APRIORI

Algoritmo utile a individuare ricorsivamente gli itemset frequenti.

Apriori Property : DOWNWARD CLOSURE

Dato un itemset frequente I, qualsiasi sottoinsieme non vuoto di I è frequente anch'esso.
In questo senso se un itemset non passa il test, non lo passeranno anche i suoi super-set.

Apriori Pruning Principle

Se un set è NON-frequente è inutile generare e testare i suoi superset.

Algoritmo:

Abbiamo due set C_k e L_k, il primo indica il set di itemset candidati a essere frequenti, il secondo (L_k) sono gli itemset effettivamente frequenti fin lì.

1) Primo Passo

Scannerizzare il dataset per trovare tutti gli 1-itemset frequenti:

C_1 = tutti gli itemset con supp
L_1 = item frequenti.

2) Secondo passo

Partendo da L_1, costruiamo C_2 da cui avremo L_2 e così via.

Generalizzando, per trovare C_i basta fare join di L_{i-1} con se stesso:

$$C_i = L_{i-1} \bowtie L_{i-1}$$

Non si contano i duplicati.

Per convenzione gli oggetti in una transazione vengono ordinati. Questo viene fatto anche per motivi di efficienza.

Prune Step: bisogna usare la proprietà di Apriori per potare C_i.

Prendiamo ogni itemset in C_i e deriviamo da esso tutti i $(i-1)$-itemset. Se tutti gli $(i-1)$-itemset trovati NON sono in L_{i-1}, allora l'itemset in questione non è frequente e può essere tolto da C_i.

3) Stop Rule

Quando arriviamo a $L_i = \emptyset$.

L'OPERAZIONE DI JOIN

Per fare join fra $L_k \bowtie L_k$ Apriori assume che gli item in un itemset siano ordinati in ordine lessicografico, ovvero un ordine che è fissato da chi usa l'algoritmo. Il che vuol dire che per un $(k-1)$-itemset e_i avremo:

$$e_i[1] < e_i[2] < \ldots < e_i[k-1]$$

Esempio

Prendiamo il 5-itemset $= \{a, b, c, d, e\}$ abbiamo fissato l'ordine alfabetico per cui:

$$a < b < c < d < e$$

join

Due $(k-1)$-itemset e_i e e_j possono fare join se condividono i primi $(k-2)$ items, per cui:

$$(e_i[1] == e_j[1]) \wedge (e_i[2] == e_j[2]) \wedge \ldots \wedge (e_i[k-2] == e_j[k-2])$$

$$\wedge (e_i[k-1] < e_j[k-1])$$

il risultato dell'operazione di join sarà:

$$\{e_i[1], e_i[2], \ldots, e_i[k-1], e_j[k-1]\}$$

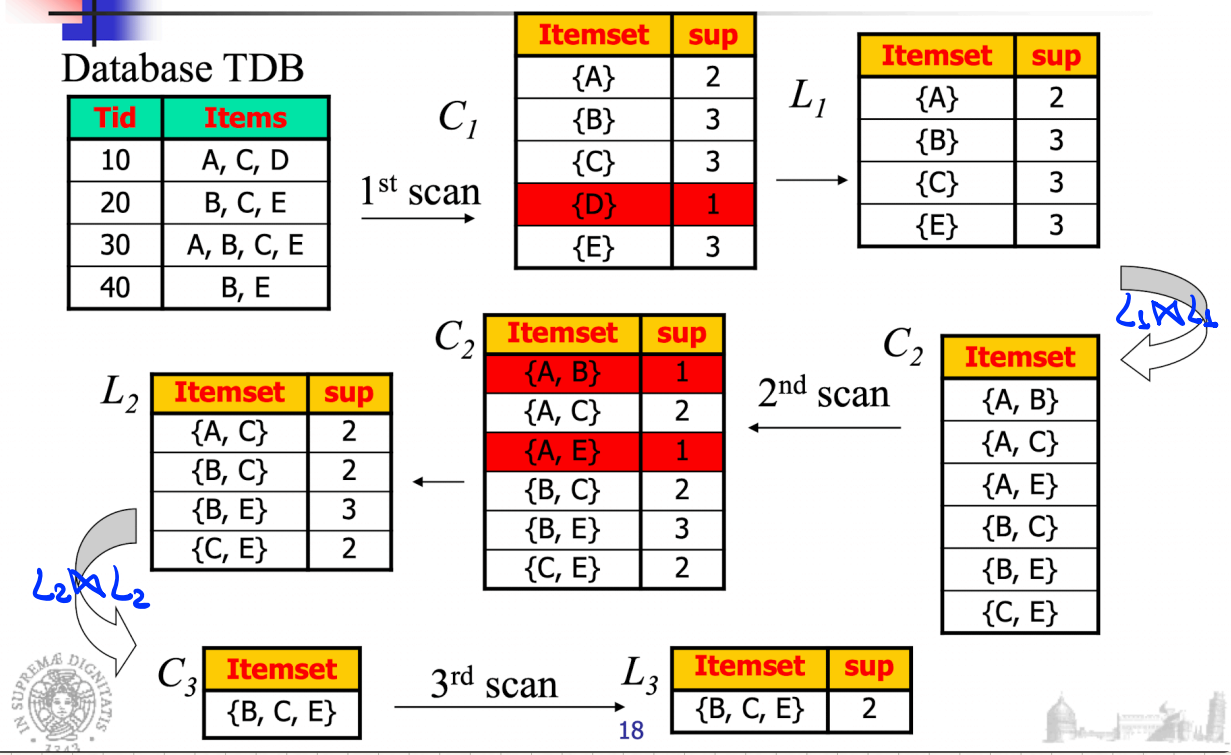
Esempio

$e_i = \{a, b, c, d\}$

$e_j = \{a, b, c, e\}$

$e_i \bowtie e_j = \{a, b, c, d, e\}$

The Apriori Algorithm—An Example



GENERAZIONE DELLE ASSOCIATION RULE

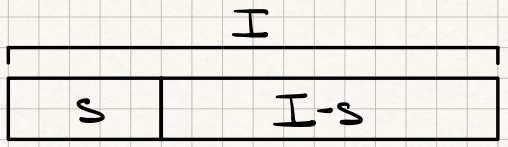
Prendiamo gli Itemset frequenti I e creiamo ogni sottoinsieme non vuoto di I.

$\forall s \in I$ cerchiamo $s \Rightarrow (I-s)$ tale per cui:

$$\frac{supp(I)}{supp(s)} \geq min\_conf$$

min_conf → threshold

Dove s è il sottoinsieme di I, mentre $(I-s)$ è l'altro sottoinsieme che non contiene s.



La formula è praticamente quella della confidence di una regola $A \Rightarrow B$:

$$Conf(A \Rightarrow B) = P(A|B) = \frac{supp(A \cup B)}{supp(A)}$$

perché $(s) \cup (I-s) = I$

OTTIMIZZAZIONI

1) Ogni itemset che è potenzialmente frequente per il dataset D, lo è in almeno una delle partizioni di D.
L'idea è quella di dividere D in partizioni, trovare gli itemset frequenti per ogni partizione. Poi combinare gli itemset trovati per generare il set di itemset candidati a essere frequenti.
Basta poi fare un ultimo scan di D per verificare e trovare gli itemset frequenti in D fra i candidati.

2) Hash-based technique
Questo metodo viene utilizzato per diminuire la dimensione del set di k-itemset candidati: C_k.
Iniziamo costruendo L_1. (minsupp = 2)

TID	List of item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

I_1 : 6
I_2 : 7
I_3 : 6
I_4 : 2
I_5 : 2

Nel mentre che scorriziamo D per creare L_1, possiamo costruire anche C_2, costruendo i vari 2-itemset per transazione:

T100	$\{I_1, I_2\}, \{I_1, I_5\}, \{I_2, I_5\}$
T200	$\{I_2, I_4\}$
T300	$\{I_2, I_3\}$
T400	$\{I_1, I_2\}, \{I_1, I_4\}, \{I_2, I_4\}$
T500	$\{I_1, I_3\}$
T600	$\{I_2, I_3\}$
T700	$\{I_1, I_3\}$
T800	$\{I_1, I_2\}, \{I_1, I_3\}, \{I_1, I_5\}, \{I_2, I_3\}$ $\{I_2, I_5\}, \{I_3, I_5\}$
T900	$\{I_1, I_2\}, \{I_1, I_3\}, \{I_2, I_3\}$

Trovati questi candidati, dobbiamo hasharli nella tabella hash nei rispettivi bucket e incrementare il bucket count. Per fare l'hashing seguiamo la formula

$$h(I_i, I_j) = \left| (i \cdot 10) + j \right|_{\#\text{BUCKET}} \rightarrow 7$$

Create hash table H_2 using hash function $h(x, y) = ((order\ of\ x) \times 10 + (order\ of\ y))\ mod\ 7$

H_2

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{I1, I4}	{I1, I5}	{I2, I3}	{I2, I4}	{I2, I5}	{I1, I2}	{I1, I3}
	{I3, I5}	{I1, I5}	{I2, I3}	{I2, I4}	{I2, I5}	{I1, I2}	{I1, I3}
			{I2, I3}			{I1, I2}	{I1, I3}
			{I2, I3}			{I1, I2}	{I1, I3}

Non penso sia importante come implementare $h(\cdot)$, a sentimento va bene!

La cosa importante è che così possiamo facilmente eliminare i bucket con bucket count < supporto, riducendo così il numero di candidati

3) Sampling

Cercare gli itemset frequenti in $S \supset D$. Così da avere meno dati da lavorare, a discapito dell'accuratezza.