Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Un toolbox ottimizzato per la scoperta di itemset frequenti e statisticamente significativi

An optimized toolbox for the discovery of statistically significant frequent itemsets

Laureando:
Federico Menegon

Relatori:
Geppino Pucci
Andrea Pietracaprina

23 ottobre 2012

Alla mia famiglia e ad Eleonora,
con amore e riconoscenza

Sommario

Il lavoro sviluppato in questo elaborato riguarda la realizzazione di un toolbox software per la scoperta di itemset statisticamente significativi all'interno di un dataset assegnato. Il toolbox permette di identificare quegli itemset il cui supporto si discosta significativamente da ciò che ci si potrebbe aspettare in un modello random del dataset stesso e quindi tali che il valore del supporto associato non risulti essere dovuto al caso. Il formato con cui i dataset di ingresso devono essere memorizzati è quello imposto dal workshop sul *Frequent Itemset Mining Implementations (FIMI)* del 2003. Il toolbox è stato sviluppato utilizzando *C* come linguaggio di programmazione. Questa scelta è stata preferita per consentire una gestione efficiente della memoria e per poter ottimizzare ulteriormente il codice in fase di compilazione, contenendo in questo modo i tempi di esecuzione del programma anche nel caso di dataset in ingresso di grosse dimensioni. La particolarità del toolbox realizzato riguarda l'implementazione al suo interno di una nuova strategia che consente di migliorare le prestazioni complessive, intese come diminuzione dei tempi d'esecuzione dell'algoritmo, senza che ciò però possa influire sulla qualità della soluzione restituita. Il programma è stato sviluppato avendo tra gli obiettivi quello della modularità, agevolando in questo modo l'estensione o la modifica in futuro di porzioni del codice, e quello del miglioramento delle prestazioni e aggiunta di funzionalità rispetto ad un tool software precedentemente creato per lo stesso scopo. Infine il toolbox è stato realizzato in maniera da facilitare la configurazione dei parametri d'ingresso tra cui appunto le misure di qualità inerenti l'insieme delle soluzioni restituite al termine dell'esecuzione.

Indice

Capitolo 1

Introduzione

Gli ultimi anni sono stati caratterizzati da un aumento vertiginoso della quantità di dati disponibile in formato digitale. Questo aumento ha riguardato numerosi ambiti, come ad esempio quello commerciale, sociale e scientifico.

Da una ricerca condotta di recente dall'*International Data Corporation* (IDC), è emerso che la quantità di file memorizzati su supporti digitali ha raggiunto nel 2011 il traguardo di 1.8 *zettabyte* ($1.8 * 10^{18}$Byte) che, più chiaramente, equivalgono ad oltre mille miliardi di gigabyte.

> *Il numero di bit nell'universo digitale è pari quasi quanto il numero di stelle nel nostro universo fisico.*

In particolare, negli ultimi dieci anni la quantità di dati memorizzati in formato digitale è cresciuta del 900% circa, si pensi infatti che, limitatamente al 2005, la quantità di file stimata era approssimativamente pari a 0.1 *zettabyte*. Dal 2005 ad oggi il costo per creare, gestire e memorizzare informazioni si è ridotto dell'80%. Nella prossima decade si prevede che la quantità di file digitali crescerà del 7500%, raggiungendo i 7.9 *zettabyte* totali nel 2015.

In questo scenario assume un ruolo di rilevanza il *Data Mining*, ovvero la scienza che si occupa di estrarre informazione utile, nuova ed interessante da grandi moli di dati. Negli ultimi anni sono stati condotti molti passi in avanti per quanto riguarda l'estrazione di *pattern* frequenti da grandi moli di dati realizzando diversi tool software che permettono di raggiungere tale scopo. Lo stesso però non si può dire per quello che riguarda la valutazione della significatività statistica dei *pattern* estratti. Solo di recente è stato presentato un nuovo metodo per la scoperta di *pattern* statisticamente significativi all'interno di grandi moli di dati che potrebbe portare ad una inversione di tendenza.

Il presente documento descrive un toolbox sviluppato in linguaggio *C*, il cui obiettivo consiste nello scoprire in maniera efficiente ed efficace gli itemset di taglia k statisticamente significativi all'interno di un dataset reale D. Il toolbox consente di valutare la significatività statistica di un itemset, confrontando il supporto di quest'ultimo con quello atteso in un dataset random appartenente allo spazio di probabilità di D. Per raggiungere tale scopo l'algoritmo utilizza due differenti approcci:

- Il primo prevede l'applicazione del metodo standard di *multi-comparison test* su un insieme di *null hypothesis*, applicando la tecnica descritta in (Benjamini e Yekutieli, 2001) per il controllo di *False Discovery Rate*.

- Il secondo invece applica il recente metodo descritto in (Kirsch *e altri*, 2009) per stabilire una soglia di supporto s^* oltre la quale tutti i k-itemset frequenti, con supporto almeno pari a tale soglia, possono essere considerati significativi con buona confidenza e con *False Discovery Rate* limitato.

Il maggiore contributo che viene dato in questo documento, per quanto concerne la scoperta di itemset frequenti e statisticamente significativi, riguarda lo sviluppo di una nuova strategia attraverso la quale è possibile ottenere un aumento significativo dell'efficienza del toolbox realizzato, ovvero una diminuzione dei tempi di esecuzione di quest'ultimo, garantendo allo stesso tempo che l'insieme di itemset statisticamente significativi restituiti soddisfi ancora i parametri di qualità imposti. La nuova strategia sviluppata consiste nella riduzione del numero di transazioni necessarie alla costruzione di un modello random del dataset in ingresso al toolbox dato che tale operazione si è dimostrata essere la più onerosa, dal punto di vista computazione, tra quelle presenti all'interno dell'algoritmo implementato. Verrà dimostrato empiricamente come i vincoli di confidenza e *False Discovery Rate* imposti sull'insieme di itemset statisticamente significativi restituito rimangano validi anche dopo l'applicazione di tale strategia.

Il documento, oltre a questo capitolo introduttivo, è composto da altri quattro capitoli strutturati nel seguente modo:

- Nel Capitolo 2 è riportata la descrizione della teoria alla base del toolbox realizzato, ossia l'approssimazione di Poisson, la simulazione di Monte Carlo, gli approcci utilizzati per la scoperta di itemset significativi e l'algoritmo Apriori per la scoperta di itemset frequenti;

- Nel Capitolo 3 è riportata la descrizione dettagliata dell'implementazione del toolbox e delle strutture dati utilizzate, nonché la descrizione

di come impostare i parametri d'ingresso e di come interpretare i valori in uscita;

- Nel Capitolo 4 sono descritti e discussi i test condotti in fase sperimentale al fine di confrontare i due differenti approcci utilizzati per l'identificazione di itemset significativi, valutare i vantaggi e gli svantaggi prodotti dalla nuova strategia proposta per il miglioramento dell'efficienza e infine dimostrare i progressi fatti rispetto ad un precedente tool realizzato con il medesimo scopo;

- Nel Capitolo 5 sono tratte le conclusioni del lavoro di tesi e presentati alcuni spunti per sviluppi futuri.

Nel documento sono presenti inoltre due appendici necessarie per poter utilizzare e sviluppare ulteriormente il toolbox realizzato.

- Nella Appendice A sono descritte esaustivamente alcune porzioni di codice ritenute fondamentali per la corretta esecuzione del toolbox sviluppato.

- Nella Appendice B è riportato il manuale utente del toolbox nel quale si spiega come installare il toolbox, come configurarlo e come interpretare i risultati ottenuti.

Capitolo 2

Cenni teorici

Il toolbox sviluppato è stato costruito su solide basi teoriche. In questo capitolo vengono illustrati i principali risultati teorici su cui basa la metodologia utilizzata per la scoperta di itemset statisticamente significativi.

2.1 Itemset frequenti

La scoperta di itemset frequenti all'interno di un multiinsieme di transazioni rappresenta uno dei problemi più importanti del *Data Mining*, ossia la scienza che si occupa dell'estrazione di informazione utile, nuova ed interessante da grandi moli di dati. Con il termine dataset si intende un multiinsieme di transazioni $D = \{t_1, t_2, \ldots, t_{|D|}\}$ in cui ciascuna transazione t_j è un sottoinsieme di item appartenenti all'insieme $I = \{i_i, i_2, \ldots, i_n\}$. Con il termine itemset invece si intende una qualunque collezione X di uno o più item appartenenti all'insieme I, più formalmente $X \subseteq I$. Se il generico itemset X è costituito da k item allora il valore della taglia di X coincide con k è X è definito anche con il nome di k-itemset. Un'importante proprietà degli itemset è data dal loro supporto il quale indica il numero di transazioni che contengono un particolare itemset X e cioè il numero di transazioni in cui X è sottoinsieme. Se consideriamo ad esempio il generico itemset $X \subseteq I$ ed un dataset $D = \{t_1, t_2, \ldots, t_{|D|}\}$, dove $t_j \subseteq I$ con $1 \leq j \leq |D|$, abbiamo che il supporto di X in D coincide con

$$s_D(X) = |\{j : 1 \leq j \leq |D| \wedge X \subseteq t_j\}|.$$

Per comprendere meglio quanto sopra esposto si consideri il seguente esempio.
ESEMPIO. Si consideri un negozio di alimentari il quale tiene traccia di tutti gli acquisti fatti dai propri clienti registrando in un database interno tutti gli scontrini emessi dal negozio.

Supponiamo che contenuto del database sia il seguente:

Tabella 2.1: Dataset di un alimentari

TID	**Items**
1	{Pane, Latte}
2	{ Pane, Biscotti, Acqua, Uova}
3	{Latte, Biscotti, Acqua, Caffè}
4	{Pane, Latte, Biscotti, Acqua}
5	{Pane, Latte, Biscotti Caffè}

allora in questo scenario il dataset è costituito da cinque transazioni, una per ogni riga della tabella, costruite sopra l'insieme di item

$$I = \{\text{Pane, Latte, Biscotti, Acqua, Uova, Caffè}\},$$

dove un generico itemset X è una qualunque collezione di uno o più item del dataset.

Il problema della scoperta di itemset frequenti è stato formalmente introdotto per la prima volta in (Agrawal *e altri*, 1993) e consiste in: dato un multiinsieme di transazioni $D = \{t_1, t_2, \ldots, t_{|D|}\}$ e dato un insieme I di n item per il quale ciascuna transazione t_j è un sottoinsieme, ricavare l'insieme di itemset frequenti $F(D, \sigma)$ per una data soglia minima di supporto σ, ossia restituire l'insieme di itemset con supporto almeno pari a σ in D.

Ritornando all'esempio precedente, estraendo dal dataset del negozio di alimentari gli itemset frequenti è possibile individuare delle correlazioni nascoste all'interno dei dati che permettono, ad esempio, di venire a conoscenza delle preferenze e delle abitudini d'acquisto dei clienti, le quali potrebbero rivelarsi utili per promuovere attività di marketing, per migliorare i rapporti con la clientela oppure per gestire efficientemente le scorte di magazzino. Quanto appena descritto è solo una delle numerose applicazioni in cui l'estrazione di pattern significativi dai dataset risulta valida per la scoperta di informazioni utili ma ancora nascoste all'interno di grandi moli di dati.

L'algoritmo Apriori, introdotto per la prima volta in (Agrawal *e altri*, 1993), permette di risolvere in maniera efficiente il problema appena descritto. Questo algoritmo, per limitare il numero di itemset da analizzare durante la ricerca, sfrutta la proprietà di antimonotonicità del supporto, che afferma che il supporto di un itemset non può essere superiore al supporto di uno qualsiasi dei suoi sottoinsiemi di item.

Le implicazioni dovute a tale proprietà sono fondamentalmente due:

- Se un generico itemset X non è frequente, allora un qualsiasi suo sovrainsieme Y non è frequente.
- Se X è frequente allora per ogni sottoinsieme Y di X si ha che Y è frequente.

L'algoritmo Apriori, descritto dal seguente pseudocodice, risulta essere molto efficiente in quanto opera al più $L + 1$ scansioni del dataset, dove L è la massima taglia di itemset (frequenti) scoperti.

```
Input: D = {t_1, t_2, ..., t_|D|}, |D|, I = {i_i, i_2, ..., i_n}, minsup
Output: F = {(X, s_D(X)/|D|) : X ⊆ I ∧ s_D(X) ≥ |D| * minsup}

    k ← 1
    F_1 ← {i ∈ I; s_D({i}) ≥ t * minsup}
    repeat
        k ← k + 1
        C_k ← APRIORI-GEN(F_{k-1})
        for each c ∈ C_k do s_D(c) ← 0
        for each transaction t ∈ D do
            C_t ← {c ∈ C_k : c ⊆ t}
            for each c ∈ C_t do s_D(c) ← s_D(c) + 1
        F_k ← {c ∈ C_k : s_D(c) ≥ |D| * minsup}
    until F_k = ∅
    return F = ⋃_{k≥1} F_k


APRIORI-GEN(F_{k-1}) :
    C_k ← ∅
    for each X, Y ∈ F_{k-1} s.t X ≠ Y ∧ X[1...k-2] = Y[1...k-2] do
        add X ∪ Y to C_k
    for each c ∈ C_k do
        for each Y ⊂ c s.t. |Y| = k - 1 ∧ Y ∉ F_{k-1} do
            remove c from C_k
    return C_k
```

In ingresso all'algoritmo Apriori si deve fornite il dataset $D = \{t_1, t_2, \ldots, t_{|D|}\}$, il numero di transazioni contenute in esso e l'insieme di item $I = \{i_1, i_2, \ldots, i_n\}$ su cui le transazioni del dataset D sono costruite. Inoltre si deve fornire il valore di frequenza *minsup* ovvero la percentuale minima di transazioni nelle

quali un generico itemset deve essere contenuto per poter essere considerato frequente. In output l'algoritmo restituisce tutti gli itemset frequenti scoperti assieme alla frequenza con cui tali itemset compaiono all'interno delle transazioni del dataset D.

Per comprendere come l'algoritmo Apriori funzioni è utile cominciare analizzando la procedura denominata *APRIORI-GEN*. Prima di procedere è necessario ipotizzare che le transazioni e gli itemset siano ordinati in accordo con un qualunque ordinamento arbitrario degli item. Lo scopo di *APRIORI-GEN* è quello di generare l'insieme di k-itemset candidati ad essere frequenti a partire dell'insieme F_{k-1} cioè l'insieme di itemset frequenti di taglia $k-1$. La procedura esegue nell'ordine le seguenti operazioni:

- Generazione dei k-itemset candidati, ovvero a partire dall'insieme di $k-1$-itemset frequenti rispetto la soglia minima *minsup* vengono costruiti tutti i k-itemset candidati ad essere frequenti. Per fare ciò si considera ciascuna coppia di itemset $<X, Y>$ appartenente all'insieme F_{k-1}, se $X \neq Y$ e il prefisso di lunghezza $k-2$ di X coincide con il prefisso anch'esso di lunghezza $k-2$ di Y, allora il k-itemset ottenuto dall'unione di X con Y viene inserito all'interno dell'insieme di itemset candidati C_k;

- Pruning dei candidati sulla base del principio di antimonotonicità del supporto, ovvero per ogni candidato c in C_k si verifica che tutti i suoi sottoinsiemi di taglia $k-1$ appartengano all'insieme F_{k-1} e che siano perciò frequenti.

APRIORI-GEN garantisce che:

- La generazione di itemset "inutili" venga evitata, ossia itemset per i quali "a priori" si può dire che saranno eliminati dall'operazione di pruning, perché non frequenti, non vengono generati;

- L'insieme di candidati C_k sia completo, ovvero che $F_{k-1} \subseteq C_k$ e quindi C_k contiene tutti gli itemset frequenti di taglia k.

- La generazione dei candidati sia efficiente, ovvero ciascun itemset viene generato una volta sola.

A questo punto è possibile iniziare con la descrizione vera e propria dell'algoritmo Apriori. Per prima cosa si inizializza a 1 il valore della variabile k la quale indica la taglia di itemset frequenti attualmente studiata. Dopo di che per ciascun item $i \in I$ si verifica se il supporto associato sia almeno pari a $|D| * minsup$, ovvero il minimo supporto richiesto ad un itemset per

essere considerato frequente, e in caso positivo si inserisce l'item i all'interno dell'insieme F_1. Successivamente si entra in un ciclo *while* che terminerà solo quando l'insieme di k-itemset frequenti calcolato sarà vuoto. Il ciclo contiene le seguenti operazioni in sequenza:

- Per prima cosa all'interno del ciclo *while* viene incrementato il valore di k per indicare la nuova taglia di itemset frequenti che si vuole studiare;
- Si applica la procedura *APRIORI_GEN* all'insieme di itemset frequenti F_{k-1} salvando il risultato nell'insieme C_k;
- Per ciascun candidato $c \in C_k$ si associa un supporto iniziale $s_D(c)$ nullo;
- A questo punto si applica la procedura per il conteggio del supporto di ciascun candidato $c \in C_k$. Tale procedura prevede che, per ciascuna transazione $t \in D$, si calcoli l'insieme C_t ovvero l'insieme di candidati di taglia k contenuti in t. Al fine di ridurre il numero di confronti necessari per il calcolo dell'insieme di candidati contenuti in ciascuna delle transazioni del dataset D, è utile memorizzare l'insieme di itemset candidati C_k in una struttura di tipo Hash Tree. Una struttura di tipo Hash Tree è definita in funzione di tre parametri:
  - τ, taglia massima per un sottoinsieme di candidati associati ad una foglia;
  - n, numero massimo di figli che un nodo interno può avere (arietà);
  - h, funzione hash che mappa gli item in interi tra 0 e $n-1$.

  Dato C_k e data una terna (τ, n, h), un Hash Tree per C_k soddisfa le seguenti proprietà:
  - Gli archi che collegano un nodo interno ai suoi figli sono etichettati con valori distinti nell'intervallo $[0, n-1]$;
  - Per ogni foglia V a profondità p, se $l_1, l_2, \ldots, l_p$ sono le etichette incontrate nel cammino dalla radice alla foglia V, allora si ha che il sottoinsieme dei candidati associati alla foglia V è univocamente determinato nel seguente modo:

$$C^V \leftarrow \{c \in C_k : h(c[i]) = l_i, 1 \leq i \leq p\}$$

  A questo punto si costruisce l'insieme di candidati $C'_k \subseteq C_k$ ottenuto dall'unione di alcune foglie dell'albero, più precisamente dall'unione dei sottoinsiemi di candidati mappati nelle foglie raggiunte applicando la

transazione t alla funzione di hash h. Una volta ottenuto C'_k si ricava l'insieme $C_t \leftarrow \{c \in C'_k : c \subseteq t\}$ e si aggiorna il supporto per tutti i candidati appartenenti a C_t.

- A questo punto è possibile procedere a calcolo degli itemset frequenti verificando quali tra i candidati contenuti nell'insieme C_k abbiano supporto almeno pari a $|D| * minsup$ e salvando il risultato nell'insieme F_k.

- Se F_k è non nullo allora si passa all'iterazione successiva del ciclo *while*, altrimenti il ciclo termina qui.

Una volta che il ciclo *while* è terminato è possibile ritornare gli insiemi di itemset frequenti calcolati.

Dalla seguente descrizione, è possibile notare come i fattori che influenzano maggiormente l'algoritmo Apriori sono:

La scelta della soglia di supporto minimo. L'abbassamento del valore della soglia *minsup* infatti causa un maggior numero di itemset candidati (che si traduce in un Hash Tree più grande) e potenzialmente quindi un aumento della lunghezza massima degli itemset frequenti (che si traduce in un maggior numero di scansioni necessarie nel dataset).

Numero di transazioni nel dataset. Dato che l'algoritmo Apriori esegue più scansioni del dataset, il tempo di esecuzione dell'algoritmo risentirà anche del numero maggiore di transazioni presenti.

Lunghezza media delle transazioni. Transazioni più lunghe fanno sì che la lunghezza media degli itemset frequenti aumenti e quindi viene richiesto maggiore spazio per l'Hash Tree e più scansioni del dataset.

Numero di item. Maggiore è il numero di item maggiore sarà il tempo richiesto per il conteggio del supporto.

2.2 Itemset significativi

Un generico itemset X viene considerato statisticamente significativo se risulta improbabile che il valore del supporto dello stesso all'interno del dataset D sia dovuto al caso, ovvero se tale valore devia significativamente da ciò che ci si aspetta in un modello random del dataset. Di conseguenza, se l'itemset X risulta frequente all'interno del dataset D non significa che questo debba essere considerato necessariamente interessante.

Fra le questioni ancora aperte che tutt'oggi interessano la scena del *Data Mining* il problema della scoperta di itemset statisticamente significativi è sicuramente una di quelle più dibattute.

Dato un generico dataset $D = \{t_1, t_2, \ldots, t_{|D|}\}$ di $|D|$ transazioni costruite su un insieme $I = \{i_1, i_2, \ldots, i_n\}$ di n item, il problema della scoperta di itemset statisticamente significativi richiede di individuare l'insieme di itemset tali per cui il supporto all'interno del dataset D è significativamente maggiore del supporto atteso in un generico dataset random $\hat{D}$ appartenente allo spazio di probabilità di D. Lo spazio di probabilità di D è definito come l'insieme di dataset in cui ciascun item $i \in I$ compare all'interno delle $|D|$ transazioni di ciascun dataset con frequenza pari a $s_D(i)/|D|$.

La procedura classica per la risoluzione di questo problema prevede che l'utente decida arbitrariamente, salvo particolari conoscenze sul dataset di riferimento, il valore di supporto σ oltre il quale ritenere significativi gli itemset frequenti rispetto tale soglia. È evidente però che in questo modo non vengono date garanzie sulla correttezza della soluzione restituita (Tan *e altri*, 2006).

Per risolvere il problema, in (Kirsch *e altri*, 2009) è stato proposto un nuovo approccio il quale attraverso il calcolo di una soglia si supporto s^* garantisce, in senso statistico, che la famiglia di itemset frequenti rispetto a tale soglia sia significativa con livello di confidenza $1 - \alpha$ e con *FDR* al più β.

La teoria a sostegno di questo nuovo approccio è basata sul metodo di approssimazione di Poisson, sviluppato in (Arratia *e altri*, 1990), e sul metodo di simulazione di Monte Carlo. Si consideri un dataset $D = \{t_1, t_2, \ldots, t_{|D|}\}$ di $|D|$ transazioni costruite su un insieme $I = \{i_1, i_2, \ldots, i_n\}$ di n item dove:

$\hat{D}$ è un dataset random appartenente allo spazio di probabilità associato a D;

$Q_{k,s}$ è il numero osservato di k-itemset con supporto almeno s in D;

$\hat{Q}_{k,s}$ è la corrispondente variabile aleatoria per $\hat{D}$.

La simulazione di Monte Carlo consente di studiare le probabilità empiriche associate agli itemset del modello random del dataset D al fine di permettere il calcolo, attraverso il metodo di approssimazione di Poisson, di una soglia di supporto s_{min} tale che, per tutti i valori di supporto s maggiori di s_{min}, la distribuzione della variabile aleatoria $\hat{Q}_{k,s}$ è bene approssimata da una distribuzione di Poisson con il medesimo valore atteso. Il nuovo approccio presentato permette di individuare gli itemset statisticamente significativi tra quelli il cui supporto è almeno pari a s_{min}, cioè solo tra quegli itemset il cui supporto è relativamente alto all'interno del dataset. Questa scelta è stata preferita dato che:

- Come evidenziato nella precedente sezione, diminuendo la soglia minima di frequenza *minsup* il numero di candidati da calcolare ad ogni iterazione dell'algoritmo Apriori cresce considerevolmente comportando un degrado nelle prestazioni dell'algoritmo.

- Il numero di itemset frequenti da esaminare, nel caso si studiasse la significatività anche per itemset con basso supporto, sarebbe esponenziale comportando un aumento dei costi computazionali non indifferente.

2.3 Approssimazione di Poisson

Il metodo di approssimazione di Poisson sviluppato in (Arratia *e altri*, 1990) è un potente strumento statistico utilizzato per limitare con una distribuzione di Poisson l'errore dovuto all'approssimazione delle probabilità associate ad una sequenza di eventi. Per applicare tale metodo al caso della scoperta di k-itemset significativi è necessario fissare i parametri k e s e definire una collezione di $\binom{n}{k}$ variabili aleatorie di Bernoulli $\{Z_{X,s} : X \subset I, |X| = k\}$ tali che

- $Z_{X,s} = 1$ se $s_{\hat{D}}(X) \geq s$;

- $Z_{X,s} = 0$ altrimenti.

Per ogni k-itemset X si definisce:

- $p_X = \mathbf{Pr}(Z_{X,s} = 1)$ come la probabilità empirica che l'itemset X abbia supporto almeno s in un generico dataset random;

- l'*insieme dei vicini* di X come

$$I(X) = \{X' : X \cap X' \neq \emptyset, |X'| = |X|\};$$

- $p_{XY} = \mathbf{Pr}(Z_{X,s} = 1 \wedge \mathbf{Pr}(Z_{Y,s} = 1)$ come la probabilità empirica che gli itemset X e Y abbiano entrambi supporto almeno s in un generico dataset random. Si noti che se $Y \notin I(X)$ allora le variabili $Z_{Y,s}$ e $Z_{X,s}$ sono indipendenti e di conseguenza $p_{XY} = 0$.

La validità del metodo di approssimazione di Poisson utilizzato nell'approccio sviluppato in (Kirsch *e altri*, 2009) per il calcolo di itemset significativi è data dal seguente teorema ottenuto adattando al caso della ricerca di itemset statisticamente significativi il *Teorema 1* esposto in (Arratia *e altri*, 1990). Si noti che

$$\hat{Q}_{k,s} = \sum_{X:|X|=k} Z_{X,s}.$$

Teorema. *Sia U una variabile aleatoria di Poisson tale che $\mathbf{E}[U] = \mathbf{E}[\hat{Q}_{k,s}] = \lambda < \infty$. La variazione di distanza fra le distribuzioni $L(\hat{Q}_{k,s})$ di $\hat{Q}_{k,s}$ e $L(U)$ di U è regolata dalla seguente equazione:*

$$||L(\hat{Q}_{k,s}) - L(U)|| = \sup_A |\mathbf{Pr}(\hat{Q}_{k,s} \in A) - \mathbf{Pr}(U \in A)| \leq b_1 + b_2$$

dove

$$b_1 = \sum_{X:|X|=k} \sum_{Y \in I(X)} p_X p_Y$$

e

$$b_2 = \sum_{X:|X|=k} \sum_{X \neq Y \in I(X)} p_{XY}.$$

Il teorema consente perciò di calcolare l'errore dovuto all'approssimazione della distribuzione della variabile aleatoria $\hat{Q}_{k,s}$ con una variabile aleatoria di Poisson U, tale che $\mathbf{E}[U] = \mathbf{E}[\hat{Q}_{k,s}] = \lambda < \infty$, attraverso il calcolo dei coefficienti b_1 e b_2 per un dato valore di supporto s.

È possibile notare che per valori di supporto s crescenti entrambe le quantità b_1 e b_2 diminuiscono. Se si fissa un limite superiore alla somma $b_1 + b_2$ è possibile, aumentando il supporto, ottenere una soglia di supporto per la quale l'errore di approssimazione sia inferiore a tale limite. La simulazione di Monte Carlo, descritta nel paragrafo seguente, si pone come obiettivo proprio quello di calcolare una soglia di supporto s_{min} per la quale l'errore commesso con l'approssimazione della variabile aleatoria $\hat{Q}_{k,s_{min}}$ sia inferiore a un limite ϵ fissato.

2.4 Simulazione di Monte Carlo

Dal paragrafo precedente si evince come il calcolo dei coefficienti b_1 e b_2 dipenda dal valore delle probabilità empiriche p_X e p_{XY}. L'approccio sviluppato in (Kirsch *e altri*, 2009) utilizza la simulazione di Monte Carlo per il calcolo di tali probabilità ed il calcolo di una soglia di supporto s_{min} oltre la quale l'errore dovuto all'approssimazione della distribuzione di $\hat{Q}_{k,s}$ con una variabile aleatoria di Poisson con il medesimo valore atteso sia inferiore ad un coefficiente ϵ fissato. Formalmente:

$$s_{min} = \min\{s \geq 1 : b_1(s) + b_2(s) \leq \epsilon\}, \text{ con } 0 < \epsilon < 1.$$

Dal precedente teorema segue che, per ogni valore di supporto $s \geq s_{min}$, il numero di k-itemset con supporto almeno pari ad s è bene approssimato da una variabile aleatoria di Poisson con lo stesso valore atteso quando ϵ è molto piccolo (0,01 ad esempio).

La simulazione di Monte Carlo opera nella seguente maniera: per una data configurazione di frequenze di item e numero di transazioni, sia $\tilde{s}$ il massimo supporto atteso di un k-itemset all'interno di un generico dataset random $\hat{D}$ calcolato come prodotto delle k frequenze di item più elevate. Ragionevolmente il valore di $b_1(\tilde{s})$ è alquanto considerevole, quindi ha senso cercare un valore per s_{min} più grande di $\tilde{s}$. Per questo motivo si generano Δ dataset random e per ciascuno di essi si identificano i k-itemset frequenti con supporto almeno pari ad $\tilde{s}$. Sia W l'insieme di itemset estratti in questa maniera da tutti i dataset random generati, per ogni valore di supporto $s \geq \tilde{s}$ si stimano i valori di $b_1(s)$ e $b_2(s)$ calcolando per ciascun itemset $X \in W$ la probabilità empirica p_X dell'evento $Z_{X,s} = 1$ e, per ogni coppia di itemset $X, Y \in W$, con $X \cap Y \neq \emptyset$ e $X \neq Y$, la probabilità empirica $p_{X,Y}$ dell'evento $Z_{X,s}Z_{Y,s} = 1$. Si noti che per ogni itemset non presente in W tali probabilità sono nulle. Se al termine di questa procedura dovesse risultare che $b_1(\tilde{s}) + b_2(\tilde{s}) \leq \epsilon/4$ allora si ha che:

$$\hat{s}_{min} = \min\{s : b_1(s) + b_2(s), s \geq \tilde{s}\}.$$

Nel caso in cui $b_1(\tilde{s}) + b_2(\tilde{s}) > \epsilon/4$ si ripete la procedura sopra esposta partendo da $\tilde{s}/2$.

La simulazione di Monte Carlo consente quindi di stimare i valori di b_1 e b_2 al variare del supporto s e calcolare perciò la soglia di supporto s_{min}.

Viene ora fornito lo pseudocodice relativo la simulazione di Monte Carlo che sintetizza fondamentalmente quanto appena esposto:

`Input:` Dataset D di t transazione costruite su un insieme I di n item, vettore $\vec{f}$ delle frequenze degli item, k, Δ, ϵ
`Output:` Stima $\hat{s}_{min}$ di s_{min}

$\tilde{s} \leftarrow$ massimo supporto atteso per un k-itemset
$s_{max} \leftarrow 0$
`second`
$W \leftarrow \emptyset$
`first`
`for` $i \leftarrow 1$ `to` Δ `do`
  $\hat{D}_i \leftarrow$ random dataset con parametri $t,\ n,\ \vec{f}$
  $W \leftarrow W \cup \{k$-itemset frequenti in $\hat{D}_i$ con supporto almeno $\tilde{s}\}$
`if` $W = \emptyset$ `then`
  $\tilde{s} \leftarrow \tilde{s}/2$
  `goto first`
`if` $(s_{max} = 0)$ `then`
  $s_{max} \leftarrow \max_{X \in W, \hat{D}_i}\{$supporto di X in $\hat{D}_i\} + 1$
`for` $s \leftarrow \tilde{s}$ `to` s_{max} `do`
  `for all` $X \in W$ `do`
    $p_X(s) \leftarrow$ probabilità empirica che $\{Z_{X,s} = 1\}$
  `for all` $X, Y \in W : X \cap Y \neq \emptyset$ `do`
    $p_{XY}(s) \leftarrow$ probabilità empirica che $\{Z_{X,s}Z_{Y,s} = 1\}$

$$b_1(s) \leftarrow \sum_{X,Y \in W; Y \in I(X)} p_X(s)p_Y(s)$$

$$b_2(s) \leftarrow \sum_{X,Y \in W; X \neq Y \in I(X)} p_{XY}(s)$$

`if` $b_1(\tilde{s}) + b_2(\tilde{s}) \leq \epsilon/4$ `then`
  $s_{max} \leftarrow \tilde{s}$
  $\tilde{s} \leftarrow \tilde{s}/2$
  `goto second`
$\hat{s}_{min} \leftarrow \min\{s > \tilde{s} : b_1(s) + b_2(s) \leq \epsilon/4\}$
`return` $\hat{s}_{min}$

Il seguente teorema fornisce un limite alla probabilità che il valore $\hat{s}_{min}$, calcolato attraverso la procedura sopra esposta, sia una stima conservativa di s_{min}, tale che $\hat{s}_{min} \geq s_{min}$.

TEOREMA. *Se* $\Delta = O(log(1/\delta)/\epsilon)$, *il valore* $\hat{s}_{min}$ *restituito dal metodo di Monte Carlo soddisfa la seguente disuguaglianza:*

$$\mathbf{Pr}(b_1(\hat{s}_{min}) + b_2(\hat{s}_{min}) \leq \epsilon) \geq 1 - \delta.$$

DIMOSTRAZIONE. Si assuma che $b_1(\hat{s}_{min}) + b_2(\hat{s}_{min}) > \epsilon$. Si noti che $b_1(\hat{s}_{min}) \leq b_2(\hat{s}_{min})$, perciò abbiamo che $b_2(\hat{s}_{min}) \geq \epsilon/2$. Sia B la variabile aleatoria corrispondente a Δ volte la stima di $b_2(\hat{s}_{min})$ ottenuta dalla simulazione di Monte Carlo, allora $\mathrm{E}[B] > \Delta\ \epsilon/2$. Dal momento che la simulazione di Monte Carlo ritorna $\hat{s}_{min}$ come stima di s_{min}, abbiamo che $B \leq \Delta\ \epsilon/4$. Sia

$$\Delta = \frac{8\log(1/\delta)}{\epsilon},$$

e sia $c < 1$ tale che:

$$(1 - c)\ \mathrm{E}[B] = \Delta\ \epsilon/4.$$

Poiché $\mathrm{E}[B] > \Delta\ \epsilon/2$, allora $c \geq 1/2$. Utilizzando il limite di Chernoff, abbiamo:

$$\mathbf{Pr}(B \leq \Delta\ \epsilon/4) \leq e^{-\frac{c^2 E[B]}{2}} \leq e^{-\frac{1}{4}\frac{8\ log(1/\delta)}{2}} \leq \delta.$$

Perciò $\mathbf{Pr}(b_1(\hat{s}_{min}) + b_2(\hat{s}_{min}) > \epsilon) \leq \delta$.

Tale teorema stabilisce quindi che: la probabilità con cui la variabile aleatoria $\hat{Q}_{k,\hat{s}_{min}}$ devii dalla variabile aleatoria di Poisson U per un valore superiore al limite massimo fissato ϵ è inferiore a δ.

2.5 Multi-comparison test

L'approccio classico per l'identificazione di itemset statisticamente significativi all'interno di un dataset D è basato sull'applicazione standard del *multi-hypothesis testing*.

L'*hypothesis testing* (Pietracaprina, 2011) è un metodo ampiamente utilizzato per l'analisi statistica dei dati di un esperimento. Il metodo prevede la formulazione di due differenti ipotesi:

- La prima ipotesi, detta *null hypothesis* H_0, rappresenta l'assunzione che i dati siano coerenti con il modello random;
- La seconda ipotesi, detta *alternate hypothesis* H_1, assume invece una significativa discrepanza dei dati rispetto al modello random.

Se la *null hypothesis* viene rigettata si deve accettare la *alternate hypothesis* e concludere che c'è quindi una differenza significativa tra i dati dell'esperimento e il modello random associato, altrimenti, accettando la *null hypothesis*, si conclude che tale differenza non sussiste. È importante sottolineare che non è mai possibile provare che una *null hypothesis* sia effettivamente vera, infatti è sempre possibile che esista una differenza che non è ancora stata trovata. Dal momento che non si possiede la certezza assoluta che esista o non esista una differenza si adotta un approccio statistico all'interno dell'*hypothesis testing*. Esistono due tipi di errori che vengono associati alla *null hypothesis*:

- L'errore di *Tipo I* occorre quando la *null hypothesis* H_0 viene rigettata anche se corretta e perciò si conclude che c'è una differenza quando in realtà questa non sussiste. La probabilità di commettere un errore di *Tipo I* è conosciuta come *livello di significatività* del test e viene indicata anche con $\alpha = \mathbf{Pr}$ (Type I Error). Il *livello di significatività* misura la probabilità di rigettare la *null hypothesis* quando invece questa è corretta (falso positivo).
- L'errore di *Tipo II* occorre quando la *null hypothesis* non viene rigettata anche se esiste una differenza tra le famiglie di dati analizzate e quindi la *null hypothesis* H_0 è falsa (falso negativo).

L'abilità di individuare le differenze è nota come *potenza* del test (definita come $1 - \mathbf{Pr}$(Type II Error)) e fornisce perciò la probabilità di rigettare correttamente la *null hypothesis*. Più alta quindi è la *potenza* di un test, più alta sarà la probabilità di individuare delle differenze qualora queste esistessero. Il *p-value* associato ad una *null hypothesis* H_0 indica il minimo livello di significatività per il quale H_0 non viene rifiutata.

In un *multi-hypothesis test* il risultato di un esperimento viene utilizzato per testare contemporaneamente un insieme di ipotesi. Se si vuole ad esempio ricercare quali sono i k-itemset significativi all'interno di dataset assegnato, bisogna testare $\binom{n}{k}$ *null hypothesis* simultaneamente, dove ciascuna *null hypothesis* corrisponde all'ipotesi che il supporto di un dato itemset non sia statisticamente significativo. Nel contesto del *multi-hypotesis testing*, il livello di significatività non può essere valutato considerando individualmente ciascuna delle ipotesi, ma bisogna considerare nell'insieme tutte le ipotesi avanzate. Se quindi il livello di significatività fissato per ciascuna ipotesi testata è α, risulterebbe errato concludere che non si commettono errori di *Tipo I* con probabilità α e cioè che con probabilità $1-\alpha$ tutti gli itemset per i quali la *null hypothesis* viene rigettata siano statisticamente significativi. Per garantire un livello di significatività complessivo al più α è necessario in questo caso che la somma dei livelli di significatività di ciascuna delle *null hypothesis* considerate sia al più α.

Per comprendere maggiormente il ragionamento alla base di questo approccio si faccia riferimento al seguente esempio.

ESEMPIO. Sia D un dataset di 1.000.000 di transazioni costruite su un insieme di 1000 item, ognuno con frequenza associata pari a 1/1000, e poniamo che la coppia di item <*i,j*> appaia in almeno 7 transazioni. Ci chiediamo se l'itemset $\{i, j\}$ possa essere considerato significativo. Per rispondere dobbiamo considerare un dataset random dove ciascun item viene posto in ciascuna transazione con probabilità 1/1000 indipendentemente dagli altri item. La probabilità che la coppia di item <*i,j*> venga inclusa all'interno della stessa transazione è pari ad 1/1.000.000, perciò il numero di transazioni attese contenenti gli item i e j è pari ad 1, e la probabilità che $\{i, j\}$ appaia in almeno 7 transazioni è pari a circa 0,0001. Perciò sembra che il supporto di $\{i, j\}$ nel dataset D sia statisticamente significativo. Tuttavia, ciascuna delle 499.500 coppie di item ha probabilità 0,0001 di apparire all'interno di almeno 7 transazioni nel dataset random. Quindi, sotto l'assunzione che gli item siano posti indipendentemente l'uno dall'altro all'interno di ciascuna transazione, il numero atteso di coppie di item con supporto almeno pari a 7 è circa 50. Di conseguenza, se nel dataset ci sono solo 50 coppie di item con supporto almeno pari a 7, restituire la coppia <*i,j*> come itemset statisticamente significativo è un errore. Se invece all'interno del dataset D sono presenti 300 coppie disgiunte di item ognuna con supporto almeno pari a 7, dal momento che la probabilità che tale evento si verifichi è circa $1/2^{300}$, allora è verosimile ipotizzare che il supporto della maggior parte delle coppie sia statisticamente significativo.

Il *False Discovery Rate (FDR)* è stato introdotto per la prima volta in (Benjamini e Hochberg, 1995). Sia V il numero di errori di *Tipo I*, e quindi il numero di *null-hypothesis* rigettate erroneamente, e sia R il numero totale di *null-hypothesis* rigettate durante il *multi-hypothesis test*, allora si definisce con il termine *FDR* il rapporto atteso fra il numero di *null-hypothesis* rigettate erroneamente e il numero totale di *null-hypothesis* rigettate, ossia $FDR = E[V/R]$.

Come anticipato nel capitolo introduttivo, l'approccio classico per la scoperta di itemset significativi applica il teorema descritto in (Benjamini e Yekutieli, 2001) per limitare il valore di *FDR*:

TEOREMA. *Assumiamo di testare m null hypothesis. Siano*

$$p_{(1)} \leq p_{(2)} \leq \cdots \leq p_{(m)}$$

i p-value osservati in ordine crescente per le m null hypothesis. Per una data costante β, con $0 < \beta < 1$, definiamo

$$l = max \left\{ i \geq 0 \ : \ p_{(i)} \leq \frac{i}{m \sum_{j=1}^{m} 1/j} \beta \right\},$$

e rigettiamo le null hypothesis corrispondenti ai test $(1), (2), \ldots, (l)$.
Allora l'FDR per l'insieme di null hypothesis rigettate è limitato superiormente da β.

Viene ora fornito lo pseudocodice relativo all'approccio appena descritto:

`Input:` Dataset D di t transazioni costruite su di n item, vettore $\vec{f}$ di frequenze degli item, k, $\beta \in (0, 1)$
`Output:` Famiglia di k-itemset statisticamente significativi con $FDR \leq \beta$

Determinare s_{min} e ricavare $F_{(k)}(s_{min})$ dal dataset D
`for all` $X \in F_{(k)}(s_{min})$ `do`
  $s_X \leftarrow$ supporto di X in D
  $f_X \leftarrow \prod_{i \in X} f_i$
  $p^X \leftarrow \Pr(\text{Bin}t, f_X) \geq s_X)$
Sia $p_{(1)}, p_{(2)}, \ldots,$ la sequenza dei *p-value* $p^{(X)}$ ordinati per valori crescenti, con $X \in F_{(k)}(s_{min})$
$m \leftarrow \binom{n}{k}$
$l = max \left\{ 0, i : \ p_{(i)} \leq \frac{i}{m \sum_{j=1}^{m} 1/j} \beta \right\}$
`return` $\{X \in F_{(k)}(s_{min}) : p^{(X)} = p_{(i)}, 1 \leq i \leq l\}$

2.6 Approccio basato sul supporto s^*

Siano α e β due costanti appartenenti all'intervallo $(0, 1)$. L'approccio sviluppato in (Kirsch *e altri*, 2009) calcola una soglia di supporto minima s^* tale per cui, con confidenza $1-\alpha$, i k-itemset appartenenti all'insieme $F_{(k)}(s^*)$, ossia l'insieme di k-itemset con supporto almeno pari ad s^*, possono essere etichettati come statisticamente significativi con *FDR* al più β.

Il valore del supporto s^* viene determinato attraverso un approccio statistico il quale tiene conto delle caratteristiche globali del dataset D assegnato. Sia s_{min} la soglia si supporto minima per la quale l'approssimazione di Poisson applicata alla distribuzione della variabile aleatoria $\hat{Q}_{k,s}$ risulti valida e sia inoltre s_{max} il massimo supporto che un qualunque k-itemset assume nel dataset D. L'approccio qui descritto esegue $h = \lfloor \log_2(s_{max} - s_{min}) \rfloor + 1$ confronti per determinare il valore di s^* (qualora questo esistesse) attraverso i seguenti passi:

1. Si definisce $s_0 = s_{min}$ e $s_i = s_{min} + 2^i$, per $1 < i < h$;

2. Al confronto i-esimo, con $0 \leq i < h$, si verifica la correttezza della *null hypothesis* H_0^i secondo la quale il valore osservato Q_{k,s_i} è approssimato dalla stessa variabile aleatoria di Poisson che approssima $\hat{Q}_{k,s_i}$;

3. Si assegna alla soglia s^* il minimo valore di supporto s_i per il quale la *null hypothesis* H_0^i associata è stata rigettata. Nel caso in cui non dovesse essere rigettata nessuna delle h *null hypothesis* formulate si restituisce ∞ come valore di soglia s^*.

Per garantire la correttezza della procedura appena descritta, è importante specificare le condizioni per le quali le *null hypothesis* H_0^i vengono rigettate. Per fare in modo che la famiglia di itemset restituiti sia statisticamente significativa con confidenza pari ad $1 - \alpha$ si fissano i valori $\alpha_0, \alpha_1, \ldots, \alpha_{h-1}$ in modo tale che $\sum_{i=0}^{h-1} \alpha_i \leq \alpha$ e per $0 \leq i < h$ si rigetta la *null hypothesis* H_0^i se il *p-value* del valore osservato Q_{k,s_i} è minore di α_i. In questo modo la probabilità di rigettare una generica *null hypothesis* corretta è al più α. Tuttavia questo accorgimento non assicura un *FDR* basso per l'insieme di itemset contenuti in $F_{(k)}(s^*)$. Infatti si può verificare che alcuni itemset non significativi, caratterizzati però da un elevato supporto, possono appartenere all'insieme $F_{(k)}(s^*)$ rappresentando quindi una falsa identificazione. L'impatto che questo evento ha sulla qualità degli itemset restituiti può essere limitato assicurando che l'*FDR* complessivo sia al di sotto di uno specifico limite β. Si fissano quindi i valori $\beta_0, \beta_1, \ldots, \beta_{h-1}$ in modo che $\sum_{i=0}^{h-1} \beta_i^{-1} \leq \beta$

e per $0 \leq i < h$ si definisce $\lambda_i = E[\hat{Q}_{k,s_i}]$. L'ipotesi H_0^i verrà rigettata quando il *p-value* di Q_{k,s_i} è al più α_i e quando il valore di Q_{k,s_i} è almeno pari al prodotto $\beta_i * \lambda_i$ garantendo in questo modo un livello di confidenza pari a $1 - \alpha$ ed un valore di *FDR* superiormente limitato da β.

Il seguente teorema stabilisce la correttezza dell'approccio appena descritto.

TEOREMA. *Con confidenza $1 - \alpha$, $F_{(k)}(s^*)$ è una famiglia di k-itemset statisticamente significativi con FDR al più β.*

DIMOSTRAZIONE. Dato che $\sum_{i=0}^{h-1} \alpha_i \leq \alpha$, tutte le *null hypothesis* vengono rigettate correttamente con probabilità $1-\alpha$. Sia E_i l'evento "H_0^i *è rigettata*" o equivalentemente "*il p-value di Q_{k,s_i} è minore di α_i e $Q_{k,s_i} \geq \beta_i \lambda_i$*". Supponiamo che H_0^i è la prima *null hypothesis* rigettata, quindi $s^* = s_i$ e Q_{k,s_i} itemset sono etichettati come significativi. Sia V_i il numero di false scoperte lungo tutti i Q_{k,s_i} itemset con supporto almeno pari a s_i, il valore atteso per V_i è $E[X_i | E_i, \bar{E}_{i-1}, \ldots, \bar{E}_0]$, dove X_i è una variabile aleatoria di Poisson con valore atteso λ_i. Dato che $Q_{k,s_i} \geq \beta_i \lambda_i$ quando H_0^i viene rigettata, per la *legge della probabilità totale* abbiamo che:

$$
\begin{aligned}
FDR &\leq \sum_{i=0}^{h-1} E\left[\frac{V_i}{Q_{k,s_i}}\right] \mathbf{Pr}(E_i, \bar{E}_{i-1}, \ldots, \bar{E}_0) \\
&\leq \sum_{i=0}^{h-1} \frac{E[V_i]}{\beta_i \lambda_i} \mathbf{Pr}(E_i, \bar{E}_{i-1}, \ldots, \bar{E}_0) \\
&\leq \sum_{i=0}^{h-1} \frac{E[X_i | E_i, \bar{E}_{i-1}, \ldots, \bar{E}_0]}{\beta_i \lambda_i} \mathbf{Pr}(E_i, \bar{E}_{i-1}, \ldots, \bar{E}_0) \\
&= \sum_{i=0}^{h-1} \frac{\sum_{j \geq 0} \mathbf{Pr}(X_i = j, E_i, \bar{E}_{i-1}, \ldots, \bar{E}_0)}{\beta_i \lambda_i} \\
&\leq \sum_{i=0}^{h-1} \frac{\lambda_i}{\beta_i \lambda_i} = \sum_{i=0}^{h-1} \frac{1}{\beta_i} \leq \beta.
\end{aligned}
$$

In conclusione, il valore della soglia di supporto s^* assicura che il numero di k-itemset appartenenti all'insieme $F_{(k)}(s^*)$ devii significativamente da ciò che ci si potrebbe aspettare in un modello random del dataset assegnato D.

Viene ora fornito lo pseudocodice relativo all'approccio appena descritto:

`Input:` Dataset D di t transazioni costruite su di n item, vettore $\vec{f}$ delle frequenze degli item, k, $\alpha \in (0,1)$, $\beta \in (0,1)$
`Output:` s^* tale che, con confidenza $1-\alpha$, $F_{(k)}(s^*)$ è una famiglia statisticamente significativa di k-itemset con $FDR \leq \beta$

Determinare s_{min} e ricavare $F_{(k)}(s_{min})$ dal dataset D
$s_{max} \leftarrow$ massimo supporto di un k-itemset in D
$i \leftarrow 0$
$s_0 \leftarrow s_{min}$
$h \leftarrow \lfloor \log_2(s_{max} - s_{min}) \rfloor + 1$
Fissare $\alpha_0, \ldots, \alpha_{h-1} \in (0,1)$ s.t. $\sum_{i=0}^{h-1} \alpha_i = \alpha$
Fissare $\beta 0, \ldots, \beta h - 1 \in (0,1)$ s.t. $\sum_{i=0}^{h-1} \beta_i^{-1} = \beta$
`for` $i \leftarrow 0$ to $h-1$ `do`
    Calcolare $\lambda_i = E[\hat{Q}_{k,s_i}]$
`while` $i < h$ `do`
    Calcolare Q_{k,s_i}
    `if` $(\Pr(\text{Poisson}(\lambda_i) \geq Q_{k,s_i}) \leq \alpha_i)$ `and` $(Q_{k,s_i} \geq \beta_i \lambda_i)$ `then`
        `return` $s^* \leftarrow s^i$
    $s_{i+1} \leftarrow s_{min} + 2^{i+1}$
    $i \leftarrow i + 1$
`return` $s^* \leftarrow \infty$

Capitolo 3

Toolbox sviluppato

Nel seguente capitolo vengono descritti gli obiettivi, le modalità di configurazione ed utilizzo del toolbox, nonché le strutture dati utilizzate per processare gli itemset all'interno dei dataset.

3.1 Obiettivi

Il toolbox presentato in questo documento è stato sviluppato con lo scopo di identificare quali sono gli itemset di taglia k statisticamente significativi all'interno di un dataset assegnato, ponendo particolare attenzione all'efficienza dell'algoritmo implementato.

Dai metodi ed approcci precedentemente esposti è possibile comprendere perciò come lo scopo principale del toolbox si traduca nei seguenti obiettivi:

- definire una soglia di supporto minima s_{min} tale per cui la distribuzione della variabile aleatoria $\hat{Q}_{k,s}$ è bene approssimata da una distribuzione di Poisson con il medesimo valore atteso quando $s \geq s_{min}$;

- applicare l'approccio basato sul *multi-comparison test* sull'insieme di itemset frequenti rispetto la soglia si supporto s_{min}, limitando il valore massimo di *FDR* attraverso il metodo sviluppato in (Benjamini e Yekutieli, 2001);

- determinare una soglia di supporto $s^* \geq s_{min}$ in modo che tutti i k-itemset frequenti oltre tale soglia possano essere considerati significativi con basso *FDR* secondo il metodo sviluppato in (Kirsch *e altri*, 2009);

- implementare il toolbox facendo attenzione all'uso delle risorse impiegate ed ottimizzando il codice e le strutture dati utilizzate al fine di ridurre i tempi di esecuzione complessivi.

3.2 Implementazione

Il toolbox presentato in questo documento si compone di un file di configurazione al cui interno è possibile specificare il dataset che si intende processare, le taglie degli itemset significativi che si vogliono studiare e altri parametri relativi alla qualità della soluzione che si vuole ottenere. Per il calcolo di tale soluzione, il toolbox, realizzato in linguaggio di programmazione *C*, fa affidamento al metodo Apriori implementato da Christian Borgelt per l'estrazione di itemset frequenti dal dataset assegnato e dai dataset random generati durante la simulazione di Monte Carlo ed inoltre utilizza le librerie scientifiche GSL per il calcolo delle distribuzioni di probabilità di Bernoulli e Poisson durante la scoperta di itemset frequenti e statisticamente significativi. L'insieme di itemset significativi scoperti al termine della procedura costituisce invece l'output principale del programma che viene salvato all'interno di un apposito file. Nei seguenti paragrafi vengono trattati in maniera più dettagliata ognuno degli aspetti del toolbox appena citati.

3.2.1 Parametri d'ingresso

L'input del toolbox è costituito da un file di configurazione al cui interno sono riportati tutti i parametri necessari al fine di ottenere una corretta esecuzione. Il file di configurazione è strutturato nel seguente modo:

```
Dataset = nome_dataset
K       = 2-4
Delta   = 1000
Epsilon = 0.01
Alpha   = 0.05
Beta    = 0.05
Rate    = 100
```

- `Dataset` deve contenere il nome del file nel quale è salvato il dataset D, compreso il percorso per raggiungere tale file qualora fosse necessario;

- `K` indica la taglia degli itemset significativi che si vuole studiare. Nel caso si fosse interessati a studiare itemset significativi appartenenti ad un intervallo di taglie è sufficiente indicare l'estremo inferiore e superiore dell'intervallo separandoli per mezzo del carattere '-';

- `Delta` indica il numero di dataset random che si vuole generare durante l'esecuzione del metodo di Monte Carlo;

- `Epsilon` è un indice di qualità relativo all'approssimazione della distribuzione di $\hat{Q}_{k,s}$ con una distribuzione di Poisson con il medesimo valore atteso;

- `Alpha` rappresenta un indice di qualità relativo alla confidenza con cui i k-itemset frequenti possono essere contrassegnati come statisticamente significativi nel secondo approccio;

- `Beta` rappresenta un indice di qualità riguardante il limite superiore del valore di *FDR* relativamente all'estrazione di k-itemset significativi in entrambi gli approcci;

- `Rate` serve ad indicare la percentuale di transazioni che si vuole generare per costruire i dataset random (valore di default 100).

3.2.2 Output restituito

L'output del toolbox si compone di due parti:

- La prima parte riguarda i tempi di esecuzione dell'algoritmo, sia complessivi che relativi a ciascuna fase dell'algoritmo. Vengono inoltre fornite le soglie di supporto stimate e attese durante la simulazione di Monte Carlo. Infine vengono forniti tutti i dati relativi agli itemset significativi identificati da ciascuna delle due procedure, come ad esempio il numero di itemset scoperti e le soglie di supporto calcolate. Questa prima parte dell'output viene visualizzata su *console*, il che permette di verificare in tempo reale lo stato di avanzamento delle diverse fasi dell'algoritmo.

- La seconda parte di output riguarda unicamente gli itemset statisticamente significativi estratti. Al termine dell'esecuzione del toolbox vengono infatti creati uno o più file contenenti gli itemset significativi scoperti attraverso ciascuno dei due approcci implementati e suddividendo ulteriormente gli itemset in file differenti in base alla taglia degli stessi. Questi file vengono memorizzati all'interno della cartella denominata *outcome*. Si noti che ad ogni esecuzione del toolbox, i file contenuti all'interno di questa cartella, relativi alla precedente esecuzione, vengono cancellati per lasciare spazio ai nuovi file di output.

3.2.3 Struttura dell'algoritmo

L'algoritmo ideato per la realizzazione del toolbox si compone di quattro fasi fondamentali ognuna delle quali necessaria al raggiungimento degli obiettivi fissati. La struttura del toolbox può essere schematizzata nel seguente modo:

FASE I: ***Pre-processing***
FASE II: **Simulazione di Monte Carlo**
1. Calcolo del massimo supporto atteso per un k-itemset.
2. Generazione di Δ dataset random.
3. Calcolo del valore di supporto massimo effettivo.
4. Calcolo dei valori di $b_1(s)$ e $b_2(s)$.
5. Calcolo della stima del supporto minimo.

FASE III: **Calcolo della soglia di supporto sStar**
1. Calcolo del supporto massimo e del valore di h.
2. Calcolo del valore di $L[i]$ al variare delle soglia di supporto.
3. Calcolo del valore di $Q[i]$ al variare delle soglia di supporto.
4. Calcolo del valore di sStar.

FASE IV: ***multi-comparison test***
1. Calcolo del supporto atteso per ciascun itemset frequente.
2. Calcolo Pr(Bin(t, F1[i].freq) $\geq$ F1[i].support).
3. Calcolo del valore di l.

i. La prima fase consiste in un *pre-processing* del dataset assegnato D fornito in input al programma sotto forma di file di testo. Il formato da rispettare per l'utilizzo del toolbox è quello specificato nel workshop sul *Frequent Itemset Mining Implementations (FIMI)*, tenutosi in Florida nel 2003. Il file di testo deve contenere solo caratteri di scrittura semplici e ad ogni riga del file deve corrispondere una ed una sola transazione del dataset, separando gli item tra loro mediante uno spazio. Gli item, che per ovvie ragioni devono essere tutti differenti tra loro, possono essere sia dei valori numerici che delle stringhe.
   L'algoritmo utilizza una struttura dati di tipo dizionario, descritta nel paragrafo 3.3, per analizzare l'intero dataset e ricavare da questo i valori delle costanti T e N, che indicano rispettivamente il numero di transazioni contenute nel dataset assegnato e il numero di item su cui le transazioni sono state costruite. Attraverso l'analisi si ricava inoltre l'identificativo e il valore di supporto associato a ciascun item, tali valori vengono poi memorizzati insieme all'item stesso in un determinato nodo della struttura a dizionario implementata.
   Durante questa prima fase viene generato un file di testo contenente il

dataset reale D in cui tutte le istanze degli item sono state sostituite dall'identificativo associato all'item stesso. Il salvataggio degli item all'interno della struttura ad albero consentirà in un secondo momento di ottenere l'item vero e proprio a partire dal suo identificativo e quindi effettuare la conversione inversa per ciascuno degli item.
In questa prima fase il toolbox opera una sola lettura del dataset D.

ii. La seconda fase è focalizzata invece sul calcolo della soglia di supporto minimo s_{min} la quale assicura che, per ogni valore di supporto $s \in [s_{min}, +\infty)$, la distribuzione della variabile aleatoria $\hat{Q}_{k,s}$, che esprime il numero di k-itemset con supporto almeno pari ad s all'interno del dataset random $\hat{D}$, sia bene approssimata da una distribuzione di Poisson con lo stesso valore atteso.
Per determinare una buona approssimazione del valore di s_{min}, in questa fase viene applicata la simulazione di Monte Carlo. Questa seconda fase si compone di cinque sezioni:

Calcolo del massimo supporto atteso per un *k*-itemset. In questa prima sezione si calcola il valore di $\tilde{s}$, cioè il massimo supporto atteso per un generico k-itemset. Il calcolo di $\tilde{s}$ prevede di moltiplicare tra loro le k frequenze più elevate fra gli N item contenuti nel dataset e di memorizzare il risultato all'interno della variabile *maxExpectedSupport*.
Qualora si fosse interessati ad itemset di taglia compresa in un determinato intervallo di estremi $minK$ e $maxK$, la variabile *maxExpectedSupport* consiste in un array di dimensioni $maxK - minK + 1$ in cui vengono memorizzati i supporti massimi attesi per ognuna delle taglie di itemset comprese nell'intervallo.

Generazione di Δ dataset random. Successivamente si passa alla generazione di Δ dataset random, dove la probabilità con cui un generico item i può comparire all'interno di una transazione coincide con la frequenza associata a tale item, calcolata come il rapporto tra il supporto associato a i e il numero N di transazioni contenute nel dataset D.
È importante sottolineare che la probabilità con cui un item compare all'interno di una transazione è indipendente dalla probabilità degli altri item del dataset. Ogni qual volta venga generato un dataset random $\hat{D}$, prima di generare il successivo, vengono estratti gli itemset frequenti con supporto almeno pari a $maxExpectedSupport$, cioè il massimo supporto atteso per la taglia di itemset che si intende studiare, memorizzando il risultato

all'interno di W (una struttura dati di tipo dizionario appositamente creata per la memorizzazione degli itemset frequenti).
Nel caso si fosse interessati allo studio di itemset di taglia differente, W coincide con un array di $maxK - minK + 1$ puntatori contenenti gli indirizzi delle radici di altrettante strutture dizionario, ognuna associata ad una precisa taglia k di itemset.
Per ogni itemset frequente presente in W, si memorizza il massimo supporto ottenuto e si associa una lista contenente i supporti e i relativi indici dei random dataset in cui tali itemset sono risultati essere frequenti.

Calcolo del valore di supporto massimo effettivo. In questa sezione si sfrutta parte del lavoro svolto precedentemente per calcolare il massimo supporto effettivo, ovvero il massimo supporto tra quelli dei k-itemset frequenti precedentemente estratti. In questa fase vengono perciò scanditi tutti gli itemset memorizzati in W per determinare il valore del supporto massimo effettivo che al termine della procedura è memorizzato all'interno della variabile *maxActualSupport*.
Qualora si fosse interessati ad itemset di taglia compresa in un determinato intervallo di estremi $minK$ e $maxK$, *maxActualSupport* consiste allora in un array di dimensioni $maxK - minK + 1$, per permettere di memorizzare i supporti massimi effettivi per qualunque taglia di itemset compresa nell'intervallo.

Calcolo dei valori di $b_1(s)$ e $b_2(s)$. Vengono calcolati i valori delle variabili $b_1(s)$ e $b_2(s)$ la cui somma definisce un *upper bound* sulla differenza tra le distribuzioni delle variabili aleatorie $\hat{Q}_{k,s}$ e la variabile aleatoria di Poisson con lo stesso valore atteso. Dato che i valori di b_1 e b_2 variano al variare del supporto s, tali valori vengono memorizzati rispettivamente negli array $b1[\ ]$ e $b2[\ ]$, di dimensione fissata pari a 50 entry, dato che il calcolo dei valori di b_1 e b_2 viene eseguito ad intervalli di supporto di ampiezza 50 per evitare la saturazione della memoria disponibile quando si lavora con dataset di grosse dimensioni.
Il calcolo dei valori di $b_1(s)$ e $b_2(s)$ richiede di conoscere la probabilità con cui ciascun itemset $X \in \mathcal{F}_{(k)}(maxExpectedSupport)$ assume supporto almeno s nei Δ dataset random quando

$$s \in [maxExpectedSupport, maxActualSupport].$$

Si procede perciò al calcolo della probabilità $p_X = P\{Z_{X,s} = 1\}$ per ogni itemset X dove Z è una variabile aleatoria di Bernoulli

che assume valore 1 se l'itemset X ha supporto almeno s in un generico dataset random e 0 altrimenti.
Per il calcolo di tali probabilità è stata utilizzata la seguente relazione:

$$\frac{\#\text{ dataset random in cui il supporto di } X \text{ è almeno } s}{\Delta}$$

Successivamente, attraverso il metodo *isNeighborhood*, vengono valutate tutte le coppie possibili di itemset frequenti $<X, Y>$ considerate "vicine", ossia tali per cui la loro intersezione non coincide con l'insieme vuoto $\emptyset$.
Questo punto è di fondamentale importanza per il calcolo dei valori di $b_2(s)$ dato che è necessario valutare la probabilità

$$p_{X,Y}(s) = P\{Z_{X,s}Z_{Y,s} = 1\},$$

ossia la probabilità che entrambi gli itemset X e Y abbiamo supporto almeno pari ad s in uno stesso dataset random. Per il calcolo di $p_{X,Y}(s)$ si applica la seguente relazione:

$$\frac{\#\text{ dataset random in cui sia } X \text{ che } Y \text{ hanno supporto almeno } s}{\Delta}$$

A questo punto è possibile procedere al calcolo di $b_1(s)$ e $b_2(s)$ attraverso le formule seguenti:

$$b_1(s) \leftarrow \sum_{X,Y \in W;\ X \cup Y \neq \emptyset} p_X(s) p_Y(s)$$

$$b_2(s) \leftarrow \sum_{X,Y \in W;\ X \cup Y \neq \emptyset;\ X \neq Y} p_{X,Y}(s)$$

Calcolo della stima del supporto minimo. Dal momento che si dispone dei valori di $b_1(s)$ e $b_2(s)$ al variare del supporto s, è possibile procedere al calcolo della stima di s_{min}, qualora la somma dei coefficienti $b_1(0)$ e $b_2(0)$ risultasse superiore a $\epsilon/4$, attraverso la seguente relazione:

$$\hat{s}_{min} \leftarrow \min\{s < \tilde{s} :\ b_1(s) + b_2(s) \leq \epsilon/4\}$$

Il valore $\hat{s}_{min}$ rappresenta la stima del supporto minimo s_{min} e viene memorizzato all'interno della variabile *minEstimatedSupport*.

Qualora si fosse interessati ad itemset di taglia compresa in un determinato intervallo di estremi $minK$ e $maxK$, le operazioni svolte in questa sezione e nella precedente vanno eseguite per ciascuna taglia di itemset che si intende studiare e in questo caso *minEstimatedSupport* consiste in un array di dimensioni $maxK - minK + 1$, per permettere di memorizzare i supporti minimi stimati per qualunque taglia di itemset compresa nell'intervallo. Nel caso invece in cui $b_1(s) + b_2(s) \leq \epsilon/4$ allora l'intera procedura di Monte Carlo va ripetuta imponendo questa volta

$$maxActualSupport \leftarrow maxExpectedSupport$$

e

$$maxExpectedSupport \leftarrow maxExpectedSupport/2.$$

iii. Come anticipato in precedenza, l'algoritmo utilizza due differenti approcci per identificare gli itemset frequenti significativi all'interno di un dataset assegnato. Dato che ora si dispone del valore di supporto s_{min}, grazie al quale poter studiare la significatività degli itemset frequenti rispetto tale soglia, in questa terza fase viene applicato l'approccio descritto in (Kirsch *e altri*, 2009) per raggiungere tale scopo.
È importante far notare fin da subito che se la taglia degli itemset che si intende studiare varia all'interno di un intervallo prefissato, le operazioni che seguono vengono ripetute per ciascuna delle taglie scelte. Questa fase si compone di quattro sezioni strutturate nel seguente modo:

FASE III: **Calcolo della soglia di supporto sStar**

1. Calcolo del supporto massimo e del valore di h.
2. Calcolo del valore di $L[i]$ al variare delle soglia di supporto.
3. Calcolo del valore di $Q[i]$ al variare delle soglia di supporto.
4. Calcolo del valore di sStar.

Calcolo del supporto massimo e del valore di *h*. In questa sezione si calcola il supporto massimo fra quelli dei k-itemset frequenti con supporto almeno pari ad s_{min} nel dataset reale D. Si procede quindi con l'applicazione dell'algoritmo Apriori sul dataset D memorizzando il risultato all'interno di una struttura a dizionario denominata $F2$. Successivamente vengono visitati tutti gli itemset memorizzati in $F2$ alla ricerca del massimo supporto ottenuto e salvando poi il risultato all'interno della variabile *maxSupport* al termine della procedura.
Infine si passa al calcolo del valore di h, ovvero il numero massimo

di iterazioni necessarie al calcolo della soglia di supporto s^*. Per il calcolo del valore di h si applica la seguente relazione:

$$h \leftarrow \lfloor \log_2(\text{maxSupport} - \text{minEstimatedSupport}) \rfloor + 1$$

Calcolo del valore di *L[i]* al variare delle soglie di supporto. Nella seconda sezione vengono calcolati i valori λ_i al variare di i nell'intervallo $[0, h]$; λ_i rappresenta il valore atteso della variabile aleatoria $\hat{Q}_{k,s_i}$, la quale ricordiamo esprime il numero di k-itemset frequenti, in un dataset random, con supporto almeno pari ad $s_i = \text{minEstimatedSupport} + 2^i$.
Il calcolo dei valori λ_i, con $i \in [0, h]$, richiede perciò di determinare il numero medio di itemset frequenti con supporto almeno s_i in un generico dataset random. Per ottenere tali valori quindi si conta il numero cumulato di itemset frequenti con supporto almeno pari ad s_i nei Δ dataset random e successivamente si suddivide tale numero per Δ stesso. I valori risultanti vengono memorizzati all'interno dell'array $L[\,]$.

Calcolo del valore di *Q[i]* al variare delle soglie di supporto. Nella terza sezione viene calcolato il numero di k-itemset frequenti, nel dataset reale D, con supporto almeno pari a

$$s_i = \text{minEstimatedSupport} + 2^i$$

con i che varia nell'intervallo $[0, h]$. Tali valori vengono calcolati contando quanti itemset, memorizzati all'interno dell'albero AVL $F2$, hanno supporto almeno pari a s_i e inserendo successivamente i risultati all'interno dell'array $Q[\,]$.

Calcolo del valore di sStar. L'obiettivo, che si intende raggiungere con il secondo approccio, consta nell'individuazione di una soglia di supporto minima s^* oltre la quale, con buona confidenza, tutti gli itemset frequenti possono essere considerati significativi. In questa ultima sezione si procede perciò al calcolo del valore di s^*, il quale però richiede di conoscere il valore della probabilità $\mathbf{Pr}(\text{Poisson}(\lambda_i) \geq Q_{k,s_i})$. Come anticipato precedentemente, il toolbox si rapporta alle librerie scientifiche GSL per il calcolo dei valori di distribuzione di probabilità associati alla variabile aleatoria $\text{Poisson}(\lambda_i)$.

Per il calcolo di s^* si associa a ciascuna soglia di supporto s_i la *null hypothesis* H_0^i, ovvero l'ipotesi secondo la quale la distribuzione dei valori della variabile aleatoria Q_{k,s_i} è regolata dalla stessa distribuzione di Poisson che disciplina la variabile aleatoria $\hat{Q}_{k,s_i}$.

Successivamente si assegna a s^* il valore della soglia minima s_i, con $i \in [0, h]$, tale per cui le seguenti condizioni vengono rispettate:

- la *null hypothesis* H_0^i viene rigettata, ovvero il *p-value* di Q_{k,s_i}, che coincide con $\mathbf{Pr}(\text{Poisson}(\lambda_i) \geq Q_{k,s_i})$, è minore di α_i, dove α_i è scelto in modo che $\sum_{i=0}^{h-1} \alpha_i = \alpha$;
- $Q_{k,s_i} \geq \beta_i \lambda_i$, dove β_i è scelto tale che $\sum_{i=0}^{h-1} \beta_i^{-1} \leq \beta$, il che permette di garantire un $FDR \leq \beta$.

Una volta ottenuto il valore di s^*, questo viene memorizzato all'interno della variabile *sStar* e viene calcolato l'insieme di k-itemset frequenti con supporto almeno pari ad s^* nel dataset reale.

iv. La quarta ed ultima fase si occupa infine di applicare l'approccio basato sul *multi-hypothesis test*, limitando l'*FDR* attraverso la procedura descritta in (Benjamini e Yekutieli, 2001) per la scoperta di itemset frequenti significativi all'interno di un dataset.
È importante far notare fin da subito che se la taglia degli itemset che si intende studiare varia all'interno di un intervallo prefissato, le operazioni che sono descritte qui di seguito vengono ripetute per ciascuna delle taglie scelte. Questa fase si compone di tre sezioni strutturate nel seguente modo:

FASE IV: ***multi-comparison test***

1. Calcolo del supporto atteso per ciascun itemset frequente.
2. Calcolo Pr(Bin(t, F1[i].freq) $\geq$ F1[i].support).
3. Calcolo del valore di l.

Calcolo del supporto atteso per ciascun itemset frequente. In questa prima sezione si determina innanzitutto l'insieme di k-itemset frequenti $\mathcal{F}_{(k)}(s_{min})$ con supporto almeno pari ad s_{min} nel dataset reale D. Il risultato viene poi memorizzato in $F1$, una struttura dati di tipo dizionario. In seguito, per ogni itemset $X \in \mathcal{F}_{(k)}(s_{min})$ si valuta il supporto atteso, calcolato come prodotto delle frequenze associate a ciascuno degli item che lo compongono e memorizzando il risultato all'interno della variabile *expSprt* contenuta nel nodo di $F1$ associato ad X.

Calcolo Pr(Bin(t, F1[i].freq) $\geq$ F1[i].support). Per ogni itemset $X \in \mathcal{F}_{(k)}(s_{min})$ si calcola il valore di *p-value* della relativa *null hypothesis* H_0^X, ovvero l'ipotesi secondo la quale il valore del supporto di X nel dataset reale D è guidato da una distribuzione binomiale di parametri t e f_X, dove t è il numero di transazioni

nel dataset reale D, mentre f_X è ottenuto come prodotto delle frequenze dei singoli item che compongono X.
Come anticipato in precedenza, durante questa operazione il toolbox fa riferimento alle librerie scientifiche GSL per il calcolo dei valori di distribuzione di probabilità associati alla variabile aleatoria $\mathrm{Bin}(t, f_X)$. Il valore di *p-value* di H_0^X per il generico itemset $X \in \mathcal{F}_{(k)}(s_{min})$ coincide con:

$$p^{(X)} \leftarrow \mathbf{Pr}(\mathrm{Bin}(t, f_X) \geq s_X),$$

dove s_X è il supporto di X nel dataset reale D. Il risultato viene poi memorizzato all'interno della variabile *pvalue* contenuta all'interno del nodo di $F1$ associato ad X. Infine gli itemset memorizzati all'interno dell'albero $F1$ vengono ordinati per valori di *p-value* crescenti applicando l'algoritmo di *qsort* implementato nelle librerie *stdlib* di C.

Calcolo del valore di *l*. In questa ultima sezione si calcola il valore della soglia l attraverso la seguente relazione:

$$l = \max\left\{ i \geq 0 : p_{(i)} \leq \frac{i}{m \sum_{j=1}^{m} \frac{1}{j}} \beta \right\}$$

con $m = |\mathcal{F}_{(k)}(s_{min})|$ e $p_{(1)} \leq p_{(2)} \leq \cdots \leq p_{(m)}$. Tale soglia assicura che rigettando le prime l *null hypotheses*, ossia quelle con *p-value* minori, gli itemset a loro associati possano essere considerati significativi con garanzia che l'*FDR* sia inferiore a β.

3.3 Strutture dati

Il toolbox utilizza una struttura dati di tipo dizionario come struttura dati principale per il salvataggio in memoria di tutte le informazioni inerenti gli item presenti all'interno del dataset reale e gli itemset frequenti estratti, mediante l'algoritmo Apriori, dai diversi dataset random e dal dataset reale. La struttura dati dizionario è stata implementata attraverso un albero AVL

Un albero AVL è fondamentalmente un albero binario bilanciato, dove la differenza di altezza fra due sotto-alberi figli di uno qualunque dei nodi dell'albero differisce al più per un solo livello. Le operazioni di ricerca, inserimento e cancellazione hanno tutte costo pari ad $O(\log x)$, dove x è il numero di nodi all'interno dell'albero precedentemente all'esecuzione dell'operazione.

3.3.1 avlItem

Un dataset è sempre caratterizzato da un numero T di transazioni e da un numero N di item su cui le transazioni sono state costruite, entrambi questi valori sono inizialmente non noti al programma. Questo toolbox per l'estrazione di itemset frequenti significativi consente di lavorare su un generico dataset i cui item possono essere sia dei numeri che delle stringhe. In entrambi i casi è richiesta una conversione di tali item in numeri compresi nell'intervallo $[0, N)$ al fine di semplificare le operazioni successive di ricerca e di ridurre l'utilizzo di memoria.

Durante la creazione dell'albero AVL, implementato attraverso la struttura avlItem, vengono conteggiati tutti gli inserimenti di nuovi item all'interno dell'albero attraverso un contatore che al termine della procedura conterrà l'esatto valore di N. Contemporaneamente un secondo contatore tiene traccia del numero di transazioni che vengono analizzate durante la scansione del dataset, ottenendo in questo modo il valore di T associato al dataset in esame. Ad ogni item è associato un valore di supporto il quale viene incrementato ogniqualvolta la ricerca di quell'item all'interno dell'albero vada a buon fine. Ad ogni item inoltre è associato un valore di indice compreso tra 0 e $N - 1$, in modo da ottenere per ciascun item un numero univoco, un identificativo quindi, che agevolerà le successive fasi di ricerca dei supporti associati a ciascun item. Nel dataset reale convertito verranno infatti riportati gli identificativi associati a ciascun item in alternativa agli item stessi che, come già detto precedentemente, possono essere sia numeri che stringhe.

La costruzione dell'albero consente quindi di determinare i valori di N e T, i valori di supporto associati a ciascun item del dataset reale e inoltre di convertire tale dataset in uno equivalente per favorire il lavoro nelle successive fasi del programma, il tutto eseguendo un'unica scansione del dataset reale.

Una volta ottenuto l'albero, viene costruito un array di interi dove ogni entry contiene il supporto di un item prefissato, più precisamente l'indice della entry associata a ciascun item corrisponde all'identificativo dell'item stesso, in questo modo le operazioni di lettura dei supporti associati a ciascun item verranno eseguite in tempo $O(1)$.

3.3.2 avlItemset

Il programma sviluppato ricorre ad un tool esterno per la ricerca di itemset frequenti all'interno di un dataset assegnato. Questo tool applica l'algoritmo Apriori e restituisce in output un file in cui ad ogni riga corrisponde un itemset frequente e la relativa frequenza associata.

Per far sì che tali dati siano utilizzabili dal programma, sono state create tre versioni differenti di strutture dati di tipo dizionario per il salvataggio di tali itemset in memoria. Tutte e tre le strutture sono implementate attraverso un albero AVL e differiscono tra loro solamente per le variabili associate a ciascuno dei nodi. Questa scelta è stata condotta con lo scopo di ridurre l'utilizzo di memoria, così da conservare solamente le informazioni necessarie associate a ciascun itemset a seconda dei casi. Le strutture sono di seguito elencate.

avlItemsetC Questa struttura dati viene utilizzata per il salvataggio in memoria di quegli itemset frequenti che sono stati ottenuti attraverso l'applicazione dell'algoritmo Apriori sui dataset random generati durante l'esecuzione del metodo di Monte Carlo. In particolare tale struttura dati permette di memorizzare per ciascun itemset frequente, il massimo supporto associato e il supporto ottenuto in ciascun dataset random in cui tale itemset è risultato essere frequente. In questo modo vengono conservate tutte le informazioni necessarie per eseguire il calcolo del minimo supporto stimato per una determinata taglia k di itemset.

avlItemsetE La procedura classica basata sul *multi-comparison test* per l'estrazione di itemset frequenti significativi, necessita del salvataggio per ciascun itemset dei relativi valori di supporto, reale e atteso, e del valore di *p-value* relativo alla *null-hypothesis* associata. Per questo motivo la struttura avlItemsetE riserva per ciascun nodo tre variabili destinate alla memorizzazione di tali informazioni.

avlItemsetS La procedura invece sviluppata in (Kirsch *e altri*, 2009) per l'estrazione di itemset frequenti significativi non necessita solamente del supporto associato a ciascun itemset per poter essere applicata e quindi per ciascun itemset si memorizza nel dizionario solo il valore che

tale itemset assume nel dataset reale. È possibile notare come questa procedura, proprio per il tipo di struttura dati che richiede, sia più efficiente in termini di consumo della memoria rispetto alla procedura basata sul *multi-comparison test*.

Capitolo 4

Sperimentazione

In questo capitolo vengono analizzati e discussi i risultati ottenuti a seguito delle sperimentazioni svolte sul toolbox sviluppato e relative a:

- confronto di efficacia tra la procedura per la scoperta di itemset statisticamente significativi basata sul *multi-comparison test* e la procedura basata sul supporto minimo s^* sviluppata in (Kirsch *e altri*, 2009);

- miglioramento dell'efficienza del toolbox riducendo il numero di transazioni e di item necessari alla creazione dei dataset random durante l'applicazione del metodo di Monte Carlo;

- confronto di efficienza con un algoritmo precedentemente creato per l'applicazione del metodo di Monte Carlo.

4.1 Confronto tra le procedure

Come esposto sopra, il toolbox sviluppato applica due differenti approcci per la scoperta di itemset statisticamente significativi all'interno di un dataset D.

La prima proceduta consiste in un'applicazione standard del metodo di *multi-comparison test*, mentre la seconda procedura, presentata di recente in (Kirsch *e altri*, 2009), consiste nel calcolo di una soglia di supporto minimo s^* oltre la quale tutti gli itemset frequenti con supporto almeno pari a tale soglia possono essere considerati significativi.

In questo paragrafo vengono confrontati i due approcci dal punto di vista dell'efficacia, esaminando il numero di itemset statisticamente significativi scoperti da entrambe le procedure, al fine di studiare quale dei due approcci identifica un numero maggiore di itemset significativi.

I dataset utilizzati per l'esperimento e le relative dimensioni sono riportati in Tabella 4.1.

Tabella 4.1: Proprietà dei dataset selezionati

Dataset	**n**	$\|D\|$
Retail	16470	88162
Pumsb*	2088	49046
Bmspos	1657	515597
Kosarak	41270	990002
Plants	15148	34781
Pumsb	2113	49046
Bms1	497	59601
Bms2	3340	77512
Data_400k	964	397487

Sono stati condotti nove test, uno per ciascun dataset selezionato, impostando i parametri di ingresso con seguenti valori:

```
K       = 2-4
Delta   = 1000
Epsilon = 0.01
Alpha   = 0.05
Beta    = 0.05
Rate    = 100
```

Al variare del valore di k all'interno dell'intervallo $[2, 4]$ in ciascun test è stato calcolato il numero di itemset significativi scoperti dalla prima procedura, memorizzando il risultato all'interno della variabile R, e il numero di itemset significativi scoperti invece dalla seconda procedura, memorizzando questa volta il risultato all'interno della variabile Q_{k,s^*}. In seguito per ciascuna coppia <*dataset,k*> è stato calcolato il valore di $r = Q_{k,s^*}/R$ che esprime il rapporto tra il numero di itemset significativi individuati dalla seconda procedura rispetto al numero di quelli individuati dalla prima. Di conseguenza, se $r \geq 1$ significa che la seconda procedura ha individuato un numero maggiore o uguale di itemset statisticamente significativi rispetto alla prima procedura, altrimenti il numero di itemset significativi individuati è inferiore.

Dai test condotti si evincono i seguenti risultati:

Tabella 4.2: Numero di itemset significativi scoperti dalle procedure

	k=2		**k=3**		**k=4**	
Dataset	Q_{2,s^*}	R	Q_{3,s^*}	R	Q_{4,s^*}	R
Retail	1	0	3	2	6	5
Pumsb*	25	24	241	240	2504	2503
Bmspos	2	1	24	23	843	842
Kosarak	1	0	1	0	12	11
Plants	364	363	36464	36463	802144	802142
Pumsb	0	0	0	0	2	0
Bms1	75	73	311994	311993	10637864	9136369
Bms2	347	346	36080	36021	369251	52090
Data_400k	3	2	1081	1080	1051747	1051746

Tabella 4.3: Valori di r calcolati

Dataset	**k=2**	**k=3**	**k=4**
Retail	-	1,5	1,2
Pumsb*	1,04	1,004	1,0004
Bmspos	2,0	1,043	1,001
Kosarak	-	-	1,09
Plants	1,002	1,0	1,0
Pumsb	-	-	-
Bms1	1,027	1,0	1,1643
Bms2	1,003	1,002	7,0887
Data_400k	1,5	1,001	1,0

Alla luce di quanto esposto in Tabella 4.3, in ciascuno dei test condotti si evince come il valore di r sia sempre maggiore o uguale ad 1, il che significa che l'approccio sviluppato in (Kirsch *e altri*, 2009) restituisce in tutti gli esperimenti eseguiti un numero di itemset significativi pari o superiore all'approccio basato sul *multi-comparison test*.

Entrambe le procedure identificano i k-itemset statisticamente significativi tra quelli il cui supporto è almeno pari ad s_{min}. Dato che i valori della soglia di supporto s_{min} e del limite superiore di *FDR* garantito sono i medesimi per entrambi gli approcci, i risultati ottenuti dimostrano come la seconda

procedura utilizzata, cioè quella basata sull'approccio descritto in (Kirsch *e altri*, 2009), sia la più efficace.

Questo risultato è dovuto fondamentalmente al fatto che tale procedura valuta la significatività dell'insieme $F_{(k)}(s^*)$ comparando Q_{k,s^*} con $\hat{Q}_{k,s^*}$ e tenendo perciò in considerazione le caratteristiche globali del dataset e non solo le singole scoperte fatte su ciascun k-itemset X nel dataset D.

In alcuni test si può notare come il valore assunto da r sia piuttosto grande (ad esempio nel caso dei dataset *Bms1* e *Bms2* con $k = 4$), mentre in altri test a r è stato assegnato il simbolo '-' per indicare che il numero di itemset significativi identificati dalla prima procedura è nullo. Ciò evidenzia come l'efficacia della seconda procedura rispetto alla prima possa raggiungere picchi anche del 700% e come questa riesca ad identificare itemset significativi anche nei casi in cui la procedura classica basata sul *multi-comparison test* non abbia successo.

I risultati qui riportati confermano inoltre le considerazioni riportate nel Paragrafo 4.2 di (Kirsch *e altri*, 2009).

4.2 Miglioramento delle prestazioni

Durante l'utilizzo del toolbox sviluppato si è notato come la generazione dei dataset random durante l'applicazione del metodo di Monte Carlo rappresenti la fase che maggiormente incide sui tempi complessivi di esecuzione. Questa tendenza viene mostrata nel grafico in Figura 4.1 dove viene riportata l'incidenza percentuale dei tempi di esecuzione della fase relativa alla simulazione di Monte Carlo rispetto ai tempi totali di esecuzione dell'algoritmo su un campione di nove differenti dataset.

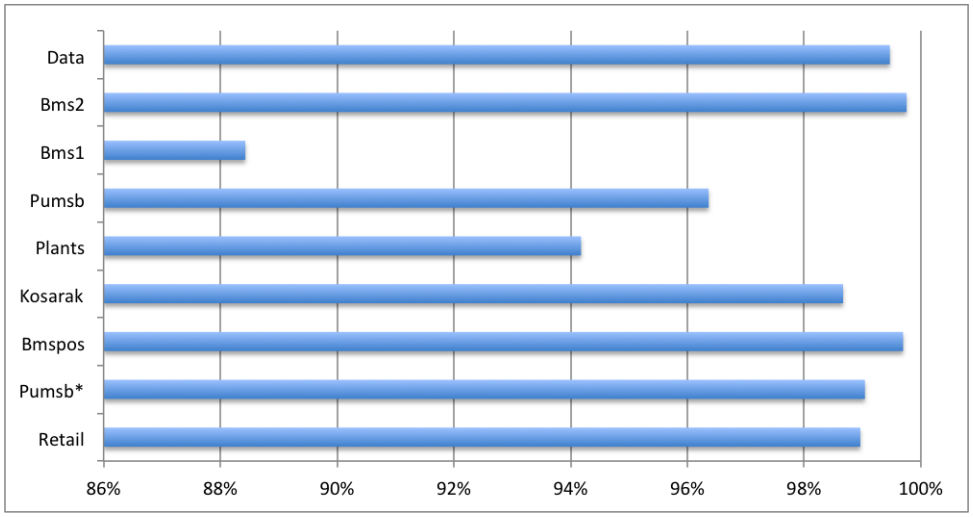

Figura 4.1: Incidenza dei tempi di esecuzione della fase di Monte Carlo

Dal grafico è possibile notare come mediamente il 94% del tempo totale di esecuzione del toolbox sia impiegato nella generazione dei Δ dataset random, per questo motivo si è deciso di agire su tale aspetto per incrementate l'efficienza complessiva del toolbox e ridurne quindi il tempo totale di esecuzione.

4.2.1 Problema

L'applicazione del metodo di Monte Carlo prevede la generazione di Δ dataset random di dimensioni equivalente a quelle del dataset di ingresso D. Durante la generazione, per ogni dataset random, si estraggono tutti i k-itemset con supporto almeno pari a $\tilde{s}$, ovvero il massimo supporto atteso per un k-itemset.

Con l'aumentare delle dimensioni del dataset D, risulta evidente come il tempo richiesto per la generazione dei dataset random aumenti di conseguenza, provocando un degrado significativo delle prestazioni generali del toolbox.

4.2.2 Soluzione proposta

Dato che il tempo di esecuzione del metodo di Monte Carlo è in funzione dei parametri Δ, n e $|D|$ si è deciso di operare su quest'ultimi due per ridurre i tempi di esecuzione complessivi del toolbox.

In particolare, prima di cominciare la generazione dei dataset random, viene effettuata un'attività di *pre-processing* che prevede di estrarre dall'insieme I di n item solo quelli con supporto maggiore o uguale a $\tilde{s}$, così che il numero di item da prendere in considerazione durante la generazione dei dataset random sia significativamente minore di n. Questa riduzione del numero di item non influenza la ricerca di itemset frequenti con supporto almeno pari a $\tilde{s}$ dato che, per la proprietà di antimonotonicità del supporto, un itemset non può avere supporto superiore ad uno qualsiasi dei suoi sottoinsiemi di item.

Il secondo accorgimento preso, invece, consiste nel generare un numero di transazioni pari solo ad una frazione f di $|D|$, dove $|D|$ indica il numero di transazioni contenute nel dataset D, per ciascuno dei dataset random, costruendo gli stessi quindi su scala ridotta rispetto al caso standard e consentendo perciò di ridurre il numero di operazioni necessarie in questa fase. Quest'ultimo accorgimento comporterà di considerare come significativi quegli itemset con supporto almeno pari a $f * \tilde{s}$. Si noti che f è un numero compreso nell'intervallo $(0, 1)$.

Per applicare tale soluzione, il metodo di Monte Carlo è stato opportunamente modificato in maniera tale da ricevere tra i parametri di ingresso, oltre al numero di transazioni $|D|$ contenute all'interno del dataset reale, anche il valore di *rate*, ossia la percentuale di transazioni che si intendono generare per la costruzione dei dataset random, in altre parole quindi la scala di riferimento per la generazione di tali dataset. Il metodo inoltre provvederà

autonomamente alla normalizzazione dei valori rendendoli compatibili con le dimensioni reali del dataset.

4.2.3 Test effettuati

I test sono stati condotti prendendo in considerazione cinque diversi dataset reali le cui proprietà sono riportate in Tabella 4.4. È stato scelto questo campione di dataset per la varietà delle dimensioni degli stessi, in modo da ottenere dei risultati con validità più ampia possibile.

Tabella 4.4: Proprietà dei dataset selezionati

Dataset	**n**	$\|D\|$
Retail	16470	88162
Pumsb*	2088	49046
Bmspos	1657	515597
Kosarak	41270	990002
Plants	15148	34781

Sono stati condotti tre test sulla base della taglia di itemset significativi estratti. Il test consiste nel valutare i tempi di esecuzione e il numero di itemset significativi restituiti al variare del valore di *rate*. Le soglie di *rate* applicate sono del 30%, 50% e 80% (oltre alla soglia di riferimento del 100%, necessaria per il calcolo dei possibili vantaggi e svantaggi dovuti alla soluzione proposta), mentre il numero di dataset random generati durante la simulazione di Monte Carlo, ossia Δ, è stato impostato al valore 1000 in ciascuna delle prove effettuate. Per ogni valore di *rate* sono stati inoltre calcolati i valori di efficienza, cioè il guadagno in termini di prestazione ottenuto rispetto al caso standard, e di efficacia, ovvero il rapporto tra il numero di itemset significativi restituiti applicando il taglio delle transazioni e quello relativo al caso standard.

Come precedentemente accennato, sono stati condotti tre test, che si differenziano tra loro per le taglie di itemset studiate: rispettivamente per gli itemset di lunghezza 2, 3 e 4.

4.2.4 Risultati ottenuti

2-itemset

Il primo test si occupa di verificare l'efficacia e l'efficienza della soluzione proposta relativamente alla scoperta di 2-itemset statisticamente significativi. In Tabella 4.5 sono riportati i tempi di esecuzione (tempo complessivo necessario all'esecuzione delle quattro fasi che costituiscono l'algoritmo) ed il numero di 2-itemset significativi scoperti relativamente al caso standard di riferimento. Tali valori saranno poi utilizzati come riferimento per il calcolo dei valori di efficienza ed efficacia.

Tabella 4.5: Valori di riferimento per 2-itemset

Dataset	Tempo (s)	Itemset
Retail	59	1
Pumsb*	183	25
Bmspos	462	2
Kosarak	876	1
Plants	138	362

Il grafico in Figura 4.2 riporta sulle ordinate i valori di efficienza ottenuti singolarmente su ciascun dataset al variare dei valori di *rate*.

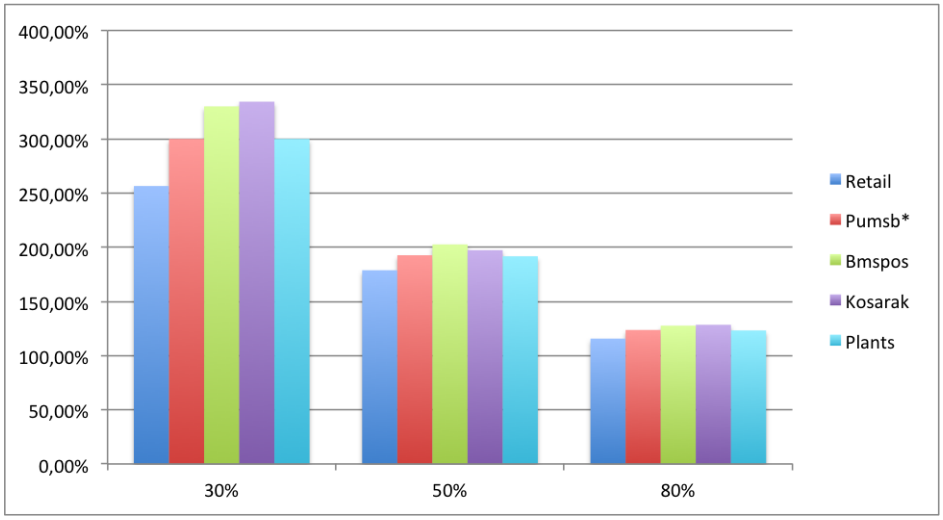

Figura 4.2: Valori di efficienza ottenuti al variare del *rate* per 2-itemset

Dai risultati ottenuti è stato calcolato, per ciascun valore di *rate* studiato, le misure di efficienza media e di deviazione standard associate alla scoperta

di 2-itemset statisticamente significativi all'interno del campione di dataset considerato. I valori calcolati sono riportati in Tabella 4.6.

Tabella 4.6: Efficienza media e varianza per 2-itemset

Rate	**Efficienza media**	**S.q.m.**
30%	304,17%	27,87%
50%	192,60%	7,93%
80%	123,72%	4,52%

Il grafico in Figura 4.3 invece riporta sulle ordinate i valori di efficacia ottenuti singolarmente su ciascun dataset al variare dei valori di *rate*.

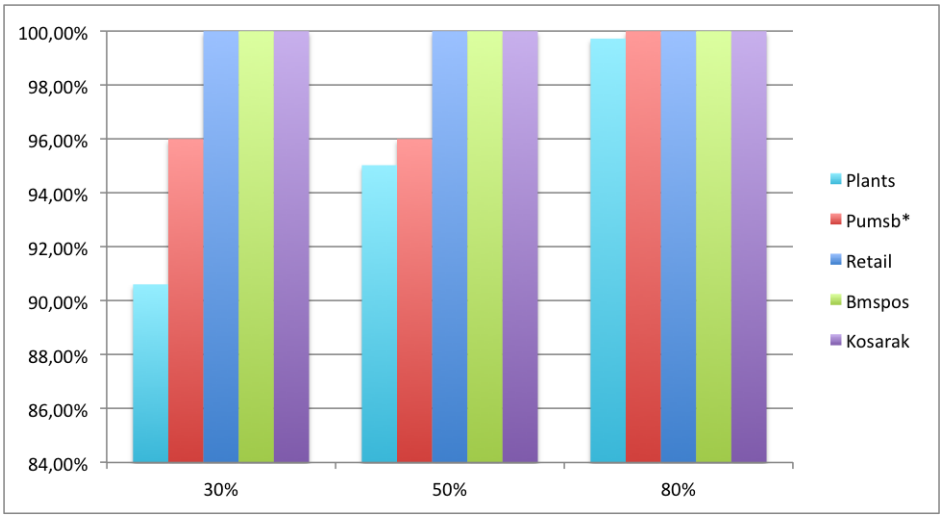

Figura 4.3: Valori di efficacia ottenuti al variare del *rate* per 2-itemset

Dai risultati ottenuti è stato calcolato, per ciascun valore di *rate* studiato, le misure di efficacia media e di deviazione standard associate alla scoperta di 2-itemset statisticamente significativi all'interno del campione di dataset considerato. I valori calcolati sono riportati in Tabella 4.7.

Tabella 4.7: Efficacia media e varianza per 2-itemset

Rate	**Efficacia media**	**S.q.m.**
30%	97,32%	3,70%
50%	98,21%	2,22%
80%	99,94%	0,11%

3-itemset

Vengono ora mostrati i risultati ottenuti relativamente al test su itemset di taglia 3. In Tabella 4.8 sono riportati i tempi di esecuzione ed il numero di 3-itemset significativi restituiti relativamente al caso standard di riferimento.

Tabella 4.8: Valori di riferimento per 3-itemset

Dataset	**Tempo (s)**	**Itemset**
Retail	83	3
Pumsb*	485	242
Bmspos	1252	24
Kosarak	966	1
Plants	174	36490

Il grafico in Figura 4.4 riporta sulle ordinate i valori di efficienza ottenuti singolarmente su ciascun dataset al variare dei valori di *rate*.

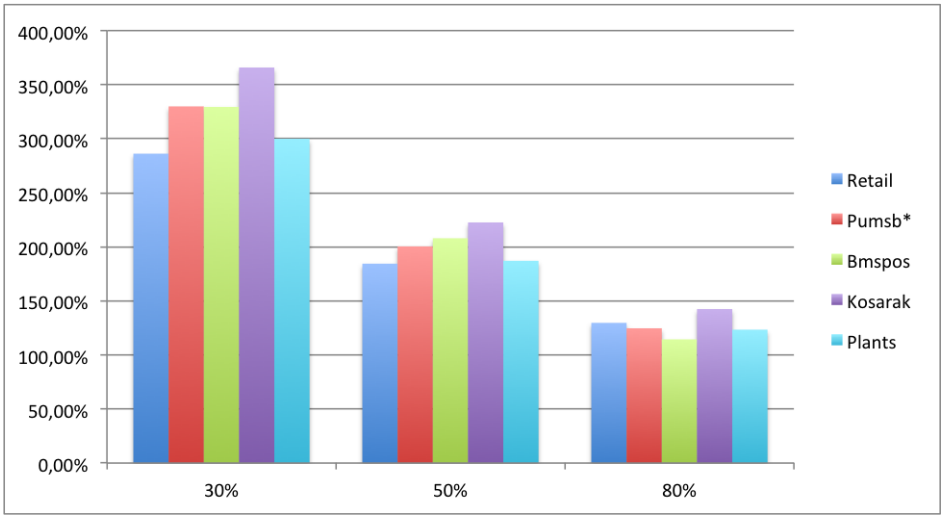

Figura 4.4: Valori di efficienza ottenuti al variare del *rate* per 3-itemset

Dai risultati ottenuti è stato calcolato, per ciascun valore di *rate* studiato, le misure di efficienza media e di deviazione standard associate alla scoperta di 3-itemset statisticamente significativi all'interno del campione di dataset considerato. I valori calcolati sono riportati in Tabella 4.9.

Tabella 4.9: Efficienza media e varianza per 3-itemset

Rate	**Efficienza media**	**S.q.m.**
30%	322,30%	27,61%
50%	200,50%	14,01%
80%	126,94%	9,20%

Il grafico in Figura 4.5 invece riporta sulle ordinate i valori di efficacia ottenuti singolarmente su ciascun dataset al variare dei valori di *rate*.

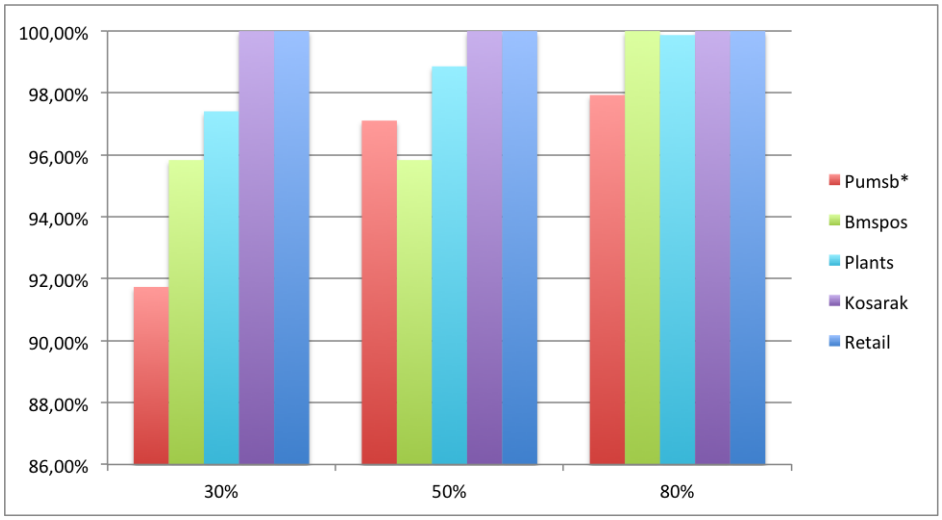

Figura 4.5: Valori di efficacia ottenuti al variare del *rate* per 3-itemset

Dai risultati ottenuti è stato calcolato, per ciascun valore di *rate* studiato, le misure di efficacia media e di deviazione standard associate alla scoperta di 3-itemset statisticamente significativi all'interno del campione di dataset considerato. I valori calcolati sono riportati in Tabella 4.10.

Tabella 4.10: Efficacia media e varianza per 3-itemset

Rate	**Efficacia media**	**S.q.m.**
30%	97,00%	3,07%
50%	98,36%	1,65%
80%	99,56%	0,82%

4-itemset

Vengono ora mostrati i risultati ottenuti relativamente al test su itemset di taglia 4. In Tabella 4.11 sono riportati i tempi di esecuzione ed il numero di 4-itemset significativi restituiti relativamente al caso standard di riferimento.

Tabella 4.11: Valori di riferimento per 4-itemset

Dataset	**Tempo (s)**	**Itemset**
Retail	267	6
Pumsb*	865	2461
Bmspos	3242	842
Kosarak	1621	12
Plants	276	802144

Il grafico in Figura 4.6 riporta sulle ordinate i valori di efficienza ottenuti singolarmente su ciascun dataset al variare dei valori di *rate*.

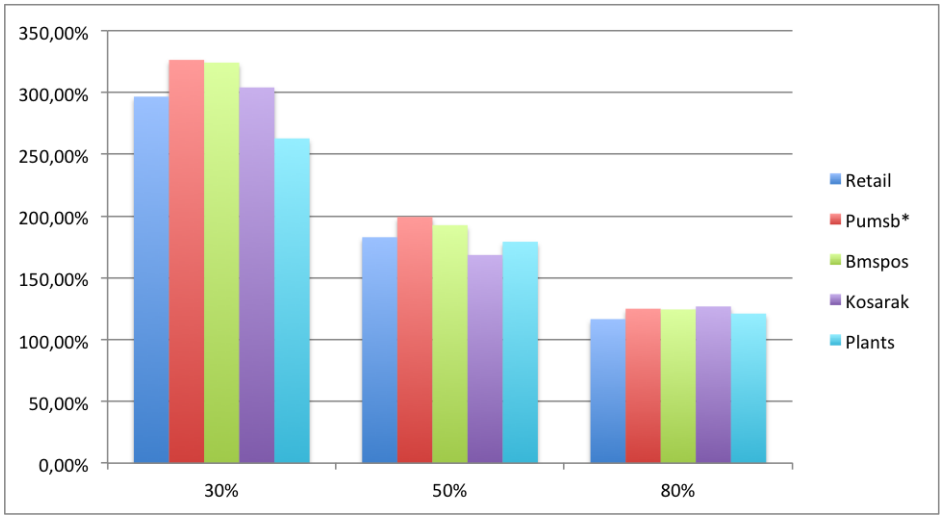

Figura 4.6: Valori di efficienza ottenuti al variare del *rate* per 4-itemset

Dai risultati ottenuti è stato calcolato, per ciascun valore di *rate* studiato, le misure di efficienza media e di deviazione standard associate alla scoperta di 4-itemset statisticamente significativi all'interno del campione di dataset considerato. I valori calcolati sono riportati in Tabella 4.12.

Tabella 4.12: Efficienza media e varianza per 4-itemset

Rate	**Efficienza media**	**S.q.m.**
30%	302,85%	23,02%
50%	184,51%	10,69%
80%	122,78%	3,61%

Il grafico in Figura 4.7 invece riporta sulle ordinate i valori di efficacia ottenuti singolarmente su ciascun dataset al variare dei valori di *rate*.

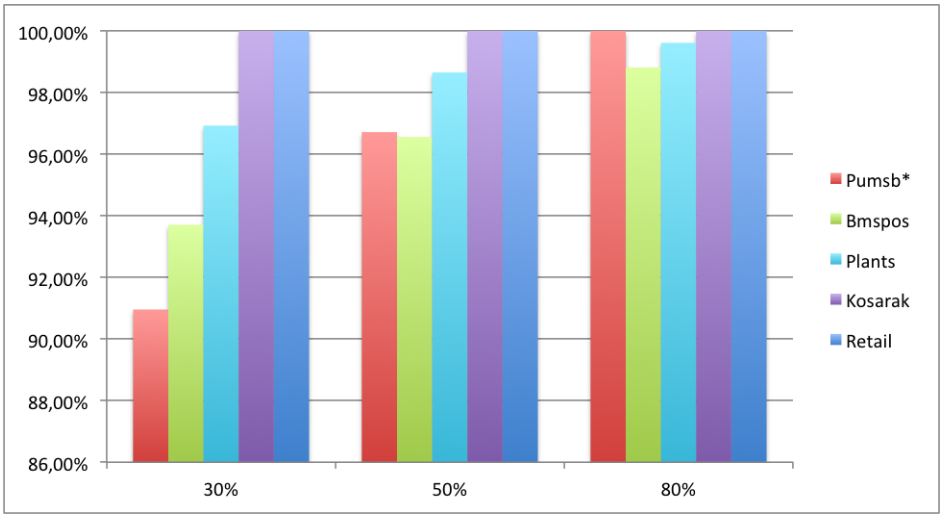

Figura 4.7: Valori di efficacia ottenuti al variare del *rate* per 4-itemset

Dai risultati ottenuti è stato calcolato, per ciascun valore di *rate* studiato, le misure di efficacia media e di deviazione standard associate alla scoperta di 4-itemset statisticamente significativi all'interno del campione di dataset considerato. I valori calcolati sono riportati in Tabella 4.13.

Tabella 4.13: Efficacia media e varianza per 4-itemset

Rate	**Efficacia media**	**S.q.m.**
30%	96,35%	3,55%
50%	98,42%	1,51%
80%	99,73%	0,46%

4.2.5 Riduzione dell'efficacia

Le transazioni di ciascun dataset random, necessarie al calcolo delle probabilità empiriche che permettono di stimare le misure di b_1 e b_2 e ottenere il valore della soglia di supporto s_{min}, sono costruite utilizzando un generatore software di numeri casuali compresi nell'intervallo $[0, 1]$ con probabilità uniforme.

Quando viene applicato l'approccio sviluppato in (Kirsch *e altri*, 2009) per la scoperta di k-itemset statisticamente significativi, vengono calcolate h soglie di supporto s_i con $0 \leq i < h$, al fine di determinare il valore di s^*, e per ognuna di esse viene definito empiricamente il valore atteso $\lambda_i = E[\hat{Q}_{k,s_i}]$ attraverso i dataset random generati. È evidente perciò come le misure del supporto s_{min} e dei valori attesi λ_i, da cui s^* dipende, vengono condizionati dal modo in cui le transazioni dei dataset random sono state costruite.

Partendo da questo presupposto vengono ora analizzati i parametri s_{min} e λ_i calcolati durante l'applicazione del toolbox al dataset *Retail* per la scoperta di 3-itemset statisticamente significativi al variare del valore di *rate* imposto. In Tabella 4.14 vengono mostrati i risultati ottenuti.

Tabella 4.14: Parametri s_{min} e λ_i ottenuti

	Rate 30%		**Rate 50%**		**Rate 80%**		**Rate 100%**	
i	s_i	λ_i	s_i	λ_i	s_i	λ_i	s_i	λ_i
0	4615	0,052	4577	0,049	4547	0,05	4537	0,048
1	4616	0,052	4578	0,049	4548	0,046	4538	0,048
2	4617	0,052	4579	0,048	4549	0,045	4539	0,043
3	4619	0,05	4581	0,046	4551	0,044	4541	0,041
4	4623	0,046	4585	0,044	4555	0,038	4545	0,037
5	4631	0,041	4593	0,035	4563	0,029	4553	0,025
6	4647	0,026	4609	0,024	4579	0,018	4569	0,014
7	4679	0,012	4641	0,008	4611	0,004	4601	0,003

I valori s_0 sono stati evidenziati in rosso dato che, come descritto nello pseudocodice di Paragrafo 2.6, vengono inizializzati al valore s_{min} stimato attraverso l'approssimazione di Poisson. Nell'approccio sviluppato in (Kirsch *e altri*, 2009) il calcolo delle soglie s_i procede finché le condizioni

$$\mathbf{Pr}(\text{Poisson}(\lambda_i) \geq Q_{k,s_i}) \leq \alpha_i) \text{ e } (Q_{k,s_i} \geq \beta_i \lambda_i)$$

non vengono soddisfatte, di conseguenza l'ultimo valore delle sequenze di soglie di supporto s_i, associate a ciascun valore di *rate*, sono state evidenziate di verde in quanto coincidono con il valore s^*.

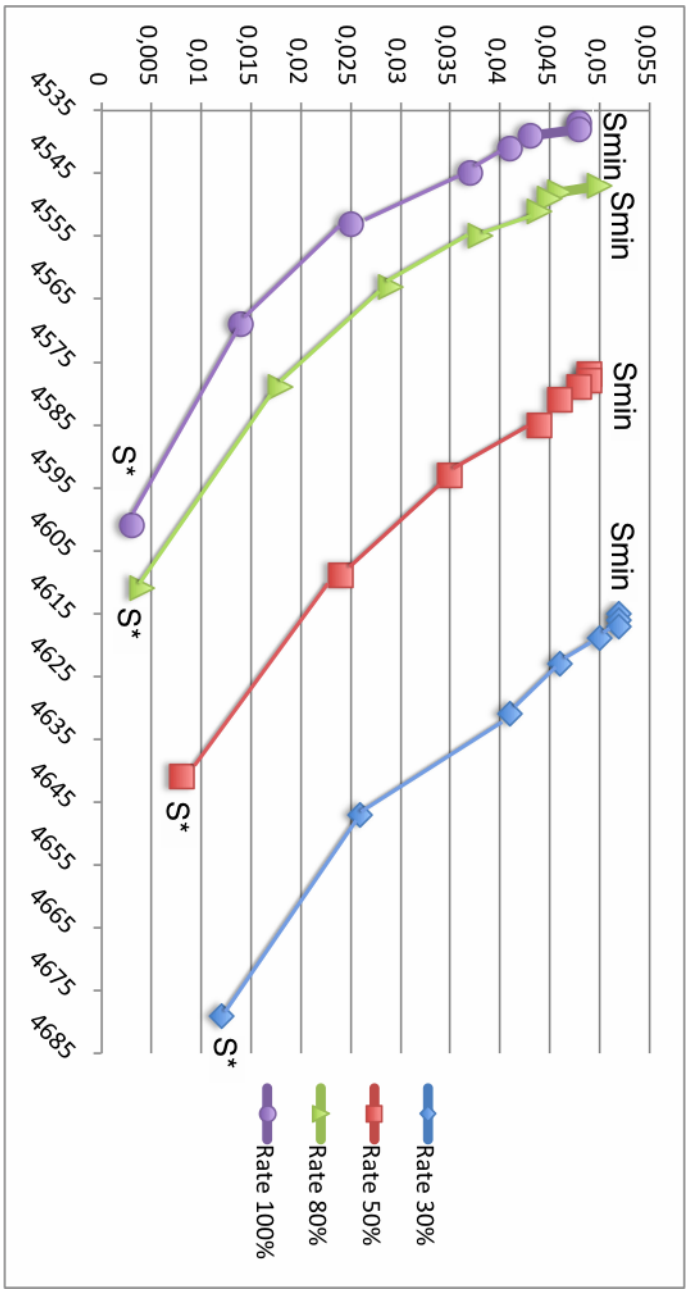

Figura 4.8: Grafico dei parametri s_{min} e λ_i ottenuti

In Figura 4.8 vengono mostrati graficamente i valori in Tabella 4.14. Sulle ascisse troviamo i valori di supporto delle soglie s_i mentre sulle ordinate i valori di λ_i ottenuti. I punti nel grafico, suddivisi per soglie di *rate* applicate, rappresentano le intersezioni delle coppie di valori $<s_i, \lambda_i>$ misurati.

Da ciascuna delle serie di punti del grafico si evince come il valore di λ_0 sia relativamente elevato e che con l'aumentare dei valori delle soglie s_i i rispettivi valori λ_i diminuiscono con andamento quasi lineare. Ciò è dovuto al fatto che man mano che le soglie di supporto crescono il numero atteso di itemset frequenti rispetto a tali soglie diminuisce di conseguenza.

Dal grafico è possibile notare come la diminuzione delle soglie di *rate* provochi un aumento dei valori di s_{min} e λ per valori di supporto equivalenti (ad esempio nel caso del supporto 4579, con *rate* 80% λ vale $0,018$ mentre con *rate* 50% λ vale $0,048$). Tale fenomeno è dovuto fondamentalmente ad un aumento generale delle probabilità empiriche con cui un qualunque itemset X può assumere supporto almeno pari ad un certo valore di supporto s in un generico dataset random, e dato che i valori s_{min} e λ sono calcolati sulla base di queste probabilità anch'essi aumentano con il diminuire delle soglie di *rate*. Il taglio delle transizioni perciò influenza i valori delle probabilità empiriche, tale situazione è dovuta fondamentalmente ai seguenti due fattori:

- Il primo riguarda appunto la diminuzione del numero di transazioni generate. Siccome la costruzione delle transazioni viene fatta attraverso l'utilizzo di un generatore software casuale di numeri uniformemente distribuiti nell'intervallo $[0, 1]$ e indipendenti fra loro, con i quali si decide di volta in volta se inserire un item all'interno di una transazione oppure no, diminuendo il numero di transazioni da generare inevitabilmente si diminuisce anche la lunghezza della sequenza di numeri casuali prodotta dal generatore software.
  Il teorema di *Glivenko-Cantelli* afferma che: Data una sequenza di variabili aleatorie $Z_1, Z_2, \ldots, Z_n$ indipendenti e uniformemente distribuite con funzione di distribuzione $F(z) = \Pr(Z_i \leq z)$, sia

  $$F_n(z) = \frac{1}{n} \sum_{i=1}^{n} 1\{Z_i \leq z\}$$

  la funzione di distribuzione empirica, allora si ha che

  $$\lim_{n \to \infty} sup_{z \in \mathbb{R}} |F(z) - F_n(z)| = 0 \text{ con probabilità } 1,$$

  e quindi la sequenza $Z_1, Z_2, \ldots, Z_n$ converge uniformemente alla distribuzione $F(z)$.

Dal seguente teorema si ricava perciò che maggiore è la lunghezza della sequenza di valori casuali restituiti dal generatore software maggiore è la convergenza di questa ad una variabile aleatoria uniforme di estremi 0 e 1. Il taglio delle transazioni quindi fa sì che la convergenza della sequenza di numeri casuali risulti più debole. Analizzando il generatore di numeri casuali utilizzato, si è notato come la varianza della sequenza dei numeri restituiti quando vengono generate un numero ridotto di transazioni sia mediamente minore rispetto al caso in cui venga generato un numero superiore di transazioni e quindi sequenze di numeri casuali di lunghezza maggiore. La varianza ridotta sta a significare che i valori casuali restituiti sono più accentrati verso la media della distribuzione (1/2). Dato che un generico item i viene inserito all'interno di una qualsiasi transazione t quando il valore casuale restituito dal generatore è inferiore alla frequenza associata a tale item, ossia al rapporto $s_D(i)/|D|$, un maggior accentramento dei valori verso la media della distribuzione non fa altro che favorire maggiormente gli item con frequenza associata maggiore a tale media, e sfavorire quelli con frequenza associata minore, ma dato che nel toolbox vengono considerati solo gli itemset con frequenza relativamente elevata si registra solamente un aumento dei supporti per tali itemset.

- Dato che le frequenze con cui alcuni item compaiono all'interno delle transazioni risulta superiore rispetto al caso standard, al momento della moltiplicazione di queste per il numero di transazioni in D in modo tale da ottenere il supporto associato a tale item, anche una minima differenza sulle frequenze viene amplificata. Risulta perciò che i supporti medi degli item, e quindi anche degli itemset, risultino significativamente maggiori rispetto al caso standard, incidendo quindi anche sulle probabilità empiriche calcolate per ciascun itemset frequente.

Concludendo, dato che i valori di $b_1(s)$ e $b_2(s)$ diminuiscono all'aumentare del supporto s, essendo questi dipendenti direttamente dalle probabilità empiriche p_X e p_{XY} per una generica coppia di itemset frequenti X e Y, la soglia di supporto s_{min} calcolata risulta essere più elevata rispetto al caso standard in modo da garantire il rispetto del limite superiore ϵ imposto sulla somma dei coefficienti $b_1(s_{min})$ e $b_2(s_{min})$. Inoltre, dato che il calcolo dei coefficienti λ dipende anch'esso dalle misure empiriche effettuate sui dataset random si registra anche in questo caso un aumento medio dei valori misurati, giustificando perciò l'aumento delle soglie di supporto s^* al diminuire delle soglie di *rate* imposte. L'aumento dei valori di s^* provoca una diminuzione del numero di itemset frequenti rispetto a tale soglia nel dataset D assegnato, e

quindi una diminuzione dell'efficienza del toolbox quantificabile in un minor numero di itemset statisticamente significativi identificati.

Nelle Tabelle 4.15 e 4.16 viene data ulteriore dimostrazione dell'aumento generale dei valori di probabilità empirica associati agli itemset dei dataset random con il diminuire delle soglie di *rate* imposte.

Tabella 4.15: Parametri ottenuti per *Kosarak* con $k = 2$

	Rate 30%		**Rate 50%**		**Rate 80%**		**Rate 100%**	
i	s_i	λ_i	s_i	λ_i	s_i	λ_i	s_i	λ_i
0	286336	0,049	285960	0,049	285786	0,048	285644	0,049
1	286337	0,049	285961	0,049	285787	0,048	285645	0,046
2	286338	0,049	285962	0,049	285788	0,048	285646	0,046
3	286340	0,049	285964	0,049	285790	0,047	285648	0,046
4	286344	0,049	285968	0,048	285794	0,047	285652	0,045
5	286352	0,048	285976	0,047	285802	0,046	285660	0,044
6	286368	0,045	285992	0,044	285818	0,041	285676	0,042
7	286400	0,042	286024	0,043	285850	0,035	285708	0,038
8	286464	0,034	286088	0,037	285914	0,023	285772	0,031
9	286592	0,027	286216	0,026	286042	0,014	285900	0,017
10	286848	0,014	286472	0,011	286298	0,005	286156	0,004
11	287360	0,001	286984	0	286810	0	286668	0

Tabella 4.16: Parametri ottenuti per *Bmspos* con $k = 2$

	Rate 30%		**Rate 50%**		**Rate 80%**		**Rate 100%**	
i	s_i	λ_i	s_i	λ_i	s_i	λ_i	s_i	λ_i
0	70402	0,042	70222	0,052	70108	0,05	69940	0,05
1	70402	0,042	70223	0,052	70109	0,049	69941	0,049
2	70404	0,042	70224	0,051	70110	0,049	69942	0,048
3	70406	0,041	70226	0,051	70112	0,047	69944	0,048
4	70410	0,04	70230	0,051	70116	0,047	69948	0,047
5	70418	0,039	70238	0,048	70124	0,042	69956	0,046
6	70434	0,037	70254	0,041	70140	0,036	69972	0,039
7	70466	0,031	70286	0,034	70172	0,028	70004	0,027
8	70530	0,026	70350	0,021	70236	0,013	70068	0,016
9	70658	0,012	70478	0,007	70364	0,005	70196	0,007
10	70914	0,001	70734	0,001	-	-	-	-

4.2.6 Itemset significativi restituiti

Il questa sezione si vuole fornire una prova evidente di come gli itemset significativi restituiti dal toolbox quando viene applicata la strategia del taglio delle transazioni sia sempre un sottoinsieme degli itemset significativi scoperti nel caso standard.

Per la prova si è deciso di analizzare i 3-itemset statisticamente significativi del dataset *Bmspos* restituiti attraverso la procedura che implementa l'approccio sviluppato in (Kirsch *e altri*, 2009) al variare delle soglie di *rate* rispettivamente per 30%, 50%, 80% e 100% (caso standard). I risultati sono riportati in Tabella 4.17.

Tabella 4.17: 3-itemset scoperti nel dataset *Bmspos*

id	3-**itemset**	**Rate 100%**	**Rate 80%**	**Rate 50%**	**Rate 30%**
1	{1, 3, 8 }	√	√	√	√
2	{1, 3, 9}	√	√	√	√
3	{1, 3, 10}	√	√	√	√
4	{1, 3, 30 }	√	√	√	√
5	{1, 3, 35 }	√	√	√	√
6	{1, 3, 42 }	√	√	√	√
7	{3, 8, 9}	√	√	√	√
8	{3, 8, 10 }	√	√	√	√
9	{3, 8, 30}	√	√	√	√
10	{3, 8, 35 }	√	√	√	√
11	{3, 8, 42}	√	√	√	√
12	{3, 8, 86}	√	√		
13	{3, 8, 109 }	√	√	√	√
14	{3, 8, 172 }	√	√	√	√
15	{3, 9, 10 }	√	√	√	√
16	{3, 9, 30}	√	√	√	√
17	{3, 9, 35}	√	√	√	√
18	{3, 9, 42}	√	√	√	√
19	{3, 10, 35}	√	√	√	√
20	{3, 23, 34}	√	√	√	√
21	{3, 30, 35}	√	√	√	√
22	{3, 33, 35}	√	√	√	√
23	{3, 35, 42}	√	√	√	√

Nel caso standard la procedura sviluppata in (Kirsch *e altri*, 2009) restituisce 23 itemset significativi di taglia pari a 3. Lo stesso insieme di itemset

viene restituito anche quando *rate = 80%*. Nei casi in cui *rate* viene impostato a 50% e 30% vengono restituiti in enteambi i casi 22 3-itemset statisticamente significativi, entrambi sottoinsieme della soluzione restituita nel caso standard. Effettuando la stessa prova sugli itemset significativi di taglia 2 del dataset *Pumsb** otteniamo i seguenti risultati riportati nelle Tabelle 4.18.

Tabella 4.18: 2-itemset scoperti nel dataset *Pumsb**

id	**2-itemset**	**Rate 100%**	**Rate 80%**	**Rate 50%**	**Rate 30%**
1	{84, 277}	√	√	√	√
2	{84, 4493}	√	√	√	√
3	{84, 4496}	√	√	√	√
4	{84, 4499}	√	√	√	√
5	{84, 4502}	√	√	√	√
6	{161, 277}	√	√	√	√
7	{161, 4493}	√	√	√	√
8	{161, 4496}	√	√	√	√
9	{161, 4499}	√	√	√	√
10	{161, 4502}	√	√	√	√
11	{168, 277}	√	√	√	√
12	{168, 4493}	√	√	√	√
13	{168, 4496}	√	√	√	√
14	{168, 4499}	√	√	√	√
15	{168, 4502}	√	√	√	√
16	{277, 4496}	√	√		
17	{277, 4499}	√	√	√	√
18	{277, 4502}	√	√	√	√
19	{4493, 4499}	√	√	√	√
20	{4493, 4502}	√	√	√	√
21	{4496, 4499}	√	√	√	√
22	{4496, 4502}	√	√	√	√
23	{4499, 4502}	√	√	√	√
24	{4933, 4937}	√	√	√	√
25	{197, 7072}	√	√	√	

Nel caso standard la procedura sviluppata in (Kirsch *e altri*, 2009) restituisce 25 itemset significativi di taglia pari a 2. Lo stesso insieme di itemset viene restituito anche quando *rate = 80%*. Nei casi in cui *rate* viene impostato a 50% e 30% vengono rispettivamente restituiti 24 e 23 2-itemset statisticamente significativi, entrambi sottoinsieme della soluzione restituita

nel caso standard.

La stessa situazione si verifica anche con gli insiemi di itemset significativi scoperti nei restanti dataset studiati, dove però la cardinalità dell'insieme delle soluzioni ottenute non consente di elencare in maniera pratica ciascun itemset significativo all'interno del documento.

4.2.7 Considerazioni

Dai test condotti è possibile notare come l'efficienza aumenti al diminuire dei valori di *rate* imposti. Questo dimostra come la generazione di dataset random rappresenti la fase più onerosa dal punto di vista computazionale e quindi quella che più influenza i tempi di esecuzione finali dell'algoritmo.

Dai grafici in Figura 4.2, 4.4 e 4.6 è possibile notare come la deviazione standard dell'efficienza aumenti con il diminuire delle soglie di *rate* imposte. Questa situazione è dovuta essenzialmente al differente numero di itemset statisticamente significativi scoperti nel dataset reale di riferimento e quindi al differente overhead richiesto successivamente all'applicazione del metodo di Monte Carlo per processare tali itemset. I valori di deviazione standard calcolati sono più elevati per la soglia di *rate* 30% dato che il peso dei tempi di esecuzione nelle fasi successive a quella relativa il metodo di Monte Carlo è maggiore rispetto al tempo di esecuzione complessivo, rendendo quindi più sensibili i tempi di esecuzione del toolbox al numero di itemset significativi scoperti nel dataset.

Per quanto riguarda l'efficacia invece, dai grafici in Figura 4.3, 4.5 e 4.7 e dall'analisi effettuata nel Paragrafo 4.2.5 è possibile notare come questa diminuisca al decrescere della soglia di *rate* imposta, ciò significa che la soluzione proposta comporta anche degli svantaggi quantificabili in un minor numero di itemset statisticamente significativi identificati all'interno del dataset assegnato.

Tuttavia dai test risulta che mediamente il numero di itemset restituiti al caso peggiore (*rate* 30%) è pari a circa il 97% rispetto al numero relativo nel caso standard (*rate* 100%), con uno scarto quadratico medio che non supera l'1%.

Si vuole ora dimostrare come il taglio delle transazioni non infici sul massimo valore di *FDR* garantito. Al momento del calcolo del valore di s^*, come descritto nel Paragrafo 2.6, viene rigettata la *null hypothesis* H_0^i associata alla soglia di supporto s_i, con $0 \leq i < h$, se e solo se le seguenti condizioni sono rispettate

$$\Pr(\text{Poisson}(\lambda_i) \geq Q_{k,s_i}) \leq \alpha_i \quad \text{e} \quad Q_{k,s_i} \geq \beta_i \lambda_i$$

$$\text{dove } \alpha_i \text{ è definita come } \sum_{i=0}^{h-1} \alpha_i \leq \alpha$$

$$\text{mentre } \beta_i \text{ è definita come } \sum_{i=0}^{h-1} \beta_i^{-1} \leq \beta.$$

Per il teorema esposto nel Paragrafo 2.6, dato che $\sum_{i=0}^{h-1} \alpha_i \leq \alpha$, tutte le *null hypothesis* H_0^i, con $0 \leq i < h$, vengono rigettate correttamente con probabilità $1 - \alpha$. Per quanto riguarda invece il limite massimo di *FDR* consentito, ovvero β, nel Paragrafo 4.2.5 si è dimostrato empiricamente come al diminuire delle soglie di *rate* le misure delle probabilità empiriche calcolate all'interno dei dataset random aumentino di conseguenza. Ne risulta quindi che, dato un generico dataset $D = \{t_1, t_2, \ldots, t_{|D|}\}$ di $|D|$ transazioni costruite su un insieme $I = \{i_1, i_2, \ldots, i_n\}$ di n item e dato un valore k, che corrisponde alla taglia di itemset frequenti e statisticamente significativi che si vuole scoprire, per un qualunque valore di *rate* inferiore al caso standard (100) si ha che la probabilità empirica di un generico itemset X di assumere supporto almeno pari alla soglia s, quando s è maggiore o uguale del supporto s_{min} calcolato applicando il taglio delle transazioni, è sempre maggiore o uguale della corrispondente probabilità empirica calcolata nel caso standard. Di conseguenza il valore di $\lambda_{cut} = E[\hat{Q}_{k,s}]$, calcolato per la soglia di supporto s, risulterà maggiore del valore $\lambda_{str} = E[\hat{Q}_{k,s}]$ calcolato imponendo il valore di *rate* a 100, perciò se $Q_{k,s} \geq \beta_i \lambda_{cut}$, siccome $\lambda_{cut} \geq \lambda_{std}$, allora sicuramente $Q_{k,s} \geq \beta_i \lambda_{std}$ e perciò l'insieme di itemset frequenti e statisticamente significativi restituiti in questo caso ha *FDR* al più β, dimostrando così la validità dei risultati restituiti anche quando il numero di transazioni da generare durante la fase relativa l'applicazione del metodo di Monte Carlo viene tagliato.

La soluzione per l'incremento dell'efficienza inizialmente illustrata è comprensiva anche di un'attività di *pre-processing* effettuata sugli n item del dataset D precedentemente all'applicazione del metodo di Monte Carlo. Tale attività consente di ridurre i tempi di esecuzione del programma indipendentemente dal valore di *rate* che si andrà ad impostare. Un semplice esempio per rendersi conto di ciò è costituito dal dataset *Kosarak*, il quale, per generare un unico dataset random, richiede solamente 2 secondi a differenza degli 8 minuti necessari in precedenza. I benefici dovuti a questo accorgimento variano a seconda del dataset, ma possiamo affermare che apporta benefici significativi senza alcun svantaggio nella maggior parte dei casi, dato che per la proprietà di antimonotonicità del supporto non vengono generate sequenze di item delle quali "a priori" si può già dire che non risulteranno essere frequenti.

In conclusione, possiamo affermare in base ai test condotti che la riduzione delle soglie di *rate* consente di diminuire considerevolmente i tempi di esecuzione del toolbox sviluppato, incrementando l'efficienza del programma anche del 330%, mentre l'efficacia della soluzione proposta si attesta quasi sempre su livelli superiori al 90%, garantendo al contempo il soddisfacimento dei parametri di qualità, relativi alla confidenza della soluzione e al massimo valore di *FDR* consentito, forniti in input al toolbox.

4.3 Miglioramenti apportati

Tra gli scopi del toolbox presentato in questo documento vi è anche quello di migliorare le prestazioni, in termini di efficienza dei tempi di esecuzione, rispetto ad un tool software (denominato *find_threshold*) precedentemente creato per la ricerca di itemset frequenti significativi attraverso l'applicazione della procedura descritta in (Kirsch *e altri*, 2009). Per questo motivo sono stati condotti alcuni test con l'obiettivo di individuare eventuali migliorie e vantaggi apportati dal nuovo toolbox realizzato.

4.3.1 Test e risultati ottenuti

I test condotti si sono concentrati fondamentalmente sull'efficienza relativa all'implementazione del metodo di Monte Carlo che, come abbiamo potuto vedere nel Paragrafo 4.2, influenza maggiormente i tempi di esecuzione del programma. Per il confronto sono stati utilizzati tre dataset, le cui proprietà sono riportate in Tabella 4.19.

Tabella 4.19: Proprietà dei dataset selezionati

Dataset	**n**	$\|D\|$
Mushroom	119	8124
Bms1	497	59601
Pumsb*	2088	49046

Come si può notare, il primo dataset, ovvero *Mushroom*, è relativamente ridotto, mentre i rimanenti due dataset, *Bms1* e *Pumsb**, sono di medie dimensioni. Questa scelta è stata fatta per poter confrontare le prestazioni dei due tool software su dataset sia di piccole che medio/grandi estensione, in modo che i risultati finali dei test possano avere validità più generale possibile.

I test sono stati condotti confrontando i tempi di esecuzione necessari

alla generazione di 10 dataset random e al calcolo della soglia di supporto minima s_{min}, per itemset di taglia k compresa nell'intervallo $[2, 4]$, variando in ciascuno dei test unicamente il dataset di ingresso.

Tabella 4.20: Tempi di esecuzione ottenuti dai test

Tool	**Dataset**	**Tempo (s)**
find_threshold	Mushroom	2250
find_threshold	Bms1	125912
find_threshold	Pumsb*	223030
toolbox	Mushroom	1
toolbox	Bms1	3
toolbox	Pumsb*	2

In Tabella 4.20 sono riportati i tempi di esecuzione, al variare del dataset reale di ingresso, di entrambi i tool software.

Analizzando i dati è possibile notare come i tempi di esecuzione del programma presentato in questo documento siano decisamente migliori rispetto ai tempi relativi al tool *find_threshold*. Questa situazione è dovuta fondamentalmente al fatto che il nuovo toolbox utilizza il metodo Apriori per la ricerca di itemset frequenti all'interno dei dataset random, mentre nel tool *find_threshold*, ad ogni transazione random generata, vengono calcolate tutte le possibili combinazioni di k item presenti all'interno della transazione stessa, potenzialmente anche quelle che non risulteranno essere frequenti al termine della procedura, e successivamente memorizzate all'interno di una apposita Hash Table.

Questa seconda soluzione oltre a dover calcolare tutte le possibili combinazioni di k item presenti in ciascuna delle transazioni generate, ripete la procedura per ogni taglia k di itemset che si intende studiare, perciò se facciamo riferimento ai test condotti, dato che Δ, ossia il numero di dataset random generati, vale 10, e che k appartiene all'intervallo $[2, 4]$, si ha che il tool *find_threshold* genera ben 30 dataset random, mentre nel toolbox sviluppato si sfrutta una delle proprietà dell'algoritmo Apriori che consente di estrarre da un dataset tutti gli itemset frequenti di taglia compresa in un determinato intervallo.

In Tabella 4.21 vengono riportati in percentuale i miglioramenti ottenuti.

Tabella 4.21: Miglioramenti ottenuti in termini di efficienza

Dataset	**Efficienza**
Mushroom	+225.000%
Bms1	+4.197.066%
Pumsb*	+11.151.500%

Si noti come in relazione all'algoritmo *find_threshold* siano stati ottenuti ottimi livelli di efficienza, evidenziando così un netto passo in avanti per quanto riguarda l'applicazione del metodo di identificazione di itemset frequenti significativi descritto in (Kirsch *e altri*, 2009) rispetto al passato.

Capitolo 5

Conclusioni

L'obiettivo principale della tesi è quello di realizzare in maniera efficiente un toolbox per l'identificazione di itemset statisticamente significativi all'interno di un generico dataset, che vada ad ampliare e migliorare le funzionalità di un tool software precedentemente sviluppato con lo stesso scopo. Il grosso del lavoro svolto si è concentrato perciò sull'implementazione della simulazione di Monte Carlo, fase in cui le prestazioni dell'algoritmo risentono maggiormente delle dimensioni dei dataset reali forniti in ingresso al programma. La soluzione esposta, come dimostrano le sperimentazioni eseguite, consente di ridurre significativamente il numero di operazioni necessarie allo svolgimento della procedura e ottenere perciò una riduzione significativa dei tempi di esecuzione complessivi del programma, a discapito però di una minore accuratezza nel calcolo dei supporti degli itemset all'interno dei dataset random che si traduce in un minor numero di itemset statisticamente significativi identificati all'interno del dataset di ingresso. Tuttavia, come dimostrato dai test eseguiti, l'efficienza al caso peggiore si mantiene a livelli medi pari al 97%, il che significa che vengono identificati circa il 97% degli itemset significativi restituiti nel caso in cui la soglia di *rate* sia impostata al valore 100. La caratteristica più importante della soluzione implementata, relativa alla validità dell'insieme di itemset statisticamente significativi restituiti, riguarda il fatto che, nonostante venga identificato un numero inferiore di itemset statisticamente significativi rispetto al caso standard, si riesce comunque a garantire un valore di FDR superiormente limitato da β e quindi l'insieme di soluzioni restituite rispetta comunque le misure di qualità imposte al momento della configurazione dei parametri d'ingresso al toolbox.

Lo sviluppo del toolbox ha richiesto, inoltre, di porre particolare attenzione anche alla gestione della memoria. Con dataset di grosse dimensioni, l'esaurimento della memoria è una eventualità di cui bisogna tenere conto. Per questo motivo, in fase di progettazione si è scelto di adoperare il lin-

guaggio *C* come linguaggio di programmazione data la sua elevata efficienza nell'uso delle risorse. In aggiunta, l'implementazione delle quattro strutture dati principali, ovvero *avlItem*, *avlItemsetC*, *avlItemsetE* e *avlItemsetS*, ha consentito di ridurre ulteriormente l'utilizzo di memoria salvando unicamente quelle informazioni considerate strettamente necessarie per il raggiungimento degli obiettivi prefissati.

Il toolbox mette a diretto confronto l'approccio standard basato sul *multi-comparison test* per la scoperta di itemset significativi e il recente approccio sviluppato in (Kirsch *e altri*, 2009) per il calcolo di una soglia di supporto s^* oltre la quale gli itemset sono, con buona confidenza, statisticamente significativi. Dai test condotti è emerso come, a parità di valore di *FDR* massimo garantito, la seconda procedura identifica un maggior numero di itemset significativi. Questo dimostra chiaramente una maggiore efficacia del nuovo approccio sviluppato. Tale risultato è stato raggiunto grazie al fatto che il nuovo metodo prende in considerazione le caratteristiche globali del dataset e non solo le singole scoperte.

5.1 Sviluppi futuri

Entrambi gli approcci utilizzati per identificare gli itemset significativi all'interno di un dataset, non tengono conto degli itemset al di sotto della soglia di supporto minima stimata s_{min} calcolata attraverso la simulazione di Monte Carlo. Eventuali itemset statisticamente significativi caratterizzati da un ridotto supporto non vengono perciò analizzati dal toolbox sviluppato. Ciò lascia aperta una porta per possibili sviluppi futuri del programma che potrebbero riguardare l'implementazione di una procedura che si occupi appositamente degli itemset con supporto inferiore alla soglia s_{min}.

Un ulteriore sviluppo futuro del toolbox può consistere nell'utilizzo di diversi formati di dataset di ingresso oltre a quello specificato nel workshop sul *Frequent Itemset Mining Implementations (FIMI)* e attualmente supportato. La suite di *data mining* denominata *Weka* utilizza ad esempio il formato *Attribute-Relation File Format (ARFF)* per la lettura di dataset. Modificare il toolbox per adattarlo alla lettura di nuovi formati, consentirebbe di estendere la popolazione di dataset su cui poter applicare il nuovo approccio sviluppato in (Kirsch *e altri*, 2009).

Appendice A

Documentazione del codice

In questa appendice vengono descritte esaustivamente alcune porzioni di codice ritenute fondamentali per la corretta esecuzione del toolbox sviluppato. In particolare vengono illustrati:

- La procedura per la costruzione di albero di tipo AVL_ITEM necessario al salvataggio in memoria degli item presenti all'interno di un dataset d'ingresso D, fornito in forma di file di testo, e al calcolo dei supporti associati a ciascuno degli item;

- La procedura per il salvataggio dei supporti di ciascun item in un array di interi;

- La procedura per la deallocazione della memoria occupata dal salvataggio di un albero AVL di tipo AVL_ITEM;

- La procedura per la ricerca di un item all'interno di un albero AVL di tipo AVL_ITEM;

- La procedura per la costruzione di albero di tipo AVL_ITEMSET_C necessario al salvataggio in memoria degli itemset frequenti ottenuti attraverso l'applicazione dell'algoritmo Apriori su un generico dataset random;

- La procedura per la generazione dei dataset random sulla base delle frequenze associate ai singoli item utilizzata nella simulazione di Monte Carlo;

- La procedura per il calcolo del massimo supporto atteso per un itemset di taglia k;

- Un estratto del codice necessario alla generazione di Δ dataset random durante la simulazione di Monte Carlo.

- Un estratto del codice necessario al calcolo dei valori di b_1 e b_2 al fine di ottenere il minimo supporto stimato per ciascuna taglia k di itemset significativi che si intende studiare;

- Un estratto del codice necessario all'applicazione del *multi-comparison test* per la scoperta di itemset statisticamente significativi;

- Un estratto del codice necessario al calcolo della soglia di supporto minima s^* durante l'applicazione dell'approccio per la scoperta di itemset statisticamente significativi sviluppato in (Kirsch *e altri*, 2009).

Questa appendice va quindi a sommarsi alla documentazione del codice già presente all'interno del toolbox al fine di facilitare la comprensione di quest'ultimo e consentire in un futuro di effettuare modifiche, miglioramenti o aggiunte di nuove funzionalità. Il toolbox si compone dei seguenti file:

main.c	
avlItem.h	*avlItem.c*
avlItemsetC.h	*avlItemsetC.c*
avlItemsetE.h	*avlItemsetE.c*
avlItemsetS.h	*avlItemsetS.c*
freqRD.h	*freqRD.c*
monteCarlo.h	*monteCarlo.c*
procedure1.h	*procedure1.c*
procedure2.h	*procedure2.c*
util.h	

Il file principale è quello denominato *main.c*: esso si occupa della lettura dei parametri di ingresso e dell'avvio dei metodi principali per l'applicazione della simulazione di Monte Carlo e l'identificazione degli itemset frequenti attraverso il *multi-comparison test* e il calcolo della soglia di supporto minima s^*. Le implementazioni dei file *avlItem.c*, *avlItemsetC.c*, *avlItemsetE.c* e *avlItemsetS.c* risultano essere molto simili tra loro. Per ridurre l'impiego di memoria da parte del toolbox è stato necessario creare quattro strutture dati di tipo albero AVL molto simili tra loro (AVL_ITEM, AVL_ITEMSET_C, AVL_ITEMSET_E e AVL_ITEMSET_S) le quali però si differenziano per le diverse informazioni che ognuna di esse consente di salvare all'interno dei nodi dell'albero. Di conseguenza in questa appendice verranno trattati solo alcuni metodi relativi ai file *avlItem.c* e *avlItemsetC.c*.

A.1 *buildAvlItem*

```
AVL_ITEM* buildAvlItem(
//Puntatore alla stringa contenente il nome del dataset in input.
char *inputFileName,
//Puntatore alla stringa contenente il nome del file che ospiterà
//il dataset convertito.
char *outputFileName,
//Puntatore alla varibile intera che ospiterà il numero totale di
//item presenti all'interno del dataset.
int *itemCounter,
//Puntatore alla variabile che ospiterà il numero totale di
//transazioni all'interno del dataset.
int *transactionCounter)
{
    bool ht_inc;
    //crea l'albero AVL che ospiterà gli item del dataset di
    //ingresso
    AVL_ITEM *root = (AVL_ITEM *)malloc(sizeof(AVL_ITEM));
    root = NULL;
    //contatore del numero di item scoperti
    *itemCounter = 0;
    //contatore del numero di transazioni scoperte
    *transactionCounter = 0;

    FILE *inputFile;
    //apertura file contenente il dataset di ingresso
    inputFile = fopen(inputFileName, "r");
    FILE *outputFile;
    //apertura file che conterrà il dataset convertito
    outputFile = fopen(outputFileName, "w");

    if (inputFile == NULL){ //controllo esistenza file
        fprintf(stdout, "Errore: il file '%s' non esiste.\n",
            inputFileName);
        return NULL;
    }
    //inizializzo il contatore di caratteri a 0
    int countInputString = 0;
    //creo il buffer per la lettura dei caratteri
    char *inputString = (char *)malloc(32 * sizeof(char));
    int c = fgetc(inputFile);
    //elimino eventutali caratteri inutili all'inizio del file
```

```
41      //di input
42      while(c == ' ' || c == '\n'){
43          c = fgetc(inputFile);
44      }
45      //variabili di stato utili alla corretta lettura del dataset
46      //di input
47      bool spaces = FALSE;
48      bool newTransaction = FALSE;
49      bool endDataset = FALSE;
50
51      while(1){ //inizio lettura
52          while(c == ' ' || c == ',' || c == '\n' || c == EOF){
53              spaces = TRUE;
54              if (c == '\n') newTransaction = TRUE;
55              if (c == EOF) {
56                  endDataset = TRUE;
57                  break;
58              }
59              c = fgetc(inputFile);
60          }
61          //fine del file
62          if (endDataset == TRUE) {
63              (*transactionCounter)++;
64              break;
65          }
66          //trovata una nuova transazione
67          if (newTransaction == TRUE){
68              fprintf(outputFile, "\n");
69              //aggiorno il contatore delle transazioni
70              (*transactionCounter)++;
71          }
72          //nessuna delle condizioni precedenti e quindi è stato
73          //trovata una nuova istanza di un item
74          else if (spaces == TRUE) fprintf(outputFile, " ");
75
76          //reinizializzo le variabili di stato
77          spaces = FALSE;
78          newTransaction = FALSE;
79          endDataset = FALSE;
80          countInputString = 0;
81          while(c != ' ' && c != ',' && c != '\n' && c != EOF){
82              //nuovo carattere letto
83              inputString[countInputString] = c;
```

```
84              countInputString++;
85              c = fgetc(inputFile);
86          }
87          //fine lettura instanza di un item
88          inputString[countInputString] = '\0';
89
90          //ricerco l'item all'interno dell'albero AVL
91          AVL_ITEM *ptrTree = (AVL_ITEM *) searchAvlItem(root,
                inputString);
92
93          if( ptrTree == NULL ){ //item non trovato
94              root = (AVL_ITEM *)insertAvlItem(inputString, root,
                    &ht_inc, *itemCounter);
95              //ricerco di nuovo l'item per poter ottenere
96              //l'identificativo
97              ptrTree = (AVL_ITEM *) searchAvlItem(root, inputString);
98              //aggiorno il dataset convertito
99              fprintf(outputFile, "%d", ptrTree->id);
100             (*itemCounter)++; //aggiorno il contatore degli item
101         }
102         else{ //item trovato
103             //aggiorno il dataset convertito
104             fprintf(outputFile, "%d", ptrTree->id);
105             ptrTree->sprt++; //aggiorno il supporto dell'item trovato
106         }
107     } //while
108     free(inputString); //libero memoria
109
110     fclose(inputFile); //chiudo il file di ingresso
111     fclose(outputFile);//chiudo il file di uscita
112     //ritorno un puntatore alla radice dell'albero AVL costruito
113     return root;
114 }
```

Il metodo *buildAvlItem* consente di costruire un albero AVL in cui ciascun nodo è associato ad un item del dataset assegnato fornito in ingresso al toolbox in forma di file di testo.

inputFileName contiene il nome del file di testo che ospita il dataset assegnato.

outputFileName contiene il nome del file di testo che ospiterà il dataset convertito, ovvero il dataset nel quale le istanze di ciascun item sono state sostituite dall'identificativo associato all'item stesso.

itemCounter è un puntatore ad un intero che al termine della procedura conterrà il numero di item differenti presenti all'interno del dataset assegnato.

transactionCounter è un puntatore ad un intero che al termine della procedura conterrà il numero di transazioni presenti all'interno del dataset assegnato.

Nelle righe da 26 a 29 vengono create le variabili *inputFile* e *outputFile*. Esse consentono rispettivamente di leggere il dataset di ingresso e di scrivere in un file esterno il dataset convertito, ovvero il dataset nel quale le istanze di ciascun item sono state sostituite dall'identificativo associato all'item stesso.
Nelle righe da 51 a 107 il dataset di ingresso viene scandito carattere per carattere. Dato che il dataset deve essere memorizzato secondo le specifiche imposte dal workshop sul *Frequent Itemset Mining Implementations* (FIMI) del 2003, il carattere di spazio viene interpretato come separatore fra due item distinti (riga 81), mentre il carattere per andare a capo è interpretato come inizio di una nuova transazione (riga 67). Le variabili di tipo *bool* definite nelle righe 47, 48 e 49 descrivono i diversi stati che il toolbox può assumere durante la lettura del dataset e consentono di interpretare correttamente la posizione degli item letti.
Nella riga 91 l'item appena letto dal dataset di ingresso viene cercato all'interno dell'albero AVL che si sta costruendo.

- Se l'item non viene trovato allora significa che questa appena letta è la prima istanza dell'item che appare all'interno del dataset e per questo deve essere creato un nuovo nodo all'interno dell'albero da poter associare all'item. Nella riga 94 il nuovo item viene inserito all'interno dell'albero AVL grazie al metodo *insertAvlItem* che provvede ad assegnare un identificativo all'item il quale coinciderà con il valore assunto dal parametro *itemCounter* in quell'esatto momento. Dato che è stato scoperto un nuovo item all'interno del dataset nella riga 100 il valore del parametro *itemCounter* viene incrementato.

- Se l'item viene trovato allora significa che è già stato inserito all'interno dell'albero AVL e quindi è necessario aggiornare il valore del supporto associato a tale item (riga 105).

Per quanto riguarda la conversione del dataset in uno equivalente, nelle righe 99 e 104 vengono scritti nel file puntato dalla variabile *outputFile* gli identificativi degli item man mano che questi vengono letti dal dataset di ingresso, portando avanti la scrittura del dataset convertito parallelamente con la lettura del dataset d'ingresso. In questo modo si ottiene un dataset

equivalente a quello d'ingresso con l'unica differenza che le istanze di ciascun item sono state sostituite dall'identificativo associato all'item stesso.

A.2 *getSprts*

```
1  void getSprts(
2  //Puntatore alla radice del sotto-albero.
3  AVL_ITEM *ptr,
4  //Array di puntatori che alla fine dell'esecuzione del metodo
5  //punteranno ciascuno ad un diverso nodo dell'albero AVL.
6  int supports[])
7  {
8      if(ptr!=NULL){
9          //applicazione ricorsiva nel sotto-albero sinistro
10         getSprts(ptr->lchild, supports);
11         supports[ptr->id] = ptr->sprt;
12         //applicazione ricorsiva nel sotto-albero destro
13         getSprts(ptr->rchild, supports);
14     }
15     return;
16 }
```

Il metodo *getSprts* consente di leggere i supporti associati a ciascun item presente all'interno dell'albero AVL e di salvare questi all'interno di un array di interi.

ptr deve puntare inizialmente alla radice dell'albero AVL che si vuole analizzare e il cui indirizzo è salvato all'interno del parametro *ptr*.

Al termine dell'esecuzione del metodo, l'array di interi *supports*, di dimensione pari al numero di item contenuti nel dataset, andrà a contenere i supporti associati agli item presenti nel dataset assegnato.

Il metodo è di tipo ricorsivo e visita i nodi all'interno dell'albero con ordinamento *in-order*.
Nella riga 11 è possibile notare come l'entry dell'array *supports* associata a ciascun item presente all'interno dell'albero AVL sia quella il cui indice corrisponde all'identificativo dell'item stesso e compreso nell'intervallo $[0, N-1]$. Tale accorgimento consente di accedere in tempo $O(1)$ ai supporti associati in ciascun item dato che, nelle successive fasi dell'algoritmo, le istanze di ciascun item saranno sostituite dall'identificativo associato all'item stesso all'interno dei dataset.

A.3 *destroyAvlItem*

```
1  void destroyAvlItem(
2  //Puntatore alla radice del sotto-albero.
3  AVL_ITEM *ptr)
4  {
5      if (ptr != NULL){
6          //applicazione ricorsiva nel sotto-albero destro
7          destroyAvlItem(ptr->rchild);
8          //applicazione ricorsiva nel sotto-albero sinistro
9          destroyAvlItem(ptr->lchild);
10         //libero la memoria riservata alla variabile item
11         free(ptr->item);
12         //libero la memoria riservata al puntatore ptr
13         free(ptr);
14     }
15     return;
16 }
```

Il metodo *destroyAvlItem* consente di liberare la memoria precedentemente occupata per il salvataggio di un albero AVL di tipo AVL_ITEM. Tale metodo è di fondamentale importanza se si vuole ridurre l'utilizzo di memoria da parte del toolbox ed evitare di incorrere in errori di tipo *segmentation fault* dovuti all'esaurimento della memoria quando si lavora con dataset di grosse dimensioni.

ptr deve puntare inizialmente alla radice dell'albero AVL che si vuole cancellare.

Se l'indirizzo contenuto nel parametro *ptr* è non nullo, allora il metodo *destroyAvlItem* viene ricorsivamente applicato ai sotto-alberi destro e sinistro del nodo riferito dal puntatore *ptr*, come è possibile notare nelle righe 7 e 9. In questo modo si procede ad una eliminazione *bottom-up* dell'albero, dove solo quando i sotto-alberi destro e sinistro di ciascun nodo sono stati eliminati si procederà con l'eliminazione del nodo stesso. Una cancellazione di tipo *top-down* non risulterebbe ugualmente efficace in quanto si perderebbero in questo modo i riferimenti ai nodi sottostanti dell'albero e non si riuscirebbe quindi a portare a termine l'eliminazione dell'albero AVL.

A.4 *searchAvlItem*

```
AVL_ITEM* searchAvlItem(
//puntatore alla radice del sotto-albero che si vuole analizzare.
AVL_ITEM *ptr,
//contiene l'item sotto forma di stringa.
char *string)
{
    if(ptr!=NULL){
        //ricerca nel sotto-albero sinistro
        if (strcmp(string, ptr->item) < 0){
            ptr=searchAvlItem(ptr->lchild, string);
        }
        //ricerca nel sotto-albero destro
        else if (strcmp(string, ptr->item) > 0){
            ptr=searchAvlItem(ptr->rchild, string);
        }
    }
    return(ptr);
}
```

Il metodo *searchAvlItem* consente di cercare un item all'interno di un albero AVL di tipo AVL_ITEM in tempo $O(x\log x)$, dove x è il numero di nodi presenti all'interno dell'albero in quel momento. Dato che gli item all'interno dell'albero sono ordinati secondo valori di stringhe crescenti, si utilizza il metodo *strcmp*, messo a disposizione dalla libreria *string.h* di C, per la ricerca di un item.

ptr deve puntare inizialmente alla radice dell'albero AVL nel quale si vuole cercare l'item.

string contiene il valore reale dell'item letto dal dataset assegnato.

Il metodo qui descritto è di tipo ricorsivo e in base al risultato ottenuto a seguito dell'applicazione del metodo *strcmp* si procede con la ricerca dell'item nel sotto-albero destro oppure nel sotto-albero sinistro nel caso in cui l'item cercato non fosse contenuto nel nodo attuale.

A.5 *buildAvlItemsetC*

```
void buildAvlItemsetC (
//Array di puntatori alle radici degli alberi AVL relativi
//alle diverse taglie di itemset che si intende studiare.
AVL_ITEMSET_C *W[],
//Puntatore alla stringa contenente il nome del file
//in cui sono memorizzati gli itemset frequenti estratti.
char *inputFileName,
//array di interi che andrà ad ospitare il numero di itemset
//scoperti per ciascuna taglia di itemset.
int sizeW[],
//Taglia minima di itemset che si intende studiare.
int minK,
//Taglia massima di itemset che si intende studiare.
int maxK,
//Numero di transazioni contenute nel dataset reale.
int T,
//Indice del dataset random a cui appartengono gli
//itemset frequenti inseriti nell'albero AVL.
int numDataset,
//Numero di dataset random che saranno generati.
int DELTA,
//Array di interi che contiene i supporti massimi attesi.
int maxExpSprt[],
//Array di interi che indica quali taglie di itemset
//prendere in considerazione.
int flag[],
//Array di interi contenente i supporti associati a
//ciascun item.
int supports[])
{
    bool ht_inc;
    //contatore del numero di item letti
    int itemCounter = 0;

    //apertura file contenente il dataset di ingresso
    FILE *inputFile;
    inputFile = fopen(inputFileName, "r");

    if (inputFile == NULL){ //controllo sull'esistenza del file
        fprintf(stdout, "Error: file '%s' does not exist.\n",
            inputFileName);
```

```
41          return;
42      }
43      //inizializzo il contatore di caratteri a 0
44      int countInputString = 0;
45      //creo il buffer per la lettura degli item
46      int *itemset = (int *) malloc(sizeof(int) * maxK);
47      //creo il buffer che ospiterà il supporto degli itemset
48      char *inputString = (char *) malloc(sizeof(char) * 7);
49      //leggo il primo carattere
50      int c = fgetc(inputFile);
51      int sprt;
52      while(1){ //inizio lettura
53          while(c != ' ' && c != '\n' && c != EOF){
54              if (c != ')' && c != '(' ){
55                  inputString[countInputString] = c;
56                  countInputString++;
57              }
58              c = fgetc(inputFile);
59          }
60          //fine del file
61          if(c == EOF){
62              break;
63          }
64          //trovato un nuovo item
65          if(c == ' '){
66              inputString[countInputString] = '\0';
67              //converto la stringa letta in un numero intero
68              itemset[itemCounter] = atoi(inputString);
69              countInputString = 0;
70              //aggiorno il contatore degli item
71              itemCounter++;
72          }
73          //lettura itemset conclusa
74          else{
75              inputString[countInputString] = '\0';
76
77              //conversione del supporto della frequenza
78              //associata all'itemset
79              float freq = atof(inputString);
80              //calcolo il supporto associato
81              sprt = (int) (freq * T / 100);
82
83              //selezione solo quegli itemset i cui item hanno
```

```
84              //tutti supporto maggiore del massimo supporto atteso
85              int item_flag = 1;
86              for (int i = 0; i < itemCounter; i++){
87                  if (supports[itemset[i]] < maxExpSprt[itemCounter -
                        minK]) {
88                      item_flag = 0;
89                  }
90              }
91              //verifico se il supporto dell'itemset è maggiore del
92              //minimo supporto consentito e se attraverso l'array
93              //flag devo oppure no inserire gli itemset di
94              //quella taglia
95              if(sprt >= maxExpSprt[itemCounter - minK] &&
                    flag[itemCounter - minK] == 0 && item_flag){
96                  itemCSort(itemset, 0, itemCounter-1, itemCounter);
97                  //cerco l'itemset al'interno dell'albero AVL
98                  AVL_ITEMSET_C *ptrTree = (AVL_ITEMSET_C *)
                        searchAvlItemsetC(W[itemCounter - minK],
                        itemset, itemCounter);
99                  //itemset non trovato
100                 if (ptrTree == NULL){
101                     //inserisco il nuovo itemset nell'albero AVL
102                     W[itemCounter - minK] = (AVL_ITEMSET_C *)
                            insertAvlItemsetC(itemset, W[itemCounter -
                            minK], &ht_inc, sprt, numDataset, DELTA,
                            itemCounter);
103                     //aggiorno il numero di itemset frequenti trovati
104                     sizeW[itemCounter - minK]++;
105                 }
106                 //itemset trovato
107                 else{
108                     //aggiorno la lista concatenata associata
109                     //all'itemset in modo che contenga il supporto
110                     //che l'itemset ha realizzato nel dataset random
111                     //identificato attraverso il valore della
112                     //variabile numDataset
113                     ptrTree->freqTail->next = (FREQ_RD *)
                            malloc(sizeof(FREQ_RD));
114                     ptrTree->freqTail = ptrTree->freqTail->next;
115                     ptrTree->freqTail->numDataset = numDataset;
116                     ptrTree->freqTail->sprt = sprt;
117                     ptrTree->freqTail->next = NULL;
118                     //aggiorno il massimo supporto se necessario
```

```
119                     if (ptrTree->maxSprt < ptrTree->freqTail->sprt){
120                         ptrTree->maxSprt = ptrTree->freqTail->sprt;
121                     }
122                 }
123             }
124             //resetto le variabili
125             countInputString = 0;
126             itemCounter = 0;
127             itemset = (int *) malloc(sizeof(int) * maxK);
128         }
129         c = fgetc(inputFile);
130     } //while
131     //libero la memoria occupata
132     free(inputString);
133     //chiudo il file di ingresso
134     fclose(inputFile);
135 }
```

Il metodo *buildAvlItemsetC* è un adattamento del metodo *buildAvlItem* al caso del salvataggio in memoria di itemset frequenti. Tale metodo consente di costruire l'albero AVL_ITEMSET_C leggendo gli itemset da un file esterno e consente inoltre di memorizzare per ciascuno di essi i supporti ottenuti lungo tutti i dataset random in cui sono risultati essere frequenti.

W è un array di puntatori alle radici degli alberi AVL relativi alle diverse taglie di itemset che si intende studiare.

inputFileName è un puntatore alla stringa contenente il nome del file in cui sono memorizzati gli itemset frequenti estratti.

sizeW è un array di interi che andrà ad ospitare il numero di itemset scoperti per ciascuna taglia di itemset.

minK è la taglia minima di itemset che si intende studiare.

maxK è la taglia massima di itemset che si intende studiare.

T è il numero di transazioni contenute nel dataset reale.

numDataset è l'indice del dataset random a cui appartengono gli itemset frequenti inseriti nell'albero AVL.

DELTA è il numero di dataset random che saranno generati.

maxExpSprt è un array di interi che contiene i supporti massimi attesi.

flag è un array di interi che indica quali taglie di itemset prendere in considerazione.

supports è un array di interi contenente i supporti associati a ciascun item.

Nelle righe da 34 a 98 il file di ingresso, contenente gli itemset frequenti estratti dal dataset random attraverso l'algoritmo Apriori, viene letto carattere per carattere. Nella riga 70 troviamo una condizione di *if* che consente di salvare in memoria un generico k-itemset appena letto se e solo se:

- Il supporto del k-itemset letto è superiore al massimo supporto atteso per itemset di taglia k il cui valore è memorizzato all'interno dell'array *maxExpSprt*.

- L'albero AVL per gli itemset di taglia k non è già stato costruito in una precedente esecuzione del metodo *buildAvlItemsetC*.

- Nessun item all'interno del k-itemset in esame presenta un supporto nel dataset assegnato inferiore al massimo supporto atteso per itemset di taglia k.

Questo accorgimento consente di risparmiare memoria e quindi di evitare di incorrere in problemi di *segmentation fault* dovuti all'esaurimento di quest'ultima. Inoltre così facendo si aumenta l'efficienza del programma dato che si riduce l'overhead richiesto per la costruzione di un albero AVL di tipo AVL_ITEMSET_C.
Nelle righe da 52 a 130 viene letto il dataset random memorizzato in forma di file di testo in modalità pressoché uguale a quanto fatto nel metodo *buildAvlItem*.
Nelle righe da 113 a 117 si può notare come, nel caso in cui l'itemset letto sia già presente all'interno dell'albero AVL, il supporto che tale itemset assume nel dataset random corrente viene memorizzato in coda alla lista concatenata associata all'itemset insieme all'indice del dataset random a cui l'istanza dell'itemset appartiene. In questo modo è possibile calcolare con esattezza la probabilità empirica che una generica coppia di itemset vicini siano entrambi frequenti all'interno di un dataset random.

A.6 *generateRandomDataset*

```
1  void generateRandomDataset(
2  //array di supporti degli item
3  int supports[],
4  //array di item da prendere in considerazione
5  //per la generazione dei dataset
6  int index_item[],
7  //dimensione di index_item[]
8  int size_item,
9  //nome del file di supporto su cui scrivere
10 //il dataset random
11 char *outputFileName,
12 //numero di item nel dataset reale
13 int N,
14 //numero di transazioni nel dataset reale
15 int T,
16 //numero di transazioni da generare
17 int num_transaction,
18 //indice del dataset random generato
19 int i,
20 //minima frequenza attesa per le taglie di
21 //itemset che si vogliono studiare
22 float minFreq)
23 {
24     //file che ospitera il dataset random generato
25     FILE *outputFile;
26     outputFile = fopen(outputFileName, "w");
27     double freq;
28     //inizializzazione del generatore random
29     const gsl_rng_type * C;
30     gsl_rng * r;
31     C = gsl_rng_default;
32     r = gsl_rng_alloc (C);
33
34     int seed = rand();
35     gsl_rng_default_seed = seed;
36     double rand_val;
37
38     //flag che indica se una transazione generata contiene oppure
           no almeno un item
39     int valid_t = 0;
40
```

```
41      for (int t = 0; t < num_transaction;){
42          valid_t = 0;
43          for (int n = 0; n < size_item; n++){
44              freq = (double) supports[index_item[n]];
45              //calcolo della frequenza dell'item
46              freq = freq / (T * 1.00);
47              //generazione valore random
48              //rand_val = gsl_rng_uniform(r);
49              rand_val = ((rand() % 10000) / 10000.00f);
50              //inserisco l'item nel dataset random
51              if (rand_val < freq){
52                  valid_t = 1;
53                  fprintf(outputFile, "%d ", index_item[n]);
54              }
55
56          }
57          //se è stata generata una transazione non vuota
58          if(valid_t == 1){
59              //passo alla transazione sucessiva
60              fprintf(outputFile, "\n");
61              t++;
62          }
63      }
64      //libero la memoria utilizzata
65      gsl_rng_free(r);
66      fclose(outputFile);
67  }
```

Il metodo *generateRandomDataset* è di fondamentale importanza per l'implementazione della simulazione di Monte Carlo ed il calcolo delle soglie di supporto s_{min} per ciascuna taglia di itemset studiati.

supports è un array di supporti degli item.

index_item è un array degli item da prendere in considerazione per la generazione dei dataset.

size_item contiene la dimensione dell'array *index_item*.

outputFileName contiene il nome del file di supporto su cui scrivere il dataset random.

N è il numero di item nel dataset reale.

T è il numero di transazioni nel dataset reale.

num_transaction è il numero di transazioni da generate.

i è l'indice del dataset random generato.

minFreq è la minima frequenza attesa per le taglie di itemset che si vogliono studiare.

Nelle righe da 29 a 36 viene inizializzato il generatore random di numeri casuali implementato attraverso l'utilizzo delle librerie scientifiche open suorce GSL. Il valore random viene poi salvato all'interno della variabile *rand_val* utilizzando il metodo *gsl_rng_uniform*. Ad ogni iterazione del doppio ciclo nidificato contenuto all'interno del metodo *generateRandomDataset* il valore random generato viene confrontato con la frequenza associata all'item i in esame (riga 51).

- Nel caso in cui il valore random dovesse essere minore della frequenza associata all'item i allora tale item viene inserito all'interno del dataset random, ossia scritto all'interno del file accessibile mediante la variabile *outputFile* precedentemente creata.

- Nel caso in cui il valore memorizzato nella variabile *rand_val* dovesse essere maggiore o uguale della frequenza associata all'item i allora tale item viene semplicemente scartato.

Si noti che durante questa operazione non vengono considerati tutti gli n item appartenenti al dataset random assegnato ma solamente quelli il cui identificativo compare all'interno dell'array *index_item*. Tale array contiene gli identificativi di quegli item il cui supporto è almeno pari al minimo supporto atteso fra le taglie di itemset che si vogliono studiare. In questo modo l'overhead richiesto per la generazione dei dataset random si riduce significativamente.
Il parametro *num_transacton* indica il numero di transazioni che devono essere generate per costruire un dataset random. Questo parametro consente di generare un numero ridotto di transazioni e quindi ridurre ulteriormente l'overhead richiesto da tale metodo il quale, come abbiamo avuto modo di vedere nel Capitolo 4, è il maggior responsabile dell'aumento dei tempi di esecuzione del toolbox con dataset di grosse dimensioni. Il parametro *num_transacton* è una conseguenza della strategia formulata per il miglioramento dell'efficienza del toolbox.

A.7 *maxExpectedSupport*

```
void maxExpectedSupport (
//array che conterrà i massimi supporti attesi.
int maxExpSprt[],
//array di supporti degli item.
int supports[],
//Taglia minima di itemset che si intende studiare.
int minK,
//Taglia massima di itemset che si intende studiare.
int maxK,
//Numero di item contenuti nel dataset reale.
int N,
//Numero di transazioni contenute nel dataset reale.
int T)
{
    for (int k = 0; k < maxK - minK + 1; k++){
        int kMaxSupports[minK + k];
        //ipotizzo che i supporti massimi siano i primi minK + k
        //nell'array supports
        for(int i = 0; i < minK + k; i++){
            kMaxSupports[i] = i;
        }
        int min;

        for(int i = minK + k; i < N; i++){
            min = 0;
            //cerco l'indice della entry contenente il minimo
            //supporto fra quelli che ho ipotizzato essere minimi
            for(int j = 1; j < minK + k; j++){
                if (supports[kMaxSupports[j]] <
                    supports[kMaxSupports[min]]){
                    min = j;
                }
            }
            //se supports[i] è maggiore allora entrerà a fare
            //parte di quei supporti che ipotizzo essere i maggiori
            if(supports[i] > supports[kMaxSupports[min]]){
                kMaxSupports[min] = i;
            }
        }
        float tmp = 1.0;
        //calcolo il supporto massimo atteso
```

```
41          for(int i = 0; i < minK + k; i++){
42              tmp = tmp * (supports[kMaxSupports[i]] * 1.0 / (T *
                    1.0));
43          }
44          maxExpSprt[k] = (int) (tmp * T) + 1.0;
45      }
46  }
```

Il metodo *maxExpectedSupport* consente di calcolare il massimo supporto atteso per gli itemset di taglia compresa fra $minK$ e $maxK$.

maxExpSprt è un array che conterrà i massimi supporti attesi.

supports è un array di supporti degli item.

$minK$ è la taglia minima di itemset che si intende studiare.

$maxK$ è la taglia massima di itemset che si intende studiare.

N è il numero di item contenuti nel dataset reale.

T è il numero di transazioni contenute nel dataset reale.

L'array *kMaxSupports* contiene gli identificativi degli itemset con supporto maggiore all'interno dell'array *supports*. Per ogni taglia k di itemset compresa nell'intervallo $[minK, maxK]$ si ipotizza che i k massimi supporti siano quelli dei primi k itemset memorizzati all'interno dell'array *supports* (righe da 19 a 21), di conseguenza i relativi identificativi, che si ricorda coincidono con gli indici delle entry dell'array *support* a loro associati, vengono memorizzati all'interno dell'array *kMaxSupports*. Ad ogni iterazione del ciclo *for* di riga 24 si scandisce per indici crescenti l'array *supports* partendo dalla k+1-esima entry e si confronta il valore letto con il minimo supporto puntato dall'array *kMaxSupports* (riga 35). Se quest'ultimo dovesse risultare inferiore allora al suo posto viene salvato l'indice del nuovo item con supporto più alto (riga 36), in caso contrario si passa all'iterazione successiva.

A.8 Generazione dei dataset random nel metodo di Monte Carlo

```
1  char *cmd = (char *) malloc(sizeof(char) * 200);
2  sprintf(cmd,"./src/apriori/apriori/src/apriori -s%.4f -S100 -ts
       -m%d -n%d %s %s", minFreq, minK, maxK, RDFileName,
       FIRDFileName);
3
4  //ciclo per la generazione di DELTA dataset random
5  for (int numDataset=0; numDataset < DELTA; numDataset++){
6          //metodo per la generazione di un singolo dataset random
7          generateRandomDataset(supports, index_item, size_item,
               RDFileName, N, T, num_transaction, numDataset,
               a_supports, minFreq);
8          //applico l'algoritmo APRIORI
9          system(cmd);
10         //aggiungo i nuovi itemset frequenti all'albero AVL
11         buildAvlItemsetC(W, FIRDFileName, sizeW, minK, maxK, T,
               numDataset, DELTA, maxExpSprt, notNull, supports);
12         //visualizzo i progressi fatti
13         loadBar(numDataset + 1, DELTA, 100);
14 }
```

Questo estratto di codice, appartenente al metodo *monteCarlo* implementato nel file *monteCarlo.c*, consente di generare *DELTA* dataset random, estrarre per ciascuno di essi gli itemset frequenti di taglia compresa nell'intervallo $[minK, maxK]$ e inserire quest'ultimi all'interno di uno degli alberi AVL di tipo AVL_ITEMSET_C i cui puntatori alle radici sono memorizzati all'interno dell'array *W*.
Nelle righe 1 e 2 si costruisce la stringa contenente il comando di sistema per l'esecuzione del programma Apriori implementato da Christian Borgelt in linguaggio C. Analizzando il comando è possibile notare che:

./src/apriori/apriori/src/apriori indica il percorso per raggiungere l'eseguibile;

-s%.4f indica il minimo supporto per il quale un itemset debba essere considerato frequente;

-S100 indica il massimo supporto per il quale un itemset debba essere considerato frequente;

-ts indica il *taget type* di tipo *frequent item set*;

-m%d indica la minima taglia di itemset frequenti che si intende studiare;

-n%d indica la massima taglia di itemset frequenti che si intende studiare;

%s %s servono ad indicare i nomi del file d'ingresso e di uscita.

Il ciclo *for* nella riga 5 termina quando *DELTA* dataset random sono stati generati.
Nella riga 7 viene invocato il metodo *generateRandomDataset* precedentemente illustrato. I dataset random vengono memorizzati all'interno del file il cui nome è contenuto nella stringa *RDFileName*.
Nella riga 9 viene eseguito il comando per l'esecuzione del programma Apriori sul dataset random precedentemente creato.
Nella riga 11 i dataset frequenti identificati vengono memorizzati all'interno degli alberi AVL di tipo AVL_ITEMSET_C precedentemente creati e le cui radici vengono puntate dall'array *W*.
Nella riga 13 viene aggiornata a video la percentuale di progresso effettuato per ciò che riguarda la generazione dei dataset random.

A.9 Calcolo dei valori di b_1 e b_2 all'interno del Monte Carlo

```
1  int s_size = 50;
2  //array per la memorizzazione dei valori di b1 al variare
3  //del supporto
4  float *b1 = (float *) malloc (sizeof(float) * s_size);
5  //array per la memorizzazione dei valori di b1 al variare
6  //del supporto
7  float *b2 = (float *) malloc (sizeof(float) * s_size);
8  //array per la memorizzazione temporanea di valori necessari
9  //al calcolo di b1 e b2
10 float **probabilities = (float **) malloc (sizeof(float *) *
       sizeW[j]);
11 for (int i = 0; i < sizeW[j]; i++){
12     probabilities[i] = (float *) malloc (sizeof(float) * s_size);
13 }
14 minEstSprt[j] = 0;
15 //calcolo dei valori di b1 e b2 al variare di s per ciascuna delle
16 //taglie di itemset studiate
17 for (int s = 0; s < size; s++){
18     //inizializzazione dei valori
```

```
19      for(int ss = 0; ss < s_size; ss++){
20          b1[ss] = 0.0;
21          b2[ss] = 0.0;
22          for(int i = 0; i < sizeW[j]; i++){
23              probabilities[i][ss] = 0.0;
24          }
25      }
26      //per ogni itemset frequente contenuto nell'albero AVL..
27      for (int i = 0; i < sizeW[j]; i++){
28          FREQ_RD *ptr = (*arrayW[i]).freqHead;
29          //calcolo la probabilità empirica che un itemset abbia
30          //supporto almeno pari a ss
31          do{
32              for (int ss = 0; ss < s_size; ss++){
33                  if (s + ss <= ptr->sprt - maxExpSprt[j]){
34                      probabilities[i][ss]++;
35                  }
36                  else break;
37              }
38              ptr = ptr->next;
39          }while(ptr != NULL);
40          //aggiorno i valori di b1 con le nuove probabilità calcolate
41          for (int ss = 0; ss < s_size; ss++){
42              probabilities[i][ss] = probabilities[i][ss] * 1.00 /
                    (DELTA * 1.00);
43              b1[ss] = b1[ss] + (probabilities[i][ss] *
                    probabilities[i][ss]);
44          }
45      }
46      for (int i = 0; i < (sizeW[j] - 1); i++){
47          for (int y = i + 1; y < sizeW[j]; y++){
48              //se i due itemset sono vicini allora entra nell'if
49              if (isNeighborhood((*arrayW[i]).itemset,
                    (*arrayW[y]).itemset, j + minK)){
50                  //serve a memorizzare il numero di eventi favorevoli
51                  //al variare del supporto
52                  float temp[s_size];
53                  for(int ss = 0; ss < s_size; ss++){
54                      temp[ss] = 0.0;
55                  }
56                  FREQ_RD *ptrI = (*arrayW[i]).freqHead;
57                  FREQ_RD *ptrY = (*arrayW[y]).freqHead;
58                  //calcolo la probabilità empirica che X e Y
```

```
59                  //siano entrambi frequenti all'interno di un
60                  //dataset random
61                  while(ptrI != NULL && ptrY != NULL){
62                      if (ptrI->numDataset <= ptrY->numDataset){
63                          //se sono stati frequenti all'interno dello
64                          //stesso dataset aggiorno la probabilità
65                          //empirica
66                          if (ptrI->numDataset == ptrY->numDataset){
67                              for (int ss = 0; ss < s_size; ss++){
68                                  if (s + ss + maxExpSprt[j] <=
                                        ptrI->sprt && s + ss +
                                        maxExpSprt[j] <= ptrY->sprt){
69                                      //aggiorno il numero di eventi
70                                      //favorevoli
71                                      temp[ss]++;
72                                  }
73                              }
74                              ptrI = ptrI->next;
75                              ptrY = ptrY->next;
76                          }
77                          else{
78                              //mi sposto avanti nella lista di I
79                              ptrI = ptrI->next;
80                          }
81                      }
82                      else{
83                          //mi sposto avanti nella lista di Y
84                          ptrY = ptrY->next;
85                      }
86                  } //while
87                  for(int ss = 0; ss < s_size; ss++){
88                      //numero di eventi favorevoli su numero
89                      //di casi possibili
90                      temp[ss] = temp[ss] * 1.00 / (DELTA * 1.00);
91                      //aggiorno b2 e b1
92                      b2[ss] = b2[ss] + temp[ss];
93                      b1[ss] = b1[ss] + (probabilities[i][ss] *
                            probabilities[y][ss]);
94                  }
95              }
96          }
97      } //for
98  }
```

Questo estratto di codice, appartenente al metodo *monteCarlo* implementato nel file *monteCarlo.c*, consente di calcolare il valore di b_1 e b_2 al variare del supporto s. Attraverso questo estratto di codice vengono calcolate le probabilità empiriche:

- p_X, probabilità che un generico dataset X sia frequente con supporto almeno s all'interno di un dataset random;

- p_{XY}, probabilità che una coppia di itemset vicini $<X, Y>$, con $X \neq Y$, risulti frequente con supporto almeno s all'interno di un generico dataset random.

Il calcolo delle probabilità empiriche viene fatto per intervalli di supporto di ampiezza 50. Ciò si deve al fatto che per dataset di grosse dimensioni si incorre in un rapido esaurimento della memoria a causa della matrice *probabilities* necessaria a velocizzare il calcolo delle probabilità empiriche p_{XY}. Per dataset di grosse dimensioni infatti il numero di itemset frequenti scoperti durante la generazione dei dataset random può crescere rapidamente e, dato che le dimensioni della matrice *probabilities* dipendono direttamente da tale numero, se la memoria venisse esaurita si incorrerebbe in errori di *segmentation fault* che arresterebbero l'esecuzione del programma. Se invece si optasse per il calcolo delle probabilità empiriche per un solo valore di supporto alla volta si avrebbe un degrado significativo delle prestazioni nel caso di itemset di grosse dimensioni dove il numero di itemset frequenti scoperti durante la generazione dei dataset random sarebbe piuttosto elevato.
Nelle righe da 27 a 45 si effettua il calcolo delle probabilità empiriche dei singoli itemset frequenti presenti all'interno dell'albero AVL e si procede con il calcolo dei singoli contributi che ciascun di essi apporta al valore di b_1. Nelle righe da 46 a 97 invece si passa al calcolo delle probabilità empiriche p_{XY} utilizzando il metodo *isNeighborhood* per individuare le coppie di itemset vicini. Quando due itemset risultano essere vicini allora si scandiscono le liste concatenate dei supporti ottenuti all'interno dei dataset random per entrambi gli itemset e si memorizza all'interno dell'array *temp* il numero di eventi favorevoli, ossia il numero di volte in cui gli itemset X e Y hanno presentato entrambi un supporto superiore ad s all'interno dello stesso dataset random. Durante questa fase viene perciò calcolato il valore di b_2 e contemporaneamente viene completato il calcolo di b_1 al variare del supporto s per entrambi.

A.10 Applicazione del *multi-comparison test*

```
1  //per ogni taglia di itemset k..
2  for (int k = 0; k < maxK - minK + 1; k ++){
3      int K = k + minK;
4      AVL_ITEMSET_E *F1 = (AVL_ITEMSET_E *)
           malloc(sizeof(AVL_ITEMSET_E));
5      F1 = NULL;
6      //creo la chiamata al programma Apriori
7      char *cmd = (char *) malloc(sizeof(char) * 200);
8      float minEstFreq = (float) minEstSprt[k] * 100.0 / (T * 1.0);
9      sprintf(cmd,"./src/apriori/apriori/src/apriori -s%.4f -S100 -ts
           -m%d -n%d %s %s", minEstFreq, K, K,
           convertedRealDatasetFileName, resultFileName);
10     //applico APRIORI per l'estrazione dei k-itemset
11     //sigificativi con supporto almeno pari a minEstSprt
12     system(cmd);
13
14     int *sizeF1 = (int *) malloc(sizeof(int));
15     *sizeF1 = 0;
16     //salvo gli itemset frequenti ottenuti nell'albero
17     //AVL denominato F1
18     F1 = buildAvlItemsetE(F1, resultFileName, sizeF1, K, T,
           supports);
19
20     sprintf(cmd,"rm %s", resultFileName);
21     system(cmd); //cancello i file di supporto
22     free(cmd);
23
24     //array di puntatori a nodi dell'albero AVL F1
25     AVL_ITEMSET_E **arrayF1 = (AVL_ITEMSET_E *) malloc ((*sizeF1) *
           sizeof(AVL_ITEMSET_E *));
26     linearizesAvlItemsetE(F1, arrayF1, 0);
27
28     /*-------------------------------------------------------------
29      Calcolo Pr(Bin(t, F1[i].freq) >= F1[i].support)
30      -------------------------------------------------------------*/
31     for(int i = 0; i < *sizeF1; i++){
32         double freq = ((*arrayF1[i]).expSprt) * 1.0 / (T * 1.0);
33         (*arrayF1[i]).pvalue = (double)
               gsl_cdf_binomial_Q((*arrayF1[i]).realSprt - 1, freq, T);
34     }
35
```

```
36      //ordinamento itemset secondo p-value crescenti
37      qsort(arrayF1, (*sizeF1), sizeof(AVL_ITEMSET_E *), cmpPvalue);
38
39      /*-------------------------------------------------------------
40       Calcolo del valore di l
41       -------------------------------------------------------------*/
42      double m = bin(N, K);
43
44      //la seria armonica da uno a k è pari al logaritmo di k più
45      //la costante di Eulero-Mascheroni più epsilon(k) che
46      //vale circa 1/(2*k)
47      //Euler-Mascheroni constant
48      long double err = 0.57721566490153286060651209;
49
50      double threshold = log(m);
51      threshold = (double) threshold + err + (1.0 / (2 * K));
52      threshold = threshold * m / BETA;
53      threshold = 1 / threshold;
54      int l = -1;
55
56      //calcolo del valore di l
57      for (int i = *sizeF1 - 1; i >= 0; i--){
58          if ((*arrayF1[i]).pvalue <= i * threshold){
59              l = i;
60              break;
61          }
62      }
63      //parte stampa
64      if (l != -1){ //se è stato trovato un valore per l
65          char *OFN = (char *) malloc(sizeof(char) * 60);
66          sprintf(OFN, "outcome/P1_%d-itemset.txt", k + minK);
67          FILE *OF = fopen(OFN, "w");
68          for (int i = 0; i <= l; i++){
69              for (int j = 0; j < K; j++){
70                  fprintf(OF, "%s \t",
                        (*item_array[(*arrayF1[i]).itemset[j]]).item);
71              }
72              fprintf(OF, "\n");
73          }
74          fclose(OF);
75          free(OFN);
76      }//stampa fine
77
```

```
78      else{
79          l = 0;
80      }
81      //stampa dei risultati
82      fprintf(stdout, "| %2d-itemset\t| \x1b[31m%3d\x1b[0m \t|\n", K,
            l);
83      destroyArrayItemsetE(arrayF1, *sizeF1);
84      free(arrayF1);
85  }
```

Questo estratto di codice, appartenente al metodo *procedure1* implementato nel file *procedure1.c*, consente di applicare il metodo di *multi-comparison test* per la scoperta di itemset significativi di taglia compresa nell'intervallo $[minK, maxK]$ all'interno di un dataset D assegnato.
Nella riga 4 viene creato l'albero AVL *F1* di tipo AVL_ITEMSET_E in cui memorizzare i K-itemset con supporto almeno pari al minimo supporto stimato attraverso la simulazione di Monte Carlo nel dataset D assegnato.
Nelle righe da 7 a 12 viene costruito il comando per l'esecuzione del programma Apriori, in maniera molto simile a quelle vista durante l'applicazione del metodo di Monte Carlo e dopo di che viene eseguito. Nella riga 18 vengono inseriti gli itemset frequenti scoperti attraverso il programma Apriori all'interno dell'albero AVL $F1$.
Nelle righe 25 e 26 viene creato un array di puntatori a strutture dati di tipo AVL_ITEMSET_E e attraverso il metodo *linearizesAvlItemsetE* le entry dell'array creato vengono fatte puntare ai nodi dell'albero $F1$ per consentire di lavorare più facilmente con gli itemset frequenti identificati.
Nelle righe da 31 a 37 si utilizzano le librerie scientifiche open source GSL per il calcolo del *pvalue* relativo alla *null hypothesis* H_0^X secondo la quale il valore del supporto di X nel dataset reale D è guidato da una distribuzione binomiale di parametri t e f_X, rispettivamente il numero di transazioni all'interno del dataset D e la frequenza associata all'itemset X.
Nelle righe da 42 e 61 gli itemset contenuti all'interno dell'albero AVL $F1$ vengono ordinati per valori di *pvalue* crescenti e successivamente viene calcolato il valore di l ossia il numero di *null hypothesis* con *pvalue* minimo rigettate.
Nelle righe da 63 e 75 vengono stampati in un file denominato "P1_K-itemset.txt" i primi l itemset per i quali le relative *null hypothesis* sono state rigettate, provvedendo però alla conversione degli identificativi di ciascun item nel reale valore assunto da questi all'interno del dataset D assegnato.

A.11 Calcolo di una soglia si supporto minima s^*

```
//per ogni taglia k di itemset..
for(int k = 0; k < maxK - minK + 1; k++){
    int K = k + minK;
    AVL_ITEMSET_S *F2 = (AVL_ITEMSET_S *)
        malloc(sizeof(AVL_ITEMSET_S));
    F2 = NULL;
    char *cmd = (char *) malloc(sizeof(char) * 200);
    //ricavo la frequenza minima di supporto (in percentuale)
    float minEstFreq = (float) minEstSprt[k] * 100.0 / (T * 1.0);
    sprintf(cmd,"\n./src/apriori/apriori/src/apriori -s%.4f -S100
        -ts -m%d -n%d %s %s", minEstFreq, K, K,
        convertedRealDatasetFileName, resultFileName);
    //applico APRIORI per estrarre dal dataset reale tutti
    //gli itemset con supporto almeno pari a minEstSprt
    system(cmd);
    free(cmd);

    int *sizeF2 = (int *) malloc(sizeof(int));
    *sizeF2 = 0;
    //salvo gli itemset frequenti ottenuti nell'albero F2
    F2 = buildAvlItemsetS(F2, resultFileName, sizeF2, K, T);
    AVL_ITEMSET_S **arrayF2 = (AVL_ITEMSET_S *) malloc ((*sizeF2) *
        sizeof(AVL_ITEMSET_S *));
    linearizesAvlItemsetS(F2, arrayF2, 0);
    //ricavo il supporto massimo effettivo fra i k-itemset estratti
    int maxSprt = 0;
    for (int i = 0; i < *sizeF2; i++){
        if (maxSprt < (*arrayF2[i]).sprt){
            maxSprt = (*arrayF2[i]).sprt;
        }
    }
    int s0 = minEstSprt[k];
    //calcolo il valore di h
    int h = floor(logf(maxSprt - minEstSprt[k]) / logf(2.0)) + 1;
    //Calcolo del valore di L[i] al variare delle soglie di
    //supporto, ossia calcolo del valore atteso delle
    //variabili aleatorie di Poisson che esprimono il numero
    //di k-itemset frequenti in un dataset random al
    //variare del supporto s
```

```
36      float L[h];
37      for (int i = 0; i < h; i ++){
38          L[i] = 0.0;
39      }
40      //memorizzo in sizeW il numero di itemset frequenti di taglia k
41      //trovati durante la simulazione di Monte Carlo
42      int sizeW = getAvlItemsetCSize(W[k], 0);
43      //costruisco un array di puntatori ai nodi dell'albero AVL
44      AVL_ITEMSET_C *arrayW[sizeW];
45      linearizesAvlItemsetC(W[k], arrayW, 0);
46      //calcolo il numero di itemset frequenti per le soglie di
47      //supporto s
48      for (int i = 0; i < sizeW; i++){
49          int s;
50          FREQ_RD *ptr = (*arrayW[i]).freqHead;
51          while (ptr != NULL){
52              s = s0;
53              int sprt = ptr->sprt;
54              for (int j = 0; j < h; j++){
55                  if (s <= sprt){
56                      L[j]++;
57                      s = s0 + (int) pow(2.0, j);
58                  }
59                  else {
60                      break;
61                  }
62              }
63              ptr = ptr->next;
64          }
65      }
66      //libero la memoria non più utilizzata
67      destroyAvlItemsetC(W[k], K);
68      for (int j = 0; j < h; j++){
69          L[j] = L[j] / DELTA;
70      }
71      //Calcolo del valore di Q[i] al variare delle soglie di
72      //supporto, ossia del numero di k-itemset frequenti
73      //nel dataset reale al variare delle soglie di supporto s
74      int Q[h];
75      for (int j = 0; j < h; j++){
76          Q[j] = 0;
77      }
78      for (int i = 0; i < *sizeF2; i++){
```

```
79          int s = s0;
80          for (int j = 0; j < h; j++){
81              if (s <= (*arrayF2[i]).sprt){
82                  Q[j]++;
83                  s = s0 + pow(2.0, j + 1);
84              }
85              else {
86                  break;
87              }
88          }
89      }
90      //libero la memoria non più utilizzata
91      destroyArrayItemsetS(arrayF2, *sizeF2);
92      free(arrayF2);
93      //Calcolo del valore di sStar
94      int s = s0;
95      int i = 0;
96      int sStar = 0;
97      while (i < h){
98          double poisson_prob = gsl_cdf_poisson_Q(Q[i] - 1, L[i]);
99          //cerco di rigettare le null hypothesis
100         if (poisson_prob <= ALPHA / h && Q[i] >= h * L[i] / BETA){
101             sStar = s;
102             //alla prima null hypothesis che trovo mi fermo
103             break;
104         }
105         //aggiorno la soglia s
106         s = s0 + pow(2.0, i);
107         i++;
108     }
109     if (sStar != 0){
110         F2 = (AVL_ITEMSET_S *) malloc(sizeof(AVL_ITEMSET_S));
111         F2 = NULL;
112         //estraggo da l dataset reale tutti quei k-itemset con
113         //supporto almeno sStar
114         cmd = (char *) malloc(sizeof(char) * 200);
115         sprintf(cmd,"./src/apriori/apriori/src/apriori -s%.4lf
                -S100 -ts -m%d -n%d %s %s", sStar * 100.0 / (T * 1.0),
                K, K, convertedRealDatasetFileName, resultFileName);
116         system(cmd);
117         //applico APRIORI per l'estrazione dei k-itemset
118         //sigificativi con supporto almeno pari a sStar
119         free(cmd);
```

```
120         *sizeF2 = 0;
121         //salvo gli itemset frequenti ottenuti nell'albero
122         //AVL denominato F2
123         F2 = buildAvlItemsetS(F2, resultFileName, sizeF2, K, T);
124         fprintf(stdout, "| %2d-itemset\t| \x1b[31m%6d\x1b[0m \t|
                %3d\t| %.2f \t|\n", K, sStar, *sizeF2, L[i]);
125         //parte stampa itemset significativi
126         arrayF2 = (AVL_ITEMSET_S *) malloc ((*sizeF2) *
                sizeof(AVL_ITEMSET_S *));
127         linearizesAvlItemsetS(F2, arrayF2, 0);
128         char *OFN = (char *) malloc(sizeof(char) * 60);
129         sprintf(OFN, "outcome/P2_%d-itemset.txt", k + minK);
130         FILE *OF = fopen(OFN, "w");
131         for (int i = 0; i < *sizeF2; i++){
132             for (int j = 0; j < K; j++){
133                 fprintf(OF, "%s \t",
                        (*item_array[(*arrayF2[i]).itemset[j]]).item);
134             }
135             fprintf(OF, "\n");
136         }
137         fclose(OF);
138         free(OFN);
139         if(F2!=NULL){
140             destroyArrayItemsetS(arrayF2, *sizeF2);
141         }
142         free(arrayF2);
143         //fine parte stampa itemset significativi
144     }
145     else{
146         fprintf(stdout, "| %2d-itemset\t| \x1b[31m +inf \x1b[0m \t|
                \t\t| \t\t|\n", K);
147     }
148     free(sizeF2);
149 }
```

Questo estratto di codice, appartenente al metodo *procedure2* implementato nel file *procedure2.c*, consente di calcolare, per ciascun itemset di taglia compresa nell'intervallo $[minK, maxK]$, una soglia di supporto minima s^* oltre la quale gli itemset frequenti possono essere considerati significativi.
Nella riga 4 viene creato l'albero AVL *F2* di tipo AVL_ITEMSET_S in cui memorizzare i K-itemset con supporto almeno pari al valore di *minEstSprt* precedentemente calcolato per gli itemset di taglia K.
Nelle righe da 9 a 12 viene costruito il comando per l'esecuzione del program-

ma Apriori, in maniera molto simile a quella vista durante l'applicazione del metodo di Monte Carlo e dopo di che viene eseguito.
Nella riga 18 vengono inseriti gli itemset frequenti scoperti attraverso il programma Apriori all'interno dell'albero AVL $F2$.
Nella riga 20 viene costruito un array di puntatori ai nodi dell'albero AVL $F2$ per facilitare le successive fasi della procedura.
Nelle righe da 22 a 27, utilizzando l'array di puntatori precedentemente creato, si calcola il massimo supporto che un K-itemset frequente assume all'interno del dataset D assegnato.
Nelle righe 28 e 30 si imposta il valore della soglia s_0 al valore minimo stimato per itemset di taglia K, calcolato durante la simulazione di Monte Carlo, e si calcola il numero massimo di confronti h che la procedura potrà eseguire per determinare la soglia di supporto s^*.
Nelle righe da 48 a 65 viene calcolato per ciascun valore di soglia s_i, con $0 \leq i < h$, il numero cumulato di itemset frequenti all'interno di ciascun dataset random precedentemente creato durante la simulazione di Monte Carlo. I valori ottenuti vengono memorizzati all'interno dell'array L e in riga 69 ciascuno di questi viene convertito nel numero medio di itemset frequenti oltre una data soglia di supporto s_i all'interno di un generico dataset random.
Nelle righe da 78 a 89 si ripetono le operazioni eseguite nelle righe da 48 a 65 ma questa volta in riferimento al dataset D assegnato.
Nelle righe da 97 a 108 viene calcolato il valore della soglia di supporto s^* memorizzando il risultato all'interno della variabile *sStar*. In riga 98 ci si appoggia alle librerie scientifiche open source GSL per il calcolo della probabilità che il numero di itemset frequenti oltre una data soglia di supporto s_i in un generico dataset random sia superiore al numero di itemset frequenti presenti nel dataset D assegnato.
Nella riga 100 vengono applicate le condizioni necessarie per poter rigettare una *null hypothesis*:

- La prima condizione prevede che la probabilità precedentemente calcolata sia inferiore al valore $ALPHA/h$ al fine di identificare gli itemset statisticamente significativi con confidenza pari ad $ALPHA$.

- La seconda condizione prevede che il numero di itemset frequenti con supporto almeno pari a s_i sia maggiore o uguale di $h * L[i]/BETA$ al fine di garantire un valore di FDR superiormente limitato da $BETA$.

Nelle righe da 114 a 123 si estraggono dal dataset D assegnato gli itemset con supporto almeno pari al valore di s^* memorizzando quest'ultimi all'interno dell'albero $F2$ precedentemente cancellato per poter essere riutilizzato.
Nelle righe da 126 e 142 vengono stampati in un file denominato "P2_K-itemset.txt"

gli itemset memorizzati all'interno dell'albero AVL $F2$, provvedendo però alla conversione degli identificativi di ciascun item nel reale valore assunto da questi all'interno del dataset D assegnato.

Appendice B

Manuale utente

La seguente appendice costituisce il manuale utente del toolbox realizzato per l'identificazione di itemset frequenti e statisticamente significativi. Il manuale è strutturato attraverso i seguenti punti:

- Organizzazione del toolbox;
- Procedura per la compilazione e installazione del toolbox;
- Configurazione dei parametri d'ingresso;
- Avvio dell'esecuzione;
- Output restituito;
- Rimozione del toolbox.

Questo manuale consente di comprendere la struttura del toolbox e di facilitarne l'utilizzo per chiunque disponga di un sistema basato su Linux o Mac OS X.

B.1 Organizzazione del toolbox

Il toolbox è organizzato all'interno di una cartella principale denominata "*MultiK*" che a sua volta è suddivisa in cartelle e sottocartelle al fine di semplificare l'utilizzo del programma. In Figura B.1 viene riportata l'organizzazione dei file all'interno della cartella *MultiK*.

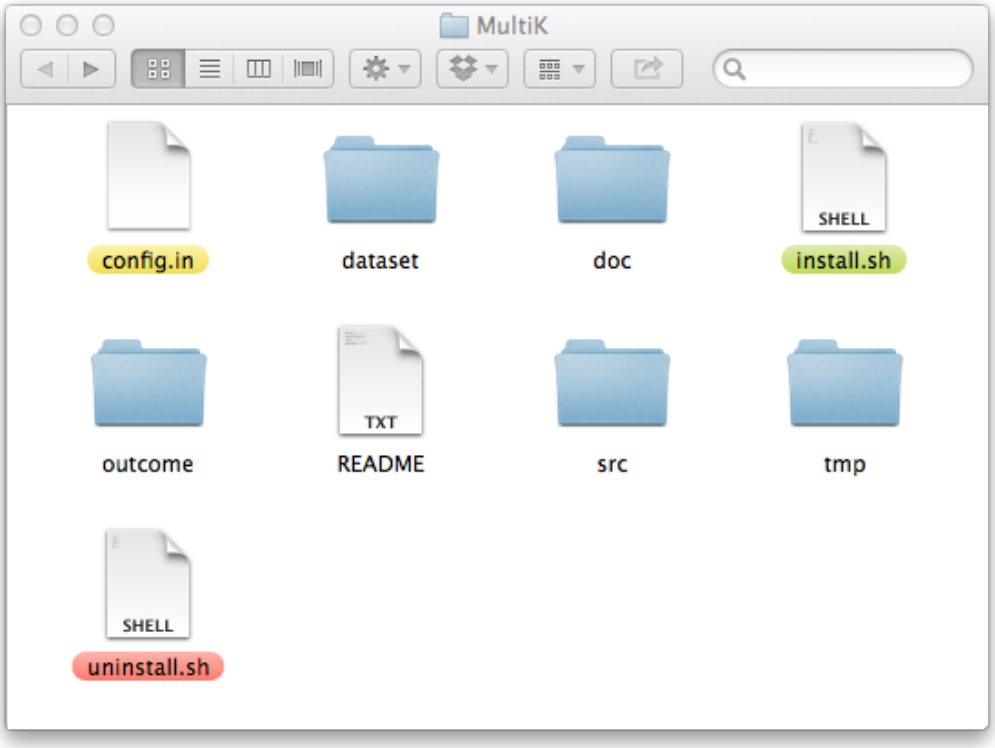

Figura B.1: Organizzazione del toolbox

config.in è il file attraverso il quale è possibile configurare i parametri d'ingresso del programma;

dataset ospita i dataset in ingresso al toolbox e per i quali si vogliono scoprire gli itemset statisticamente significativi;

doc contiene la documentazione in formato HTML dei metodi e delle strutture dati implementate all'interno del toolbox;

install.sh è uno script di shell che consente di compilare ed installare comodamente i file relativi al toolbox sviluppato;

README è un file di testo semplice che contiene un riassunto di quanto riportato in questa appendice;

outcome ospiterà al termine di ogni esecuzione i file contenenti gli itemset statisticamente significativi scoperti;

src contiene il codice sorgente del toolbox realizzato e del programma Apriori implementato da Christian Borgelt per la scoperta di itemset frequenti;

tmp contiene dei file di supporto necessari al corretto funzionamento del toolbok che non devono interessare l'utente finale e vengono cancellati al termine di ogni esecuzione;

uninstall.sh è uno script di shell che consente di cancellare rapidamente i file creati a seguito della compilazione del toolbox.

Come sopra indicato, la cartella *src* ospita il codice sorgente del toolbox realizzato e del programma esterno Apriori. In Figura B.2 viene riportata l'organizzazione dei file all'interno della cartella *src*.

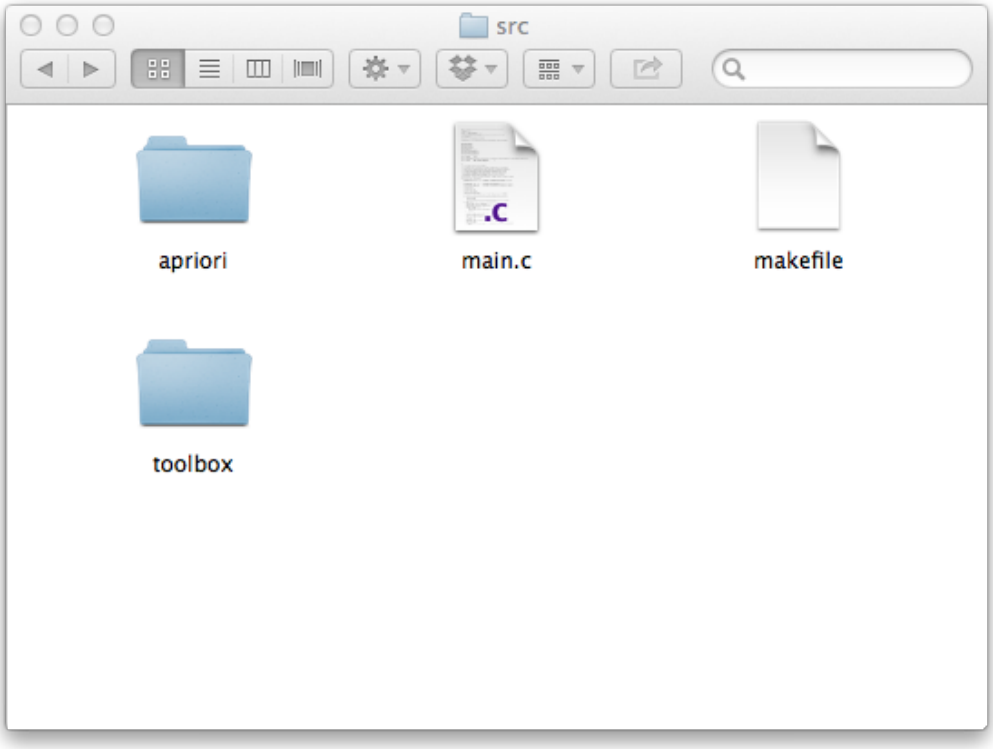

Figura B.2: Organizzazione del codice sorgente

apriori contiene il codice sorgente del programma Apriori ottenibile dal sito borgelt.net/apriori.html;

main.c è il file che contiene il codice sorgente per la gestione dell'esecuzione del toolbox e la lettura dei parametri d'ingresso;

makefile si occupa della corretta compilazione ed ottimizzazione del codice sorgente al fine di ottenere un file eseguibile adatto alle diverse piattaforme hardware e software nelle quali il toolbox verrà utilizzato;

toolbox contiene il codice sorgente relativo all'implementazione del toolbox.

Questo metodo di organizzazione consente di "mostrare" all'utente finale unicamente ciò che è di interesse per lui. Eventuali file generati a seguito della compilazione del toolbox e possibili file temporanei o di supporto creati durante l'esecuzione del programma verranno tenuti nascosti all'utente il quale potrà concentrarsi interamente sui risultati restituiti a video e all'interno della cartella *outcome*.

B.2 Installazione

Il toolbox realizzato può essere compilato ed eseguito all'interno di sistemi Linux e Mac OS X. Per una corretta esecuzione del programma è necessario disporre delle librerie scientifiche open source GSL. Prima di procedere perciò con la compilazione del toolbox è necessario che tali librerie vengano installate.

B.2.1 Librerie GSL

Sistemi Linux

Molte distribuzioni di Linux possiedono dei sistemi di gestione dei pacchetti già contenenti i riferimenti alle librerie GSL. In Ubuntu ad esempio è sufficiente digitare nel terminale di sistema il comando

```
sudo apt-get install gsl-bin libgsl0ldbl
```

per avviare l'installazione delle librerie.
Altrimenti è possibile procedere con il download e l'installazione manuale delle librerie attraverso i seguenti passi:

1. Andare all'indirizzo gnu.org/prep/ftp.html;
2. Cliccare sul link ftp più vicino alla propria località;
3. Cercare la cartella *gsl/* e cliccare su di essa;
4. Cercare il file *gsl-VERSIONE.tar.gz* dove il valore della versione sia il più alto possibile in modo da disporre della release più recente delle librerie GSL;
5. All'interno della finestra del terminale estrarre il file *tar.gz* scaricato utilizzando il comando

   ```
   tar -xzf gsl-VERSIONE.tar.gz
   ```

   e poi il comando `cd` per portarsi nella cartella *./gsl-VERSIONE*.
6. Eseguire in sequenza i comandi `./configure`, `make` e `make install` per avviare e portare a termine l'installazione delle librerie. Durante quest'ultimo passaggio potrebbe essere necessario effettuare il login come amministratore (`sudo`).

Sistemi Mac OS X

Il metodo più semplice per installare le librerie GSL all'interno di un sistema Mac OS X è di passare attraverso un programma di Homebrew. Per fare questo è necessario seguire i seguenti passi:

1. All'interno della finestra del terminale di sistema digitare il comando

   ```
   ruby -e ‘‘$(curl -fsSkL raw.github.com/mxcl/homebrew/go)’’
   ```

   e attendere il completamento del processo.
   (Attenzione: le doppie virgolette “ e ” devono essere sostituite dai doppi apici per ottenere la corretta esecuzione del comando.)

2. Infine digitare il comando

   ```
   brew install gsl
   ```

   per avviare il processo d'installazione delle librerie GSL.

Altrimenti è possibile procedere con l'installazione manuale delle librerie seguendo i medesimi passi indicati per i sistemi Linux.

B.2.2 Compilatore GCC

Una volta che l'installazione delle librerie è stato completata, prima di procedere con la compilazione del toolbox è necessario assicurarsi che all'interno del proprio sistema sia presente il compilatore GCC.

Sistemi Linux

Nella maggior parte dei sistemi Linux il compilatore GCC è già preinstallato. Per verificare ciò è sufficiente digitare nel terminale di sistema il comando:

```
gcc -v
```

e verificare che venga restituita la versione del compilatore installato, ad esempio *gcc version 4.2.1* .
Se così non dovesse accedere in Ubuntu è sufficiente digitare nel terminale di sistema il comando

```
sudo apt-get install gcc make
```

per avviare il processo d'installazione.
Altrimenti è possibile procedere con l'installazione manuale seguendo le istruzioni riportate all'indirizzo gcc.gnu.org/install/index.html

Sistemi Mac OS X

Nei sistemi Mac OS X è necessario installare il pacchetto *Xcode*.

- Per le versioni più recenti quali *Lion (10.7)* e *Mountain Lion (10.8)* tale pacchetto è reperibile su *Mac App Store*.

- Per le versioni precedenti invece è necessario disporre del DVD d'installazione di Mac OS X all'interno del quale, nella cartella *Installazioni opzionali*, è presente il file denominato *Xcode.mpkg*.

A questo punto è possibile avviare l'installazione del pacchetto. Una volta che il processo è stato completato è necessario aprire *Xcode*, entrare nel pannello di preferenze del programma, selezionare la scheda *Downloads* e nella scheda *Components* avviare l'installazione del pacchetto *Command Line Tools*.

B.2.3 Compilazione del toolbox

A questo punto è possibile procedere con la compilazione ed installazione vera e propria del toolbox realizzato. La procedura da seguire è la medesima sia per sistemi Linux che per sistemi Mac OS X.

1. Aprire il terminale di sistema;
2. Utilizzare il comando `cd` fino a portarsi nella cartella *MultiK*;
3. Digitare il comando

   ```
   sh install.sh
   ```

   per avviare la compilazione del toolbox e del programma Apriori.

A questo punto è possibile utilizzare il toolbox all'interno del proprio sistema.

B.3 Configurazione

Attraverso il file *config.in* è possibile configurare i parametri d'ingresso del toolbox. In Figura B.3 è riportato un esempio del contenuto del file di configurazione.

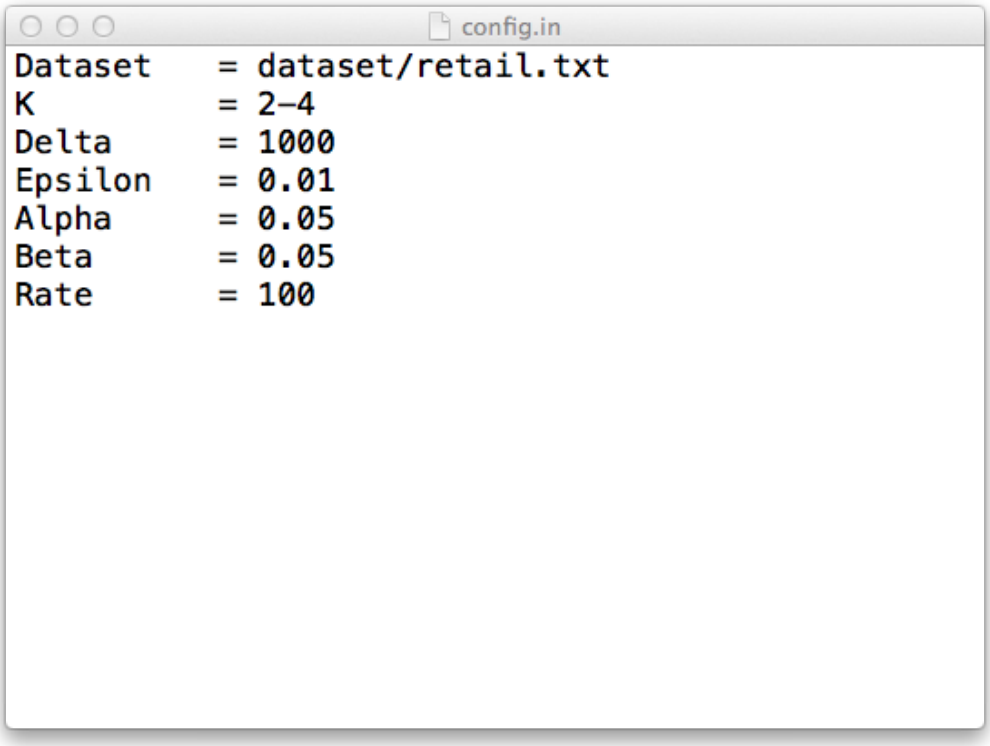

Figura B.3: File di configurazione

- *Dataset* deve contenere il nome del file nel quale è salvato il dataset d'ingresso, compreso il percorso per raggiungere tale file qualora fosse necessario. Si consiglia di memorizzare i dataset all'interno della omonima cartella al fine di mantenere un'organizzazione ordinata del toolbox;
- *K* indica la taglia degli itemset significativi che si vuole studiare. Nel caso si fosse interessati a studiare itemset significativi appartenenti ad un intervallo di taglie è sufficiente indicare l'estremo inferiore e superiore dell'intervallo separandoli per mezzo del carattere '-';
- *Delta* indica il numero di dataset random che si vuole generare durante l'esecuzione del metodo di Monte Carlo;

- *Epsilon* è un indice di qualità relativo all'approssimazione della distribuzione di $\hat{Q}_{k,s}$ con una distribuzione di Poisson con il medesimo valore atteso;

- *Alpha* rappresenta un indice di qualità relativo alla confidenza con cui i k-itemset frequenti possono essere contrassegnati come statisticamente significativi nell'approccio basato sulla soglia di supporto minima s^*;

- *Beta* rappresenta un indice di qualità riguardante il limite superiore del valore di *FDR* relativamente all'estrazione di k-itemset significativi in entrambi gli approcci;

- *Rate* serve ad indicare la percentuale di transazioni che si vuole generare per costruire i dataset random (valore di default 100). Tale parametro è legato direttamente alla strategia per l'incremento dell'efficienza presentata nel Parafrafo 4.2.

I valori dei parametri d'ingresso vanno impostati unicamente all'interno del file *config.in*. Questa modalità è stata preferita in quanto permette di modificare facilmente i parametri d'ingresso senza doverli digitare manualmente ad uno ad uno durante ogni esecuzione del programma.

Il toolbox realizzato mette perciò a disposizione le seguenti opzioni:

- Selezionare un qualsiasi dataset d'ingresso purché questo sia memorizzato all'interno di un file di testo contenente unicamente caratteri di scrittura semplici nel quale ad ogni riga del file viene fatta corrispondere una ed una sola transazione del dataset, separando gli item tra loro mediante uno spazio. Gli item, che per ovvie ragioni devono essere tutti differenti tra loro, possono essere sia dei valori numerici che delle stringhe;

- Studiare la significatività degli itemset di taglia k fissata oppure appartenente ad un intervallo di estremi fissati;

- Scegliere il numero di dataset random generati durante la procedura di Monte Carlo. Si noti che maggiore è il valore che si assegna alla variabile *Delta* migliore poi sarà la precisione con cui i supporti attesi di ciascun itemset all'interno di un generico dataset random verranno calcolati.

- Fissare dei parametri di qualità relativi all'insieme di itemset significativi restituiti;

- Aumentare l'efficienza del programma tenendo conto però degli svantaggi che ciò potrebbe comportare come descritto nel Parafrafo 4.2.

B.4 Esecuzione

La corretta compilazione del toolbox produce come risultato un file eseguibile denominato *main* all'interno della cartella *MultiK*.

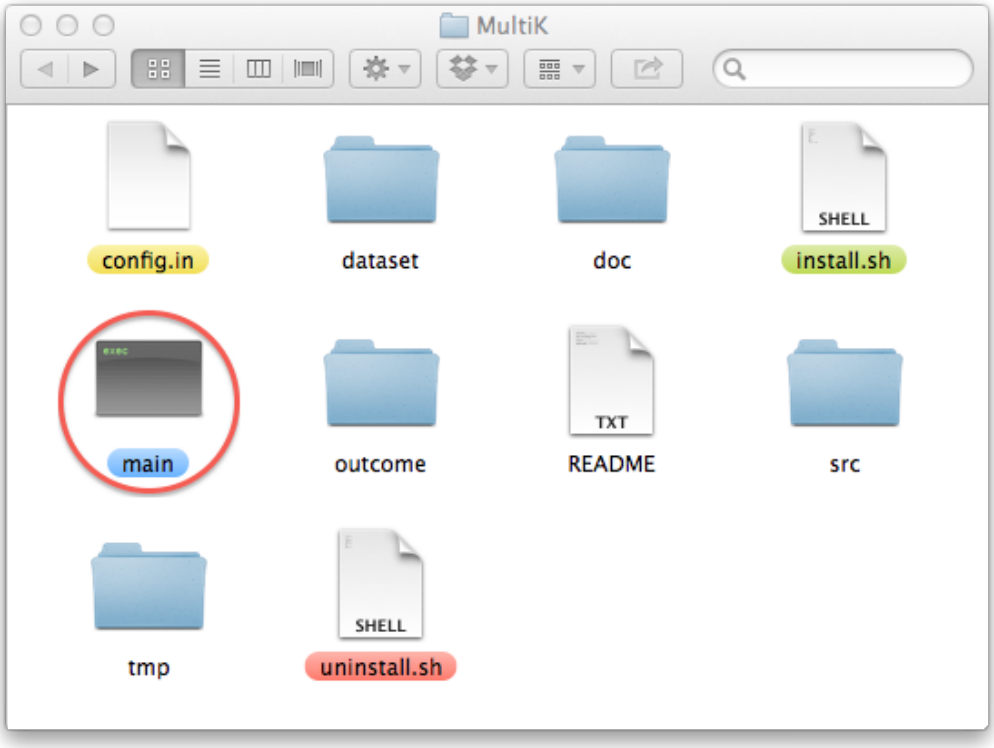

Figura B.4: Risultato della compilazione

Per avviare l'esecuzione del toolbox è necessario aprire la finestra del terminale di sistema e, utilizzando il comando `cd`, portarsi all'interno della cartella *MultiK*.
Digitando il comando

```
./main config.in
```

verrà avviato il programma. L'unico parametro a riga di comando che è necessario fornire è quello del nome del file di configurazione precedentemente descritto, eventualmente includendo il percorso nel quale il file è memorizzato nel caso questo non dovesse trovarsi all'interno della cartella *MultiK*.

B.5 Output restituito

L'output del toolbox si compone di due parti:

- La prima parte riguarda le statistiche di utilizzo del toolbox stesso, cioè le tempistiche relative a ciascuna fase dell'algoritmo ed anche quelle relative all'intera esecuzione. In questa prima parte vengono inoltre fornite le soglie di supporto stimate e attese durante la simulazione di Monte Carlo. Vengono infine forniti tutti i dati relativi agli itemset significativi identificati da ciascuna delle due procedure, come ad esempio il numero di itemset scoperti e le soglie di supporto calcolate. Questa prima parte dell'output viene visualizzata direttamente nella finestra del terminale, il che permette di verificare in tempo reale lo stato di avanzamento delle diverse fasi del programma.

- La seconda parte di output riguarda unicamente gli itemset statisticamente significativi estratti. Al termine dell'esecuzione del toolbox vengono creati uno o più file contenenti gli itemset significativi scoperti, suddivisi per taglie e tipo di procedura con cui sono stati ottenuti e memorizzati nello stesso formato utilizzato per i dataset d'ingresso.

    - I file la cui denominazione inizia con "P1" contengono gli itemset statisticamente significativi di taglia k estratti attraverso la procedura basata sul *multi-comparison test*;
    - I file la cui denominazione inizia con "P2" contengono gli itemset statisticamente significativi di taglia k estratti attraverso la procedura sviluppata in (Kirsch *e altri*, 2009).

  Questi file vengono memorizzati all'interno della cartella denominata "outcome" come riportato in Figura B.5.

Si noti che ad ogni esecuzione del toolbox i file contenuti all'interno di questa cartella, relativi alla precedente esecuzione, vengono cancellati per lasciare spazio ai nuovi file di output. In Figura B.6 viene riportato l'insieme di 4-itemset statisticamente significativi identificati dalla procedura sviluppata in (Kirsch *e altri*, 2009) relativamente al dataset *Retail*.

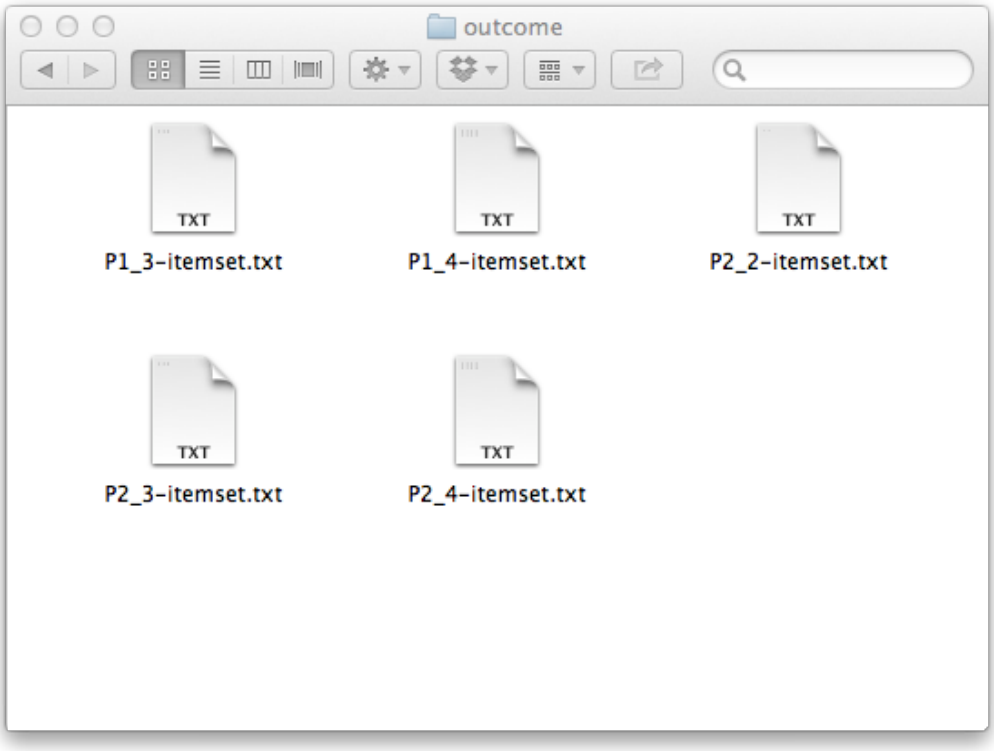

Figura B.5: Contenuto della cartella *outcome*

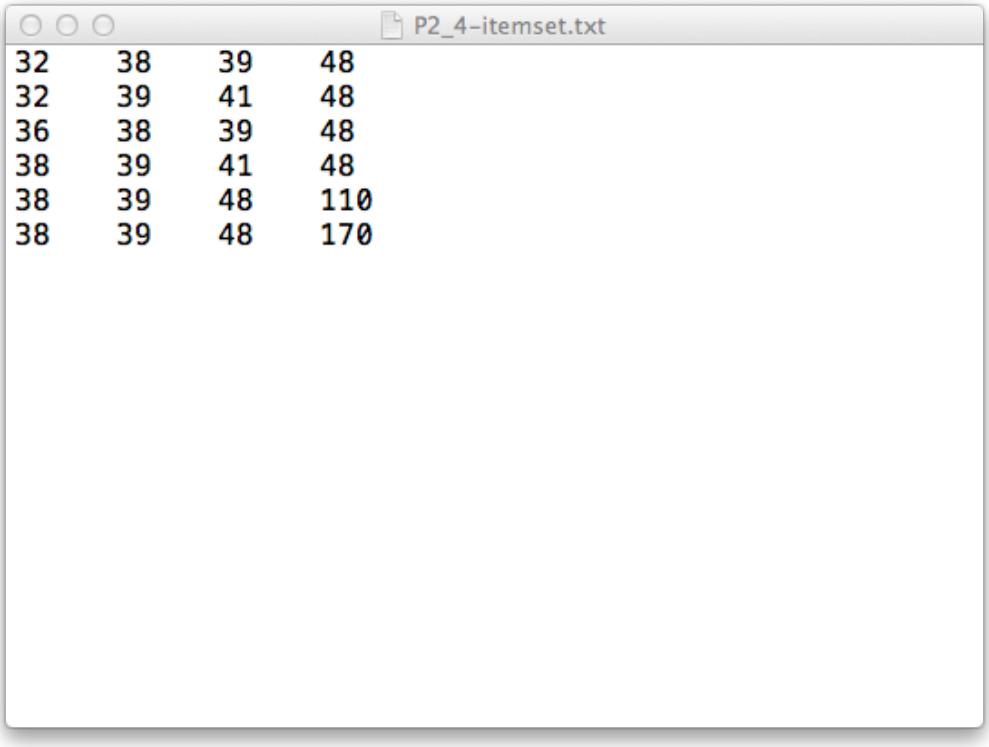

Figura B.6: Esempio del contenuto di un file di output

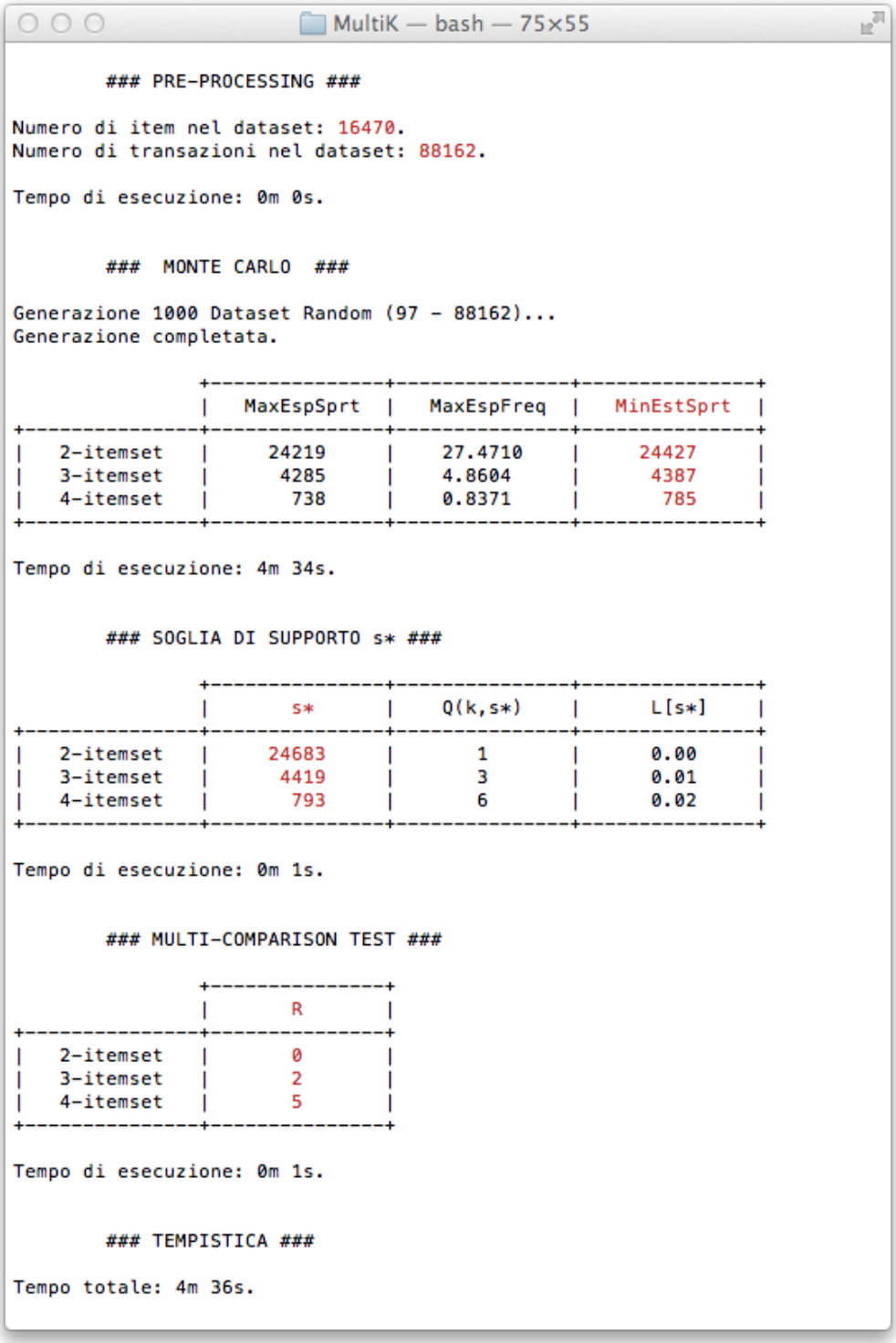

Figura B.7: Esempio di output nel terminale

In Figura B.7 viene mostrato un esempio di output stampato all'interno del terminale di sistema relativamente al dataset *Retail*.

- Nella fase di *pre-processing* viene restituito in output il numero di item e transazioni scoperte all'interno del dataset;

- Durante la fase che prevede l'applicazione del metodo di Monte Carlo vengono generati i dataset random necessari allo studio delle probabilità empiriche relative agli itemset frequenti scoperti. In output viene restituita la stringa "Generazione DDDD dataset random (NN - TTTT)..." per indicare l'inizio della generazione dei dataset.

  - Con "DDDD" si vuole indicare il numero di dataset random che verranno generati;
  - Con "NN" si vuole indicare il numero di item con supporto sufficientemente grande da poter essere presi in considerazione durante la generazione dei dataset;
  - Con "TTTT" si vuole indicare il numero di transazioni generate per costruire ciascun dataset random. Tale valore è direttamente collegato al parametro d'ingresso *Rate*.

  Per ogni taglia di itemset, nelle colonne *MaxExpSprt* e *MaxExpFreq* vengono restituiti rispettivamente i valori del massimo supporto e della massima frequenza attesi, calcolati attraverso il metodo *maxExpectedSupport*. In colonna *MinEstSprt* viene restituito il risultato più importante di questa fase ovvero la minima soglia di supporto stimata oltre la quale il numero di itemset frequenti con supporto almeno pari a tale soglia all'interno di un generico dataset random è bene approssimato da una variabile aleatoria di Poisson;

- Durante l'applicazione dell'approccio sviluppato in (Kirsch *e altri*, 2009), per ogni taglia k di itemset, in colonna s^* viene restituito il valore della minima soglia di supporto oltre la quale tutti k-itemset frequenti possono essere considerati statisticamente significativi. In colonna Q_{k,s^*} viene restituito il numero di k-itemset con supporto almeno pari a s^* all'interno del dataset d'ingresso mentre in colonna $L[s^*]$ viene restituito il numero atteso di itemset frequenti oltre la soglia si supporto s^* in un modello random del dataset;

- Durante l'applicazione dell'approccio basato sul *multi-comparison test*, per ogni taglia k di itemset, in colonna R viene restituito il numero di itemset statisticamente significativi trovati.

B.6 Rimozione del toolbox

All'interno della cartella *MultiK* è presente un file denominato *uninstall.sh* che permette di cancellare i file generati a seguito della compilazione del toolbox e del programma Apriori, incluso anche il file eseguibile *main*. La funzione del file *uninstall.sh* è esattamente complementare quindi a quella del file *install.sh*. La rimozione dei file può essere avviata attraverso il comando

```
sh uninstall.sh
```

al cui completamento si avrà che il contenuto della cartella *MultiK* sarà il medesimo di quello presentato all'inizio dell'appendice.

Elenco delle tabelle

Elenco delle figure

Bibliografia

Agrawal R.; Imielinski T.; Swami A. (1993). Mining association rules between sets of items in large databases. *Proc. of the 1993 ACM SIGMOD Conf. on Management of Data.*

Arratia R.; Goldstein L.; Gordon L. (1990). Poisson approximation and the chen-stein method. *Statistical Science 5, 4, 403–434.*

Benjamini Y.; Hochberg Y. (1995). Controlling the false discovery. *Journal of the Royal Statistic Society, Series B 57, 289–300.*

Benjamini Y.; Yekutieli D. (2001). The control of the false discovery rate in multiple testing under dependency. *Annals of Statistics 29, 4, 1165–1188.*

Kirsch A.; Mitzenmacher M.; Pietracaprina A.; Pucci G.; Upfal E.; Vandin F. (2009). An efficient rigorous approach for identifying statistically significant frequent itemsets. *Proc. 28th ACM PODS, 117–126, 2009.* To appear in Journal of the ACM.

Pietracaprina A. (2011). Significatività statistica di itemset frequenti. http://crono.dei.unipd.it/~dm/MATERIALE/significance.pdf.

Tan P.; Steinbach M.; Kumar V. (2006). *Introduction to Data Mining.* Addison-Wesley.