SS-ZG548: ADVANCED DATA MINING

03 Hashing & Pruning

Dr. Kamlesh Tiwari
Assistant Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

Aug 07, 2021 ONLINE (WILP @ BITS-Pilani July-Dec 2021)

http://ktiwari.in/adm

Recap: Association Rule Mining

Association Rule Mining does link analysis

- Set of items $I = \{i_1, i_2, ..., i_m\}$, set of transactions $T = \{t_1, t_2, ..., t_n\}$ where $t_i \subseteq I$
- An association rule $X \Rightarrow Y$ has a **support** s in set T if $s\%$ of the transactions in T contains $X \cup Y$

$$support(X \Rightarrow Y) = P(X \cup Y)$$

- The association rule $X \Rightarrow Y$ holds in the transaction set T with **confidence** c if $c\%$ of the transactions in T that contain X also contain Y.

$$confidence(X \Rightarrow Y) = P(Y|X)$$

- An association rule is an implication of the form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \phi$

Recap: It is a two-step process (Apriori[1])

1. *Find all frequent item sets:* $\{X : support(X) \geq S_{min}\}$
2. *Generate association rules from the frequent item set:* For any pair of frequent item set W and X satisfying $X \subset W$, of $support(X)/support(W) \geq C_{min}$, then $X \Rightarrow Y$ is a valid rule where $Y = W - X$.

Second step is straight forward so most of the research interest lies in solving the first part.

Apriori algorithm

- Starting from set of 1-item frequent sets L_1
- Uses k-item set to levelwise explore (k+1)-item set
- Process continues until there is no more candidate item sets

[1] Mining association rules between sets of items in large databases, *R Agrawal, T Imielinski, and A Swami*, SIGMOD, 22(2), pp 207–216, ACM-1993

Recap: Apriori at work

Consider transactions T

T_1=(A,B,C)
T_2=(A,F)
T_3=(A,B,C,E)
T_4=(A,B,D,F)
T_5=(C,F)
T_6=(A,B,C)
T_7=(A,B,C,E)
T_8=(C,D,E)
T_9=(B,D,E)

Scan T $\longrightarrow$ C_1

Item	Supp
{A}	6
{B}	6
{C}	6
{D}	3
{E}	4
{F}	3

$\longrightarrow$ L_1

Item	Supp
{A}	6
{B}	6
{C}	6
{E}	4

C_2

Item
{A,B}
{A,C}
{A,E}
{B,C}
{B,E}
{C,E}

Scan T $\longrightarrow$ C_2

Item	Supp
{A,B}	5
{A,C}	4
{A,E}	2
{B,C}	4
{B,E}	3
{C,E}	3

$\longrightarrow$ L_2

Item	Supp
{A,B}	5
{A,C}	4
{B,C}	4

C_3

Item
{A,B,C}

Scan T $\longrightarrow$ C_3

Item	Supp
{A,B,C}	4

$\longrightarrow$ L_3

Item	Supp
{A,B,C}	4

Recap: Generate Association Rules

If $X \subset W$ & $support(X)/support(W) \geq C_{min}$, then $X \Rightarrow W - X$

Transactions

T_1=(A,B,C)
T_2=(A,F)
T_3=(A,B,C,E)
T_4=(A,B,D,F)
T_5=(C,F)
T_6=(A,B,C)
T_7=(A,B,C,E)
T_8=(C,D,E)
T_9=(B,D,E)

- Our frequent item contains
  - $\{A\}_6$ $\{B\}_6$ $\{C\}_6$ $\{E\}_4$ $\{A,B\}_5$ $\{A,C\}_4$ $\{B,C\}_4$ $\{A,B,C\}_4$
- Possibilities are
  $A \Rightarrow B$, $B \Rightarrow A$, $A \Rightarrow C$, $C \Rightarrow A$, , $B \Rightarrow C$, $C \Rightarrow B$, $A \Rightarrow \{B, C\}$, $B \Rightarrow \{A, C\}$, $C \Rightarrow \{B, A\}$, $\{A, B\} \Rightarrow C$, $\{A, C\} \Rightarrow B$, $\{B, C\} \Rightarrow A$
- Let's take $C_{min} = 1.22$
- Association rules that qualifies as valid rule are shown green
  $A \Rightarrow B$, $B \Rightarrow A$, $A \Rightarrow C$, $C \Rightarrow A$, $B \Rightarrow C$, $C \Rightarrow B$, $A \Rightarrow \{B, C\}$, $B \Rightarrow \{A, C\}$, $C \Rightarrow \{B, A\}$, $\{A, B\} \Rightarrow C$, $\{A, C\} \Rightarrow B$, $\{B, C\} \Rightarrow A$

Hash Based Algorithm DHP

- Direct Hashing and Pruning (DHP[2]) uses hash functions to reduce the size of set C's. (it is similar to Apriori)

[2] An effective hash-based algorithm for mining association rules, *JS Park, MS Chen, and PS Yu* 24(2), ACM-1995

Hash Based Algorithm DHP

- Direct Hashing and Pruning (DHP[2]) uses hash functions to reduce the size of set C's. (it is similar to Apriori)
- This improve the computing efficiency while the number of candidate item sets to be checked is reduced.

[2] An effective hash-based algorithm for mining association rules, *JS Park, MS Chen, and PS Yu* 24(2), ACM-1995

Hash Based Algorithm DHP

- Direct Hashing and Pruning (DHP[2]) uses hash functions to reduce the size of set C's. (it is similar to Apriori)
- This improve the computing efficiency while the number of candidate item sets to be checked is reduced.
- Consider a hash function $h(x, y) = (10 * ID(x) + ID(y)) \bmod 7$

[2] An effective hash-based algorithm for mining association rules, *JS Park, MS Chen, and PS Yu* 24(2), ACM 1995

Hash Based Algorithm DHP

- Direct Hashing and Pruning (DHP[2]) uses hash functions to reduce the size of set C's. (it is similar to Apriori)
- This improve the computing efficiency while the number of candidate item sets to be checked is reduced.
- Consider a hash function $h(x, y) = (10 * ID(x) + ID(y)) \bmod 7$

Transactions T

$T_1 = (A, B, C)$	$\{A, B\}\{A, C\}\{B, C\}$
$T_2 = (A, F)$	$\{A, F\}$
$T_3 = (A, B, C, E)$	$\{A, B\}\{A, C\}\{A, E\}\{B, C\}\{B, E\}\{C, E\}$
$T_4 = (C, F)$	
$T_5 = (A, B, C)$	
$T_6 = (A, B, C, E)$	
$T_7 = (C, D, E)$	
$T_8 = (C, D, E)$	
$T_9 = (B, D, E)$	

[2] An effective hash-based algorithm for mining association rules, *JS Park, MS Chen, and PS Yu* 24(2), ACM 1995

Hash Based Algorithm DHP

- Direct Hashing and Pruning (DHP[2]) uses hash functions to reduce the size of set C's. (it is similar to Apriori)
- This improve the computing efficiency while the number of candidate item sets to be checked is reduced.
- Consider a hash function $h(x, y) = (10 * ID(x) + ID(y)) \bmod 7$

Transactions T	
$T_1 = (A, B, C)$	$\{A, B\}\{A, C\}\{B, C\}$
$T_2 = (A, F)$	$\{A, F\}$
$T_3 = (A, B, C, E)$	$\{A, B\}\{A, C\}\{A, E\}\{B, C\}\{B, E\}\{C, E\}$
$T_4 = (C, F)$	
$T_5 = (A, B, C)$	
$T_6 = (A, B, C, E)$	
$T_7 = (C, D, E)$	
$T_8 = (C, D, E)$	
$T_9 = (B, D, E)$	

Hash	Bucket	
0	{E,C}{A,D}{C,E}{C,E}	[4]
1	{A,E}{C,F}{A,E}	[3]
2	{B,C}{A,F}{B,C}{A,F}{B,C}{B,C}{D,E}{D,E}	[8]
3	{B,D}{B,D}	[2]
4	{B,F}{D,F}{B,E}{B,E}	[4]
5	{A,B}{A,B}{B,C}{B,A}{A,B}	[5]
6	{A,C}{A,C}{A,C}{A,C}{C,D}	[5]

[2] An effective hash-based algorithm for mining association rules, *JS Park, MS Chen, and PS Yu* 24(2), ACM-1995

Hash Based Algorithm DHP

- Direct Hashing and Pruning (DHP[2]) uses hash functions to reduce the size of set C's. (it is similar to Apriori)
- This improve the computing efficiency while the number of candidate item sets to be checked is reduced.
- Consider a hash function $h(x, y) = (10 * ID(x) + ID(y)) \bmod 7$

Transactions T

$T_1 = (A, B, C)$	$\{A, B\}\{A, C\}\{B, C\}$
$T_2 = (A, F)$	$\{A, F\}$
$T_3 = (A, B, C, E)$	$\{A, B\}\{A, C\}\{A, E\}\{B, C\}\{B, E\}\{C, E\}$
$T_4 = (C, F)$	
$T_5 = (A, B, C)$	
$T_6 = (A, B, C, E)$	
$T_7 = (C, D, E)$	
$T_8 = (C, D, E)$	
$T_9 = (B, D, E)$	

Hash	Bucket	
0	{E,C}{A,D}{C,E} {C,E}	[4]
1	{A,E}{C,F}{A,E}	[3]
2	{B,C}{A,F}{B,C}{A,F} {B,C}{B,C}{D,E}{D,E}	[8]
3	{B,D}{B,D}	[2]
4	{B,F}{D,F}{B,E}{B,E}	[4]
5	{A,B}{A,B}{B,C}{B,A} {A,B}	[5]
6	{A,C}{A,C}{A,C}{A,C} {C,D}	[5]

Note: total counts for buckets 1 and 3 may not satisfy the minimum support constraint, therefore, {A E}, should not be included in C_2.

[2] An effective hash-based algorithm for mining association rules, *JS Park, MS Chen, and PS Yu* 24(2), ACM-1995

Partition Based Algorithm

- Essentially based on a partition-based heuristic.
- A frequent item set in database D having n partitions $p_1, p_2, ..., p_n$ must be a frequent item set in at least one of the n partitions

Partition Based Algorithm

- Essentially based on a partition-based heuristic.
- A frequent item set in database D having n partitions $p_1, p_2, ..., p_n$ must be a frequent item set in at least one of the n partitions
- It first scans partitions $p_1, p_2, ..., p_n$ to find local frequent item sets

Partition Based Algorithm

- Essentially based on a partition-based heuristic.
- A frequent item set in database D having n partitions $p_1, p_2, ..., p_n$ must be a frequent item set in at least one of the n partitions
- It first scans partitions $p_1, p_2, ..., p_n$ to find local frequent item sets
- Set of candidate item sets is constructed by taking the union

Partition Based Algorithm

- Essentially based on a partition-based heuristic.
- A frequent item set in database D having n partitions $p_1, p_2, ..., p_n$ must be a frequent item set in at least one of the n partitions
- It first scans partitions $p_1, p_2, ..., p_n$ to find local frequent item sets
- Set of candidate item sets is constructed by taking the union
- Scans the D second time to calculate the support of each item set

Partition Based Algorithm

- Essentially based on a partition-based heuristic.
- A frequent item set in database D having n partitions $p_1, p_2, ..., p_n$ must be a frequent item set in at least one of the n partitions
- It first scans partitions $p_1, p_2, ..., p_n$ to find local frequent item sets
- Set of candidate item sets is constructed by taking the union
- Scans the D second time to calculate the support of each item set

$T_1 = (A, B, C)$
$T_2 = (A, F)$
$T_3 = (A, B, C, E)$
P_1

$T_4 = (A, B, D, F)$
$T_5 = (C, F)$
$T_6 = (A, B, C)$
P_2

$T_7 = (A, B, C, E)$
$T_8 = (C, D, E)$
$T_9 = (B, D, E)$
P_3

Partition Based Algorithm

- Essentially based on a partition-based heuristic.
- A frequent item set in database *D* having *n* partitions $p_1, p_2, ..., p_n$ must be a frequent item set in at least one of the *n* partitions
- It first scans partitions $p_1, p_2, ..., p_n$ to find local frequent item sets
- Set of candidate item sets is constructed by taking the union
- Scans the *D* second time to calculate the support of each item set

$T_1 = (A, B, C)$
$T_2 = (A, F)$
$T_3 = (A, B, C, E)$ P_1

$T_4 = (A, B, D, F)$
$T_5 = (C, F)$
$T_6 = (A, B, C)$ P_2

$T_7 = (A, B, C, E)$
$T_8 = (C, D, E)$
$T_9 = (B, D, E)$ P_3

Partition	Frequent Item set
P_1	$\{A\}\ \{B\}\ \{C\}\ \{AB\}\ \{AC\}\ \{BC\}\ \{ABC\}$
P_2	$\{A\}\ \{B\}\ \{C\}\ \{F\}\ \{AB\}$
P_3	$\{B\}\ \{C\}\ \{D\}\ \{E\}\ \{BE\}\ \{CE\}\ \{DE\}$

Partition Based Algorithm

- Essentially based on a partition-based heuristic.
- A frequent item set in database D having n partitions $p_1, p_2, ..., p_n$ must be a frequent item set in at least one of the n partitions
- It first scans partitions $p_1, p_2, ..., p_n$ to find local frequent item sets
- Set of candidate item sets is constructed by taking the union
- Scans the D second time to calculate the support of each item set

$T_1 = (A, B, C)$ $T_2 = (A, F)$ $T_3 = (A, B, C, E)$		P_1
$T_4 = (A, B, D, F)$ $T_5 = (C, F)$ $T_6 = (A, B, C)$		P_2
$T_7 = (A, B, C, E)$ $T_8 = (C, D, E)$ $T_9 = (B, D, E)$		P_3

Partition	Frequent Item set
P_1	$\{A\}$ $\{B\}$ $\{C\}$ $\{AB\}$ $\{AC\}$ $\{BC\}$ $\{ABC\}$
P_2	$\{A\}$ $\{B\}$ $\{C\}$ $\{F\}$ $\{AB\}$
P_3	$\{B\}$ $\{C\}$ $\{D\}$ $\{E\}$ $\{BE\}$ $\{CE\}$ $\{DE\}$

Union is taken

Candidate item sets
$\{A\}$ $\{B\}$ $\{C\}$ $\{D\}$ $\{E\}$ $\{F\}$ $\{AB\}$ $\{AC\}$ $\{BC\}$ $\{CE\}$ $\{DE\}$ $\{ABC\}$

Second scan of D examines the required support

Incremental Association Rule Mining

- Real databases are generally dynamic (not static)

Incremental Association Rule Mining

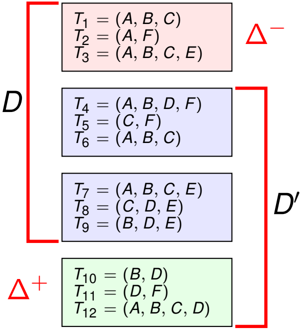
- Real databases are generally dynamic (not static)
- New transactions are generated and old transactions may be obsolete over time

Incremental Association Rule Mining

- Real databases are generally dynamic (not static)
- New transactions are generated and old transactions may be obsolete over time

Premitives:

- Let D be the initial database
- $\triangle^-$ portion of the database becomes obsolete
- Therefore, the database is left with $D^- = D - \triangle^-$
- $\triangle^+$ more transactions are added
- The database becomes $D' = D^- + \triangle^+ = D - \triangle^- + \triangle^+$



Incremental update can avoid redoing mining on updated database

Incremental Association Rule Mining (contd..)

1. The update problem can be reduced to finding the new set of frequent item sets. After that, the new association rules can be computed from the new frequent item sets.
2. An old frequent item set has the potential to become infrequent in the updated database.
3. Similarly, an old infrequent item set could become frequent in the new database.
4. In order to find the new frequent item sets "exactly", all the records in the updated database, including those from the original database, have to be checked against every candidate set.

We would evaluate two algorithms *viz*. FUP and FUP2

Algorithm FUP

- FUP stands for **F**ast **UP**date. [3]
- Can handle insertions only
- Specifically, given the original database D and its corresponding frequent item set $L = \{L_1, L_2, ..., L_k\}$.
  - The goal is to reuse the information to efficiently obtain $L' = \{L'_1, L'_2, ..., L'_k\}$ for new database $D' = D \cup \triangle^+$

By utilizing the definition of support and constraint of minimum support S_{min}, following is used by FUP.

- An original frequent item set $X \in L$, becomes infrequent in D' iff $support(X)_{D'} < S_{min}$
- An item set $X \notin L$, becomes frequent in D' iff $support(X)_{\triangle^+} \geq S_{min}$
- If a k-item set X whose $(k-1)$-subset(s) becomes infrequent, $i.e.$, the subset is in L_{k-1} but not in L'_{k-1}, then X must be infrequent in D'.

[3] Maintenance of discovered association rules in large databases: An incremental updating technique, *DW Cheung, and J Han, and V Ng, and CY Wong*, International conference on data engineering, pp 106–114, IEEE 1996

FUP at work

Consider the database D and the related frequent set discovered with Apriori

$T_1 = (A, B, C)$
$T_2 = (A, F)$
$T_3 = (A, B, C, E)$
$T_4 = (A, B, D, F)$
$T_5 = (C, F)$
$T_6 = (A, B, C)$
$T_7 = (A, B, C, E)$
$T_8 = (C, D, E)$
$T_9 = (B, D, E)$

Item set	Support
{A}	6/9
{B}	6/9
{C}	6/9
{E}	4/9
{A B}	5/9
{A C}	4/9
{B C}	4/9
{A B C}	4/9

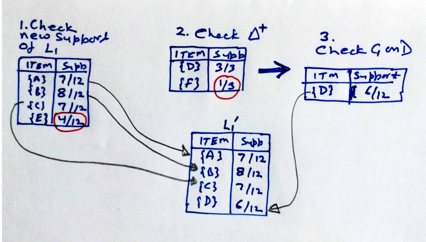
Consider the arrival of $\triangle^+$ more transactions

$T_1 = (A, B, C)$
$T_2 = (A, F)$
$T_3 = (A, B, C, E)$
$T_4 = (A, B, D, F)$
$T_5 = (C, F)$
$T_6 = (A, B, C)$
$T_7 = (A, B, C, E)$
$T_8 = (C, D, E)$
$T_9 = (B, D, E)$

Δ^+

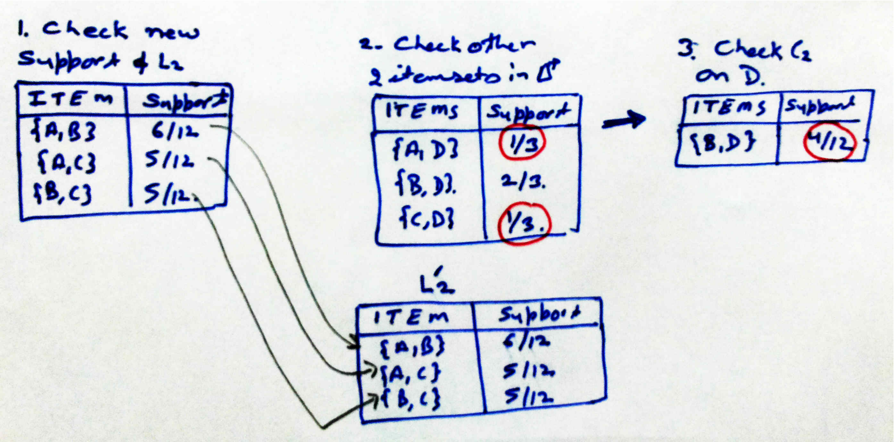
$T_{10} = (B, D)$
$T_{11} = (D, F)$
$T_{12} = (A, B, C, D)$

The first iteration, is as below.



FUP at work (contd...)

The second iteration, is as below.



Similarly it is executed for next levels.

FUP_2

- FUP_2 [4] can work for both $\triangle^-$ and $\triangle^+$
- L_k from previous mining result is used for dividing candidate itemset C_k into two parts
  - $P_k = C_k \cap L_k$
  - $Q_k = C_k - P_k$
- Itemset that is frequent in $\triangle^-$, must be infrequent in D^-.
- Further if itemset in Q_k in infrequent in $\triangle^+$ then it is infrequent in D^-.
- This technique helps to effectively reduce number of candidate itemsets.

[4] A general incremental technique for maintaining discovered association rules, *DW Cheung, SD Lee, and B Kao*, Database Systems For Advanced Applications, pp: 185–194, World Scientific-1997

FUP_2 at work

D

Δ^-
$T_1 = (A, B, C)$
$T_2 = (A, F)$
$T_3 = (A, B, C, E)$

$T_4 = (A, B, D, F)$
$T_5 = (C, F)$
$T_6 = (A, B, C)$

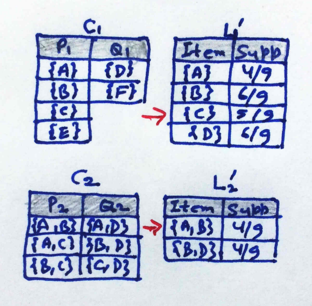
$T_7 = (A, B, C, E)$
$T_8 = (C, D, E)$
$T_9 = (B, D, E)$

Δ^+
$T_{10} = (B, D)$
$T_{11} = (D, F)$
$T_{12} = (A, B, C, D)$

D'

Item set	Support
{A}	6/9
{B}	6/9
{C}	6/9
{E}	4/9
{A B}	5/9
{A C}	4/9
{B C}	4/9
{A B C}	4/9

Frequent itemsets of D



- C_1 is set of all items. It is divided in P_i and Q_i
- Being frequent, support for all items in P_i is known. It could be updated using $\triangle^-$ and $\triangle^+$ only.
- $\text{Count}(\{A\})_{D'} = \text{Count}(\{A\})_D - \text{Count}(\{A\})_{\triangle^-} + \text{Count}(\{A\})_{\triangle^+}$
  $= 6 - 3 + 1 = 4$

FUP$_2$ at work

- In some cases only the scan of $\triangle^-$ and $\triangle^+$ is required.
- For example, $\text{Count}(\{\text{F}\})_{\triangle^+} - \text{Count}(\{\text{F}\})_{\triangle^-} = 0$ showing that support of $\{\text{F}\}$ can not be improved.
- Consequently, fewer itemsets have to be further scanned
- An iteration finishes when all the itemsets in P_i and Q_i are verified, and new set of frequent itemsets L_i' is generated

D:

Δ^-:

$T_1 = (A, B, C)$
$T_2 = (A, F)$
$T_3 = (A, B, C, E)$

D':

$T_4 = (A, B, D, F)$
$T_5 = (C, F)$
$T_6 = (A, B, C)$

$T_7 = (A, B, C, E)$
$T_8 = (C, D, E)$
$T_9 = (B, D, E)$

Δ^+:

$T_{10} = (B, D)$
$T_{11} = (D, F)$
$T_{12} = (A, B, C, D)$

FUP$_2$H

Uses hashing for performance improvement

Variations of FUP

- **Update With Early Pruning (UWEP):** Occurrence of potentially huge set of candidate itemset and multiple scans of the database is the issue
  - If a k-itemset is frequent in $_{\triangle^+}$ but infrequent in D', it is not considered when generating C_{k+1}
  - This can significantly reduce the number of candidate itemsets, with the trade-off that an additional set of unchecked itemsets has to be maintained.
- **Utilizing Negative Borders):** Negative border set consists of all itemsets that are closest to be frequent
  - Negative border consists of all itemsets that were candidates of level-vise method but did not have enough support

$$Bd^{-}(L) = C_k - L_k$$

  - Find negative border set for $L = \{\{A\}, \{B\}, \{C\}, \{E\}, \{AB\}, \{AC\}, \{BC\}, \{ABC\}\}$
  - Full scan of dataset is only required when *itemsets outside negative border set* get added to frequent itemsets or negative border set.

Thank You!

Thank you very much for your attention!

Queries ?