1)\left(n - \frac{i}{2}\right) + j - i$$

- If counts are stored as 4 byte integers, approximately $2n^2$ bytes are needed for *n* items

Storing counts in memory

– Storing triplets
- Triplet { *i*, *j*, *c* } is stored so that *i* < *j* are item indices and c is the itemset count
- Hash table { *i*, *j* } → *c*
- If counts are stored as 4 byte integers, 12 bytes are needed to store a singe itemset count
- However, itemset counts are stored **only** when ***c* > 0**
- More efficient if significantly fewer then **1/3** of the possible itemsets occur in the baskets (realistic scenario!)

A-priori algoritam – dvosmjerni algoritam koji limitira potrebu za glavnom memorijom
Glavna ideja: monotonost skupa stvari
Ako je skup stvari I čest, onda svaki od njegovih podskupova stvari je isto čest
Ako se I pojavljuje s puta, tada se njegov podskup J pojavljuje barem s puta
Ako nijedan nadskup stvari I nije čest, I se zove maksimalni skup stvari

Prvi korak
Napraviti tablicu koja prevodi imena stvari u integere – hash tablica
Pobrojati
Pobrojati pojavu svake stvari kroz sve košare – broj redova je proporcionalan broju stvari n..bla bla

Međukorak
Generirati tablicu čestih stvari

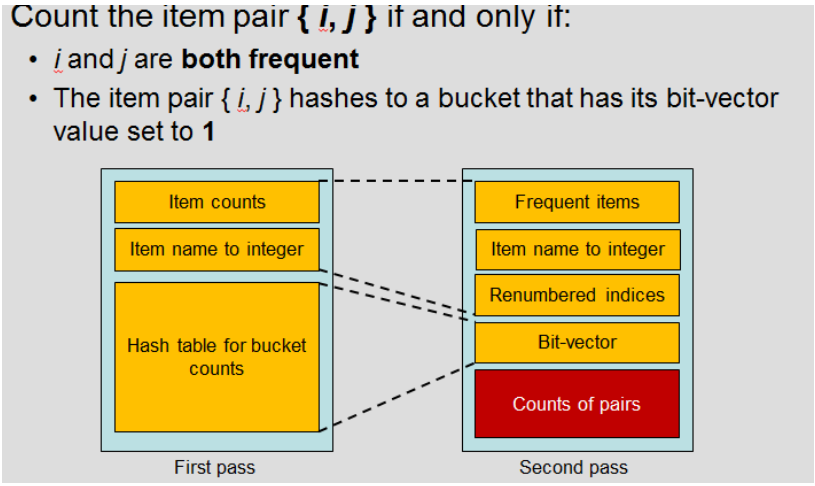
Drugi korak
za svaku košaru pogledati tablicu čestih stvari i vidjeti koje su česte
generirati sve parove stvari koristeći **samo česte stvari** u košari
dodati
pogledati 32-35

PCY algoritam
Ideja: većina glavne memorije u prvom koraku apriori algoritma je slobodna. Taj prazan prostor može biti iskorišten za reducirati memorijske zahtjeve iz drugog koraka(spremmanje skupa stvari može biti resursno zahtjevno – ovisno o resursu).

Prvi korak
Pobrojati stvari, te generirati imen u index-translacijsku tablicu
Za svaki skup kreirati parove stvari u double loop
Hash svaki par u skup i povećati brojač

Međukorak
Prevesti brojače skupova u bit-vektore
Vrijednost 1 je dodijeljena čestim skupovima, npr. Onima čiji je brojač veći ili jednak da podrži s?
Vrijednost 0 dodijeljena obrnuto
4. bajtni integer brojači su zamijenjeni sa 1 bitnim, 1/32 memorije je potrebna za spremiti bit-vektor

Drugi korak
Pobrojati parove stvari {i,j} ako i samo ako:
I i j su oba česti
Par stvari {i,j} hashes u skup koji ima bit-vektor vrijednost postavljenu na 1.



PCY algoritam je često efikasniji, ali ne uvijek

Trijangularna matrica nije prikladna za pohraniti brojače, zato jer parovi koje se može preskočiti su se slučajno raspršili, nema poznatog načina kako izbjeći neiskorišten prostor u matrici
Brojači trebaju biti pohranjeni u hash mape.
PCY je koristani samo ako dopušta izbjegavanje brojanja minimalno za 2/3 parova stvari, inače je apriori bolji.

RANDOM SAMPLING I SON ALGORITAM IZ PREZENTACIJA