Esercitazione 2 – Classi, Aggregazione, Ereditarietà, Classi Astratte: Implementazione di APRIORI per la scoperta di pattern frequenti e /pattern emergenti

Si definiscano le classi Java necessarie a modellare i pattern e realizzare l'algoritmo per la scoperta di pattern frequenti e pattern emergenti.

Si ricorda che gli studenti devono definire le visibilità di attributi, metodi e classi

- *Definire la classe **LinkList** che modella una struttura dati lista linkatada usare come contenitore per i pattern frequenti e le regole confidenti (classe fornita dal docente)*
- *Definire la classe **Cella** che modella un elemento singolo della struttura dati lista linkata (classe fornita dal docente)*
- *Definire la classe **Puntatore** che modella il puntatore all'elemento successivo nella struttura dati lista linkata (classe fornita dal docente)*
- *Definire la classe **Queue** che modella una struttura coda che è poi usata come contenitore a modalità FIFO per i pattern frequenti scoperti a livello k da usare per generare i pattern candidati a livello k+1 (classe fornita dal docente)*

- *Definire la classe astratta **Item** che modella un generico item (coppia attributo-valore, per esempio Outlook=”Sunny”)*

  Attributi

  Attribute attribute; *attributo coinvolto nell'item*

  Object value; *valore assegnato all'attributo*

Metodi

Item(Attribute attribute, Object value)

Input: riferimenti per inizializzare i campi attribute e value

Output:

Comportamento: inizializza i valori dei membri attributi con i parametri passati come argomento al costruttore

Attribute getAttribute()

Input:

Output : attributo membro dell'item

*Comportamento: restituisce il membro **attribute**;*

Object getValue()

Input:

Output : valore coinvolto nell'item

*Comportamento: restituisce il membro **value**;*

abstract boolean checkItemCondition(Object value)

Input:

Output:

Comportamento: da realizzare nelle sottoclassi.

public String toString()

Input:

Output:

Comportamento: restituisce una stringa nella forma <attribute>=<value>

- *Definire la classe concreta **DiscreteItem** che estende la classe **Item** e rappresenta la coppia <Attributo discreto - valore discreto> (Outlook="Sunny")*

  Metodi

  DiscreteItem(DiscreteAttribute attribute, String value)

  Input: attributo discreto e un suo valore

  Output :

  Comportamento: Invoca il costruttore della classe madre per avvalora i membri

  boolean checkItemCondition(Object value)

  Input: value dichiarato di tipo Object (in realtà sarà di tipo String)

  Output:

  *Comportamento: verifica che il membro **value** sia uguale (nello stato) all'argomento passato come parametro della funzione*

- *Definire la classe FrequentPattern che rappresenta un itemset (o pattern) frequente*

  Attributi

  *Item fp[]; array che contiene riferimenti a oggetti istanza della classe **Item** che definiscono il pattern*

  *float support; valore di supporto calcolato per il pattern **fp***

  Metodi

  FrequentPattern()

  Input:

Output:

*Comportamento: costruttore che alloca **fp** come array di dimensione 0 (fornito dal docente)*

FrequentPattern(FrequentPattern FP)

Input:

Output:

*Comportamento: costruttore che alloca **fp** e **support** come copia del frequent pattern FP passato (fornito dal docente)*

void addItem(Item item)

Input: oggetto Item da aggiungere al pattern

Output:

*Comportamento: si estende la dimensione di **fp** di 1 e si inserisce il ultima posizione l'argomento della procedura (fornito dal docente)*

Item getItem(int index)

*Input: posizione in **fp***

*Output: Item che occupa la posizione indicata in **fp***

*Comportamento: restituisce l'item in posizione **index** di **fp***

float getSupport()

Input:

Output: valore di supporto del pattern

*Comportamento: restituisce il membro **support***

int getPatternLength()

Input:

Output: lunghezza del pattern

*Comportamento: restituisce la dimensione (lunghezza) di **fp***

float computeSupport(Data data)

Input: valore di supporto del pattern nel dataset data

Output:

Comportamento: calcola il supporto del pattern rappresentato dall'oggetto this rispetto al dataset data passato come argomento (fornita dal docente).

void setSupport(float support)

Input: valore di supporto del pattern

Output:

*Comportamento: assegna al membro **support** il parametro della procedura*

public String toString() (fornito dal docente)

Input:

Output: stringa ripresentante lo item set e il suo supporto

*Comportamento: si scandisce **fp** al fine di concatenare in una stringa la rappresentazione degli item; alla fine si concatena il supporto*

■ *Definire la classe EmergingPattern che estende FrequenPattern e modella un pattern emergente*

Attributi

float growrate;

grow rate del pattern

Metodi

EmergingPattern(FrequentPattern fp, float growrate)

Input: pattern e grow rate del pattern

Output:

*Comportamento: chiama il costruttore della superclasse passandogli fp e inizializza il membro **growrate** con l'argomento del costruttore*

float getGrowRate()

Input:

Output: grow rate del pattern

*Comportamento: restituisce il valore del membro **growrate***

public String toString()

Input:

Output: stringa contenente il pattern emergente nella forma

"pattern [supporto] [growrate]

Comportamento: Si crea e restituisce la stringa che rappresenta la il pattern,il suo supporto e il suo growrate (fare uso del toString() ereditato da FrequentPattern

- *Definire la classe FrequentPatternMiner che include i metodi per la scoperta di pattern frequenti con Algoritmo APRIORI*
- *Attributi*
- *LinkList outputFP[]; lista che contiene riferimenti a oggetti istanza della classe **FrequentPattern** che definiscono il pattern*

```
frequentPatternDiscovery(D,minS)→P
begin
   P= ∅
   L1= {1-item che compaiono in minD×|D| transazioni di D}
   K=2
   while LK-1 ≠∅ do
   begin
        CK=  candidati generati da Lk-1 aggiungendo un nuovo item
        LK=∅
        for each (p ∈ Ck) do
               if (supporto(p, D) >= minS) then
                     LK=LK∪p
        P=P∪Lk
        K=K+1
   end
   return P
end
```

Metodi

FrequentPatternMiner(Data data,float minSup) (già fornito per il caso attributi discreti)

Input: l'insieme delle transazioni (data) e il minimo supporto (minSuUp)

Output:

Comportamento: Costruttore che genera tutti i pattern k=1 frequenti e per ognuno di questi genera quelli con k>1 richiamando expandFrequentPatterns () (fornita dal docente) I pattern sono memorizzati nel membro OutputFP

FrequentPattern refineFrequentPattern(FrequentPattern FP, Item item)

Input: pattern FP da raffinare, item da aggiungere ad FP

Output: nuovo pattern ottenuto per effetto del raffinamento

Comportamento: Crea un nuovo pattern a cui aggiunge tutti gli item di FP e il parametro item

LinkList expandFrequentPatterns(Data data, float minSup, Queue fpQueue, LinkList outputFP)

Input: l'insieme delle transazioni (data), minimo supporto (minSup), coda contente i pattern da valutare (fpQueue) e lista dei pattern frequenti già estratti (outputFP)

Output: : lista linkata popolata con pattern frequenti a k>1

Comportamento: Finché fpQueue contiene elementi, si estrae un elemento dalla coda fpQueue, si generano i raffinamenti per questo (aggiungendo un nuovo item non incluso). Per ogni raffinamento si verifica se è frequente e, in caso affermativo, lo si aggiunge sia ad fpQueue sia ad outputFP (fornita dal docente)

public String toString()

Input:

Output: Stringa rappresentante il valore di OutputFP

Comportamento: Scandisce OutputFp al fine di concatenare in un'unica stringa i pattern frequenti letti.

- *Definire la classe EmergingPatternMiner che modella la scoperta di emerging pattern partire dalla lista di frequent pattern*
- *Attributi*
- *LinkList epList[]; lista che contiene riferimenti a oggetti istanza della classe **EmergingPattern** che definiscono il pattern*

Metodi

EmergingPatternMiner(Data dataBackground,FrequentPatternMiner fpList,float minG)

Input: il dataset di background (dataBackground) su cui calcolare il growrate di tutti i pattern presenti in fpList considerando con minimo grow rate minG

Comportamento: Si scandiscono tutti i frequent pattern in fpList , per ognuno di essi si calcola il grow rate usando dataBackground e se tale valore è maggiore uguale di minG allora il pattern è aggiunto ad epList (fare uso del metodo computeEmergingPattern).

float computeGrowRate(Data dataBackground, FrequentPattern fp)

Input: l'insieme delle transazioni di background (dataBackground) e il pattern frequente (fp) di cui calcolare il growrate

Outpu0t: grow rate di fp

Comportamento: Si ottiene da fp il suo supporto relativo al dataset target. Si calcola il supporto di fp relativo al dataset di background. Si calcola il grow rate come rapporto dei due supporti.

EmergingPattern computeEmergingPattern(Data dataBackground ,FrequentPattern fp,float minGR)

Input: insieme di transazioni di background (dataBackground), frequent pattern , minimo grow rate (minGr)

Output: : restituisce l'emerging pattern creato da fp se la condizione sul grow rate è soddisfatta, null altrimenti

Comportamento: Verifica che il gorw rate di fp sia maggiore di minGR. In caso affermativo crea un oggetto EmemrgingPattern da fp.

public String toString()

Input:

Output: *Stringa rappresentante il valore di epList*

Comportamento: Scandisce epList al fine di concatenare in un'unica stringa le stringhe rappresentati i pattern emergenti letti.

Si definisca una classe MainTest con metodo main che consenta il test delle classi implementate, ed, in particolare, che permetta la scoperta dei pattern frequenti con minimo supporto 0.3 e pattern emergenti con minimo grow rate 1.0 (fornita dal docente)

Si modifichi il costruttore di Data usando il codice fornito dal docente.

Per l'output completo vedere il file output.txt