0.1 cardinalità

I possibili frequent itemsets (FI) con N item di dimensione 1 sono, ovviamente, 20. I FI di dimensione 2 sono $\binom{N^2}{2!}$, di dimensione 3 $\binom{N^3}{3!}$ e così via. Il totale dei possibili itemset sono $\binom{N^2}{2!} * \binom{N^3}{3!} * ... * \binom{N^N}{N!} = \prod_{k=1}^{N} \binom{N^k}{k!}$
non è banale conservare in memoria $\binom{N}{2}$ contatori per una coppia i, j. Possiamo per esempio tradurre gli items in interi tramite una funzione di hash.
Si possono rappresentare i contatori come una matrice triangolare vettorializzata. Nella posizione $a[k]$ si mette il contatore per la coppia i, j dove $k = (i-1)(n - \frac{i}{2}) + j - i$

0.2 Monotonicità

se un itemset I è frequente, lo è anche ogni sottoinsieme di I.
Con questo possiamo definire un set massimale come un itemset che, data una soglia s di supporto, non ha set più grandi supportati.
Difficilmente ci sono più tripli set candidati per essere supportati di quante siano le coppie.

0.3 apriori

Facendo due passaggi sui dati invece che uno possiamo ridurre il numero di coppie da contare.

Nel primo passaggio si creano due tabelle: una per la traduzione del nome degli item in numeri, da 1 a N; l'altra per i conteggi.
l'elemento iesimo dell'array conta le occorrenze dell'elemento i.

Fatto il primo passaggio controlliamo quali item sono frequenti come singoletti usando il threshold s .

Per il secondo passaggio creiamo le $\binom{m^2}{2!}$ coppie di item dove m sono gli item frequenti (singoletti). Usiamo solo questi perchè una coppia non può essere frequente se non lo sono entrambi i suoi elementi.

Si alternano continuamente passaggi di generazione e filtraggio dei set di item:

1. Generazione C_k, set di itemset candidati di dimensione k
2. Filtraggio e produzione di L_k, il set di itemset frequenti di dimensione k

termina quando L_k è vuoto.

0.4 limited-pass

Algoritmi che approssimano i frequent itemsets usando solo 2 passate. SON usa 2 passate in media, ottiene un risultato esatto e raramente rischia di non terminare.

0.4.1 SON

Si estrae un sottoinsieme di transazioni (basket) e si riduce S per la frazione in cui viene partizionato il dataset. Dato S la soglia del supporto, se prendiamo un partizionamento che corrisponde all'1% del dataset, moltiplichiamo anche S per 1%: $s * \frac{1}{100} = \frac{s}{100}$.
Se le transazioni si presentano nel dataset senza correlazione o ordine possiamo tranquillamente. prendere le prime pm transazioni, dove p è la frazione di file su cui si lavora per volta.
SON non ha problemi con falsi positivi e falsi negativi perchè usa due passate invece che una sola.
Una volta che i pezzi sono stati processati e prodotti i frequent itemsets all'interno di ogni pezzo, vengono uniti e vengono trattati come candidati. Gli itemset che non compaiono mai (frequente in nessun pezzo) avranno una media di supporto inferiore a ps e quindi non frequente in totale.
Ogni itemset frequente in totale deve essere frequente in almeno un pezzo di dataset.
Dopo il primo passaggio si contano gli itemset candidati e si selezionano quelli con supporto almeno s.

0.4.2 mapreduce

Dato N la dimensione del dataset e S il supporto richiesto per definire un itemset "frequente":

1. first cycle
   (a) map: Si partiziona il dataset in p parti (ciascuna lunga $\frac{N}{p}$ elementi). Considerando il supporto $s = \frac{S}{p}$. Viene emesso ogni itemset e il valore 1.
   (b) reduce: restituisce tutti gli itemset che compaiono almeno una volta. Questi sono itemset candidati
2. second cycle
   (a) map Prende l'intero output del passaggio precedente (gli itemset candidati) e una porzione di dataset. Conta le occorrenze e le restituisce.
   (b) reduce Per ogni itemset somma tutte le occorrenze e calcola il supporto totale. Infine restituisce l'itemset e il relativo supporto. Se il supporto non è $\geq S$ viene scartato.