Scoperta di sequenze di comportamento

Vincenzo Antonio Manganaro

vincenzomang@virgilio.it, www.statistica.too.it

Indice

Fra le tecniche di DM grande importanza riveste l'utilizzo di modelli che permettono la scoperta di forme di comportamento sequenziali. Lo studio di queste tecniche sarà oggetto del seguente capitolo. L'approccio che seguiremo fa riferimento al mondo della grande distribuzione (caso tipico di market basket analysis), tuttavia possiamo facilmente trasporre le metodologie studiate a qualunque altro settore ove sia presente un rapporto di tipo acquisto/vendita.

1 Definizione del problema ed esempi.

Le regole di associazione, viste nel capitolo 2, si riferiscono alla scoperta di regole che nel caso tipico della basket analysis riguardano il comportamento di più clienti in una singola transazione e più in generale in un medesimo contesto temporale. Il miglioramento nelle tecnologie relative all'uso dei lettori dei codici a barre, nonché la sempre maggiore disponibilità di dati contenenti il codice cliente (CUSTOMER ID),

grazie all'uso sempre più massiccio di carte di credito e carte fedeltà, ha permesso lo sviluppo di tecniche che vanno oltre l'analisi del comportamento nella singola transazione e che riescono a costruire dei modelli noti nella letteratura del DM come modelli sequenziali/temporali. Tali modelli estraggono dai dati delle sequenze di comportamento interessanti e legati a più contesti temporali. Il problema può essere impostato nel modo seguente: sia dato un database D di transazioni, nel quale ogni transazione è caratterizzata dai campi CODICE CLIENTE, TEMPO DELLA TRANSAZIONE, OGGETTI ACQUISTATI NELLA TRANSAZIONE. L'obiettivo è quello di studiare il comportamento degli acquirenti nell'ottica di più periodi, ovvero quello di estrarre dai dati delle sequenze di comportamento del tipo "i clienti prendono a noleggio il film A, quindi B e poi C" con A,B,C non necessariamente consecutivi. A tal scopo diamo ora una serie di definizioni utili per una impostazione formale del problema.

- Si definisce *itemset* un insieme non vuoto di items (o oggetti). Possiamo associare ad ogni item un intero positivo in modo da indicare con $i(i_1 i_2 \dots i_m)$ il generico itemset. L'elemento i_j è un singolo item dell'itemset, corrispondente ad un intero.
- Una sequenza è una lista ordinata di itemsets. $\langle s_1 s_2 \dots s_n \rangle$ rappresenta la generica sequenza s, con s_j un itemset.
- La sequenza $\langle a_1 a_2 \dots a_m \rangle$ "è contenuta in" un'altra sequenza $\langle b_1 b_2 \dots b_m \rangle$ se esistono gli interi $i_1 < i_2 \dots < i_n$ tali che $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, $\dots$, $a_n \subseteq b_{i_n}$. Useremo il simbolo " $\prec''$ per denotare la relazione "è contenuta in ". Ad esempio la sequenza $\langle (3)(4\ 5)(8) \rangle \prec \langle (7)(3\ 8)(9)(4\ 5\ 6)(8) \rangle$ poichè $(3) \subseteq (3\ 8)$, $(4\ 5) \subseteq (4\ 5\ 6)$ e $(8) \subseteq (8)$. Però la sequenza $\langle (3)(5) \rangle$ non è contenuta in $\langle (3\ 5) \rangle$ e viceversa, poichè la prima rappresenta gli items 3 e 5 comprati in due diversi istanti temporali e l'altra gli items 3 e 5 comprati insieme.
- In un insieme di sequenze, una sequenza s è *massimale* se non è contenuta in ogni altra sequenza dell'insieme.

Tutte le transazioni di un cliente possono essere viste come una sequenza, dove ogni transazione corrisponde ad un insieme di items e la lista di transazioni, ordinate rispetto al tempo, corrisponde alla sequenza. Ad ogni singolo cliente (riconosciuto attraverso il campo CUSTOMER ID) si può dunque associare una sequenza, che chiameremo *sequenza cliente*. Formalmente se indichiamo con t_1, t_2, $\dots, t_n$ gli istanti temporali delle transazioni e se indichiamo con $itemset(t_i)$ l'in-

sieme di items acquistati nell'istante t_i, la sequenza cliente può essere indicata con $\langle itemset(t_1), itemset(t_2), \ldots, itemset(t_n)\rangle$.
- Un cliente "supporta" una sequenza s se s è contenuta nella sequenza cliente associata a quel particolare cliente.
- Il *supporto* per una sequenza è definito come la frazione sul totale dei clienti che supportano questa sequenza.
- La lunghezza di una sequenza è definita come il numero di itemsets nella sequenza stessa. Una sequenza di lunghezza k la chiameremo una *k-sequence*.
- Il supporto per un generico itemset i è definito come la frazione sul totale dei clienti che comprano tutti gli items in i in una singola transazione. In tal modo l'itemset i e la 1-sequence $\langle i\rangle$ hanno il medesimo supporto.
- Nel database di transazioni D il problema di estrarre da questo dei percorsi sequenziali si riduce alla ricerca delle sequenze massimali fra tutte le sequenze che hanno un supporto minimo specificato dall'utilizzatore. Ogni sequenza massimale rappresenta un percorso sequenziale mentre una sequenza che soddisfa il valore di supporto minimo la chiameremo una *large sequence*.
- Un itemset con il minimo supporto è chiamato *large itemset* o *litemset*. Da notare che ogni *itemset* di una *large sequence* deve avere il supporto minimo. Da qui segue necessariamente che ciascuna *large sequence* è una lista ordinata di *litemsets*.

Prima di introdurre gli algoritmi per la ricerca di sequenze, illustriamo con un esempio le definizioni appena introdotte. A tale scopo, sia dato il seguente database di transazioni, nel quale ciascun item è codificato attraverso un intero positivo:

TRANSAZIONE (TEMPO)	**CUSTOMER ID**	**OGGETTI ACQUISTATI**
10 GIU	2	10, 20
12 GIU	5	90
15 GIU	2	30
20 GIU	2	40, 60, 70
25 GIU	4	30
25 GIU	3	30, 50, 70
25 GIU	1	30
30 GIU	1	90
30 GIU	4	40, 70
25 LU	4	90

Dal seguente database si passa al database ordinato secondo il CUSTOMER ID e il TEMPO DELLA TRANSAZIONE:

CUSTOMER ID	**TRANSAZIONE (TEMPO)**	**OGGETTI ACQUISTATI**
1	25 GIU	30
1	30 GIU	90
2	10 GIU	10, 20
2	15 GIU	30
2	20 GIU	40, 60, 70
3	25 GIU	30, 50, 70
4	25 GIU	30
4	30 GIU	40, 70
4	25 LU	90
5	12 GIU	90

Con un supporto minimo del 25%, pari a 2 clienti, solo due sequenze $\langle(30)(90)\rangle$ e $\langle(30)(40\ 70)\rangle$, fra tutte quelle che soddisfano il valore di supporto minimo su indicato, sono massimali. Il percorso sequenziale $\langle(30)(90)\rangle$ è supportato dai clienti 1 e 4. Il cliente 4 compra gli items (40 70) nel periodo compreso tra l'acquisto dell'item 30 e l'acquisto dell'item 90, ma supporta la sequenza $\langle(30)(90)\rangle$ dato che comunque ci interessiamo a modelli che non sono necessariamente contigui. Il percorso sequenziale $\langle(30)(40\ 70)\rangle$ è supportato dai clienti 2 e 4. Il cliente 2 compra l'item

60 insieme agli items 40 e 70, ma supporta questa sequenza poichè (40 70) è un sottoinsieme di (40 60 70).
Un esempio di sequenza che non ha il supporto minimo è la sequenza $\langle(10\ 20)(30)\rangle$ che è supportata solo dal cliente 2. Le sequenze $\langle(30)\rangle$, $\langle(40)\rangle$, $\langle(70)\rangle$, $\langle(90)\rangle$, $\langle(30)(40)\rangle$, $\langle(30)(70)\rangle$ e $\langle(40)(70)\rangle$, sebbene hanno un supporto maggiore o uguale al supporto minimo, non fanno parte del "set di risposta" seguente poichè non sono massimali:

Percorsi sequenziali con supporto $\geq$ 25%
$\langle(30)(90)\rangle$
$\langle(30)(40\ 70)\rangle$

Riportiamo inoltre nella seguente tabella le sequenze cliente per ciascun cliente:

CUSTOMER ID	**Sequenze cliente**
1	$\langle(30)(90)\rangle$
2	$\langle(10\ 20)(30)(40\ 60\ 70)\rangle$
3	$\langle((30\ 50\ 70)\rangle$
4	$\langle(30)(40\ 70)(90)\rangle$
5	$\langle(90)\rangle$

2 Scoperta di sequenze.

Il problema della scoperta di sequenze è generalmente risolto nelle seguenti 5 fasi:

1. *Sort phase*
2. *Litemset phase*
3. *Transformation phase*
4. *Sequence phase*
5. *Maximal phase*

Descriviamo brevemente le fasi su indicate, rimandando ai paragrafi successivi per quanto riguarda gli approfondimenti alla 4° fase con i relativi algoritmi.
SORT PHASE: il database D viene ordinato rispetto al CUSTOMER ID, come criterio principale, e al TEMPO DELLA TRANSAZIONE come criterio secondario. Questa fase converte in maniera implicita il database di transazioni originale in un database di sequenze cliente (si veda l'esempio del paragrafo precedente).
LITEMSET PHASE: è la fase in cui si trova l'insieme L di tutti i litemsets, ovvero l'insieme che ha come elementi tutti gli itemsets con supporto maggiore o uguale al valore di supporto minimo. Contemporaneamente si trova anche l'insieme di tutte le large sequences di lunghezza 1 (large 1-sequences), essendo quest'ultimo insieme costituito da $\{\langle l\rangle | l \in L\}$, ovviamente coincidente con l'intero L.
Il problema della ricerca di large itemsets in un dato database di transazioni è per certi versi analogo a quello visto per la ricerca di regole di associazione; tuttavia la differente definizione di supporto non permette l'estensione degli algoritmi utilizzati nella ricerca di regole di associazione al caso delle sequenze. Nel primo caso il supporto di un itemset è definito come la frazione sul totale delle transazioni nel quale l'itemset è presente[19] (si veda la definizione di supporto nel caso di regole di associazione), mentre nel caso delle sequenze il supporto è definito come la frazione sul totale dei clienti che hanno comprato gli oggetti dell'itemset in almeno una delle loro transazioni. Questa diversa definizione di supporto comporta una differenza sostanziale per quanto riguarda il calcolo del supporto stesso. Infatti nel caso delle sequenze il supporto dovrebbe essere incrementato solo una volta per ciascun cliente anche se quest'ultimo compra lo stesso insieme di oggetti in due o più transazioni. In questa fase, inoltre, all'insieme di litemset viene associato un insieme di interi positivi contigui: trattando infatti ciascun litemset come una singola entità (un numero intero) è possibile ridurre il tempo richiesto per controllare se una sequenza è contenuta in una sequenza cliente. In tal modo ciascun litemset viene mappato con un intero; nell'esempio del paragrafo precedente si avrebbe la seguente mappatura con a sinistra l'insieme di litemsets e a destra l'insieme di interi corrispondenti a

[19] In realtà abbiamo definito il supporto di una regola di associazione, ma in ogni caso se s è il supporto della regola $A \Rightarrow B$, si ha che $s(A \Rightarrow B) = s(A, B)$ essendo (A, B) l'itemset. Infatti il supporto è definito come la frazione sul totale delle transazioni di quelle che contengono sia gli elementi dell'antecedente che quelli del conseguente.

ciascun litemset:

Litemset	**Numero corrispondente**
(30)	1
(40)	2
(70)	3
(40 70)	4
(90)	5

TRANSFORMATION PHASE: la fase della sequence phase, successiva alla transformation phase, richiede di contollare continuamente ed iterativamente se determinate sequenze di lunghezza $k+1$ (sequenze candidate), ottenute a partire dall'insieme di large k-sequences, sono esse stesse delle large sequences, ovvero sono contenute in un numero di sequenze cliente maggiore o uguale al numero minimo derivato dal valore di supporto minimo. La transformation phase prepara il terreno per effettuare questa operazione di "conteggio" che consiste nel contare per ciscuna sequenza candidata quanti clienti la supportano. Per velocizzare il conteggio, in questa fase ogni sequenza cliente viene trasformata utilizzando una rappresentazione alternativa. Tale rappresentazione prevede che in una sequenza cliente trasformata, ogni transazione è sostituita dall'insieme di tutti i litemset contenuti in quella transazione. Se una transazione non contiene nessun litemset, essa viene esclusa dalla sequenza trasformata e nel caso in cui un'intera sequenza cliente è costituita da transazioni che non contengono dei litemsets, si procede alla eliminazione della sequenza cliente dal database trasformato, fermo restando che tale sequenza cliente eliminata contribuisce ugualmente al conteggio del totale clienti anche nel database trasformato. Una sequenza cliente è adesso rappresentata da una lista di insiemi di litemsets. Ogni insieme di litemsets è rappresentato da $\{l_1, l_2, \ldots, l_n\}$, dove l_i è un litemset. Questo database trasformato, che generalmente indicheremo con D_t, nel caso dell'esempio del paragrafo precedente, è indicato nel seguente schema dove vengono illustrate tutte le fasi della trasformazione:

ID	**SEQUENZA CLIENTE ORIGINALE**	**SEQUENZA CLIENTE TRASFORMATA**	**DOPO LA MAPPATURA**
1	$\langle(30)(90)\rangle$	$\langle\{(30)\}\{(90)\}\rangle$	$\langle\{1\}\{5\}\rangle$
2	$\langle(10\ 20)(30)(40\ 60\ 70)\rangle$	$\langle\{(30)\}\{(40),(70),(40\ 70)\}\rangle$	$\langle\{1\}\{2,3,4\}\rangle$
3	$\langle(30\ 50\ 70)\rangle$	$\langle\{(30),(70)\}\rangle$	$\langle\{1,3\}\rangle$
4	$\langle(30)(40\ 70)(90)\rangle$	$\langle\{(30)\}\{(40),(70),(40\ 70)\}\{(90)\}\rangle$	$\langle\{1\}\{2,3,4\}\{5\}\rangle$
5	$\langle(90)\rangle$	$\langle\{(90)\}\rangle$	$\langle\{5\}\rangle$

Ad esempio, durante la trasformazione della sequenza cliente relativa al cliente 2 la transazione (10 20) è eliminata poichè non contiene nessun litemset e la transazione (40 60 70) è sostituita dall'insieme di litemsets {(40), (70), (40 70)}.
SEQUENCE PHASE: si tratta della fase specifica della scoperta di sequenze che include gli algoritmi che tratteremo nei prossimi paragrafi.
MAXIMAL PHASE: in quest'ultima fase l'obiettivo è la ricerca delle sequenze massimali nell'insieme delle large sequences. In alcuni algoritmi, che vedremo nei paragrafi successivi, tale fase viene eseguita all'interno della sequence phase allo scopo di ridurre il tempo impiegato nel conteggio di sequenze non massimali.
Avendo trovato l'insieme S di tutte le large sequences nella sequence phase, il seguente algoritmo può essere usato per trovare le sequenze massimali. In esso la lunghezza della sequenza più lunga è indicata con n:

for$(k = n;\ k > 1;\ k--)$ **do**
foreach k-sequence s_k **do**
Cancella da S tutte le sottosequenze[20] s_k

Decriveremo successivamente il modo in cui la maximal phase viene eseguita all'interno degli algoritmi che introdurremo di seguito.

3 Algoritmi per la sequence phase.

Gli algoritmi per la ricerca di large sequences si articolano in diversi passi. Ogni passo è caratterizzato da un insieme iniziale di sequenze, che hanno il supporto minimo

[20]Ovviamente l'algoritmo presuppone la presenza di un ulteriore algoritmo per la ricerca di sottosequenze di una data sequenza s_k di lunghezza k.

richiesto (large sequences), che viene utilizzato per generare sequenze di maggiore lunghezza candidate ad avere anch'esse il supporto minimo (candidate large sequences). In ogni passo viene calcolato il supporto per queste sequenze candidate, in modo che alla fine del passo si vede quali sequenze sono effettivamente delle large sequences (operazione di conteggio). Queste ultime costituiscono l'insieme di partenza per il passo successivo. Nel primo passo tutte le sequenze di lunghezza 1 col supporto minimo, ottenute nella litemset phase, formano l'insieme di sequenze iniziale.

Gli algoritmi che presenteremo appartengono a due famiglie che potremmo grosso modo indicare come la famiglia del *conteggio totale* e la famiglia del *conteggio parziale*[21]. Negli algoritmi appartenenti alla prima famiglia vengono conteggiate tutte le large sequences, incluse quelle non massimali; successivamente nella maximal phase vengono individuate le sole sequenze massimali. Gli algoritmi della seconda famiglia si basano, invece, sull'intuizione che se si è interessati alle sole sequenze massimali è possibile evitare di conteggiare sequenze che sono contenute in sequenze più lunghe (e qui a parte il verificare se sono o meno delle large sequences, queste sequenze non potranno essere massimali) se si opera prima sulle sequenze più lunghe. Tuttavia, si deve aver cura nell'evitare le sequenze lunghe che non hanno il supporto minimo richiesto, altrimenti il tempo risparmiato in un primo momento nel non conteggiare sequenze non massimali viene successivamente perso nell'escludere le sequenze lunghe che non hanno il supporto minimo e che negli algoritmi appartenenti alla prima famiglia non sarebbero state mai inserite per il semplice fatto che alcune, o magari tutte, le sottosequenze non avrebbero avuto anch'esse il supporto minimo richiesto. Presenteremo in questo contesto un algoritmo appartenente alla prima famiglia, l'algoritmo *AprioriAll*, e due algoritmi appartenenti alla seconda, gli algoritmi *AprioriSome* e *DynamicSome*. Questi ultimi algoritmi presentano una forward phase ed una backward phase. Nella forward phase si trovano tutte le sequenze con determinate lunghezze, mentre nella backward phase si trovano tutte le restanti sequenze. La differenza fra questi due algoritmi risiede, come vedremo, nella procedura che essi usano per generare le candidate sequences nella forward phase.

[21] Traduzione dall'inglese count-all e count-some.

3.1 L'algoritmo AprioriAll.

L'algoritmo AprioriAll è descritto nel seguente schema:

L_1 = {Large 1-sequences}; *Risultato della litemset phase*
for ($k = 2$; $L_{k-1} \neq \oslash$; $k++$) **do**
  begin
  C_k = nuove candidate generate da L_{k-1}
  foreach sequenza cliente c nel database **do**
    Incrementa il conteggio di tutte le candidate in C_k che sono contenute in c.
  L_k = candidate in C_k con il supporto minimo.
  end
Output= sequenze massimali nell'insieme $\cup_k L_k$;

Ad ogni passo l'algoritmo utilizza le large sequences ottenute al passo precedente per generare le sequenze candidate, delle quali calcola il supporto. Le sequenze candidate, il cui supporto risulta non minore del valore soglia stabilito, costituiranno le large sequences del passo successivo. Il primo passo dell'algoritmo è semplicemente costituito dall'insieme L_1 che rappresenta il risultato della litemset phase, ovvero l'insieme di large sequences di lunghezza 1.
Una considerazione a parte merita la funzione per la generazione dell'insieme C_k (vedi riga 4 dell'algoritmo), ovvero l'insieme di sequenze candidate a partire dall'insieme L_{k-1}, cioè dall'insieme di large sequences di lunghezza $k-1$. Tale funzione (nota come **apriori-generate** function) prende come argomento l'insieme L_{k-1} e restituisce l'insieme C_k. Essa è essenzialmente costituita dalle seguenti due fasi:

1. Fase di congiungimento;
2. Fase di pruning.

La fase di congiungimento, che può essere implementata in svariati modi[22], è equivalente all'estendere ogni sequenza dell'insieme L_{k-1} con ogni large itemset e quindi cancellare quelle sequenze la cui sottosequenza ottenuta eliminando il $(k-1)^{\circ}$-imo itemset non appartiene all'insieme L_{k-1}. Il ragionamento è semplice: se accade che la sottosequenza di lunghezza $k-1$, ottenuta come descritto dalla sequenza di lunghezza k, non fa parte dell'insieme L_{k-1}, ciò implica che esiste una sequenza di lunghezza k candidata ad essere una large sequence che ammette una sottosequenza di lunghezza $k-1$ che non è una large sequence. Il che è evidentemente assurdo, per cui la sequenza di lunghezza k non può appartenere all'insieme C_k.
Nella fase di pruning si eliminano dall'insieme C_k, ottenuto con la fase di congiungimento, quelle sequenze di lunghezza k che ammettono almeno una sottosequenza di lunghezza[23] $k-1$ che non appartiene all'insieme L_{k-1}. Il ragionamento è analogo al precedente.
L'esempio seguente dovrebbe chiarire il meccanismo costituito dalle due fasi appena illustrate.
Supponiamo che il seguente insieme L_3 di large 3-sequences sia l'input della funzione apriori-generate. Mostriamo l'output ottenuto, rispettivamente dopo la fase di congiungimento e dopo la fase di pruning, nella seconda e terza colonna del seguente prospetto:

---

[22] Almeno in questa sede non è nostro compito scendere nei particolari dei diversi sottoalgoritmi, nonchè fare delle considerazioni dettagliate circa la loro differente performance. In ogni caso si veda l'articolo "Mining Sequential Patterns" di Rakesh Agrawal and Ramakrishnan Srikant IBM Research Division Almaden Research Center 650 Harry Road San Jose, CA 95120-6099.

[23] E' ovvio che si considerano tutte le sottosequenze di lunghezza $k-1$ e non si considera quella che si ottiene eliminando il $(k-1)^{\circ}$-imo itemset, dal momento che quest'ultima è stata considerata nella fase precedente e con il risultato che qualora essa non appartenesse all'insieme L_{k-1}, la sequenza di origine non sarebbe stata inclusa già prima nell'insieme C_k.

Insieme L_3	**Supporto**	**Insieme C_4 dopo il congiungimento**	**Insieme C_4 dopo il pruning**
$\langle 1\ 2\ 3\rangle$	2	$\langle 1\ 2\ 3\ 4\rangle$	$\langle 1\ 2\ 3\ 4\rangle$
$\langle 1\ 2\ 4\rangle$	2	$\langle 1\ 2\ 4\ 3\rangle$	
$\langle 1\ 3\ 4\rangle$	3	$\langle 1\ 3\ 4\ 5\rangle$	
$\langle 1\ 3\ 5\rangle$	2	$\langle 1\ 3\ 5\ 4\rangle$	
$\langle 2\ 3\ 4\rangle$	2		

Chiarito il meccanismo con il quale vengono determinate le sequenze candidate di generica lunghezza k riprendiamo l'algoritmo AprioriAll; esso opera nel seguente modo:

L_1 viene assegnato come l'insieme di large sequences di lunghezza 1 ottenuto dalla litemset phase. Il ciclo for, con l'assegnazione $k = 2$, la condizione $L_{k-1} \neq \oslash$ e l'ncremento $k++$, trova per ogni k l'insieme C_k di sequenze candidate a partire dall'insieme L_{k-1} e sfruttando la funzione apriori-generate descritta prima. Inoltre sempre all'interno dello stesso ciclo for l'algoritmo considera ciascuna sequenza cliente c del database trasformato D_t e controlla se ogni sequenza appartenente a C_k è contenuta in quella particolare sequenza cliente incrementando di una unità il supporto di tale sequenza quando questa è contenuta nella sequenza cliente. In tal modo si perviene all'insieme L_k, cioè all'insieme di sequenze $\subseteq C_k$ che hanno il supporto minimo. L'output dell'algoritmo è infine costituito dall'insieme di sequenze massimali contenuto nell'insieme $\cup_k L_k$.

Consideriamo ora il seguente database di sequenze cliente trasformate, cioè delle sequenze cliente nelle quali ogni transazione è stata sostituita dall'insieme di litemsets contenuti nella transazione e i litemsets sostituiti da numeri interi:

SEQUENZE CLIENTE TRASFORMATE
$\langle\{1\ 5\}\{2\}\{3\}\{4\}\rangle$
$\langle\{2\}\{3\}\{4\}\{3\ 5\}\rangle$
$\langle\{1\}\{2\}\{3\}\{4\}\rangle$
$\langle\{1\}\{3\}\{5\}\rangle$
$\langle\{4\}\{5\}\rangle$

Il supporto minimo specificato è del 40% corrispondente a 2 sequenze clienti. Come si comporta l'algoritmo AprioriAll per individuare sequenze di comportamento all'interno di un tale database trasformato?
Il primo passo è svolto nella litemset phase, dove si determinano le large sequences di lunghezza 1 che mostriamo nel seguente prospetto insieme ai valori di supporto:

Large 1-sequences (Set L_1)	**Supporto**
$\langle 1 \rangle$	4
$\langle 2 \rangle$	2
$\langle 3 \rangle$	4
$\langle 4 \rangle$	4
$\langle 5 \rangle$	2

Successivamente mostriamo l'insieme L_k con i relativi valori di supporto alla fine del 2°, 3° e 4° passo dell'algoritmo e dunque con k rispettivamente uguale ai valori 2, 3 e 4 (per $k = 5$ non viene generata nessuna sequenza candidata, per cui l'insieme $L_5 = \oslash$ e l'algoritmo si arresta):

Set L_2	**Supporto**	**Set L_3**	**Supporto**	**Set L_4**	**Supporto**
$\langle 1\ 2 \rangle$	2	$\langle 1\ 2\ 3 \rangle$	2	$\langle 1\ 2\ 3\ 4 \rangle$	2
$\langle 1\ 3 \rangle$	4	$\langle 1\ 2\ 4 \rangle$	2		
$\langle 1\ 4 \rangle$	3	$\langle 1\ 3\ 4 \rangle$	3		
$\langle 1\ 5 \rangle$	3	$\langle 1\ 3\ 5 \rangle$	2		
$\langle 2\ 3 \rangle$	2	$\langle 2\ 3\ 4 \rangle$	2		
$\langle 2\ 4 \rangle$	2				
$\langle 3\ 4 \rangle$	3				
$\langle 3\ 5 \rangle$	2				
$\langle 4\ 5 \rangle$	2				

Dall'insieme $\{L_1 \cup L_2 \cup L_3 \cup L_4\}$ si ottiene infine l'insieme di sequenze massimali:

Large sequences massimali	**Supporto**
$\langle 1\ 2\ 3\ 4 \rangle$	2
$\langle 1\ 3\ 5 \rangle$	2
$\langle 4\ 5 \rangle$	2

3.2 L'algoritmo AprioriSome.

L'AprioriSome è un algoritmo appartenente alla famiglia del conteggio parziale e costituisce in qualche modo una generalizzazione del precedente algoritmo AprioriAll. La sua struttura è la seguente:

- Forward phase
  $L_1 = \{$large 1-sequences$\}$; *Risultato della litemset phase*
  $C_1 = L_1$; *assegnazione per evitare che l'algoritmo cada in un loop *
  $last = 1$; * la variabile *last* contiene il numero indicante la lunghezza
  delle sequenze conteggiate nel passo precedente*
  for$(k = 2;\ C_{k-1} \neq \oslash;\ k++)$ **do**
  begin
  if (L_{k-1} è noto) **then**
  $C_k =$ Nuove sequenze candidate generate da L_{k-1};
  else
  $C_k =$ Nuove sequenze candidate generate da C_{k-1};
  if $(k == next(last))$ **then begin**
  foreach sequenza cliente c nel database **do**
  Aumenta di una unità il numero di tutte le candidate
  sequences $\in C_k$ che sono contenute in c.
  *si tratta dell'operazione di conteggio che consiste nel
  calcolo del supporto per ciascuna sequenza*.
  $L_k =$ Sequenze candidate in C_k con supporto minimo.
  $last = k$;
  end
  end

- Backward phase

  for($k = k_{max}$; $k >= 1$; $k--$;) **do**

  if (L_k non è stato determinato nella forward phase) **then begin**

  Cancella tutte le sequenze in C_k contenute in qualche L_i, con $i > k$;

  foreach sequenza cliente c in D_t **do**

  Aumenta di una unità il numero di tutte le candidate sequences $\in C_k$ che sono contenute in c.

  L_k = Sequenze candidate in C_k con supporto minimo.

  end

  elsebegin *caso in cui L_k è noto*

  Cancella tutte le sequenze in L_k contenute in qualche L_i, con $i > k$.

  end

  Output= Sequenze massimali nell'insieme $\cup_k L_k$;

L'algoritmo prevede, come già accennato, una forward phase ed una backward phase. Nella forward phase vengono conteggiate solo le sequenze che hanno una determinata lunghezza; ad esempio, potremmo conteggiare sequenze di lunghezza 1,2,4 e 6 nella forward phase e sequenze di lunghezza 3,5 nella backward phase. La funzione next (nella forward phase) prende come parametro di input la lunghezza delle sequenze conteggiate nell'ultimo passo e restituisce in output la lunghezza delle sequenze da conteggiare nel passo successivo. In tal modo questa funzione determina esattamente quali sequenze devono essere conteggiate nella forward phase[24].
Nella backward phase vengono conteggiate le sequenze con i valori di lunghezza saltati nella precedente fase. Il conteggio avviene, però, dopo aver eliminato tutte le sequenze contenute in determinate large sequences. Infatti le sequenze più piccole non possono appartenere al set di risposta dell'algoritmo poichè l'utilizzatore è interessato alle sole sequenze massimali. Inoltre in questa fase vengono eliminate tutte le large sequences della fase precedente che non sono massimali.

[24]Si ricordi che il conteggio di sequenze consiste nel calcolo del relativo supporto.

Di seguito è riportata una descrizione dettagliata delle due fasi considerate.
Nella forward phase, partendo dal set L_1 ovvero dal set di large sequences di lunghezza 1 ottenuto dalla litemset phase, si considera una variabile di tipo intero, denominata *last*, inizializzata ad 1. Tale variabile rappresenta la lunghezza delle ultime sequenze conteggiate (1 perchè nella litemset phase si calcola il supporto di sequenza di lunghezza 1); l'assegnazione $C_1 = L_1$ è un'assegnazione di comodo che non influisce sull'output dell'algoritmo e che serve semplicemente a non far cadere in un loop senza uscita il successivo ciclo for (nel caso in cui $k = 2$). Il ciclo for è costituito dall'inizializzazione $k = 2$, dalle condizioni $C_{k-1} \neq \oslash$ e $L_{last} \neq \oslash$ e dall'incremento $k++$ che forza il ciclo a considerare valori successivi di k. Il ciclo for opera in questo modo:
se L_{k-1} è noto (nel senso che è stato determinato), allora con un meccanismo di generazione analogo a quello desritto nell'algoritmo precedente (vedi funzione apriori-generate) si genera l'insieme C_k a partire dal set L_{k-1}; qualora L_{k-1} non sia noto, si genera ugualmente l'insieme C_k a partire però dall'insieme C_{k-1} sempre attraverso il meccanismo visto. Il successivo if è caratterizzato dalla presenza della funzione *next* che è tipica di tale algoritmo. La funzione *next* ha come parametro iniziale la lunghezza delle sequenze conteggiate nell'ultimo passo e fornisce come output la lunghezza delle sequenze da conteggiare nel passo successivo. Se dunque k risulta uguale all'output della funzione *next*, per quel valore di k, cioè per le candidate sequences di lunghezza k, deve essere effettuato il conteggio in modo da determinare il sottoinsieme L_k di C_k, ovvero il sottoinsieme di sequenze candidate di lunghezza k con il supporto minimo. La funzione *last* sarà posta ovviamente uguale a k per iniziare un nuovo passo, finché si arriva al non soddisfacimento di una delle due condizioni del ciclo for.
La lunghezza delle sequenze che devono essere conteggiate dipende dalla funzione *next*. Vi possono essere dei casi estremi di funzioni *next*. Ad esempio, la funzione $next(k) = k + 1$ è tale che vengono conteggiate tutte le sequenze nella forward phase e in questo senso potremmo dire che l'algoritmo coincide con l'AprioriAll. La funzione $next(k) = 100 * k$ è tale che vengono conteggiate sequenze la cui lunghezza è un multiplo di 100.
Nella pratica la funzione *next* può dipendere da valori che si possono ricavare dall'esecuzione dell'iterazione k-esima dell'algoritmo. Ad esempio, in ogni passo k con

$k \geq 2$, nel quale è determinato sia il set C_k sia il set L_k, consideriamo il rapporto $|L_k|/|C_k|$, ovvero il rapporto tra il numero di sequenze candidate di lunghezza k e il numero di large sequences di lunghezza k. Ad esempio, potremmo costruire una funzione *next* il cui output, a parità di input, dipende dal rapporto su indicato in questo modo:

$$next(k) = \begin{cases} k+1 & \text{se } |L_k|/|C_k| < 0.666 \\ k+2 & \text{se } |L_k|/|C_k| < 0.75 \\ k+3 & \text{se } |L_k|/|C_k| < 0.80 \\ k+4 & \text{se } |L_k|/|C_k| < 0.85 \\ k+5 & \text{se } |L_k|/|C_k| < 1 \end{cases}$$

In tal caso la funzione *next* dipende dalla percentuale di large sequences sul totale di sequenze candidate. Questa dipendenza è tale che la funzione next fornirà come output valori di lunghezza tanto più elevati quanto più elevato è il rapporto $|L_k|/|C_k|$. In teoria per un valore del rapporto pari al 90% per sequenze di lunghezza 3, la funzione *next* dirà all'algoritmo di conteggiare al passo successivo sequenze di lunghezza pari a 3+5 = 8, non conteggiando di fatto sequenze di lunghezza 4,5,6 e 7. L'intuizione che sta alla base di questo ragionamento è semplice: se tale percentuale è elevata, è probabile che tali large sequences siano delle sottosequenze di sequenze più lunghe di quelle di lunghezza $k+1$, $k+2$, $k+3$ e $k+4$, per cui l'algoritmo va a conteggiare direttamente sequenze di lunghezza $k+5$[25].

L'output della backward phase è costituito dall'insieme di large sequences che sono anche massimali (l'algoritmo AprioriSome include al suo interno anche la maximal phase). Il ciclo for è caratterizzato da un'inizializzazione ($k = k_{max}$) che assegna a k il massimo valore assunto nella forward phase, da una condizione che stabilisce $k \geq 1$ e da un decremento ($k--$) che forza il ciclo a considerare valori interi decrescenti a partire dal k iniziale. Se L_k non è stato determinato nella forward phase nel ciclo for vengono cancellate tutte le sequenze in C_k contenute in qualche L_i con $i \geq k$; inoltre per ogni sequenza c nel database D_t (database delle sequenze cliente trasformate) si vede se le candidate sequences in C_k sono contenute in c, effettuando

[25]Come già detto in precedenza la filosofia che sta alla base della famiglia del conteggio parziale è che se si è interessati alle sole sequenze massimali è possibile evitare di conteggiare sequenze che sono poi contenute in altre sequenze; quindi si tende il più possibile a conteggiare sequenze abbastanza lunghe.

di fatto il conteggio dopo aver eliminato tutte le sottosequenze. Si ottiene in tal modo l'insieme L_k cioè le candidate sequences in C_k che hanno il supporto minimo. Nel caso in cui L_k è noto, perchè determinato nella forward phase, si eliminano tutte le sequenze in L_k contenute in qualche L_i con $i > k$ effettuando di fatto una sorta di maximal phase.

Per vedere in pratica il comportamento dell'algoritmo AprioriSome riprendiamo il database di sequenze cliente trasformate di pag. 60 insieme al risultato della litemset phase. Per semplicità consideriamo una funzione *next* del tipo $next(k) = 2k$. Nel secondo passo verranno quindi conteggiate le sequenze in C_2 per ottenere l'insieme L_2 proprio come avveniva nell'algoritmo precedente; tuttavia riportiamo nuovamente l'insieme L_2 con i relativi valori di supporto:

Set L_2	**Supporto**
$\langle 1\ 2 \rangle$	2
$\langle 1\ 3 \rangle$	4
$\langle 1\ 4 \rangle$	3
$\langle 1\ 5 \rangle$	3
$\langle 2\ 3 \rangle$	2
$\langle 2\ 4 \rangle$	2
$\langle 3\ 4 \rangle$	3
$\langle 3\ 5 \rangle$	2
$\langle 4\ 5 \rangle$	2

Successivamente nel terzo passo viene utilizzato l'insieme L_2 per ottenere l'insieme C_3 attraverso il meccanismo di generazione di candidate sequences più volte menzionato. Mostriamo l'insieme C_3:

Set C_3
$\langle 1\ 2 \rangle$
$\langle 1\ 3 \rangle$
$\langle 1\ 4 \rangle$
$\langle 1\ 5 \rangle$
$\langle 2\ 3 \rangle$
$\langle 2\ 4 \rangle$
$\langle 3\ 4 \rangle$
$\langle 3\ 5 \rangle$
$\langle 4\ 5 \rangle$

Sull'insieme C_3 non viene però effettuata alcuna operazione di conteggio, poiché la funzione *next* per come è definita non assume mai il valore 3. In sostanza si usa l'insieme C_3 per generare direttamente l'insieme C_4 che dopo l'operazione di pruning viene ad essere costituito dalla sola sequenza $\langle 1\ 2\ 3\ 4 \rangle$. Sull'insieme C_4 viene svolta l'operazione di conteggio, perchè previsto dalla funzione *next*, per ottenere infine l'insieme $L_4 = \langle 1\ 2\ 3\ 4 \rangle$, avendo la sequenza $\langle 1\ 2\ 3\ 4 \rangle$ supporto pari a 2. Successivamente il tentativo di generazione dell'insieme C_5 a partire dall'insieme L_4 porta ad un insieme vuoto che conclude il ciclo for e l'intera forward phase.
Nel primo passo della backward phase l'algoritmo considera il massimo valore di k nella fase precedente e dunque nel caso specifico $k = 4$. L'insieme L_4 è noto, in quanto determinato nella precedente forward phase, tuttavia da esso non viene eliminata nessuna sequenza dal momento che non esiste un insieme L_i con $i > 4$. L'insieme L_3 invece non esiste perché sull'insieme C_3 non sono state effettuate operazioni di conteggio, per cui si eliminano dall'insieme C_3 tutte le sequenze che sono sottosequenze di sequenze in L_4 e quindi sottosequenze di $\langle 1\ 2\ 3\ 4 \rangle$. Dopo questa operazione rimangono le sequenze $\langle 1\ 3\ 5 \rangle$ $\langle 3\ 4\ 5 \rangle$ sulle quali, effettuata l'operazione di conteggio, si ottiene la sequenza $\langle 1\ 3\ 5\ \rangle$ come large sequence massimale di lunghezza 3. Successivamente dall'insieme L_2 vengono eliminate tutte le sequenze tranne $\langle 4\ 5 \rangle$, essendo quest'ultima non contenuta in nessun insieme di sequenze L_i con $i \geq 2$. Per la stessa ragione si eliminano tutte le sequenze in L_1, ottenendo l'insieme di sequenze massimali uguale a quello ottenuto con l'algoritmo AprioriAll e che per comodità riscriviamo:

Set di sequenze massimali L_2	**Supporto**
$\langle 1\ 2\ 3\ 4 \rangle$	2
$\langle 1\ 3\ 5 \rangle$	2
$\langle 4\ 5 \rangle$	2

I due algoritmi visti portano allo stesso output, ma evidentemente già in questo semplice esempio l'algoritmo AprioriSome comporta dei vantaggi computazionali: infatti in quest'ultimo algoritmo e per quanto riguarda le sequenze di lunghezza 3 l'operazione di conteggio viene fatta per sole due sequenze contro le sei dell'algoritmo precedente[26]. Tuttavia ciò non è vero in generale; infatti vi possono essere dei casi nei quali l'insieme C_k generato a partire dall'insieme C_{k-1} e non dall'insieme L_{k-1} (per il fatto che $k-1$ non è un valore assunto dalla funzione *next*) sia costituito da un numero di sequenze maggiore di quello che si sarebbe ottenuto generando l'insieme da L_{k-1}, anche dopo aver eliminato le sequenze contenute negli insiemi L_i con $i > k$. Dunque può accadere che l'algoritmo AprioriSome effettui il conteggio di sequenze candidate che non sarebbero mai state prese in considerazione dall'algoritmo AprioriAll, di fatto procurando un maggiore carico computazionale.
In generale, il confronto fra le performance dei due algoritmi è abbastanza complesso e non è possibile stabilire in assoluto quale dei due comporta un maggiore carico computazionale. Nell'esempio visto l'AprioriSome garantiva una migliore performance, ma non potremmo certo generalizzare questo risultato.

3.3 L'algoritmo DynamicSome.

La struttura dell'algoritmo DynamicSome è mostrata in questo prospetto:

- Initialization phase
  step **è un intero ≥ 1*
  $L_1 = \{$large 1-sequences$\}$; *Risultato della litemset phase*
  for($k = 2$; $k <=$ *step* e $L_{k-1} \neq \oslash$; $k++$) **do**
  begin
  $C_k =$ Nuove sequenze candidate generate da L_{k-1};

[26]Inoltre l'algoritmo non effettua il conteggio su sequenze candidate che non siano state considerate anche nel precedente algoritmo.

foreach sequenza cliente c in D_t **do**
Aumenta di una unità il numero di tutte le sequenze candidate $\in C_k$ che sono contenute in c.
L_k = Sequenze candidate in C_k con supporto minimo.
end

- Forward phase
for($k = step$; $L_k \neq \oslash$; $k+ = step$) **do**
Si tratta del ciclo for con il quale si trova il set L_{k+step} a partire dagli insiemi L_k e L_{step}
begin
$C_{k+step} = \oslash$;
foreach sequenza cliente c in D_t **do**
begin
X = otf-generate [27] (c, L_k, L_{step});
Per ogni sequenza $x \in X$ incrementa di una unità il relativo supporto e aggiungi, se necessario, tale sequenza all'insieme C_{k+step}[28]
end
L_{k+step} = Sequenze candidate in C_{k+step} con supporto minimo.
end

- Intermediate phase
for($k = k_{max}$; $k > 1$; $k - -$) **do**
if (L_k non è stato ancora determinato) **then**
if (L_{k-1} è noto) **then**
C_k = Nuove candidate generate da L_{k-1};

[27] otf-generate sta per On-The-Fly Candidate Generation: si tratta di una funzione generatrice di sequenze candidate con un meccanismo diverso da quello finora visto che prevedeva l'uso della funzione apriori-generate.

[28] Il "se necessario" si riferisce al fatto che una determinata sequenza di lunghezza $k + step$ può essere ottenuta attraverso la funzione otf-generate considerando come parametri di input l'insieme L_k, l'insieme L_{step} e due differenti sequenze cliente c_1 e c_2. In tal caso si procederà ad aumentare di una unità il supporto della sequenza, ma non si inserirà questa all'interno dell'insieme C_{k+step} poiché già presente all'interno dell'insieme.

else

C_k = Nuove candidate generate da C_{k-1};

- Backward phase: Identica alla backward phase dell'algoritmo AprioriSome.

Allo stesso modo dell'algoritmo AprioriSome, nella forward phase viene saltato il conteggio di candidate sequences di determinate lunghezze. Ma a differenza di questo, la lunghezza delle candidate sequences che devono essere conteggiate è determinata da un passo (*step*) variabile (un intero ≥ 1). In particolare, l'algoritmo è composto da una Initialization phase in cui tutte le sequenze candidate di lunghezza compresa tra 1 e il passo scelto vengono conteggiate. Nella forward phase vengono conteggiate tutte le sequenze candidate che hanno lunghezza pari ad un multiplo del passo scelto. Infatti, la forward phase prevede un ciclo for che presenta l'assegnazione $k = step$, la condizione $L_k \neq \oslash$ e l'incremento $k+ = step$, il quale indica che k si dovrà muovere secondo i multipli di *step* (*step*, 2*step*, 3*step*, ...). Inizialmente si assegna l'insieme $C_{k+step} = \oslash$ e successivamente per ogni sequenza c all'interno del database trasformato D_t si calcola l'insieme X attraverso la nuova funzione otf-generate, che ha come parametri iniziali c, L_k ed L_{step}.
Chiariamo ora, senza entrare nei dettagli, il meccanismo della funzione otf-generate che viene usata in tale algoritmo e che differisce dal metodo di generazione di sequenze candidate visto per i due precedenti algoritmi. In generale, la funzione otf-generate prende come argomenti L_k, l'insieme di large k-sequences, L_j, l'insieme di large j-sequences, e la sequenza cliente c. Essa fornisce come output le candidate sequences di lunghezza $k + j$ contenute in c. L'intuizione che sta alla base di questa funzione è semplice: se s_k è una sequenza dell'insieme L_k e s_j è una sequenza dell'insieme L_j entrambe contenute in c e tali che non hanno itemsets in comune (cioè non siano sequenze del tipo $\langle 1\ \ 2\ \ 3\rangle$ e $\langle 3\ \ 4\ \ 5\rangle$ che presentano l'itemset $\langle 3\rangle$ in comune) la sequenza che si ottiene unendo le due precedenti, ovvero la sequenza $\langle s_k.s_j\rangle$ o anche la sequenza $\langle s_j.s_k\rangle$, è una sequenza candidata di lunghezza $k + j$.
In particolare l'algoritmo utilizza il primo meccanismo di generazione di sequenze candidate visto (apriori-generate) nella initialization phase e la funzione otf-generate nella forward phase. La ragione di ciò è che l'apriori-generate genera meno candidate rispetto all'otf-generate quando si genera l'insieme C_{k-1} a partire dall'insieme L_k.

Tuttavia ciò potrebbe non essere vero quando si cerca di trovare l'insieme L_{k+step} a partire dagli insiemi L_k ed L_{step}, cioè nel caso della forward phase. Inoltre se il numero di elementi dell'insieme L_k più il numero di elementi dell'insieme L_{step} ($|L_k| + |L_{step}|$) è minore del numero di elementi nell'insieme C_{k+step} ($|C_{k+step}|$) generato dall'apriori-generate, può essere più veloce trovare tutti i membri di L_k e L_{step} contenuti in una sequenza c rispetto al trovare tutti i membri di C_{k+step} contenuti in c, cioè la seconda funzione è più conveniente in termini di tempo computazionale. Ritornando nuovamente alla forward phase dell'algoritmo in questione, X è proprio l'insieme di sequenze di lunghezza $k + step$ con $k = step$, $2step$, $3step$, ... contenute in ogni iterazione in ciascuna sequenza cliente c considerata. Successivamente l'algoritmo incrementa di una unità il supporto di ogni sequenza $x \in X$ all'interno dell'insieme C_{k+step}. In particolare l'algoritmo aggiunge quella particolare sequenza x all'insieme C_{k+step} nel caso questa non sia già presente nell'insieme. Da qui si ottiene l'insieme di large sequences di lunghezza $k + step$ indicato con L_{k+step} dato dalle candidate sequences in C_{k+step} aventi il supporto minimo richiesto.
Vi è inoltre una intermediate phase con un ciclo for che pemette il calcolo dei rimanenti insiemi C_k. Tali insiemi vengono ottenuti dagli insiemi L_{k-1} se questi sono noti o dagli insiemi C_{k-1} in caso contrario. La backward phase finale ricalca nella sostanza il meccanismo della medesima backward phase nell'algoritmo AprioriSome. In definitiva, come nell'AprioriSome, durante la fase backward vengono conteggiate quelle sequenze con i valori di lunghezza saltati durante la forward phase. Ma a differenza del primo algoritmo le candidate sequences sono generate nella ulteriore intermediate phase e non nella forward phase.
Riprendiamo nuovamente il database di sequenze trasformate D_t di pag. 60 e descriviamo brevemente il funzionamento di questo algoritmo. Nella initialization phase, considerando un passo pari a 2, l'algoritmo determina L_2 come nell'AprioriAll. Successivamente, nella forward phase, si ottiene l'insieme C_4 costituito dalle sequenze $\langle 1\ 2\ 3\ 4\rangle$ e $\langle 1\ 3\ 4\ 5\rangle$ con supporto rispettivamente 2 ed 1. Ma di queste sequenze solo la $\langle 1\ 2\ 3\ 4\rangle$, avendo il supporto minimo richiesto, è una large sequence. Nel passo successivo l'insieme C_6 risulta vuoto, per cui si passa alla successiva intermediate phase nella quale si genera C_3 da L_2 e C_5 da L_4. Poiché l'insieme C_5 risulta vuoto nella backward phase si conteggia il solo insieme C_3 per ottenere L_3. L'output finale è ovviamente analogo a quello degli algoritmi precedenti, data la semplicità

dell'esempio.