

SEQUENTIAL PATTERNS

Lo scopo di questi algoritmi è quello di estrarre le più frequenti sottosequenze di un DB sequenziale.

Un DB sequenziale ha al suo interno dati salvati in un certo ordine temporale, anche senza definire rigorosamente il concetto di tempo.

Esempio:

Un DB di un negozio che mostra per ogni cliente delle transazioni con i prodotti che quel cliente ha acquistato ogni settimana per un mese:

[T₁: <(PANZ, LATO), (PANZ, LATO, zuc), (LATO), (TSA, zuc)>]
[T₂: <(PANZ), (zucc, TSA)>]

Dove vengono mostrati i clienti T₁ e T₂.

ITEMSET

Un **itemset** è un set di oggetti disegnati a partire da $I = (i_1 \dots i_n)$. Possono anche essere determinati eventi.

- 1) Dato un DB sequenziale
- 2) Dato lo threshold **min-sup**
- 3) Dato l'insieme di oggetti unici $I = (i_1 \dots i_n)$

Dobbiamo trovare tutte le sequenze frequenti S del DB formate dagli elementi di I.

ORDINE LESSICOGRAFICO

Gli elementi di una sequenza sono ordinati in ordine lessicografico. Il che vuol dire, che se assumiamo l'itemset $t = \{i_1, \dots, i_k\}$: $i_1 \leq i_2 \leq \dots \leq i_k$ e l'itemset $t' = \{j_1, \dots, j_l\}$: $j_1 \leq j_2 \leq \dots \leq j_l$ dove il simbolo \leq indica che l'elemento che segue **occorre temporaneamente dopo** (i_1 viene prima di i_2).

Diciamo che $t < t'$, o che t è **lessicograficamente minore** di t' se vale UNA delle seguenti:

- 1) $\exists h \in \mathbb{R} : \emptyset \leq h \leq \min\{k, l\}$ tale per cui abbiamo che $i_r = j_r$ con $r < h$ e $i_h < j_h$
- 2) $k < l$ e $i_1 = j_1, i_2 = j_2, \dots, i_k = j_k$

Esempio

$$(abc) < (abec)$$

Scegliamo h : $0 < h < \min(k=3, l=4)$. Diciamo che $h=3$.

$\forall r < 3$, quindi $r=1$ e $r=2$ abbiamo effettivamente che $i_r = j_r$: $i_1 = j_1 = a$, $i_2 = j_2 = b$.

Abbiamo poi che $i_3 < j_3 \rightarrow c < e$. Torna l'ordine $abc < abec$.

SEQUENZA

Una sequenza di k elementi è chiamata **k -sequence**.
Un item può apparire **una sola volta** in un itemset,
ma può stare in più itemset diversi della stessa
sequenza. (Seq = $\langle (a,b), a, b \in (a,b,c) \rangle$ dove l'item
“a” compare in 3 itemset diversi della sequenza)

Una sequenza $a = \langle e_{i_1} \dots e_{i_m} \rangle$ è una **sotto
sequenza** di un'altra sequenza $b = \langle e_1 \dots e_n \rangle$ se
esistono gli interi $i_1 < i_2 < \dots < i_m$ e tutti gli
eventi $e_{i_j} \in a$ (i_j : dove j sta a indicare i numeri da
 1 a m) e $e_{i_j} \in b$ e $i_1 \leq 1$ e $i_m \leq n$ tale
per cui $e_{i_j} \subseteq e_j$.

In parole povere, prendo gli elementi di a e b e
vedo se gli elementi di a appaiono in b nello
stesso ordine.

Esempio

$$a = \langle A, B, C \rangle$$

$$b = \langle X, Y, Z, A, B, C, M, N \rangle$$

Abbiamo che gli indici in a sono $i_1=1, i_2=2$ e
 $i_3=3$. Mentre gli indici per gli stessi valori in b
sono 4, 5 e 6.

Sono rispettati i vincoli $i_1 \leq 4, i_2 \leq 5, i_3 \leq 6$.

MAXIMUM SEQUENTIAL PATTERN

È una sequenza che non è sotto-sequenza di nessun
altro.

SEQUENZA IN ORDINE LESSICOGRAFICO

Supponiamo di avere un ordinamento lessicografico definito degli elementi del DB. Questo ordine è anche esteso alle sequenze e sottosequenze, definendo $S_a \leq S_b$, ovvero S_a sottosequenza di S_b .

Lessicographic Tree

Questo albero ha NULL alla radice.

Viene costruito ricorsivamente: ogni nodo m del tree ha come figli tutti i nodi che rispettano la condizione $m \leq m'$: ovvero che il padre nell'albero è una sotto-seguenza dei figli.

Oltre a questa condizione ce n'è anche una seconda che dice: $\forall m \in \text{Tree } m' \leq m \Rightarrow m \leq m$.

Una sequenza viene estesa aggiungendo o un item o un intero itemset alla fine.

SUPPORTO DI UNA SEQUENZA

È dato dal numero di sequenze che hanno come sotto-seguenza la sequenza sotto-analisi, diviso il numero totale di sequenze in D.

Una sequenza è frequente se questo numero supera una threshold δ indicata dall'utente.

FREQUENT CLOSE SEQUENCES

Prende una sequenza frequente S_a , e' detta frequent close sequence se non esiste nessuna supersequenza di S_a con lo stesso supporto.

FASI DSGCI ALGORITMI

1) FASE DI ORDINAMENTO

Dove semplicemente i dati vengono ordinati. Per esempio un DB con clienti e data dell'acquisto può essere ordinato in prima fase per l'ID del cliente e poi per data d'acquisto.

Convertire il DB in un DB sequenziale.

2) Cercare tutti i Large Itemset: LItemset.

Ovvero quegli itemset che hanno supporto superiore a una certa soglia. Il supporto è misurato considerando il DB diviso per clienti nell'esempio.

3) FASE DI TRASFORMAZIONE

Trasformare le transazioni non considerando per ogni transazione gli elementi non LItemset. Eliminare le eventuali transazioni che non hanno nessun LItemset.

4) FASE SEQUENZA

Cercare le sequenze da analizzare sulla base dei LItemset.

- Count - All: Considera tutte le sequenze frequenti anche quelle non massime.

- Count - Some: Cerca di non considerare le sequenze frequenti NON massime.

5) FASE MASSIMA

Trovare le sequenze frequenti massime.

APRIORI-ALL

- 1) Trovare il set L_3 con i 3-Itemset in D.
- 2) Ciclaree costruendo C_k tramite L_{k-1} .

Per cercare C_k faccio join fra L_{k-1} con se stesso e vedo a cercare tutte le sequenze in L_{k-1} che condividono un prefisso grande ($k-2$), poi le unisco:

Esempio

$$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 & 2 & 4 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 2 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 2 & 4 & 3 \end{pmatrix}$$

Elimino poi da C_k tutte le sequenze che non presentano sotto-seguenza in L_{k-1}

Esempio

$$L_3 = (123)(234)(124)(134)(135)$$

In C_4 ho la sequenza (1243) che presenta sotto-seguenza (243) non presente in L_3 . La elimino.

- 4) Trovo L_k considerando le sole sequenze di C_k che soddisfano il min-sup.

- 5) Output = $\bigcup_k L_k$

APRIORI SOME

Apriori-Some si divide in due fasi

1) FORWARD PHASE

Dove vengono trovate le grandi sequenze di una data dimensione (δ_S : sequenze di lunghezza 1, 2, 4, 6)

2) BACKWARD PHASE

Dove vengono analizzate le lunghe sequenze di lunghezza non analizzata nella fase precedente (δ_S lunghezza 3, 5)

Per capire quale lunghezza analizzare APRIORI-ALL usa una funzione $\text{next}(k)$ che sulla base della lunghezza dell'iterazione precedente. Difatti valuta:

$$\frac{|L_{k-1}|}{|C_{k-1}|}$$

Più questa grandezza si avvicina ad uno, più viene restituito un k più grande.

Creazione di C_k nello phase di forward.

Questa volta abbiamo due scenari, il caso in cui abbiamo L_{k-1} e procediamo come sempre.

Nel caso in cui la funzione $\text{next}(k-1)$ ha fatto sì di sollevare la eccezione di L_{k-1} , allora possiamo generare C_k partendo da C_{k-1} , poiché tanto vale la proprietà che $L_{k-1} \subseteq C_{k-1}$.

Da notare che i vari C_k vengono generati tutti.

Fase di Backward nel dettaglio.

Qui per ogni C_k da cui non è stato ricavato L_k , ovvero per tutti i K saltati nella fase di forward:

- Eliminare in C_k le sequenze che sono contenute negli L_i con $i > k$. Questo perché sono state già analizzate.
- Eliminare le sequenze rimaste e trovare quelle che soddisfano il supporto.

Se invece analizzo un L_k trovato nella fase di forward elimino le sue sequenze se contenute in un L_i con $i > k$.

Esempio

Dataset:

{1 5}	{2}	{3}	{4}
{1}	{3}	{4}	{3 5}
{1}	{2}	{3}	{4}
{1}	{3}	{5}	
{4}	{5}		

Forward Phase:
 $\text{next}(k) = 2k$
 $\text{minsup} = 2$

L_1	
1-Sequences	Support
{1}	4
{2}	2
{3}	4
{4}	4
{5}	4

L_2	
2-Sequences	Support
{1 2}	2
{1 3}	4
{1 4}	3
{1 5}	3
{2 3}	2
{2 4}	2
{3 4}	3
{3 5}	2
{4 5}	2

C_3	
3-Sequences	
{1 2 3}	
{1 2 4}	
{1 2 5}	
{1 3 4}	
{1 3 5}	
{2 3 4}	
{3 4 5}	

L_4	
4-Sequences	Support
{1 2 3 4}	2

Backward Phase:

L_4	
4-Sequences	Support
{1 2 3 4}	2

Quelle eliminate in rosso non sono presenti nel DB

C_3	
3-Sequences	
{1 2 3}	
{1 2 4}	
{1 2 5}	
{1 3 4}	
{1 3 5}	
{2 3 4}	
{3 4 5}	

L_2	
2-Sequences	Support
{1 2}	2
{1 3}	4
{1 4}	3
{1 5}	3
{2 3}	2
{2 4}	2
{3 4}	3
{3 5}	2
{4 5}	2

L_1	
1-Sequences	Support
{1}	4
{2}	2
{3}	4
{4}	4
{5}	4

L_3	
1-Sequences	
{1 3 5}	

OUTPUT FINAL

$\langle 1 2 3 4 \rangle \cup \langle 1 3 5 \rangle \cup \langle 4 5 \rangle$

APRIORI DYNAMIC SONG

Questo algoritmo si basa sul cercare gli L_k per K multiplo di un numero α di ingresso. Per farlo viene diviso in 4 parti:

FASE INIZIALE

Praticamente uguale a Apriori All, dove genera i primi $\alpha - L_k$. Questo perché dobbiamo necessariamente costruire L_d per analizzare i vari L_{kd} con $k = 1, \dots, n$.

FASE DI FORWARD

Vengono creati gli altri L_k per i vari K multipli di α .

L_{k+d} ricavato da L_k e L_d .

Di fatti posso cercare gli L_{k+d} usando L_d .

Esempio: Da L_3 e L_6 posso generare L_9 .

OTF - GENERATO

Dobbiamo usare un'altro tipo di join.

Consideriamo di fare join fra due sequenze se la fine della sequenza a è più piccola dell'inizio della sequenza b:

Esempio

Consideriamo le sequenze:

a: < 1 2 >

b: < 1 3 >

c: < 1 4 >

d: < 2 3 >

e: < 2 4 >

f: < 3 4 >

Condizione di join: $s_1.end < s_2.start$

$a \bowtie b$: $a.end < b.start ? 2 < 1 \text{ NO}$

$a \bowtie c$: $2 < 1 \text{ NO}$

Continuando, l'unico join che rispecchia la condizione è: $a \bowtie f$ con $2 < 3$.

$C_4 = C_2 \bowtie C_2 = a \bowtie f = < 1 2 3 4 >$

INTERMEDIATE PHASE

Qui vengono creati i vari C_k non creati nella fase di forward. I C_k sono costituiti o da C_{k-1} o da C_{k-2} .

BACKWARD PHASE

Identica ad APRIORI-SOLVE