ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages : 68 – 72 (2014)
Special Issue of ICETETS 2014 - Held on 24-25 February, 2014 in Malla Reddy Institute of Engineering and Technology, Secunderabad– 14, AP, India

Efficient Attribute Selection to stand out in the market

Mr.Murlidher Mourya*, Mr. J.phani prasad**
* Computer Science, Vardhaman College of Engineering,India,murli_cool9@yahoo.com
**Computer Science, Vardhaman College of Engineering,India,phanimtechcse@gmail.com

Abstract:-Mining of frequent item sets is one of the most fundamental problems in data mining applications. My proposed algorithm which guides the seller to select the best attributes of a new product to be inserted in the database so that it stands out in the existing competitive products, due to budget constraints there is a limit, say m, on the number of attribute that can be selected for the entry into the database. Although the problems are NPcomplete.The Approximation algorithm are based on greedy heuristics. DCIP algorithm uses data-set condensing and intersection pruning to find the maximal frequent item set. The condensing process is performed by deleting items in infrequent 1-itemset and merging duplicate transactions repeatedly; the pruning process is performed by generating intersections of transactions and deleting unneeded subsets recursively. This algorithm differs from all classical maximal frequent item set discovering algorithms; experiments show that this algorithm is valid with moderate efficiency; it is also easy to code for use in KDD applications.

Keywords: Association rules, Data mining, Mining frequent item sets, intersection pruning, and data-set condensing

I.INTRODUCTION:

In recent years there has been development of ranking functions and efficient top-k retrieval algorithms which help the users in mining Frequent items set which plays a major role in many data mining applications .examples include: users wishing to search databases and catalogs of products such as homes, cars, cameras, or articles such as news and job ads. Users browsing these databases typically execute search queries via public front-end interfaces to these databases. Typical queries may specify sets of keywords in case of text databases or the desired values of certain attributes in case of structured relational databases. The query answering system answers `such queries by either returning all data objects that satisfy the query conditions, or may rank and return the top-k data objects, or return the results that are on the query's skyline. If ranking is employed, the ranking may either be simplistic—e.g., objects are ranked by an attribute such as Price; or more sophisticated—e.g., objects may be ranked by the degree of "relevance" to the query.

***Attributes selection*:**

There are two types of users of these databases. Buyers of products who search such database trying to locate objects of interest ,while the latter type of user are sellers of products who insert new objects into these databases in the hope that they will be easily discovered by the buyers i.e it must stands out in the existing competitive products. To understand it a little better consider the following scenario: If a real estate seller wants to give an add on the news paper about sale of flats, He has to choose the best features of the flats, that are the most of the customers are interested. If he has given an add with some features (or attributes), and if no customer is interested on those features, then the add may not add value to his advertisement .If he has a system, that can suggest top k attributes (or features) of the product, then he can give a very good add, and that add will be referred by more number of customers. General problem also arises in domains beyond e-commerce applications. For example, in the design of a new product, a manufacturer may be interested in selecting the 10 best features from a large wish-list of possible features—e.g., a homebuilder can find out that adding a swimming pool really increases visibility of a new home in a certain neighborhood. *The problem here is selecting the proper and the best attributes of the flats, to give a good advertisement that is more number of customers are interested.*
To define our problem more formally, we need to develop a few abstractions. Let D be the database of products already being advertised in the marketplace (i.e., the "competition"). Let Q be the set of search queries that have been executed against this database in the recent past—thus Q is the "workload" or "query log." The query log is our primary model of what past potential buyers have been interested in. For a new product that needs to be inserted into this database, we assume that the seller has a complete "ideal" description of the product. But due to budget constraints, there is a limit, say m, on the number of attributes/keywords that can be selected for entry into the database. Our problem can now be defined as follows.

II. PROBLEM FRAMEWORK

Given a database D, a query log Q, a new tuple t, and an integer m, determine the best (i.e., top-m) attributes of t to retain such that if the shortened version of t is inserted into the database, the number of queries of Q that retrieve t is maximized.

PRELIMINARIES

First we provide some useful definitions
Let car attribute set A={AC, four door, turbo, power door, remote keyless entry, anti lock brakes, auto trans, GPS system, power breaks,side bags} are represented as {I1,I2,I3,I4.I5,I6,I7,I8,I9,I10} respectively.

Table 1: Database

CAR ID	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
1	0	1	0	1	0	0	0	1	0	1
2	0	1	1	0	0	1	0	0	0	1
3	1	0	0	1	1	0	1	1	1	0
4	1	1	0	1	1	1	0	0	1	0
5	1	1	0	0	0	1	0	0	0	1
6	0	1	0	1	1	0	0	1	0	1
7	0	0	1	1	0	0	0	1	0	0

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages : 68 – 72 (2014)
Special Issue of ICETETS 2014 - Held on 24-25 February, 2014 in Malla Reddy Institute of Engineering and Technology, Secunderabad– 14, AP, India

Boolean database

Let D = {t1 . . . tN} be a collection of Boolean tuples over the attribute set A = {a1 . . . aM}, where each tuple t is a bit-vector where a 0 implies the absence of a feature and a 1 implies the presence of a feature. A tuple t may also be considered as a subset of A, where an attribute belongs to t if its value in the bit-vector is 1.

Table 2: Query Log Q

T ID	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
001	1	1	0	1	1	0	1	0	0	0
002	1	1	0	0	1	1	1	0	0	0
003	0	0	1	0	1	0	1	0	0	1
004	0	0	1	0	0	0	0	1	0	1
005	1	1	1	1	0	0	1	0	0	0
006	0	1	1	0	0	0	1	1	0	0
007	0	0	1	0	0	1	0	0	1	0
008	1	0	1	0	1	0	0	0	1	0
009	1	1	0	0	0	1	0	0	0	0
010	0	0	1	1	0	0	0	1	1	0

Table 3: New Tuple T to be inserted

Car ID	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
8	1	1	1	1	0	1	1	0	0	0

Tuple domination.

Let t1 and t2 be two tuples such that for all attributes for which tuple t1 has value 1, tuple t2 also has value 1. In this case, we say that t2 dominates t1.

Tuple compression.

Let t be a tuple and let t' be a subset of t with m attributes. Thus, t' represents a compressed representation of t. equivalently, in the bit-vector representation of t, we retain only m 1s and convert the rest to 0s. *Query log*. Let Q = {q1 . . . qS} be collection of queries where each query q defines a subset of attributes. The following running example will be used throughout the paper to illustrate various concepts

Conjunctive Boolean - Query Log (CB-QL): Given a query log Q with Conjunctive Boolean Retrieval semantics, a new tuple t, and an integer m, computes a compressed tuple t′ having m attributes such that the number of queries that retrieve t′ is maximized. Intuitively, for buyers interested in browsing products of interest, we wish to ensure that the compressed version of the new product is visible to as many buyers as possible.

Selecting the threshold value:

There are two alternate approaches to setting the threshold. One approach is essentially a heuristic, where we set the threshold to a reasonable fixed value dictated by the practicalities of the application. Threshold enforces that attributes should be selected such that the compressed tuple is satisfied by a certain minimum number of queries. For example, a threshold of 1 percent means that we are not interested in results that satisfy less than 1 percent of the queries in the query log.

III. RELATED WORK

A large corpus of work has tackled the problem of ranking the results of a query. In the documents world, the most popular techniques are count based ranking functions, like BM25, as well as link-structure-based techniques like Page Rank if such links are present (e.g., the Web). In the database world, automatic ranking techniques for the results of structured queries have been proposed. Also there has been recent work on ordering the displayed attributes of query results.

Both of these tuple and attribute ranking techniques are inapplicable to our problem. The former inputs a database and a query, and outputs a list of database tuples according to a ranking function, and the latter inputs the list of database results and selects a set of attributes that "explain" these results. In contrast, our problem inputs a database, a query log, and a new tuple, and computes a set of attributes that will rank the tuple high for as many queries in the query log as possible.

Although the problem of choosing attribute is related to the area of feature selection, our work differs from the work on feature selection because our goal is very specific—to enable a new tuple to be highly visible to the database users and not to reduce the cost of building a mining model such as classification or clustering.

PINCER SEARCH ALGORITHM

Most of the algorithms used for mining maximal frequent item sets perform fairly well when the length of the maximal frequent item set is small. However, performance degrades when the length of the maximal frequent item set is large, since in the bottom-up approach, the maximal frequent item set is obtained only after traversing all its subsets.

The Pincer-search algorithm (Lin and Kedem, 1998, 2002), proposes a new approach for mining maximal frequent item sets. It reduces the complexity by combining both top-down and bottom-up methods for generating maximal item sets. The bottom-up search starts from 1-itemset and proceeds up to *n*-item sets as in Apriori while the top-down search starts from n- item sets and proceeds up to 1 item set. Both bottoms-up and top-down searches identify the maximal frequent itemsets by examining its candidates individually. Bottom-up search moves one-level up during a single pass whereas top-down search moves many levels down during a single pass. During the execution, all the item sets are classified into 3 categories

Frequent: Item sets whose support is greater than min_sup are classified as frequent *Infrequent*: Item sets whose support is less than min_sup are classified as infrequent *Unclassified:* All other Item sets are said to be unclassified.

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages : 68 – 72 (2014)
Special Issue of ICETETS 2014 - Held on 24-25 February, 2014 in Malla Reddy Institute of Engineering and Technology, Secunderabad– 14, AP, India

The Pincer algorithm is illustrated for the sample data source given in Table 4. The same data source will be used later for illustrating the proposed algorithm. An example of pincer search is shown in figure 1.

Pincer algorithm uses the following two properties to classify the unclassified item sets *Property 1:* If an item set is infrequent, all its supersets must be infrequent and they need not be examined further *Property 2:* If an item set is frequent, all its subsets must be frequent and they need not be examined further

Table 4: Sample data source

TID	ITEMS
T1	a,d,e,g,j
T2	a,b
T3	a,b,c,e,h
T4	a,b,c,d
T5	a,b,c,d,f,i
T6	a,b,c
T7	a,b,c,d,f,i
T8	a,b,c,e
T9	j

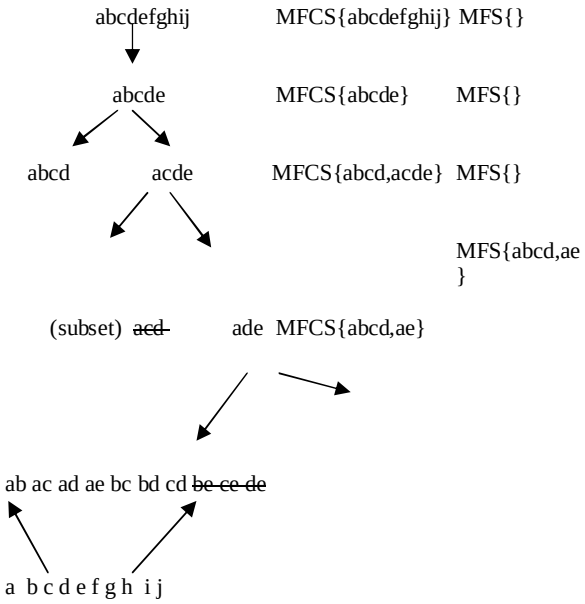

Figure 1: Pincer search

IV. PROPOSED TECHNIQUE: DCIP ALGORITHM

The first step of DCIP algorithm is to reduce the length of itemsets and the volume of data-set. According to Lemma 1, any maximal frequent itemset is also a maximal frequent itemset corresponding to one transaction in D, so find all maximal frequent itemsets correspond to every transaction through intersection pruning, merge them into one set (denoted as FS hereinafter), then delete all infrequent maximal itemset in FS, and the remaining set is maximal frequent itemset. The two main processes are described as follows.

A. Condensing the Data-set

This process first sorts the data-set with descending order according to the length of its itemsets, then moves those high-dimensional transactions whose support are bigger than minimal support threshold to a frequent itemset, and deletes all subsets of those transactions to condense the data-set. These steps are as follows:

Step 1: Scan the data-set, finding all frequent 1-itemset;

Step 2: Scan the data-set, deleting all items infrequent 1-itemset from all transactions; then add up identical transactions (i.e., if transaction T1=T2, let support(T1) = support(T1) + support(T2), and delete T2 from data-sets). Sorting the data-set descendingly according to the length of itemsets to form a new data-set which we denote as C;

Step 3: Process every transaction Ti in C whose support are bigger than minimal support threshold: move T_i to FS and delete all T_j ($T_j \subset T_i$, >i);

Step 4: Delete non-MFI from FS;

Step 5: End.

B. Intersection Pruning

Any maximal frequent itemset is also the maximal frequent itemset corresponding to a certain transaction in D; merge all maximal frequent itemset corresponding to every transaction into one set (which we denote as FS), then delete all non-frequent maximal itemsets in FS, and the remaining set is the maximal frequent itemset. These steps are as follows:

Assume we have a data-set denoted as D, and the minimal support threshold is S.

Step 1: Condense data-set D using the method described in 3.1; if |D|<S, terminate the processing for the current data-set;

Step 2: Find intersection of T_1 and T_i($1<i\leqslant n$); merge all intersections into a new data-set D1; establish the vertical data format of D; delete transaction T_i ($T_i \subset T_1$); if $|D1| \geqslant S$, then go to step 1 to perform another intersection pruning circle for D1;

Step 3: Use the vertical data format of D to find the intersection of T_j and T_i (j=2, 3, 4, ..., m<n; $j<i\leqslant n$), merge all intersections into a new data-set D1, go to step 1 to perform another intersection pruning circle for D1; when the volume of the remaining data-set is less than S, stop finding intersections of T_j and T_i, terminate the process for current data-set.

Step 4: End;

Note: Data-set condensing can be performed at the beginning of the intersection pruning process, as well as in the process of step 3.

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages : 68 – 72 (2014)
Special Issue of ICETETS 2014 - Held on 24-25 February, 2014 in Malla Reddy Institute of Engineering and Technology, Secunderabad– 14, AP, India

C. Instance Analysis

The following example shows how to discover MFI using DCIP for transaction database D (Table I) with minimum support threshold as 4 (i.e., minsup=4).

TABLE I.
TRANSACTION DATA-SET D

TID	Items
001	I1,I2,I4,I5,I7
002	I1,I2,I5,I6,I7
003	I0,I3,I5,I7
004	I0,I3,I8
005	I1,I2,I3,I4,I7
006	I2,I3,I7,I8
007	I3,I6,I9
008	I1,I3,I5,I9
009	I1,I2,I6
010	I3,I4,I8,I9

Step 1: Condense transaction data-set D using the method in 3.1, the result is shown in Table II;

TABLE II.
RESULT OF CONDENSED D

TID	items	Count	del
1	I1,I2,I5,I7	2	
2	I1,I2,I3,I7	1	
3	I3,I5,I7	1	
4	I2,I3,I7	1	1
5	I1,I3,I5	1	
6	I1,I2	1	1

Attribute Count is the count of corresponding transactions; attribute del indicate whether the corresponding transaction can be ignored in later processing, for example, after step 2, T6 can be ignored.

Step 2: Find intersections of T_1 and T_i (i=2, 3... 7), merge all intersections into data-set D1, as shown in Table III:

TABLE III.
INTERSECTION DATA-SET FOR T1 IN TABLE II

TID	items	Count	del
1	I1,I2,I7	1(+2)	
2	I5,I7	1(+2)	
3	I2,I7	1(+2)	1
4	I1,I5	1(+2)	
5	I1,I2	1(+2)	1

Establish vertical data format for D; because in Table II, $T_6 \subset T_1$, T_6.del=1 (see Table II) . The (+2) for attribute Count in table III is the count of T_1 in Table II.

Step 3: Condense the data-sets in Table III; as this example, the result remains no change.

Step 4: Find intersections of T_1 and T_i (i=2, 3, 4, 5) in Table III respectively, merge them into a new data-set D1, as shown in Table IV.

TABLE IV.
INTERSECTION DATA-SET FOR T1 IN TABLE III

TID	items	Count	del
1	I2,I7	1(+2+1)	
2	I1,I2	1(+2+1)	

Because T_3 and T_5 are subset of T_1 in Table III, delete T_3 and T_5;

Step 5: Condense the data-set in Table IV, produce frequent itemset {{I2, I7}:4, {I1, I2}:4}; Table IV is now empty after condensing;

Step 6: Back to Table III, T_3 and T_5 has been deleted, we only need to find the intersection of T_2 and T_4; but the length of T_2 and T_4 are both 2, no need to find intersection of them.

Step 7: Back to Table II, because T_6 has been deleted, we only need to find the intersections of T_2 and T_i (i=3, 4, 5); merge all intersections into a new data-set D1, as shown in Table V.

Because $T_4 \subset T_2$ in Table II, it should be deleted.

Step 8: Condense the data-set in Table V; after Condensing the result is empty;

Step 9: The original data- set D has 10 transactions; Table II shows that 30% (3 transactions) of them has been processed; condense again the remaining data-set in Table II, and the result is empty. The process ends.

Step 10: Merge all resulting frequent item sets, and delete all non-frequent maximal item sets, the final result of MFI is {{I2, I7}: 4, {I1, I2}: 4}.

The steps above use 14 times of intersection calculations for MFI; compared with other Apriori-like algorithms, its simplicity and efficiency is explicit.

Note: Because the volume of the example data-set D is small (only 10), the above process does not include the utilizing of vertical data format; the reason of introducing vertical data format is to reduce the number of times of finding the intersections.

TABLE V.
INTERSECTION DATA-SET FOR T2 IN TABLE 2

TID	items	Count	del
1	I3,I7	1(+1)	
2	I2,I3,I7	1(+1)	
3	I1,I3	1(+1)	

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages : 68 – 72 (2014)
Special Issue of ICETETS 2014 - Held on 24-25 February, 2014 in Malla Reddy Institute of Engineering and Technology, Secunderabad– 14, AP, India

V. PERFORMANCE STUDY

If the length of the longest transaction item set is L, the depth of recursive calling of the algorithm itself is less than L-2. The number of calculations for intersections is negatively correlated with support, number of deleted transactions, and number of duplicated transactions. This algorithm can also be implemented parallels for each transaction's maximal frequent itemset to get more efficiency. It is valid for both long and short frequent pattern mining applications; for vast volume of data-set, its usability retains because of the time & space cost increases not very drastically.

Another advantage of DCIP algorithm is its easy implementation. It is coded and tested using PowerBuilder script language on a microcomputer with Pentium IV/1.80GHz CPU, 512M memory running Windows XP operation system. Testing dataset is extracted from a supermarket's sales record.3000, 5000, 10000 and 20000 transactions are tested respectively with each record having 2-10 categories of commodity (the average number of categories is 6). Figure 1 shows the running time for different volume of datasets with minimum support threshold of 5%, 20% and 50%respectively. The bigger minimum support threshold, the lesser time needed for MFI.

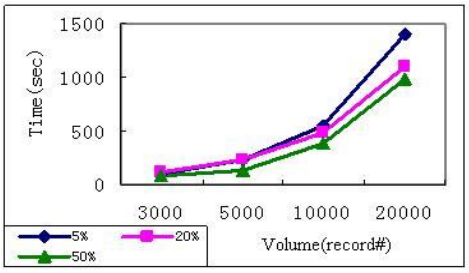

Figure 1. Performance test for multiple data-set & supports

VI. CONCLUSION

DCIP provides a new and efficient algorithm for discovering MFI; it condenses data-set by deleting items in infrequent 1-itemsets and merging duplicate transactions repeatedly, and utilizes the intersections of (1-s)*|D|+1 transactions with other transaction item sets to perform pruning; along with the discovering process, with the increasing of the number of deleted transactions, the number of times needed for calculating intersections will decrease rapidly. It's time & space cost increases not drastically when data-set volume increases, so its usability retains for MFI applications for high volume data-sets.

The DCIP algorithm can be further optimized in various aspects, such as keep a record of all resulting intersections to avoid duplicated generation of identical intersections to further improve the efficiency of this algorithm.

While the problems considered in this paper are novel and important to the area of ad hoc data exploration and retrieval, we observe that our specific problem definition does have limitations. After all, a query log is only an approximate surrogate of real user preferences, and moreover, in some applications neither the database, nor the query log may be available for analysis; thus, we have to make assumptions about the nature of the competition as well as about the user preferences. Finally, in all these problems,

VII. REFERENCES

[1]*D. Burdick, M. Calimlim, and J. Gehrke,* (2001)"MAFIA: A Maximal Frequent Item Set Algorithm for Transactional Databases," Proc. Int'l Conf. Data Eng. (ICDE), 2001.

[2]*M.R. Garey and D.S. Johnson* (1979), "Computers and Intractability: A Guide to the Theory of NP-Completeness".

[3]W.H. Freeman, K. Gouda and M.J. Zaki, (2001) "Efficiently Mining Maximal Frequent Itemsets," Proc. Int'l Conf. Data Mining (ICDM),

[4]J. Han, J. Pei, and Y. Yin, (2000) "Mining Frequent Patterns without Candidate Generation," Proc. SIGMOD Conf., pp. 1-12,.

J. Han, J. Wang, Y. Lu, and P. Tzvetkov, (2002) "Mining Top-k Frequent Closed Patterns without Minimum Support," Proc. Int'l Conf. Data Mining (ICDM),.

[5]M.D. Morse, J.M. Patel, and H.V. Jagadish, (2007) "Efficient Skyline Computation over Low-Cardinality Domains," Proc. Int'l Conf.Very Large Data Bases (VLDB),.

[6]M. Miah, G. Das, V. Hristidis, and H. Mannila, (2008) "Standing Out in a Crowd: Selecting Attributes for Maximum Visibility," Proc. Int'l Conf. Data Eng. (ICDE), pp. 356-365, 2008.