Frequent Itemsets Mining with Differential Privacy over Large-scale Data

Ms.V.R.SWETHA[1], P.POOJITHA[2]

[1]Associate Professor, Dept of MCA, Audisankara College of Engineering &Technology (AUTONOMOUS), Gudur, Tirupati (Dt), AP, India

[2]PG Scholar, Dept of MCA, Audisankara College Of Engineering &Technology (AUTONOMOUS) Gudur, Tirupati (Dt), AP, India.

Abstract:

Frequent Item set mining is one of the enormous operations which reestablish all thing sets in the trade table that occur as a subset of decided segment of the trades. The current counts disregard to deal with unremitting thing sets successfully on colossal data, since they either require various pass analyzes on the table or manufacture complex data structures which ordinarily outperform the open memory on massive data. This paper proposes a novel pre computation based item set mining (PFIM) count to enlist the persistent thing sets quickly on enormous data. PFIM views the trade table as two segments: a gigantic old table taking care of chronicled data and a by and large minimal new table taking care of as of late delivered data. PFIM first pre-builds up the semi progressive thing sets on the old table whose supports are over the lower-bound of the sensible assistance level. Given the foreordained assistance limit, PFIM can quickly reestablish the important ordinary thing

sets on the table by utilizing the semi unremitting thing sets. Three pruning rules are acquainted with decrease the size of the included up-and-comers. A slow update method is devised to capably re-assemble the semi consistent thing sets when the tables are consolidated. The expansive test outcomes, coordinated on designed and veritable datasets, show that PFIM has an immense favored situation over the current counts and run two critical degrees speedier than the latest estimation.

1. INTRODUCTION

Frequent item set mining is a significant activity, which has been broadly concentrated in numerous useful applications, for example, information mining, programming bug discovery, spatiotemporal information investigation and natural examination. Given an exchange table, wherein every exchange contains a lot of things, successive item set mining restores all arrangements of things whose frequencies (likewise alluded to as help of the arrangement of things) in the table are over a given limit.

Because of its functional significance, since initially proposed successive item set mining has gotten broad attentions and numerous calculations are proposed. The current continuous item set mining calculations can be classified into two gatherings: applicant age based algorithms and design development based calculations. The up-and-comer age based calculations first generate up-and-comer item sets and these competitors are approved against the exchange table to recognize incessant item sets. The counter droning property is used in competitor age based calculations to prune search space. However, the up-and-comer-age based calculations require different pass table sjars and this will acquire a high I/O cost on huge information. The example development based calculations don't produce up-and-comers unequivocally. They build the unique tree-based information structures to keep the fundamental data about the incessant item sets of the exchange table. By utilization of the build ed information structures, the successive item sets can be figured productively. Notwithstanding, design development based calculations have the issue that the built information structures are unpredictable and typically surpass the accessible memory on monstrous information. To summarize, the current calculations can't process continuous item sets on gigantic information productively.

In successive item set mining, the quantity of the continuous item sets regularly is touchy to the estimation of the help limit. In the event that the help limit is little, there will be an enormous number of successive item sets and it is hard for the clients to settle on productive choices. Despite what might be expected, if the help edge is enormous, it is conceivable that no successive item sets can be found or the intriguing item sets may be missed. Along these lines, a legitimate help edge is essential for the reasonable continuous item set mining and the clients regularly need to perform incessant item set digging for a few times before the acceptable help edge is resolved. The cycle regularly is intuitive. On gigantic information, the current calculations regularly need a long execution time to process successive item sets and this will influence clients' working efficiency genuinely. The focal point of this paper is to locate another effective calculation to figure successive item sets on monstrous information rapidly.

One helpful stunt, which is embraced to accelerate the execution in the current calculations, is to reuse the work done in the tallying activity of the shorter item sets for that of the more drawn out item sets. In this paper, we need to use this reuse thought to an a lot bigger degree. In ordinary huge information applications, with the expanding information volume and the plate I/O bottleneck, information typically is put away in read/attach just mode. Accordingly, the general informational collection can be separated into two sections: the a lot bigger old informational collection putting away the recorded information, and the overall little new informational collection putting away the recently produced information. In light of the depiction over, this paper devises another PFIM calculation (Pre-computation-based Frequent Item set Mining calculation) on enormous information, which uses the pre-developed regular item sets on the old informational collection to return the successive item sets rapidly. Since the too little estimation of help limit will produce too many incessant item sets, we accept in this paper that there exists a lowerbound! of the help limit determined by the clients in down to earth applications. Due to the genuine/annex just mode, given the old table TO, PFIM first pre-builds the continuous item sets (allude to as semi incessant item sets in this paper) whose supports are no not exactly! The new exchanges are collected in the new table T . Taking advantage of the pre-built semi continuous item sets, given the predetermined help limit, PFIM can figure the regular item sets on TO [T rapidly. During the time spent execution of PFIM, three pruning rules are contrived in this paper to diminish the quantity of up-and-comer incessant item sets. A steady update procedure is proposed in this paper to rapidly refresh the semi regular item sets when To and T are blended. The broad examinations are led on engineered and genuine informational collections. The test results show that, PFIM outflanks the current calculations essentially; it runs two significant degrees quicker than the most recent calculation.

2. LITERATURE SURVEY

Candidate-Generation-Based Algorithms

The candidate-generation-based algorithms firstly generate the candidates of the frequent item sets, then the candidates are validated against the transaction table, and the frequent item sets are discovered.

Apriori algorithm adopts a level-wise execution mode. It uses the downward closure property, i.e. any super-set of an infrequent item set must also be infrequent, to prune the search space. By a pass of scan on the transaction table, it first counts the item occurrences to find the frequent 1-item sets F_1. Subsequently, the frequent k-item sets in F_k are used to generate the candidates C_{k+1} of the frequent (k + 1)-item sets. Another pass of scan is needed to compute the supports of candidates in C_{k+1} to find the frequent (k + 1)-item sets F_{k+1}. This process iterates similarly until the F_{k+1} is empty. Apriori algorithm often needs multiple passes over table, it will incur a high I/O cost on massive data.

Savasere et al. propose Partition algorithm to generate frequent item sets by reading the transaction table at most two times. The execution of Partition consists of two stages. In the first stage, Partition algorithm divides the table into a number of non-overlapping partitions in terms of the allocated memory, and the local frequent item sets for each partition are computed. All the local frequent item sets are merged at the end of first stage to generate the candidates of frequent item sets. In the second phase, another pass over table is performed to acquire the support of the candidates and the global frequent item sets can be discovered. The useful property adopted in Partition is that, every global frequent item sets must be appeared in local frequent item sets of at least one partition. Partition algorithm utilizes vertical table representation of transaction table and the support counting is performed by recursive TID (transaction identifier) list intersection. In the first phase, Partition may generate many false positives, i.e. the item sets are frequent locally but not frequent globally. Therefore, it needs another table scan to remove the false positives.

Zaki proposes another vertical mining algorithm E-clat. Eclat decomposes the original search space by a lattice-theoretic approach into smaller sublattices, each of which is a group of item sets with a common prefix (referred to as prefix-based equivalence class). Depending on the allocated memory size, Eclat can recursively partition large classes into smaller ones until each class can be maintained entirely in the memory. Then, each class is processed independently in the breath-first fashion to compute the frequent item sets. Eclat processes the sublattices sequentially one by one and does not need post-processing overhead as Partition algorithms. The main problem of Eclat is that when the intermediate results of vertical TID lists can become too large for memory, especially in dense database, the performance of Eclat starts to suffer. In order to solve the problem, Zaki et al. devise a novel vertical data representation called diffset, which keeps differences in the TIDs of a candidate pattern from its generating frequent patterns. A variation (dEclat) of Eclat by diffset is presented, which performs a depth-first search of the enumeration tree. By the incorporation of diffset, the memory requirement of dEclat is cut down drastically.

Deng et al. propose PPV algorithm to integrate the advantages of vertical mining and FPgrowth. PPV utilizes a coding prefix tree structure PPC-tree to store the table. Each node in PPC-tree is associated with pre-post code via the pre-order and post-order traversal on the PPC-tree. Each frequent item can be represented by a node-list, i.e. the list of PrePost code consisting pre-order code, post-order code and the count of nodes registering the frequent item. PPV fully uses candi-date generation to discover frequent item sets, i.e. the node-lists of the candidate item sets of

length (k + 1) are generated by intersecting node-lists of frequent item sets of length k, then the frequent item sets can be reported. PPV can achieve a high execution efficiency since (1) the node-list is more compact than the vertical structure, (2) the support counting is transformed into the intersection of node-lists, (3) the ancestor-descendant relationship of two nodes can be verified efficiently by their prepost codes. proposes PrePost to improve PPV. The core difference between PrePost and PPV is that PrePost can directly find frequent item sets without generating candidates in some cases by using the single path property of Nlist. points out that node-list and N-list need to encode each node of PPC-tree with both preorder code and post-order code, thus they are memory-consuming. A more efficient data structure, Node Set, is adopted, which only requires the pre-order code (or post-order code) of each node. And based on Node Set, an algorithm FIN is devised to compute frequent item sets. The algorithm dFIN is presented to improve FIN further. The algorithm dFIN uses an enhanced Node Set, DiffNode Set, which is combined by the idea of diffset. Aryabarzan et al. find that the calculation of the difference between DiffNode Set takes a long time on some tables. They propose a new data structure, NegNode Set, which also uses prefix tree.
NegNode Set employs a set-bitmaprepresentation-based encoding model for nodes. By using NegNode Set data structure, negFIN is proposed. Three key advantages of negFIN are: (1) employing bitwise operator to generate new sets of nodes,

Pattern-Growth-Based Algorithms Pattern-growth-based algorithms do not generate candidate item sets explicitly but compress the required information for frequent item sets in specific data structure. The frequent item sets can be acquired quickly with the notion of projected databases, a subset of the original transaction database relevant to the enumeration node.

Agarwal et al. present DepthProject algorithm to mine long item sets in databases. DepthProject examines the nodes of the lexicographic tree in depth-first order. The examination process of a node refers to the support counting of the candi-date extension of the node. During the search, the projected transaction sets are maintained for some of the nodes on the path from the root to the node P currently being extended. Normally, the projected transaction sets only contain the relevant part of the transaction database for counting the support at the node P. In the process of depth-first search, the projected database can be reduced further at the children of P and DepthProject can reuse the counting work of its previous exploration. At the lower levels of the lexicographic tree, a specialized counting technique called bucketing is used to substantially improve the counting time.

Han et al. propose a FP-tree-based FPgrowth algorithm to mine the complete set of frequent patterns by pattern fragment growth. FP-tree (frequent-pattern tree) is a compact prefix-based trie structure to store the essential information about frequent patterns. In each transaction, only frequent length-1 items, which are sorted with the descending order of support, are used to construct the FP-tree. Then the FP-growth algorithm works on FP-tree rather than on the original database to mine frequent patterns. FP-growth algorithm starts with a frequent length-1 pattern (initial suffix pattern), and the set of frequent items cooccurring with the suffix pattern is extracted as conditional-pattern base, which is then constructed as conditional FP-tree. With the current suffix pattern and the conditional FPtree, if the conditional FP-tree is not empty, FP-growth performs mining recursively. The frequent patterns are acquired by concatenating the new ones generated from the conditional FP-tree and the suffix pattern. FP-growth transforms the problem of finding long frequent patterns to looking for shorter ones and then concatenating the suffix. An additional optimization is proposed for FPgrowth, i.e. if all the nodes of the FP-tree lie on a single path, the frequent patterns can be generated by enumeration of all the combinations of the sub-paths with the support being the minimum support of the item sets contained in the sub-path.

Grahne et al. find out that about 80 percent of the CPU time in frequent item set mining is used for traversing FP-trees. A special data structure, FP-array, is devised. Given an item set of m items, FP-array is a (m 1) (m 1) matrix, where each element of the matrix corresponds to the counter of an ordered pair of items. By the special data structure, a new FPgrowth* is proposed, which can reduce the traversal time on FP-tree and speed up the FPgrowth method significantly.

3. PROPOSED WORK

Proposes a novel pre-computation-based frequent item set mining (PFIM) algorithm to compute the frequent item sets quickly on massive data. PFIM treats the transaction table as two parts: the large old table storing historical data and the relatively small new table storing newly generated data. PFIM first pre-constructs the quasi-frequent item sets on the old table whose supports are above the lower-bound of the practical support level. Given the specified support threshold, PFIM can quickly return the required frequent item sets on the table by utilizing the quasi-frequent item sets. Following methods are presented to improve the efficiency of the system in mining data sets very fast.

i. Intuitive Idea
ii. Pre-computation Operation
iii. Basic Process iv. Pruning Operation v. Update Operation

i. Intuitive Idea

Generally the number of frequent item sets is very sensitive to the value of minsup. If the value of minsup is too small, the number of frequent item sets will be so large that the users can become overwhelmed with too many results and it is difficult for users to find the really useful information from them. Therefore, in this paper, we assume that there exists a lower-bound for the value of minsup in practical applications. During the time interval between two consecutive merging, T_O remains unchanged and only T updates frequently. Under such circumstances, given the frequent item set mining with varying support thresholds, why not we keep the pre-computed item sets whose support values in T_O are no less than ! And only compute the required frequent item sets considering the existence of T. In this way, the work done for T_O can be reused for the entire frequent item set mining in a long enough time.

ii. Pre-computation Operation

This part describes the pre-computation operation to generate the required item sets on the large old transaction table To whose supports are no less than !. The required item sets here are referred to as quasi-frequent item sets, distinguishing from the frequent item sets with the support threshold minsup specified by users. Let tno be the number of transactions in TO and tn be the number of transactions in T . Since the size of TO is much large, usually exceeds the size of the allocated memory. Therefore, the process of pre-computing the quasi-frequent item sets consists of two stages: candidate generation and result refinement.

In the stage of candidate generation, we retrieve the transactions in To sequentially and maintain the retrieved transactions in an in-memory buffer BU F , whose size is set according to the size of the allocated memory. If BUF is full, we can compute the local quasi-frequent item sets in BUF by the current vertical frequent item set mining algorithms. The quasi-frequent item sets corresponding to current BUF are kept in a file. Then we empty BU F and continue the sequential scan for the next iteration. The process is similarly executed until all transactions in TO is retrieved and all local quasi-frequent item sets are generated.

In the stage of result refinement, we first read all the local quasi-frequent item sets into the memory. Then another sequential scan on T_O be performed to compute support count, i.e. the absolute occurrence number, for each local quasi-frequent item set.

iii. Basic Process

Sequential scan on new table:

PFIM first retrieves the transactions in T1. ∀ t1 ∈T1, let t1 be the currently retrieved transaction. ∀i∈t1, i is an item in t1, we increase the count of i by 1 (initial value is 0). Due to its relatively small size of T1 and the simple computation, this sequential scan can be executed quickly. We use an array cnt1 to keep these counts. ∀i ∈ U, cnt1[i] is the count of item i in T1, cnt1[i] = 0 if i does not appear in any transaction in T1. The value of mas1 is the maximum support count for all items in T1.

Sequential scan on quasi-frequent item sets

Then, PFIM begins to retrieve Fqf .∀t ∈ Fqf, let t be the currently retrieved quasi-frequent item set in Fqf. The quasi-frequent item sets in Fqf can be divided into three classes:

(1) definitely belonging to the frequent item sets,

(2) definitely not belonging to the frequent item sets,

(3) Possibly belonging to the frequent item sets.

Given|t.IS| = 1 and t.IS = {i}, if t.SUP+cnt1[i] ≥[n×minsup], t.IS is frequent, otherwise, t.IS is not frequent. Given|t.IS|≥ 2, if t.SUP ≥[n×minsup], t.IS is frequent obviously, otherwise, if t.SUP+mas1< [n×minsup], t.IS certainly not a frequent item set. In other cases, t may be a frequent item set, depending on the transactions in T1, and PFIM maintains t in a set
STCAD

Increase supports for item sets

When all quasi-frequent item sets are retrieved already, PFIM needs to increase the support counts of quasi-frequent item sets in STCAD by their counts in T1, this can be done by a sequential scan on T1., PFIM retrieves B transactions from T1. For the current iteration, the transactions maintained in memory are transformed into vertical representation, i.e. each item is associated with the list of identifiers (TID) of transactions containing the item. ∀t ∈STCAD and t.IS = {ij1,ij2,...,ija}, the number of transactions in BUF1 containing t.IS is |Ta =1 ijb .tlist|, where ijb.tlist is the TID list corresponding to the item ijb.

iv. Pruning Operation

PFIM can reuse the pre-computation result of T0 and reduce the execution cost significantly. In this part, we discuss how to improve PFIM further to speed up its execution by pruning operation.

One main part of the cost in PFIM is to compute the support counts of the item sets of STCAD in T1. Therefore, if we can reduce the number of item sets in STCAD in step 2, the counting cost in T1 can be decreased. Use the maximum count mas1 of the single item in T1 to determine the support count range of the possible frequent item sets. Obviously, if we can narrow down the support count range, the size of STCAD can be reduced. As described in the process of step 2, PFIM can determine directly whether the quasifrequent 1-itemsets in Fqf are frequent item sets. Therefore, STCAD only needs to maintain the quasi-frequent item sets which contain at least two items. At the end of step2, PFIM maintains the possible frequent item sets in STCAD

v. Update Operation

When the size of T1 reaches a certain threshold, for example, 5% of the size of TO, the transactions in T1 and TO are merged. At this point, the quasi-frequent item sets in Fqf needs to be updated also. But the total re-construction can be expensive. Therefore, in this paper, a new an incremental update strategy is proposed, which utilizes the existing information computed already, to speed up the update operation.

The goal of the update operation is to generate the quasi-frequent item sets on T given the support level ω. The local quasi-frequent item sets of TO are kept in Fqf, O, First we need to add the occurrences of the local quasi-frequent item sets of Fqf, O in T1. Then, the local quasifrequent item sets in Fqf, O, are written into the new file Fqf, the local quasi-frequent item sets of T1 are kept in Fqf ,1. In order to avoid duplicate computation, the local quasi-frequent item sets in Fqf, 1, which have been contained in Fqf, O, are removed before the support counting. The support counts of the local quasi-frequent item sets in Fqf, 1 are calculated by another scan on TO and T1. The number of the item sets in Fqf, 1 can be reduced significantly by the containment checking in Fqf, O. The computation cost of adding the support counts of item sets in T1 and TO can be lowered accordingly. Therefore, the incremental update strategy can run much faster than the total reconstruction strategy, which also is verified in the experiments.

4. CONCLUSION

This paper thinks about the issue of figuring incessant item sets on huge information. It is discovered that the current algorithms can't perform regular item set mining on huge information effectively. This paper uses reusing the work done beforehand and devises a precalculation based PFIM calculation to rapidly secure the incessant item sets on enormous information. The exchange table comprises of

two sections: the enormous old table and the generally little new table. By the semi regular item sets pre-registered on the old table, PFIM can report the incessant item sets on huge information effectively. Three pruning rules are proposed in this paper to accelerate the execution of PFIM. The gradual update technique is introduced to re-build the semi continuous item sets immediately when combining the old table and the new table. The broad test results show that PFIM has a huge performance advantage over the current
calculations

5. REFERENCES

[1] A. Ceglar and J.F. Roddick, "Association mining," ACM Comput. Surv., 38(2):5, 2006.

[2] H. Cheng, X. Yan, J. Han, and P.S. Yu, "Direct discriminative pattern min-ing for effective classification," in Proceedings of the 24th International Conference on Data Engineering, April 7-12, 2008, pp. 169–178.

[3] H. Wang, W. Wang, J. Yang, and P.S. Yu, "Clustering by pattern similarity in large data sets," in Proceedings of the 2002 ACM SIGMOD Internation-al Conference on Management of Data, June 3-6, 2002, pp. 394–405.

[4] Z. Li and Y. Zhou, "Pr-miner: automatically extracting implicit program-ming rules and detecting violations in large software code," in Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Soft-ware Engineering, September 5-9, 2005, pp. 306–315.

[5] J.T.L. Wang, M.J. Zaki, H. Toivonen, and D.E. Shasha, editors. "Data Mining in Bioinformatics," Springer, 2005.

[6] R. Agrawal, T. Imielinski, and A.N. Swami, "Database mining: A perfor-mance perspective," IEEE Trans. Knowl. Data Eng., vol. 5, no. 6, pp.914– 925, 1993.

[7] C.C. Aggarwal, "Data Mining - The Textbook," Springer, 2015.

[8] C.C. Aggarwal and J. Han, editors, "Frequent Pattern Mining," Springer, 2014.

[9] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: current status and future directions," Data Min. Knowl. Discov., vol. 15, no. 1, pp.55–86, 2007.

[10] R. Agrawal, T. Imielinski, and A.N.

Swami, "Mining association rules between sets of items in large databases," in Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 1993, pp. 207–216.

[11] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, 1994, pp. 487–499.

Author's Profiles

Ms.V.R. SWETHA currently working as Assistant Professor in Audisankara College of Engineering & Technology AUTONOMOUS Gudur, Tirupati (Dt), Andhra Pradesh, India.

Ms.P.POOJITHA is pursuing MCA from Audisankara College of Engineering & Technology (AUTONOMOUS), Gudur, Affiliated to JNTUA . Andhra Pradesh, India.