



UNIVERSITÀ  
DEGLI STUDI  
DI PALERMO



# Mining di Pattern Frequenti

CORSO DI BIG DATA  
a.a. 2020/2021

Prof. Roberto Pirrone

# Sommario

- Generalità e definizioni
- Proprietà di Itemset e Regole
- Algoritmo Apriori e sue varianti
- Mining di pattern interessanti
- Gestione di grandi database

# Generalità

- Il problema del mining di pattern o itemset frequenti nasce nell'ambito dei dati transazionali
- Dato un database di transazioni  $\mathcal{T}=\{T_1, \dots, T_n\}$  relative a un universo  $U$  di item, ogni transazione sarà una lista di item e può essere rappresentata come una stringa di bit di lunghezza pari a  $|U|$  in cui un bit 1 rappresenta un item presente nella transazione
- Un *itemset* di cardinalità  $k$  è un insieme di  $k$  item presenti all'interno della transazione

# Generalità

- Data la natura del dato cui si applica, il mining di pattern frequenti può essere utilizzato in tutti i casi in cui ci sono dati riconducibili alla forma categorica
  - Text mining, in caso di rappresentazione *Bag Of Words (BOW)*
  - Serie temporali
  - Sequenze discrete
  - Dati geo-spatiali

# Generalità

- Il *supporto* di un itemset  $I$  è la frazione di transazioni in  $\mathcal{T}$  che lo contiene

$$sup(I) = \frac{|\{T_i : T_i \supseteq I\}|}{|\mathcal{T}|}$$

- *Frequent Pattern Mining*: dato un set di transazioni  $\mathcal{T} = \{T_i\}$ , tratte da un universo di item  $U$ , determinare gli itemset  $I$  che occorrono come sottoinsiemi di almeno una data frazione  $minsup$  di transazioni in  $\mathcal{T}$

# Generalità

- $sup(\{Eggs, Milk\}) = 3/5 = 0.6$

tid	Set of items	Binary representation
1	{Bread, Butter, Milk}	110010
2	{Eggs, Milk, Yogurt}	000111
3	{Bread, Cheese, Eggs, Milk}	101110
4	{Eggs, Milk, Yogurt}	000111
5	{Cheese, Milk, Yogurt}	001011

# Generalità

- $sup(\{Cheese, Yogurt\}) = 1/5 = 0.2$

tid	Set of items	Binary representation
1	{Bread, Butter, Milk}	110010
2	{Eggs, Milk, Yogurt}	000111
3	{Bread, Cheese, Eggs, Milk}	101110
4	{Eggs, Milk, Yogurt}	000111
5	{Cheese, Milk, Yogurt}	001011

# Generalità

- $\text{minsup} = 0.3$

tid	Set of items	Binary representation
1	{Bread, Butter, Milk}	110010
2	{Eggs, Milk, Yogurt}	000111
3	{Bread, Cheese, Eggs, Milk}	101110
4	{Eggs, Milk, Yogurt}	000111
5	{Cheese, Milk, Yogurt}	001011

# Generalità

- $\text{minsup} = 0.3$

tid	Set of items	Binary representation
1	{Bread, Butter, Milk}	110010
2	{Eggs, Milk, Yogurt}	000111
3	{Bread, Cheese, Eggs, Milk}	101110
4	{Eggs, Milk, Yogurt}	000111
5	{Cheese, Milk, Yogurt}	001011

# Generalità

- $\text{minsup} = 0.3$

tid	Set of items	Binary representation
1	{Bread, Butter, Milk}	110010
2	{Eggs, Milk, Yogurt}	000111
3	{Bread, Cheese, Eggs, Milk}	101110
4	{Eggs, Milk, Yogurt}	000111
5	{Cheese, Milk, Yogurt}	001011

# Generalità

- $\text{minsup} = 0.3$

tid	Set of items	Binary representation
1	{Bread, Butter, Milk}	110010
2	{Eggs, Milk, Yogurt}	000111
3	{Bread, Cheese, Eggs, Milk}	101110
4	{Eggs, Milk, Yogurt}	000111
5	{Cheese, Milk, Yogurt}	001011

# Generalità

- $\text{minsup} = 0.3$

tid	Set of items	Binary representation
1	{Bread, Butter, Milk}	110010
2	{Eggs, Milk, Yogurt}	000111
3	{Bread, Cheese, Eggs, Milk}	101110
4	{Eggs, Milk, Yogurt}	000111
5	{Cheese, Milk, Yogurt}	001011

# Generalità

- $\text{minsup} = 0.3$

tid	Set of items	Binary representation
1	{Bread, Butter, Milk}	110010
2	{Eggs, Milk, Yogurt}	000111
3	{Bread, Cheese, Eggs, Milk}	101110
4	{Eggs, Milk, Yogurt}	000111
5	{Cheese, Milk, Yogurt}	001011

# Generalità

- $\text{minsup} = 0.3$

tid	Set of items	Binary representation
1	{Bread, Butter, Milk}	110010
2	{Eggs, Milk, Yogurt}	000111
3	{Bread, Cheese, Eggs, Milk}	101110
4	{Eggs, Milk, Yogurt}	000111
5	{Cheese, Milk, Yogurt}	001011

# Generalità

- $\text{minsup} = 0.3$

tid	Set of items	Binary representation
1	{Bread, Butter, Milk}	110010
2	{Eggs, Milk, Yogurt}	000111
3	{Bread, Cheese, Eggs, Milk}	101110
4	{Eggs, Milk, Yogurt}	000111
5	{Cheese, Milk, Yogurt}	001011

# Generalità

- $\text{minsup} = 0.3$

tid	Set of items	Binary representation
1	{Bread, Butter, Milk}	110010
2	{Eggs, Milk, Yogurt}	000111
3	{Bread, Cheese, Eggs, Milk}	101110
4	{Eggs, Milk, Yogurt}	000111
5	{Cheese, Milk, Yogurt}	001011

# Generalità

- $\text{minsup} = 0.3$

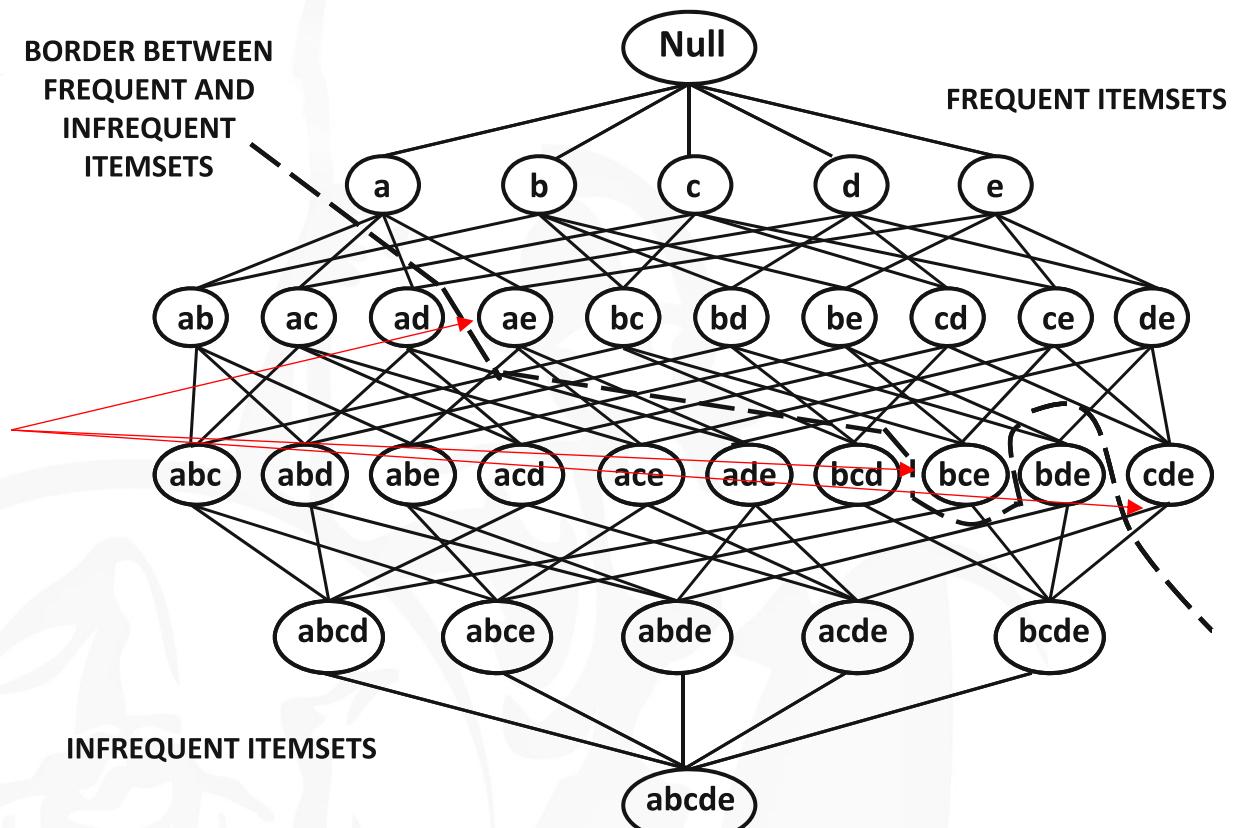
tid	Set of items	Binary representation
1	{Bread, Butter, Milk}	110010
2	{Eggs, Milk, Yogurt}	000111
3	{Bread, Cheese, Eggs, Milk}	101110
4	{Eggs, Milk, Yogurt}	000111
5	{Cheese, Milk, Yogurt}	001011

# Proprietà degli Itemset

- Monotonicità
  - Il supporto di un qualunque sottoinsieme  $J$  di un itemset  $I$  è non minore del supporto di  $I$
- Chiusura verso il basso
  - Ogni sottoinsieme di un Itemset frequente è anch'esso frequente

# Proprietà degli Itemset

- Massimo itemset frequente
  - Un itemset frequente  $I$  ad un dato supporto minimo  $minsup$  si dice massimo se è frequente e non esistono itemset frequenti che lo contengono (super-insiemi di  $I$ )



# Regole di associazione e loro proprietà

- Una regola di associazione  $X \Rightarrow Y$  è una co-occorrenza degli itemset  $X$  e  $Y$  nella stessa transazione
  - Possiamo vedere la regola in termini insiemistici:  $X \cup Y$
- La confidenza di una regola è la probabilità condizionale della sua occorrenza soggetta all'occorrenza di  $X$ 
  - La probabilità dell'occorrenza di un itemset è banalmente il suo supporto

$$conf(X \Rightarrow Y) = \frac{sup(X \cup Y)}{sup(X)}$$

# Regole di associazione e loro proprietà

- La regola  $X \Rightarrow Y$  si definisce regola di associazione al supporto minimo  $minsup$  e confidenza minima  $minconf$  se:
  - $sup(X \cup Y) \geq minsup$
  - $conf(X \Rightarrow Y) \geq minconf$
- Monotonicità della confidenza
  - Siano dati gli itemset  $X_1 \subset X_2 \subset I$ :

$$conf(X_2 \Rightarrow I - X_2) \geq conf(X_1 \Rightarrow I - X_1)$$

# Algoritmo Apriori

- Un approccio forza bruta per il mining di pattern frequenti prevede l'esplorazione di tutti i  $2^{|U|} - 1$  nodi del grafo degli itemset
- Per essi deve effettuare il calcolo del supporto e considerarli come candidati itemset frequenti
- La proprietà di chiusura verso il basso ci dice che *non ci sono item frequenti di cardinalità k+1 se non ce ne sono di cardinalità k*, ma questo mantiene lo spazio di ricerca ancora elevato
- Se  $l$  è la lunghezza massima degli itemset frequenti allora il numero di nodi da cercare diviene 
$$\sum_{i=1}^l \binom{|U|}{i}$$

# Algoritmo Apriori

- Un algoritmo efficiente di mining deve:
  - Ridurre lo spazio di ricerca
  - Effettuare un calcolo efficiente del supporto
  - Usare una struttura dati compatta per gestire i candidati itemset frequenti

# Algoritmo Apriori

**Algorithm** *Apriori*(Transactions:  $\mathcal{T}$ , Minimum Support:  $minsup$ )

**begin**

$k = 1;$

$\mathcal{F}_1 = \{ \text{ All Frequent 1-itemsets } \};$

**while**  $\mathcal{F}_k$  is not empty **do begin**

        Generate  $\mathcal{C}_{k+1}$  by joining itemset-pairs in  $\mathcal{F}_k$ ;

        Prune itemsets from  $\mathcal{C}_{k+1}$  that violate downward closure;

        Determine  $\mathcal{F}_{k+1}$  by support counting on  $(\mathcal{C}_{k+1}, \mathcal{T})$  and retaining  
            itemsets from  $\mathcal{C}_{k+1}$  with support at least  $minsup$ ;

$k = k + 1;$

**end;**

**return**( $\cup_{i=1}^k \mathcal{F}_i$ );

**end**

*Usa la proprietà di chiusura verso il basso*

# Algoritmo Apriori

- I candidati in  $\mathcal{C}_{k+1}$  sono generati da coppie di item frequenti in  $\mathcal{F}_k$  che *differiscono per un solo item*
  - $abc$  e  $abd \rightarrow abcd$
  - Si utilizza l'ordinamento lessicografico degli item negli itemset per evitare permutazioni di item che corrispondono sempre allo stesso itemset
    - $abcd$  viene generato se e solo se  $a$  e  $b$  sono stati concatenati per generare  $ab$  e via di seguito fino ad  $abc$  e  $abd$
  - L'ordinamento lessicografico garantisce che gli itemset in  $\mathcal{F}_k$  che contengono lo stesso sottoinsieme di item siano contigui e sono quindi semplici da cercare all'interno di  $\mathcal{F}_k$

# Algoritmo Apriori

- La proprietà di chiusura verso il basso garantisce che un itemset candidato  $I \in \mathcal{C}_{k+1}$  è frequente se e solo se *tutti i suoi k-sottoinsiemi* sono presenti in  $\mathcal{F}_k$ 
  - Se questa ricerca in  $\mathcal{F}_k$  da esito negativo,  $I$  viene rimosso da  $\mathcal{C}_{k+1}$
- Il calcolo del supporto si effettua ricercando l'occorrenza del candidato  $I$  in tutte le transazioni

# Algoritmo Apriori

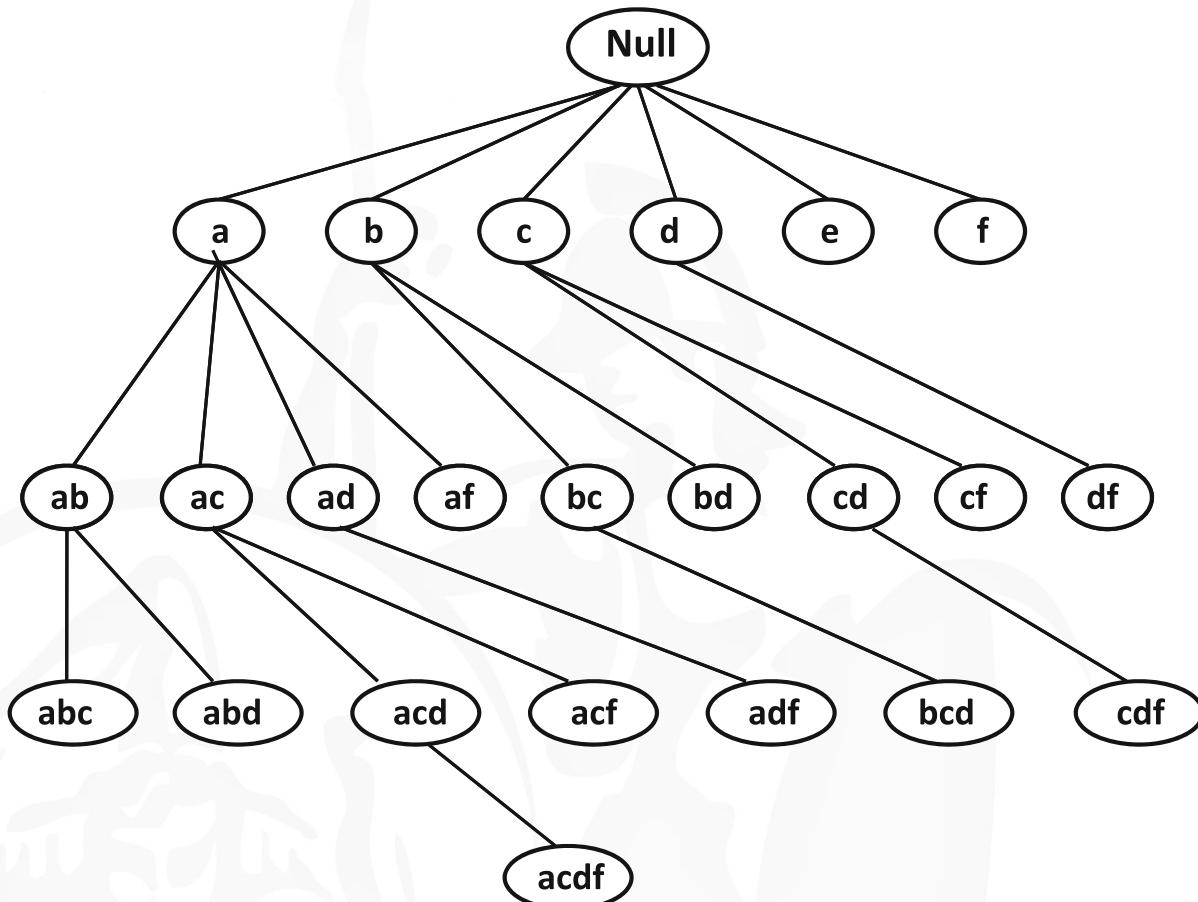
- Calcolo efficiente del supporto tramite un *hash tree* cioè un albero in cui ogni nodo ha un numero fissato  $h$  di figli e ogni nodo interno è una tabella hash, mentre i nodi foglia contengono tutti i candidati
- Ogni itemset  $I \in \mathcal{C}_{k+1}$ , ordinato lessicograficamente, è così associato ad un percorso nell'albero
- Una data funzione casuale di hashing  $f(.) \rightarrow [0, \dots, h-1]$  è applicata per un dato nodo a profondità  $i$ , all' $i$ -esimo item di  $I$  per decidere quale sarà il discendente successivo per continuare l'indicizzazione di  $I$ .

# Algoritmo Apriori

- Per ogni transazione  $T_i$  in  $\mathcal{T}$  si analizzano ricorsivamente i suoi item come segue:
  - Al livello 1 (nodo radice) si seguono tutti i rami in modo che qualunque item in  $T_i$  possa essere successivamente preso in considerazione
  - Ad ogni livello di profondità k, che assumiamo raggiunto attraverso l'hashing dell'item  $j$  in  $T_i$ :
    - Si esegue l'hashing di tutti i successivi di  $T_i^{(j)}$  per seguire tutti i possibili percorsi che contengano gli itemset di  $T_i$
    - Tutti i nodi foglia rilevanti (che *possono* contenere un itemset di  $T_i$ ) vengono raccolti, ordinati e confrontati con  $T_i$  per vedere se sono in essa contenuti e, in questo caso, se ne incrementa il relativo conteggio del supporto

# Algoritmo Apriori

- L'enumeration tree consente una ricerca efficiente tra gli itemset frequenti sotto la condizione del loro *ordinamento lessicografico*
- La generazione dei candidati si effettua facendo accrescere l'enumeration tree



# Algoritmo Apriori

- Procedura di accrescimento dell'enumeration tree:
  - Sia  $Q$  il genitore di  $P$ , itemset frequente, e sia  $F(Q)$  il suo set di estensioni frequenti cioè singoli item, ordinati lessicograficamente che estendono  $Q$  verso altri itemset frequenti
  - Sia  $i \in F(Q)$  l'item che estende  $Q$  in  $P$ , cioè  $P = Q \cup \{i\}$
  - Allora, per l'ordinamento lessicografico, l'insieme  $C(P)$  degli item candidati ad estendere  $P$  per creare  $F(P)$  sono **solo e soltanto i successori di  $i$  in  $F(Q)$**
  - Vale sempre  $F(P) \subseteq C(P) \subset F(Q)$
  - $F(P)$  si genera da  $C(P)$  facendo il conteggio esplicito del supporto dei candidati

# Algoritmo Apriori

- Procedura di accrescimento dell'enumeration tree:

```
Algorithm GenericEnumerationTree(Transactions:  $\mathcal{T}$ ,  
                                Minimum Support:  $minsup$ )  
begin  
    Initialize enumeration tree  $\mathcal{ET}$  to single Null node;  
    while any node in  $\mathcal{ET}$  has not been examined do begin  
        Select one or more unexamined nodes  $\mathcal{P}$  from  $\mathcal{ET}$  for examination;  
        Generate candidates extensions  $C(P)$  of each node  $P \in \mathcal{P}$ ;  
        Determine frequent extensions  $F(P) \subseteq C(P)$  for each  $P \in \mathcal{P}$  with support counting;  
        Extend each node  $P \in \mathcal{P}$  in  $\mathcal{ET}$  with its frequent extensions in  $F(P)$ ;  
    end  
    return enumeration tree  $\mathcal{ET}$ ;  
end
```

- Fare il join di due  $k$ -itemset per generare un  $(k+1)$ -itemset corrisponde qui a fare il join di due fratelli immediati a profondità  $k$  dell'enumeration tree

# Algoritmo Apriori

- Tree projection:
  - Quando un itemset frequente, nodo dell'enumeration tree, non è frequente in una certa transazione  $T_i$ , allora  $T_i$  non sarà rilevante per il conteggio del supporto anche dei suoi discendenti
  - Un projected database  $\mathcal{T}(P)$  su un nodo  $P$  è una proiezione di  $\mathcal{T}$  in cui sono presenti solo le transazioni rilevanti per  $P$ 
    - Problemi di efficienza per l'occupazione di spazio su disco
  - $\mathcal{T}(P)$  può essere espresso solamente dai candidati di  $P$   $C(P)$  perché è fatto dalle transazioni che contengono l'itemset  $P$  e ogni sua possibile estensione verso itemset frequenti
    - Non è necessario mantenere l'intero universo  $U$  degli item

# Algoritmo Apriori

- Tree projection:

```
Algorithm ProjectedEnumerationTree(Transactions:  $\mathcal{T}$ ,  
Minimum Support:  $minsup$ )  
begin  
    Initialize enumeration tree  $\mathcal{ET}$  to a single (Null,  $\mathcal{T}$ ) root node;  
    while any node in  $\mathcal{ET}$  has not been examined do begin  
        Select an unexamined node  $(P, \mathcal{T}(P))$  from  $\mathcal{ET}$  for examination;  
        Generate candidates item extensions  $C(P)$  of node  $(P, \mathcal{T}(P))$ ;  
        Determine frequent item extensions  $F(P) \subseteq C(P)$  by support counting  
            of individual items in smaller projected database  $\mathcal{T}(P)$ ;  
        Remove infrequent items in  $\mathcal{T}(P)$ ;  
        for each frequent item extension  $i \in F(P)$  do begin  
            Generate  $\mathcal{T}(P \cup \{i\})$  from  $\mathcal{T}(P)$ ;  
            Add  $(P \cup \{i\}, \mathcal{T}(P \cup \{i\}))$  as child of  $P$  in  $\mathcal{ET}$ ;  
        end  
    end  
    return enumeration tree  $\mathcal{ET}$ ;  
end
```

# Algoritmo Apriori

- Tree projection:
  - Il conteggio del supporto degli antenati di  $P$  può essere propagato lungo l'albero e allora per i successori di  $P$  **basta contare il supporto di tutte le estensioni di un solo item di  $P$**
  - Si possono ordinare i nodi per supporto crescente così i rami più grandi dell'albero si portano appresso meno transazioni
  - Gli approcci all'accrescimento dell'enumeration tree possono essere sia breadth-first sia depth-first
    - Riduzione dei costi di accesso alla memoria di massa vs efficienza che però mantiene solo i database proiettati lungo il percorso esaminato in profondità

# Algoritmo Apriori

- Rappresentazioni *verticali* del database:

Item	Set of tids	Binary representation
Bread	{1, 3}	10100
Butter	{1}	10000
Cheese	{3, 5}	00101
Eggs	{2, 3, 4}	01110
Milk	{1, 2, 3, 4, 5}	11111
Yogurt	{2, 4, 5}	01011

- $sup(\{Eggs, Milk\}) = |\{2, 3, 4\} \cap \{1, 2, 3, 4, 5\}| = |\{2, 3, 4\}| = 3 \quad ( / 5 = 0.6)$

# Algoritmo Apriori

- Rappresentazioni *verticali* del database:

```
Algorithm VerticalApriori(Transactions:  $\mathcal{T}$ , Minimum Support:  $minsup$ )
begin
     $k = 1$ ;
     $\mathcal{F}_1 = \{ \text{All Frequent 1-itemsets} \}$ ;
    Construct vertical  $tid$  lists of each frequent item;
    while  $\mathcal{F}_k$  is not empty do begin
        Generate  $\mathcal{C}_{k+1}$  by joining itemset-pairs in  $\mathcal{F}_k$ ;
        Prune itemsets from  $\mathcal{C}_{k+1}$  that violate downward closure;
        Generate  $tid$  list of each candidate itemset in  $\mathcal{C}_{k+1}$  by intersecting
             $tid$  lists of the itemset-pair in  $\mathcal{F}_k$  that was used to create it;
        Determine supports of itemsets in  $\mathcal{C}_{k+1}$  using lengths of their  $tid$  lists;
         $\mathcal{F}_{k+1} = \text{Frequent itemsets of } \mathcal{C}_{k+1} \text{ together with their } tid \text{ lists}$ ;
         $k = k + 1$ ;
    end;
    return( $\bigcup_{i=1}^k \mathcal{F}_i$ );
end
```

# Algoritmo Apriori

- Crescita ricorsiva del *suffisso* dell'itemset
  - Il database  $\mathcal{T}$  viene scansionato ed espresso in termini di un insieme di item (ovvero 1-itemset) frequenti e ordinati lessicograficamente per supporto decrescente, già all'inizio dell'algoritmo
  - Si accrescono gli itemset ricorsivamente in ordine lessicografico inverso cioè dall'ultimo item verso il primo
  - La chiusura verso il basso ci garantisce che, via via che il suffisso dell'itemset si accresce, il supporto dell'itemset diminuirà, **ma certamente ogni singolo item di un itemset frequente avrà supporto maggiore o uguale a *minsup***
  - Non appena non si possono aggiungere più item frequenti l'itemset non deve più essere accresciuto perché non è frequente

# Algoritmo Apriori

- Crescita ricorsiva del *suffisso* dell'itemset
  - Ad ogni livello di ricorsione abbiamo in ingresso:
    - un suffisso  $P$  di item che è di per sé frequente
    - un database proiettato  $\mathcal{T}$  di item frequenti di lunghezza 1 relativi alle transazioni che contengono  $P$
  - Vogliamo ottenere i suffissi più lunghi di un item  $P_i = \{i\} \cup P$  anch'essi frequenti nonché la proiezione di  $\mathcal{T}$  nei relativi database  $\mathcal{T}_i$

# Algoritmo Apriori

- Crescita ricorsiva del *suffisso* dell'itemset
  - In virtù dell'ordinamento lessicografico decrescente per supporto, il database  $\mathcal{T}_i$ , relativo al suffisso  $P_i$ , è costituito come segue:
    - Si considerano le transazioni in  $\mathcal{T}_i$  (nella sua forma consueta) che contengono l'item  $i$
    - Si escludono da ogni transazione  $i$  ed i suoi successori (a supporto più elevato)
    - Si mantengono in  $\mathcal{T}_i$ , solo gli item frequenti (a supporto maggiore o uguale a  $minsup$ )
  - Siamo certi che solo le transazioni in  $\mathcal{T}_i$  sono quelle che contengono gli itemset che terminano con il suffisso  $P_i$

# Algoritmo Apriori

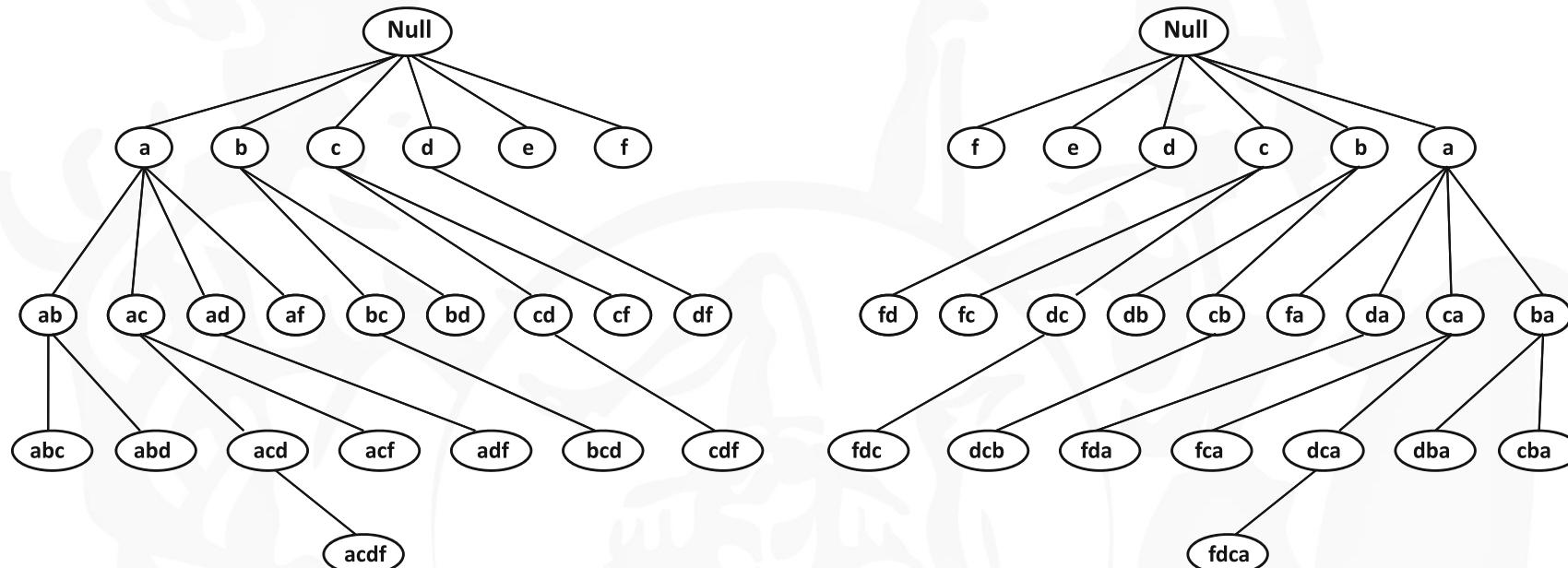
- Crescita ricorsiva del *suffisso* dell'itemset
  - Si accrescono gli itemset ricorsivamente in ordine lessicografico inverso cioè dall'ultimo item verso il primo

**Algorithm** *RecursiveSuffixGrowth*(Transactions in terms of frequent 1-items:  $\mathcal{T}$ ,  
Minimum Support:  $minsup$ , Current Suffix:  $P$ )

```
begin
    for each item  $i$  in  $\mathcal{T}$  do begin
        report itemset  $P_i = \{i\} \cup P$  as frequent;
        Extract all transactions  $\mathcal{T}_i$  from  $\mathcal{T}$  containing item  $i$ ;
        Remove all items from  $\mathcal{T}_i$  that are lexicographically  $\geq i$ ;
        Remove all infrequent items from  $\mathcal{T}_i$ ;
        if ( $\mathcal{T}_i \neq \emptyset$ ) then RecursiveSuffixGrowth( $\mathcal{T}_i, minsup, P_i$ );
    end
end
```

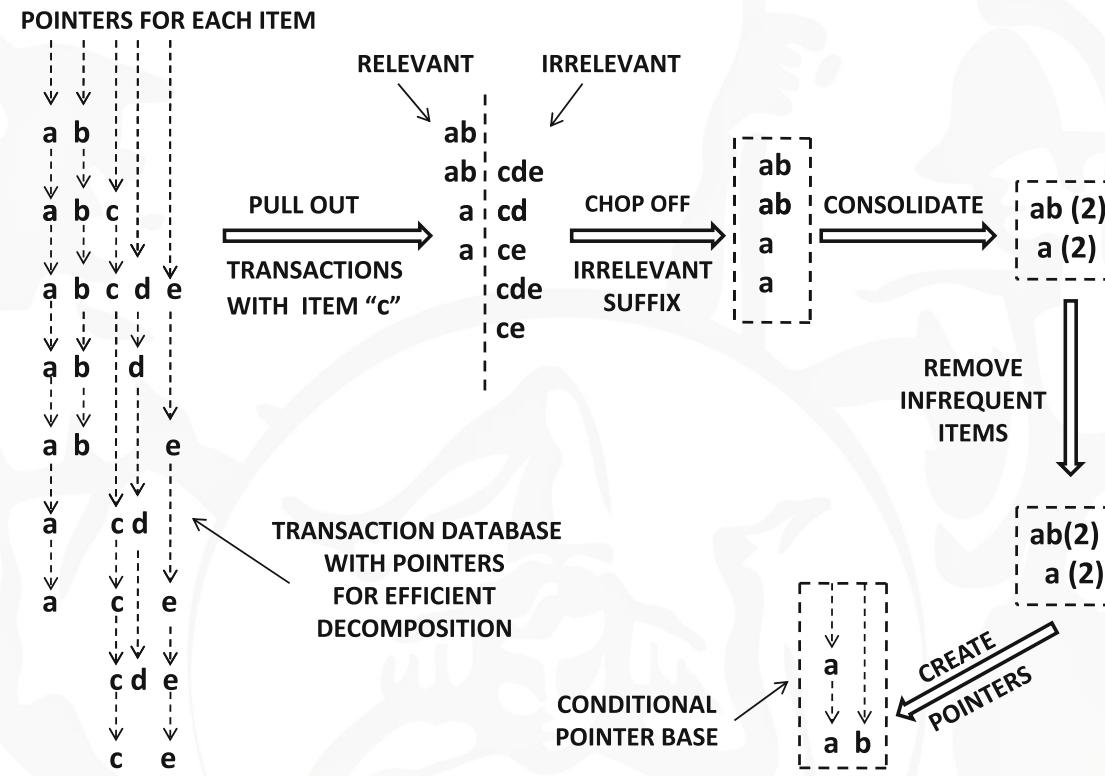
# Algoritmo Apriori

- Crescita ricorsiva del *suffisso* dell'itemset
  - Generano l'enumeration tree con ordine inverso



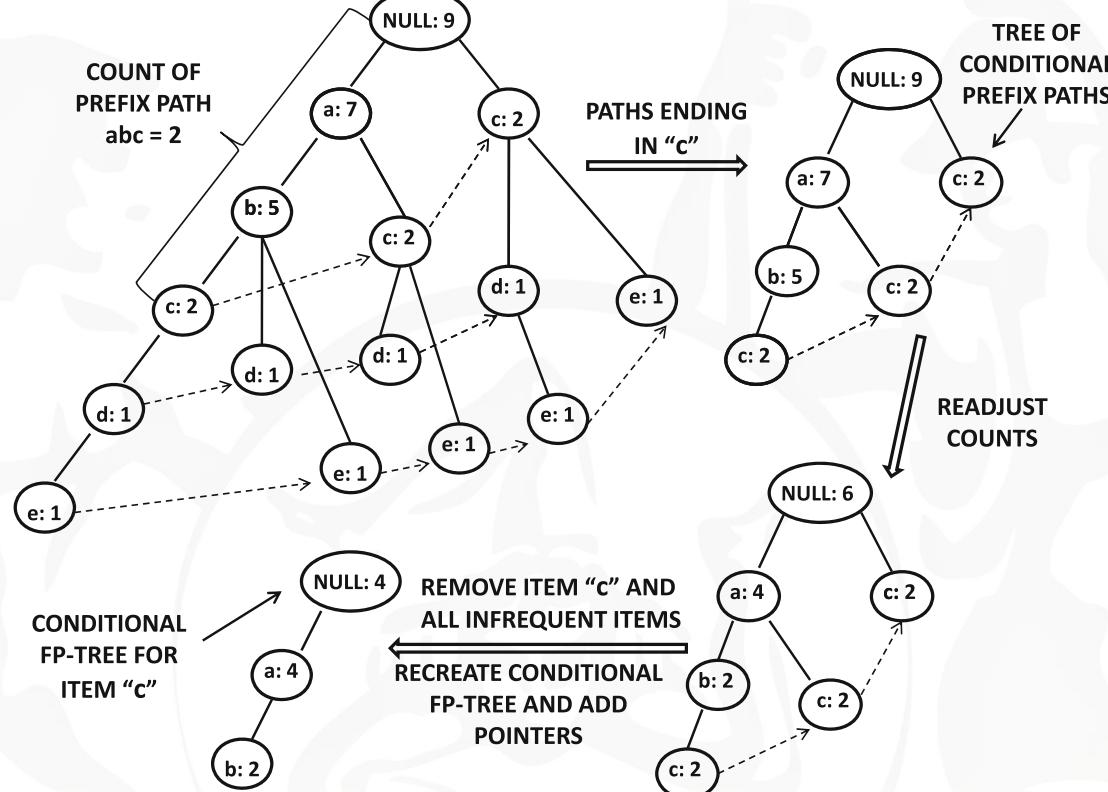
# Algoritmo Apriori

- Crescita ricorsiva del *suffisso* dell'itemset
  - Generazione efficiente dei  $\mathcal{T}_i$  – uso dei puntatori



# Algoritmo Apriori

- Crescita ricorsiva del *suffisso* dell'itemset
  - Generazione efficiente dei  $\mathcal{T}_i$  – uso del FP-Tree



# Algoritmo Apriori

- Crescita ricorsiva del *suffisso* dell'itemset
  - Generazione efficiente dei  $\mathcal{T}_i$  – uso del FP-Tree

**Algorithm**  $FP\text{-growth}$ (FP-Tree of frequent items:  $\mathcal{FPT}$ , Minimum Support:  $minsup$ ,  
Current Suffix:  $P$ )

```
begin
    if  $\mathcal{FPT}$  is a single path
        then determine all combinations  $C$  of nodes on the
            path, and report  $C \cup P$  as frequent;
    else (Case when  $\mathcal{FPT}$  is not a single path)
        for each item  $i$  in  $\mathcal{FPT}$  do begin
            report itemset  $P_i = \{i\} \cup P$  as frequent;
            Use pointers to extract conditional prefix paths
                from  $\mathcal{FPT}$  containing item  $i$ ;
            Readjust counts of prefix paths and remove  $i$ ;
            Remove infrequent items from prefix paths and reconstruct
                conditional FP-Tree  $\mathcal{FPT}_i$ ;
            if  $(\mathcal{FPT}_i \neq \emptyset)$  then  $FP\text{-growth}(\mathcal{FPT}_i, minsup, P_i)$ ;
        end
    end
```

# Mining di pattern interessanti

- Il semplice calcolo frequentista delle occorrenze di un pattern può non essere discriminativo dell'importanza di una regola per una certa applicazione
- La regola  $X \Rightarrow \{Milk\}$  nel nostro database ha confidenza 100%, qualunque sia il supporto di  $X$ , ma a che serve?
  - $X$  non è discriminativo di *Milk*, qualunque sia l'itemset, perché  $\text{sup}(\{Milk\})$  è di per sé il 100%
- Possono essere più rilevanti altri criteri di misura dell'interesse del pattern

# Mining di pattern interessanti

- Coefficiente di correlazione ( $X$  e  $Y$  sono variabili binarie che indicano la presenza di un item)

$$\rho = \frac{E[X \cdot Y] - E[X] \cdot E[Y]}{\sigma(X) \cdot \sigma(Y)}$$

- Queste variabili sono descritte dalla distribuzione di Bernoulli il cui parametro è  $\text{sup}(i)$  per l'i-esimo item, per cui:

$$\rho_{ij} = \frac{\text{sup}(\{i, j\}) - \text{sup}(i) \cdot \text{sup}(j)}{\sqrt{\text{sup}(i) \cdot \text{sup}(j) \cdot (1 - \text{sup}(i)) \cdot (1 - \text{sup}(j))}}$$

# Mining di pattern interessanti

- Coefficiente di correlazione

$$\rho_{ij} = \frac{\text{sup}(\{i, j\}) - \text{sup}(i) \cdot \text{sup}(j)}{\sqrt{\text{sup}(i) \cdot \text{sup}(j) \cdot (1 - \text{sup}(i)) \cdot (1 - \text{sup}(j))}}$$

- È una misura simmetrica poiché si può mostrare che  $\rho_{i,j} \equiv \rho_{\neg i, \neg j}$
- Poco significativo per piccoli valori di supporto

# Mining di pattern interessanti

- Test  $\chi^2$ 
  - È un classico test simmetrico per misurare l'indipendenza statistica tra variabili o per comparare distribuzioni
  - Si assume che  $X$  sia una variabile statistica associata all'osservazione di un itemset descritto come una stringa di  $k$  variabili binarie che indicano la presenza/assenza di un certo item per cui possono avversi  $2^k$  stati diversi, corrispondenti a  $2^k$  possibili itemset effettivamente osservati

# Mining di pattern interessanti

- Test  $\chi^2$

$$\chi^2(X) = \sum_{i=1}^{2^{|X|}} \frac{(O_i - E_i)^2}{E_i}$$

- $O_i$  è il numero di osservazioni dello stato  $i$ -esimo, mentre  $E_i$  è quello atteso cioè il prodotto del numero di transazioni per i supporti assoluti dei suoi item
- Rispetta la proprietà di chiusura *verso l'alto* → si possono implementare algoritmi di mining (in forma duale)
- Non da informazioni sul segno della dipendenza tra gli item

# Mining di pattern interessanti

- Interest ratio

$$I(\{i_1, \dots, i_k\}) = \frac{\text{sup}(\{i_1, \dots, i_k\})}{\prod_{j=1}^k \text{sup}(i_j)}$$

- $I = 1$  implica indipendenza statistica
  - $I > 1 \rightarrow$  correlazione positiva
  - $I < 1 \rightarrow$  correlazione negativa
- Valori di supporto estremamente bassi per singoli item generano valori di  $I$  elevati, ma statisticamente poco significativi

# Mining di pattern interessanti

- Coefficiente coseno tra colonne del database
  - Riorganizzando il database  $\mathcal{T}$  in maniera verticale, si possono calcolare similarità tra item rappresentati come vettori binari corrispondenti ai *tid* che contengono l'item

$$\text{cosine}(i, j) = \frac{\text{sup}(\{i, j\})}{\sqrt{\text{sup}(i)} \sqrt{\text{sup}(j)}}$$

- I valori di supporto  $\text{sup}(\{i, j\})$  possono essere calcolati come intersezione delle *tid* list di  $i$  e  $j$
- Il coefficiente coseno può essere visto come la media geometrica delle confidenze delle regole  $\{i\} \Rightarrow \{j\}$  e  $\{j\} \Rightarrow \{i\}$

# Mining di pattern interessanti

- Coefficiente di Jaccard
  - Riorganizzando il database  $\mathcal{T}$  in maniera verticale, sia  $S_i$  la *tid* list dell' $i$ -esimo item:
$$J(S_1 \dots S_k) = \frac{|\cap S_i|}{|\cup S_i|}$$
$$J(S_1 \dots S_k) \geq J(S_1 \dots S_{k+1})$$
- L'algoritmo Apriori e le sue varianti possono essere implementati in termini del coefficiente di Jaccard

# Mining di pattern interessanti

- Collective strength
  - Il coefficiente di collective strength di un itemset  $I$ , denominato  $C(I)$ , si basa sul concetto di *violation rate*  $v(I)$  dell'itemset
    - Violation rate: la frazione di transazioni in  $\mathcal{T}$  che contengono solo alcuni item di  $I$ , ma non tutti

$$C(I) = \frac{1 - v(I)}{1 - E[v(I)]} \cdot \frac{E[v(I)]}{v(I)}$$

- $C(I) = 0 \rightarrow$  correlazione negativa;  $C(I) = 1 \rightarrow$  indipendenza;  $C(I) = \infty \rightarrow$  correlazione positiva

# Mining di pattern interessanti

- Collective strength
  - Il valore atteso del violation rate  $E[v(I)]$  si può stimare dalle frazioni di transazioni  $p_i$  in cui ogni suo item  $i$  è presente

$$E [v(I)] = 1 - \prod_{i \in I} p_i - \prod_{i \in I} (1 - p_i)$$

- Si assume indipendenza statistica tra gli item
- In genere si impone una proprietà di chiusura per cui si ricercano itemset / tali che:  
 $C(I) \geq s, C(J) \geq C(I) \forall J \subset I$

# Gestione di grandi database

- Gli algoritmi di mining di pattern frequenti o interessanti necessitano di dati residenti preferibilmente in memoria centrale
  - Algoritmi di tipo depth first su enumeration tree
- Una soluzione può essere il *campionamento delle transazioni* per ridurre il numero di elementi su cui lavorare
  - Falsi positivi: itemset a supporto elevato all'interno dei dati campionati, ma non sull'intero database
  - Falsi negativi: itemset a supporto basso all'interno dei dati campionati, ma con supporto elevato sull'intero database

# Gestione di grandi database

- Una soluzione per la riduzione di falsi positivi e falsi negativi è l'uso di *tecniche di partizionamento*
  - Il database è suddiviso in  $k$  partizioni disgiunte e il mining si applica separatamente su ognuna di esse
  - *Un itemset dev'essere interessante almeno in una partizione* → rimozione dei falsi negativi
  - I falsi positivi si eliminano con un conteggio esplicito a posteriori del supporto di ogni itemset su tutte le partizioni